KATEDRA INFORMATYKI
TECHNICZNEJ
POLITECHNIKI WROCŁAWSKIEJ

Raport serii Sprawozdania
nr: / 2019

Programowanie aplikacji czasu rzeczywistego w systemie QNX6 Neutrino z wykorzystaniem platformy PC104 Vortex

Jędrzej UŁASIEWICZ

Słowa kluczowe:
- Systemy czasu rzeczywistego
- Systemy wbudowane
- Platforma Momentics
- System QNX 6 Neutrino
- POSIX 1003
- Komputer PC104

Wrocław 2019

Spis treści

1. Podstawy posługiwania się systemem QNX 6 Neutrino

1.1 Wstęp

System QNX6 Neutrino jest rozproszonym systemem czasu rzeczywistego. Charakteryzuje się on wysoką niezawodnością działania, prostotą konfiguracji użytkowania i programowania, małym zapotrzebowaniem na zasoby. System jest od podstaw zaprojektowany jako system sieciowy i wykonany w technologii mikrojądra (*ang. microkernel*). Nadaje się dobrze do tworzenia aplikacji rozproszonych i zastosowań krytycznych jak na przykład sterowanie procesami przemysłowymi. System w większości zgodny jest ze standardem POSIX . Toteż osoby mające pewną wprawę w posługiwaniu się systemem Linux nie powinny mieć większych problemów w pracy z systemem QNX6 Neutrino. Posługiwanie się systemem QNX6 opisane jest w [4], [5], [6], [9]. System QNX akceptuje większość poleceń i narzędzi znanych z systemu Linux. Tak więc osoba mająca pewną wprawę w posługiwaniu się systemem Linux nie powinna mieć kłopotów przy pracy z systemem QNX6. Poniżej podane zostały podstawowe informacje umożliwiające posługiwanie się systemem w zakresie uruchamiana prostych programów napisanych w języku C.

1.2 Instalacja systemu

System QNX6 Neutrino można bezpłatnie pobrać z witryny http://www.qnx.com/products/getmomentics/. Po zarejestrowaniu się na podany adres poczty elektronicznej przesyłany jest klucz aktywacyjny potrzebny przy instalacji systemu. System można zainstalować na:

1. Oddzielnej partycji dyskowej
2. W maszynie wirtualnej np. VMware lub innej .

System zajmuje w podstawowej konfiguracji około 300 MB przestrzeni dyskowej ale lepiej jest zarezerwować 2-3 GB.System pracuje poprawnie już na komputerze Pentium 600MHz z 256 MB pamięci RAM.

1.3 Uzyskiwanie pomocy

Dokumentacja systemu dostępna jest w postaci HTML lub PDF. Można ją oglądać za pomocą wchodzącej w skład systemu przeglądarki Mozilla. System pomocy uruchamia się klikając w ikonę z napisem `Help` umieszczoną na belce programów. Skrócony opis poleceń systemowych można uzyskać także pisząc w oknie terminala polecenie:

```
$ use nazwa_polecenia
```

1.4 Podstawowe polecenia

Pliki i katalogi

W systemie QNX6 Neutrino prawie wszystkie zasoby są plikami. Dane, urządzenia, bloki pamięci a nawet pewne usługi są reprezentowane przez abstrakcję plików. Mechanizm plików pozwala na jednolity dostęp do zasobów tak lokalnych jak i zdalnych za pomocą poleceń i programów usługowych wydawanych z okienka terminala. Plik jest obiektem abstrakcyjnym z którego można czytać i do którego można pisać. Oprócz zwykłych plików i katalogów w systemie plików widoczne są pliki specjalne. Zaliczamy do nich łącza symboliczne, kolejki FIFO, bloki pamięci, urządzenia blokowe i znakowe.
System umożliwia dostęp do plików w trybie odczytu, zapisu lub wykonania. Symboliczne oznaczenia praw dostępu do pliku dane są poniżej:

`r` - Prawo odczytu (*ang. read*)
`w` - Prawo zapisu (*ang. write*)
`x` - Prawo wykonania (*ang. execute*)

Prawa te mogą być zdefiniowane dla właściciela pliku, grupy do której on należy i wszystkich innych użytkowników.

`u` - Właściciela pliku (*ang. user*)
`g` - Grupy (*ang. group*)
`o` - Innych użytkowników (*ang. other*)

Polecenia dotyczące katalogów

Pliki zorganizowane są w katalogi. Katalog ma postać drzewa z wierzchołkiem oznaczonym znakiem /. Położenie określonego pliku w drzewie katalogów określa się za pomocą ścieżki. Rozróżnia się ścieżki absolutne i relatywne.

Ścieżka absolutna podaje drogę jaką trzeba przejść od wierzchołka drzewa do danego pliku. Przykład ścieżki absolutnej to `/home/juka/prog/hello.c`. Ścieżka absolutna zaczyna się od znaku /. Ścieżka relatywna zaczyna się od innego znaku niż /. Określa ona położenie pliku względem katalogu bieżącego. Po zarejestrowaniu się użytkownika w systemie katalogiem bieżącym jest jego katalog domowy. Może on być zmieniony na inny za pomocą polecenia `cwd`.

Uzyskiwanie nazwy katalogu bieżącego

Nazwę katalogu bieżącego uzyskuje się pisząc polecenie `pwd`. Na przykład:

```
$pwd
/home/juka
```

Listowanie zawartości katalogu

Zawartość katalogu uzyskuje się wydając polecenie `ls`. Składnia polecenia jest następująca:

```
ls [-l] [nazwa]
```

Gdzie:

`l` - Listowanie w „długim” formacie, wyświetlane są atrybuty pliku

`nazwa` - Nazwa katalogu lub pliku

Gdy nazwa określa pewien katalog to wyświetlona będzie jego zawartość. Gdy nazwa katalogu zostanie pominięta wyświetlana jest zawartość katalogu bieżącego. Listowane są prawa dostępu, liczba dowiązań, właściciel pliku, grupa, wielkość, data utworzenia oraz nazwa. Wyświetlanie katalogu bieżącego ilustruje Przykład 1-1.

```
$ls -l
-rwxrwxr-x 1 root root  7322 Nov 14  2003 fork3
-rw-rw-rw- 1 root root   886 Mar 18  1994 fork3.c
```

typ, właściciel, grupa, inni, liczba dowiązań, właściciel, grupa, wielkość, data utworzenia, nazwa

Przykład 1-1 Listowanie zawartości katalogu bieżącego.

Zmiana katalogu bieżącego

Katalog bieżący zmienia się na inny za pomocą polecenia `cd`. Składnia polecenia jest następująca: `cd nowy_katalog` .Gdy jako parametr podamy dwie kropki `..` to przejdziemy do katalogu położonego o jeden poziom wyżej. Zmianę katalogu bieżącego ilustruje Przykład 1-2.

```
$pwd
/home/juka
$cd   prog
$pwd /home/juka/prog
```

Przykład 1-2 Zmiana katalogu bieżącego

Tworzenie nowego katalogu

Nowy katalog tworzy się poleceniem `mkdir`. Polecenie to ma postać: `mkdir nazwa_katalogu`. Tworzenie nowego katalogu ilustruje Przykład 1-3.

```
$ls
prog
$mkdir src
$ls
prog src
```

Przykład 1-3 Tworzenie nowego katalogu

Kasowanie katalogu

Katalog kasuje się poleceniem `rmdir`. Składnia polecenia `rmdir` jest następująca: `rmdir nazwa_katalogu`. Aby możliwe było usuniecie katalogu musi on być pusty. Kasowanie katalogu ilustruje Przykład 1-4.

```
$ls
prog src
$rmdir src
$ls
prog
```

Przykład 1-4 Kasowanie katalogu

Polecenia dotyczące plików

Kopiowanie pliku

Pliki kopiuje się za pomocą polecenia `cp`. Składnia polecenia `cp` jest następująca:

```
cp [-ifR] plik_źródłowy plik_docelowy
cp [-ifR] plik_źródłowy katalog_docelowy
```

Gdzie:

`i` - Żądanie potwierdzenia gdy plik docelowy może być nadpisany.
`f` - Bezwarunkowe skopiowanie pliku.
`R` - Gdy plik źródłowy jest katalogiem to będzie skopiowany z podkatalogami.

Kopiowanie plików ilustruje Przykład 1-5.

```
$ls
nowy.txt prog
$ls prog
$
$cp nowy.txt prog
$ls prog
nowy.txt
```

Przykład 1-5 Kopiowanie pliku `nowy.txt` z katalogu bieżącego do katalogu `prog`

Zmiana nazwy pliku

Nazwę pliku zmienia się za pomocą polecenia `mv`. Składnia polecenia `mv` dana jest poniżej:

```
mv [-if] stara_nazwa nowa_nazwa
mv [-if] nazwa_pliku katalog_docelowy
```

Gdzie:

`i` - Żądanie potwierdzenia gdy plik docelowy może być nadpisany.
`f` - Bezwarunkowe skopiowanie pliku.

Zmianę nazwy plików ilustruje Przykład 1-6.

```
$ls
stary.txt
$mv stary.txt nowy.txt
$ls
nowy.txt
```

Przykład 1-6 Zmiana nazwy pliku `stary.txt` na `nowy.txt`

Kasowanie pliku

Pliki kasuje się za pomocą polecenia `rm`. Składnia polecenia rm jest następująca:
`rm [-Rfi] nazwa`

Gdzie:
`i` - Żądanie potwierdzenia przed usunięciem pliku.
`f` - Bezwarunkowe kasowanie pliku.
`R` - Gdy nazwa jest katalogiem to kasowanie zawartości wraz z podkatalogami.

Kasowanie nazwy pliku ilustruje Przykład 1-7.

```
$ls
prog nowy.txt
$rm nowy.txt
$ls
prog
```

Przykład 1-7 Kasowanie pliku `nowy.txt`

Listowanie zawartości pliku

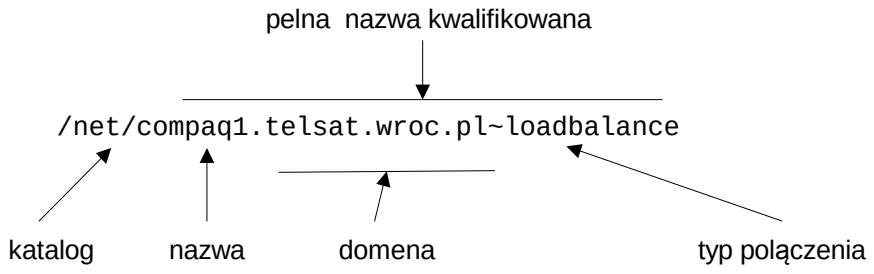
Zawartość pliku tekstowego listuje się za pomocą poleceń: `more nazwa_pliku`, `less nazwa_pliku`, `cat nazwa_pliku`. Można do tego celu użyć też innych narzędzi jak edytor `vi`, edytor `ped` lub wbudowany edytor programu `Midnight Commander`.

Używanie dysków Flash USB

System QNX6 Neutrino umożliwia korzystanie z dysków Flash USB. Po włożeniu takiego dysku do gniazda USB widziany on jest w katalogu `/fs` np. `hd10-dos-1`.

Sieć Qnet

Sieć Qnet jest rodzimą siecią systemu QNX6 Neutrino zapewniającą wysoki stopień integracji poszczególnych węzłów systemu. Każdy z komputerów połączonych siecią posiada swoją nazwę. Jego system plików widziany jest w katalogu **`/net`** komputera bieżącego.



Rys. 1-1 Struktura nazwy w sieci Qnet

Listę dostępnych węzłów można uzyskać za pomocą polecenia:

`ls /net`

Zdalne wykonanie programu

System umożliwia zdalne wykonanie programu na innym komputerze podłączonym do sieci QNET. Służy do tego celu polecenie on.

```
on –f nazwa_węzła polecenie
```

```
on –n nazwa_węzła polecenie
```

Podana wyżej konstrukcja powoduje że na węźle o nazwie nazwa_węzła wykonane zostanie polecenie. W opcji –f domyslnym systemem plików będzie system plików komputera zdalnego. W opcji –n domyslnym systemem plików będzie system plików komputera lokalnego.

```
# hostname
koziol
# on -f pc104-komp5 hostname
pc104-komp5
# _
```

Przykład 1-1 Lokalne i zdalne wykonanie (na komputerze pc104-komp5) polecenia hostname podającego nazwę węzła bieżącego.

Uzyskiwanie informacji o stanie systemu

Uruchamiając i testując programy potrzebujemy niejednokrotnie różnych informacji o stanie systemu. Dotyczy to w szczególności informacji o uruchomionych procesach, wątkach i użytych zasobach. Zestawienie ważniejszych programów i poleceń używanych do uzyskiwania informacji o stanie systemu pokazuje Tabela 1-1.

Graficzna informacja o procesach i wątkach.	`psin`
Tekstowa informacja o procesach i wątkach.	`pidin`
Informacja o procesach.	`ps`
Informacja o wykorzystaniu procesora przez procesy.	`hogs`

Tabela 1-1 Polecenia do uzyskiwania informacji o stanie systemu

Większość informacji o stanie systemu można uzyskać za pomocą programu `pidin`. Składnia polecenia jest następująca:

`pidin [`*`opcje`*`] funkcja`

Funkcja	Znaczenie
`arguments`	Argumenty uruchomienia
`environment`	Otoczenie procesu
`fds`	Deskkryptory otwartych plików dla procesów
`info`	Informacje o systemie
`irqs`	Informacje o handlerach obsługi przerwań
`mapinfo`	Informacje o mapie pamięci
`memory`	Informacje o segmentach pamięci uzywanych przez procesy
`net`	Informacje o sieci
`rc`	Nazwy procesów i węzłów dołączonych do węzła bieżącego
`sched`	Priorytety, starategie szeregowania i stany procesów/wątków
`signals`	Stany obsługi sygnałów
`threads`	Wyświetlenie wątków
`timers`	Wyświetlenie timerów
`times`	Wyświetlenie informacji o czasach zużycia procesora dla procesów
`users`	Wyświetlenie użytkowników i danych o nich

Tabela 1-2 Funkcje polecenia `pidin`

Przykład polecń `pidin` podano poniżej.

```
# pidin info
CPU:X86 Release:6.5.0  FreeMem:201Mb/255Mb BootTime:Oct 01 22:11:39 CEST 2019
Processes: 32, Threads: 93
Processor1: 586 Vortex86 SoC 586 F5M2S2 800MHz FPU
```

Przykład 1-8 Polecenie pidin info podaje informacje o systemie

Przykład polecenia `pidin net` podającego węzły systemu sieci QNET podano poniżej.

```
# pidin net
 ND    Node              CPU      Release FreeMem       BootTime
 0     new-host-3        1 X86    6.5.0   201Mb/255Mb   Oct 01 22:11:39 CEST 2019
           Processes: 34, Threads: 95
           CPU 1:         586 Vortex86 SoC 586...     800MHz FPU
 1     new-host-2        2 X86    6.5.0   837Mb/1023Mb  Oct 01 14:23:22 CEST 2019
           Processes: 37, Threads: 108
           CPU 1:       66222 Pentium III Step...    2340MHz FPU
           CPU 2:       66222 Pentium III Step...    2315MHz FPU
```

Przykład 1-9 Polecenie pidin net podaje węzły systemu QNX

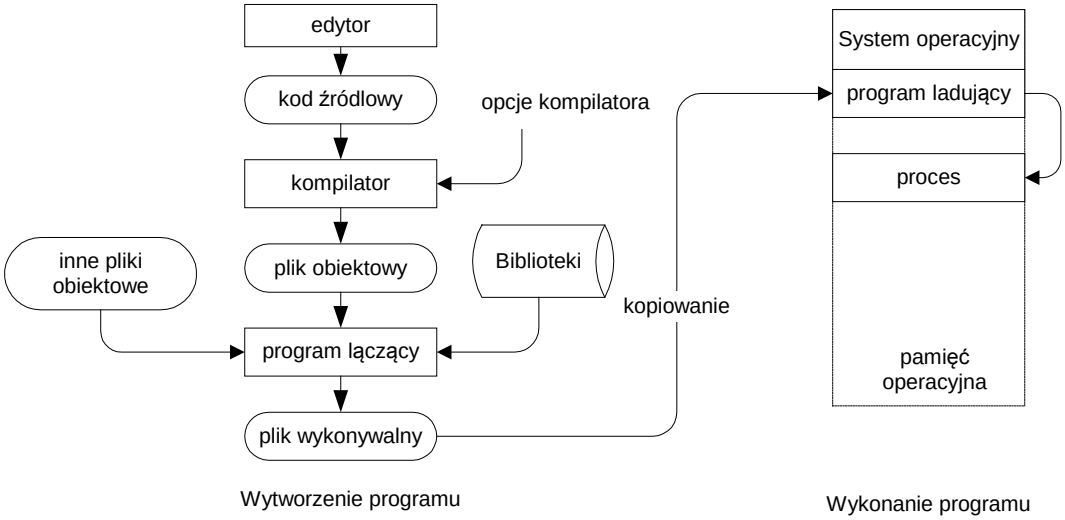
Program `psin`

Wiele informacji u stanie systemu uzyskać można za pomocą programu `psin` Program ten uruchamia się wybierając ikonę `System Infor...` umieszczoną na belce programów lub pisząc polecenie `psin` w oknie terminala (nie jest dostępny we wszystkich wersjach systemu) .

2. Kompilacja i uruchamianie programów

2.1 Jak kod źródłowy przekształca się w proces

Kod aplikacji tworzony jest zazwyczaj w języku wysokiego poziomu, my posługiwać się będziemy językiem C. W języku wysokiego poziomu tworzy się tak zwany kod źródłowy który po zapisaniu będzie plikiem z programem źródłowym. Plik źródłowy przetwarzany jest następnie przez kompilator do tak zwanego programu wykonywalnego. Jeżeli procesor (i ewentualnie system operacyjny) na którym kompilowany jest program jest inny niż ten na którym jest wykonywany to kompilator nazywa się kompilatorem skrośnym (ang. *cross compiler*). Następnie program wykonywalny przekształcany jest w wykonujący się proces, co wykonywane jest przez program ładujący systemu operacyjnego (ang. *loader*). Proces tworzenia i wykonywania programu pokazany jest na poniższym Rys. 2-1. Program wykonywalny może się wykonać na tej samej maszynie na której został utworzony, lub tak jak się to dzieje w systemach wbudowanych, jest on przesyłany do systemu docelowego i tam dopiero wykonany.



Rys. 2-1 Przebieg procesu wytworzenia i wykonania programu

Przetworzenie kodu źródłowego w wykonywany proces odbywa się w kilku etapach. Najważniejsze z nich to kompilacja, łączenie i ładowanie programu.

2.2 Kompilacja

Podstawowym narzędziem przekształcającym kod źródłowy zrozumiały dla procesora w kod wykonywalny jest kompilator. Celem kompilacji jest transformacja kodu źródłowego będącego zapisem algorytmu w języku wysokiego poziomu (który nie może być wykonany przez procesor) na kod maszynowy danego procesora. Kompilacja przebiega w kilku etapach i prowadzi ona do wytworzenia tak zwanego pliku obiektowego. Plik obiektowy zawiera kod maszynowy właściwy dla procesora na którym kod będzie wykonywany i informacje dodatkowe. Typowy plik obiektowy składa się z takich części jak: nagłówek, kod maszynowy, dane, tablica symboli, informacje o relokacji, informacje dla programu uruchomieniowego (ang. *debugger*). Na etapie kompilacji nie sposób określić pod jaki adres w pamięci należy załadować utworzony program, gdyż kompilator nie posiada informacji o stanie pamięci procesora w chwili wykonania programu. Stąd pliki obiektowe i wykonywalne zawierają tak zwaną tablicę relokacji (ang. *relocation table*) . Składa się ona z pozycji, z których każda zawiera wskaźnik do adresu w kodzie obiektowym, który musi być zmodyfikowany w procesie ładowania programu do pamięci operacyjnej. W systemie Linux plik obiektowy jak i wykonywalny tworzony jest w tak zwanym formacie ELF (ang. *Executable and Linkable Format*). Informacje o plikach w formacie ELF uzyskać można za pomocą narzędzi Linuksowych takich jak `readelf i objdump`, . Istnieje wiele kompilatorów języka C ale my używać będziemy standardowego narzędzia kompilacyjnego systemu Linux czyli kompilatora `gcc` .

2.3 Łączenie

Plik obiektowy zawiera tłumaczenie kodu źródłowego na instrukcje kodu maszynowego i dane na których te instrukcje operują ale nie jest jeszcze kompletnym programem gdyż nie zawiera bibliotek i być może innych segmentów programu. Kompletny program wykonywalny powstanie na etapie łączenia. Operację łączenia wykonuje program nazywany konsolidatorem lub linkerem (ang. *linker*). Konsolidator dołącza programu głównego inne pliki obiektowe i biblioteki w wyniku czego powstaje program wykonywalny. Jest on także w formacie ELF. W systemie Linux rolę linkera pełni program `ld`.

2.4 Ładowanie programu

Plik wykonywalny (nazywany też plikiem binarnym) kopiowany jest następnie w miejsce przeznaczenia. Może to być inny folder w tym samym komputerze, inny komputer lub też system wbudowany. Kolejną czynnością która musi być wykonana jest utworzenie procesu na podstawie pliku wykonywalnego. Czynność tę wykonuje program ładujący (ang. *loader*). Funkcje programu ładującego to:

- Weryfikacja pozwoleń, wymagań na zasoby
- Skopiowanie segmentów programu do pamięci operacyjnej
- Skopiowanie argumentów linii poleceń na stos
- Inicjalizacja rejestrów procesora
- Przekazanie sterowania do punktu startowego programu

Po wykonaniu powyższych czynności program zostaje przekształcony w proces i przystępuje do wykonywania swojej funkcji. W dalszej części tego rozdziału przedstawimy sposoby kompilacji programów i opiszemy stosowane do tego celu narzędzia.

2.5 Edycja programów

Do uzyskania praktycznych doświadczeń z systemem niezbędna jest umiejętność edycji, kompilacji i uruchamiania programów. Edytor `ped` uruchamia się wybierając go z grupy `Utilities` na belce startowej. Można używać także edytora `vi` lub `WorkSpace` (polecenie: `ws`). Naukę tworzenia programów rozpoczniemy od tradycyjnego programu `hello.c`. Najpierw uruchamiamy edytor `ped` a następnie wpisujemy tekst programu `hello` który pokazuje Przykład 2-1.

```
#include <stdlib.h>
void main(void) {
  printf("Pierwszy program w QNX Neutrino !\n");
}
```

Przykład 2-1 Program `hello.c`

Po wpisaniu programu zachowujemy go wybierając opcje `File / Save As` i podając nazwę pliku `hello.c`. Następnie otwieramy okno terminala i poprzez polecenie `ls` sprawdzamy czy plik `hello.c` istnieje w katalogu bieżącym. Gdy tak to program kompilujemy pisząc: `$gcc hello.c –o hello`. Gdy kompilator wykryje błędy to je poprawiamy. Następnie uruchamiamy program pisząc: `$./hello`. Wykonane działania pokazuje poniższy przykład.

2.6 Kompilacja programów za pomocą narzędzia make

Opisane wyżej metoda kompilacji programów jest odpowiednia dla prostych aplikacji składających się z jednego programu utworzonego z jednego bądź niewielkiej liczby plików źródłowych. Jednak w praktyce najczęściej mamy do czynienia z bardziej zaawansowanymi aplikacjami. Aplikacje te składają się z wielu programów a te z kolei składają się z wielu plików. W trakcie ich uruchamiania modyfikujemy niektóre z nich. Następnie musimy uruchomić aplikację aby sprawdzić efekty wprowadzonych zmian. Powstaje pytanie które pliki skompilować i połączyć aby wprowadzone w plikach źródłowych zmiany były uwzględnione a aplikacja aktualna. Z pomocą przychodzi nam narzędzie `make` powszechnie stosowane w tworzeniu złożonych aplikacji. W praktyce programistycznej typowa jest sytuacja gdy aplikacja składa się z pewnej liczby programów wykonywalnych zawierających jednak pewne wspólne elementy (stałe, zmienne, funkcje). Pokazuje to poniższy przykład. Aplikacja składa się z dwóch programów - `pierwszy` i `drugi`. Każdy z programów wypisuje na konsoli swoją nazwę i w tym celu korzysta z funkcji `void pisz(char * tekst)` zdefiniowanej w pliku `wspolny.c` a jej prototyp zawarty jest w pliku `wspolny.h`. Sytuację pokazuje Rys. 2-2 . Aby skompilować aplikację należy dwukrotnie wpisać polecenia kompilacji np. tak jak poniżej:

```
$gcc pierwszy.c wspolny.c –o pierwszy
$gcc drugi.c wspolny.c –o drugi
```

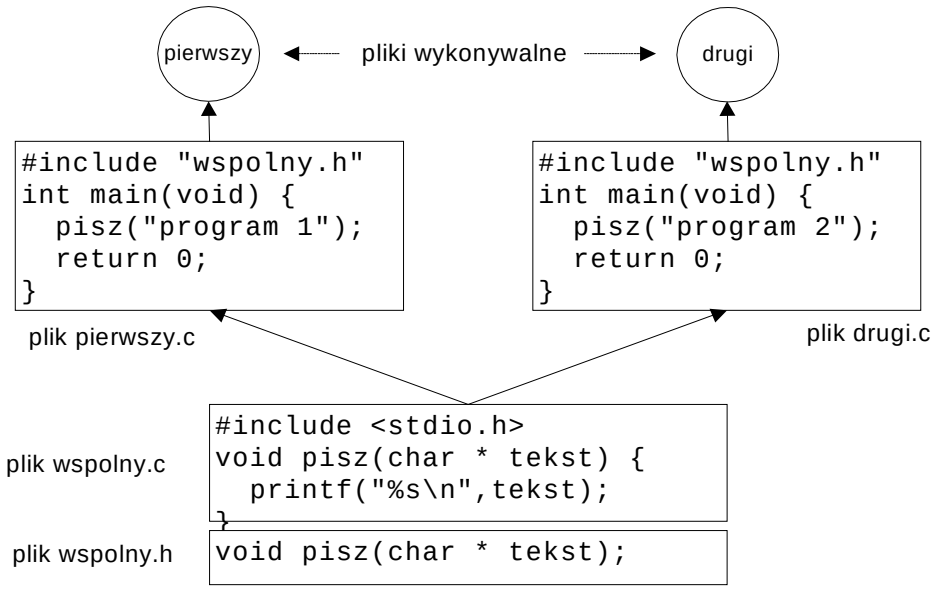
Analogiczny efekt osiągnąć można tworząc plik definicji `makefile` dla narzędzia `make` a następnie pisząc z konsoli polecenie `make`. Narzędzie `make` opisane jest obszernie w literaturze [13], [14] i dokumentacji systemu. Plik `makefile` dla powyższego przykładu pokazany jest poniżej. Należy zauważyć że plik `makefile` i pliki źródłowe muszą być umieszczone w jednym folderze. Po wpisaniu polecenia make system szuka w folderze bieżącym pliku o nazwie `Makefile` a następnie `makefile`. Gdy chcemy aby miał inną nazwę wpisujemy ja jako parametr polecenia `make:`

```
make –f nazwa_pliku.
```

```
# Plik makefile dla aplikacji skladajacej się z dwoch programow
all: pierwszy drugi
pierwszy: pierwszy.c wspolny.c wspolny.h
      gcc -o pierwszy pierwszy.c wspolny.c
drugi: drugi.c wspolny.c wspolny.h
      gcc -o drugi drugi.c wspolny.c
```

Przykład 2-2 Plik `makefile` dla aplikacji składającej się z dwóch plików

Natomiast wyniki działania polecenia `make` pokazuje .



Rys. 2-2 Aplikacja składająca się z dwóch programów

```
# gcc pierwszy.c wspolny.c -o pierwszy
# gcc drugi.c wspolny.c -o drugi
# ./pierwszy
 Program pierwszy
# ./drugi
 Program drugi
```

Przykład 2-3 Działanie polecenia `make`

Plik definicji `makefile` składa się z zależności i reguł. Zależność podaje jaki plik ma być utworzony i od jakich innych plików zależy. Na podstawie zależności program `make` określa jakie pliki są potrzebne do kompilacji, sprawdza czy ich kompilacja jest aktualna - jeśli tak, to pozostawia bez zmian, jeśli nie, sam kompiluje to co jest potrzebne. Nawiązując do omawianego przykładu występuje tam definiująca taką zależność linia:

```
pierwszy: pierwszy.c wspolny.c wspolny.h
```

Informuje ona system że plik `pierwszy` zależy od plików `pierwszy.c wspolny.c wspolny.h` toteż jakakolwiek zmiana w tych plikach spowoduje konieczność powtórnego tworzenia pliku `pierwszy`. Natomiast reguły mówią jak taki plik utworzyć. W tym przykładzie aby utworzyć plik wykonywalny `pierwszy` należy uruchomić kompilator z parametrami jak poniżej.

```
gcc -o pierwszy pierwszy.c wspolny.c
```

Należy zwrócić uwagę że powyższa linia zaczyna się niewidocznym znakiem tabulacji. W plikach `makefile` umieszczać można linie komentarza poprzez umieszczenie na pierwszej pozycji takiej linii znaku #. Gdy do programu `make` wpiszemy parametr będący nazwą pewnej zależności można spowodować wykonanie reguły odpowiadające tej zależności. Do poprzedniego pliku makefile dodać można regułę o nazwie `archiw` wykonania archiwizacji plików źródłowych co pokazuje Przykład 2-4. Wpisanie polecenia make archiw spowoduje wykonanie archiwum plików źródłowych i zapisanie ich w pliku `prace.tgz`. Narzędzie `make` ma znacznie więcej możliwości ale nie będą one potrzebne w dalszej części laboratorium i zostaną pominięte.

```
all: pierwszy drugi
pierwszy: pierwszy.c wspolny.c wspolny.h
      gcc -o pierwszy pierwszy.c wspolny.c
drugi: drugi.c wspolny.c wspolny.h
      gcc -o drugi drugi.c wspolny.c
archiw: pierwszy.c drugi.c wspolny.c wspolny.h
      tar -cvf prace.tar *.c *.h makefile
      gzip prace.tar
      mv prace.tar.gz prace.tgz
```

Przykład 2-4 Plik `make` z opcją archiwizacji plików źródłowych

```
# rm pierwszy
# rm drugi
# make
gcc -o pierwszy pierwszy.c wspolny.c
gcc -o drugi drugi.c wspolny.c
# make archiw
tar -cvf prace.tar *.c *.h makefile
drugi.c
pierwszy.c
wspolny.c
wspolny.h
makefile
gzip prace.tar
mv prace.tar.gz prace.tgz
```

Przykład 2-5 Kompilacja za pomoca narzędzia make

2.7 Uruchamianie programów za pomocą narzędzia gdb

Rzadko zdarza się by napisany przez nas program od razu działał poprawnie. Na ogół zawiera wiele błędów które trzeba pracowicie poprawiać. Typowy cykl uruchamiania programów polega na ich edycji, kompilacji i wykonaniu. Przy kompilacji mogą się ujawnić błędy kompilacji które wymagają poprawy a gdy program skompiluje się poprawnie przystępujemy do jego wykonania. Często zdarza się że uruchamiany program nie zachowuje się w przewidywany przez nas sposób. Wówczas należy uzyskać dodatkowe informacje na temat:

- Ścieżki wykonania programu
- Wartości zmiennych a ogólniej zawartości pamięci związanej z programem

Informacje takie uzyskać można na dwa sposoby:

- Umieścić w kodzie programu dodatkowe instrukcje wyprowadzania informacji o przebiegu wykonania i wartości zmiennych.
- Użyć programu uruchomieniowego (ang. *Debugger*)

Gdy używamy pierwszej metody, dodatkowe informacje o przebiegu wykonania programu są zwykle wypisywane na konsoli za pomocą instrukcji `printf` lub też zapisywane do pliku. Po uruchomieniu programu, instrukcje wypisywania dodatkowych informacji są z programu usuwane. Użycie pierwszej metody jest w wielu przypadkach wystarczające. Jednak w niektórych, bardziej skomplikowanych przypadkach, wygodniej jest użyć programu uruchomieniowego. Program taki daje następujące możliwości:

- Uruchomienie programu i ustawienie dowolnych warunków jego wykonania (np. argumentów, zmiennych otoczenia, itd)
- Doprowadzenie do zatrzymania programu w określonych warunkach.
- Sprawdzenie stanu zatrzymanego programu (np. wartości zmiennych, zawartość rejestrów, pamięci, stosu)
- Zmiana stanu programu (np. wartości zmiennych) i ponowne wznowienie programu.

W świeciesystemów klasy POSIX , szeroko używanym programem uruchomieniowym jest `gdb` (ang. *gnu debugger*) [15]który jest częścią projektu GNU Richarda Stallmana. Może on być użyty do uruchamiania programów napisanych w językach C, C++, assembler, Ada , Fortran, Modula-2 i częściowo OpenCL. Program działa w trybie tekstowym, jednak większość środowisk graficznych IDE takich jak Eclipse czy CodeBlocks potrafi się komunikować z `gdb` co umożliwia pracę w trybie okienkowym. Istnieją też środowiska graficzne specjalnie zaprojektowane do współpracy z `gdb` jak chociażby DDD (ang. *Data Display Debugger*). Program gdb posiada wiele możliwości i obszerną dokumentację podaną w [15] a tutaj podane zostaną tylko najważniejsze polecenia.

Kompilacja programu

Aby możliwe było uruchamianie programu z użyciem `gdb` testowany program należy skompilować z kluczem: `-g`. Użycie tego klucza powoduje że do pliku obiektowego z programem dołączona zostanie informacja o typach zmiennych i funkcji oraz zależność pomiędzy numerami linii programu a fragmentami kodu binarnego. Rozważmy przykładowy program `test.c` podany w Przykład 2-1. Aby skorzystać z debuggera program `test.c` należy skompilować następująco: `gcc test.c -o test -g`

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void) {
      int i,j;
      puts("Witamy w Lab PRW");
      system("hostname");
      for(i=0;i<10;i++) {
         j=i+10;
         printf("Krok  %d\n",i);
         sleep(1);
      }
      printf("Koniec\n");
      return EXIT_SUCCESS;
}
```

Przykład 2-1 Program test.c

2.7.1.1 Uruchomienie i zakończenie

Program `gdb` uruchamia się w następujący sposób:

```
gdb [opcje] [prog [obraz-pam lub pid]]
gdb [opcje] --args prog [argumenty_prog ...]
```

gdzie:

`opcje`	Opcje programu które można uzyskać pisząc: `gdb --h`
`prog`	Nazwa pliku wykonywalnego
`obraz-pam`	Obraz pamięci utworzony przy awaryjnym zakończeniu programu
`pid`	Pid procesu do którego chcemy się dołączyć
`args`	Argumenty programu

Najprostszy sposób uruchomienia debuggera `gdb` w celu uruchamiania programu zawartego w pliku `prog` to napisanie na konsoli polecenia: `gdb prog`

Można też dołączyć się do już działającego programu. W tym celu po nazwie programu należy podać jego `pid` co pokazuje `Tabela 2-1`. Program `gdb` może też służyć do analizy przyczyny awaryjnego zakończenia programu. Gdy proces jest kończony, na skutek otrzymania jednego z pewnych sygnałów, system operacyjny tworzy plik zawierający obraz pamięci procesu. Obraz ten może być analizowany przez `gdb` w celu znalezienia przyczyny awaryjnego zakończenia procesu. Aby dokonać analizy procesu `prog` który został awaryjnie zakończony, a jego oraz pamięci został zapisany w pliku `core`, `gdb` uruchamia się jak następuje: `gdb prog core`. Jeszcze jeden wariant uruchomienia programu `gdb` pozwala na podanie argumentów uruchamianego programu. Gdy program `prog` należy uruchomić z argumentami `a1 a2 ... an` to wtedy `gdb` należy uruchomić z opcją `--arg` jak następuje: `gdb --arg prog a1 a2 ... an`. Typowe sposoby uruchomienia programu `gdb` podaje `Tabela 2-1`.

`gdb prog`	Zwykłe uruchomienie `gdb` dla programu zawartego w pliku `prog`
`gdb prog core`	Uruchomienie `gdb` dla programu z pliku `prog`. Obraz pamięci znajduje się w pliku core.
`gdb prog pid`	Dołączenie się do działającego programu, oprócz nazwy podajemy też jego `pid`
`gdb --args prog argumenty`	Uruchomienie `gdb` dla programu z argumentami
`gdb -help`	Uzyskiwanie pomocy

Tabela 2-1 Różne sposoby uruchomienia programu gdb

Program `gdb` kończy się wpisując polecenie `quit`, skrót `q` lub też kombinację klawiszy `Ctrl+d`.

Możemy uruchomić program gdb w celu testowania podanego w programu test. W tym celu piszemy polecenie:

```
$gdb test
```

Po wpisaniu tego polecenia zgłasza się program `gdb` i oczekuje na wprowadzenie poleceń.

Uzyskiwanie pomocy

Program `gdb` posiada znaczną liczbę poleceń. Wpisując polecenie `help` uzyskujemy zestawienie kategorii poleceń, wpisując polecenie `help all` uzyskujemy zestawienie wszystkich poleceń.

Listowanie programu źródłowego

Uzyskanie fragmentu kodu źródłowego następuje przez użycie polecenia list. Polecenie to występować może w różnych wariantach co pokazuje `Tabela 2-2`.

`list`	Listowanie fragmentu kodu źródłowego począwszy od bieżącej pozycji
`list nr`	Listowanie fragmentu kodu źródłowego w pobliżu linii o numerze `nr`
`list pocz, kon`	Listowanie fragmentu kodu źródłowego od linii `pocz` do linii `kon`
`list +`	Listowanie nastepnego fragmentu kodu źródłowego (od pozycji bieżącej)
`list -`	Listowanie poprzedniego fragmentu kodu źródłowego (od pozycji bieżącej)

Tabela 2-2 Polecenia listowania fragmentu kodu źródłowego

Gdy mamy już uruchomiony gdb i testujemy program test możemy wylistować fragment kodu źródłowego pisząc polecenie list jak pokazuje `Ekran 2-1`.

```
(gdb) list
1
2       #include <stdio.h>
3       #include <stdlib.h>
4       #include <unistd.h>
5
6       int main(void) {
7               int i,j;
8               puts("Witamy w Lab PRW");
9               for(i=0;i<20;i++) {
10                  j=i+10;
(gdb)
```

Ekran 2-1 Listowanie kodu źródłowego

Zatrzymywanie procesu

Testowanie programów polega zwykle na próbie ich wykonania i zatrzymania, po czym następuje zbadanie stanu programu. Momenty zatrzymania określane przez tak zwane punkty zatrzymania (ang. *breakpoint*). Są trzy metody zdefiniowania punktu zatrzymania:

- Wskazanie określonej linii w kodzie programu lub też nazwy fukcji. Jest to zwykły punkt zatrzymania.
- Zatrzymanie programu gdy zmieni się wartość zdefiniowanego przez nas wyrażenia (ang. *watchpoint*).
- Zatrzymanie programu gdy zajdzie określone zdarzenie (ang. catchpoint) jak wystąpienie wyjątku czy załadowanie określonej biblioteki.

Tworzonym punktom zatrzymania nadawane są kolejne numery począwszy od 1. Po utworzeniu mogą one być aktywowane (ang *enable*), dezaktywowane (ang. *disable*) i kasowane (ang. *delete*). Najprostszym sposobem ustawienia punktu zatrzymania jest użycie polecenia: `break nr_lini`. Następuje wtedy ustawienie punktu zatrzymania w danej linii programu. Polecenie: `info break` powoduje wypisanie ustawionych punktów wstrzymania. Możemy teraz, w naszym przykładowym programie, ustawić punkt zatrzymania na linii 10 co robimy poleceniem: `break 10` jak pokazuje `Ekran 2-2`. Polecenie `info break` podaje ustawione punkty zatrzymania.

```
(gdb) break 10
Breakpoint 1 at 0x8048453: file test.c, line 10.
(gdb) info break
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x08048453 in main at test.c:10
(gdb)
```

Ekran 2-2 Ustawienie punktu zatrzymania

Punkty wstrzymania można kasować za pomocą polecenia: `clear nr_linii`.
Polecenie `break` występuje w wielu wariantach co opisane jest w dokumentacji [15]. Między innymi można ustawić zatrzymanie procesu gdy spełniony jest pewien warunek. Polecenie warunkowe ma wtedy postać:
`break nr_linii if warunek` którego skutkiem będzie zatrzymanie procesu w danej linii gdy warunek będzie spełniony. Przykładowo możemy ustawić punkt zatrzymania na linii 10 gdy zmienna i w programie z osiągnie wartość większą niż 5. Polecenie będzie miało postać: `break 10 if i > 5`. Proces ustawiania warunkowego punktu zatrzymania pokazuje Ekran 2-3.

```
(gdb) break 10 if i > 5
Breakpoint 2 at 0x8048453: file test.c, line 10.
(gdb) i b
Num     Type           Disp Enb Address    What
2       breakpoint     keep y   0x08048453 in main at test.c:10
        stop only if i > 5
(gdb)
```

Ekran 2-3 Ustawienie warunkowego punktu zatrzymania

Uruchamianie procesu

Jeżeli w programie ustawiono punkty zatrzymania to można go uruchomić. Wykonuje się to poprzez polecenie `run`.

```
(gdb) run
Starting program: /home/juka/prog/beagle/test
Witamy w Lab PRW
Krok  0
Krok  1
Krok  2
Krok  3
Krok  4
Krok  5

Breakpoint 1, main () at test.c:10
10                  j=i+10;
```

Ekran 2-4 Wykonanie programu do punktu zatrzymania

Listowanie programu	`list [numer linii]`	`skrót`
Uruchomienie programu	`run [argumenty]`	

Ustawienie punktu zatrzymania	`break nr_linii`
	`break nazwa_funkcji`
	`break nr_linii if warunek`
		`b`

Ustawienie pułapki, program zatrzyma się gdy zmieni się wartość obserwowanej zmiennej	`watch nazwa_zmiennej`	`w`
Listowanie punktów zatrzymania	`info break`	
Kasowanie punktu zatrzymania	`clear nr_linii`	
Wyprowadzenie wartości zmiennych	`print nazwa_zmienne`	`p`
Kontynuacja do następnego punktu wstrzymania	`continue`	`c`
Wykonanie następnej instrukcji, nie wchodzimy do funkcji	`next`	`n`
Wykonanie następnej instrukcji, wejście do funkcji	`step`	`s`
Uzyskanie pomocy	`help`	`h`
Ustawienie wartości zmiennej `var` na `wart`	`set var=wart`	
Zakończenie pracy programu	`quit`	`q`

Tabela 2-3 Najczęściej używane polecenia programu uruchomieniowego gdb

- b main - Put a breakpoint at the beginning of the program
- b - Put a breakpoint at the current line
- b N - Put a breakpoint at line N
- b +N - Put a breakpoint N lines down from the current line
- b fn - Put a breakpoint at the beginning of function "fn"

- d N - delete breakpoint number N
- info break - list breakpoints
- r - Run the program until a breakpoint or error
- c - continue running the program until the next breakpoint or error
- f - Run until the current function is finished
- s - run the next line of the program
- s N - run the next N lines of the program
- n - like s, but don't step into functions
- u N - run until you get N lines in front of the current line
- p var - print the current value of the variable "var"
- bt - print a stack trace
- u - go up a level in the stack
- d - go down a level in the stack
- q - Quit gdb

2.8 Zadania

Zadanie 2. 4. 1 Uzyskiwanie informacji o stanie systemu

Uruchom systemowy inspektor procesów `psin`. Zobacz jakie procesy i wątki wykonywane są aktualnie w systemie. Zbadaj zasoby zajmowane przez proces `io-net`. Porównaj uzyskane informacje i informacjami uzyskanymi za pomocą poleceń: `pidin`, `sin`, `ps`.

Zadanie 2. 4. 2 Uzyskiwanie informacji o obciążeniu systemu

Używając polecenia `hogs` zbadaj który z procesów najbardziej obciąża procesor.

Zadanie 2. 4. 3 Uruchomienie programu z wykorzystaniem narzędzia gdb

Uruchom pokazany w Przykład 2-1 program `test.c` z wykorzystaniem debuggera gdb. Przećwicz różne opcje programu.

Zadanie 2. 4. 4 Program kopiowania plików

Napisz program `fcopy` który kopiuje pliki. Program ma być uruchamiany poleceniem `fcopy file1 file2` i kopiować podany jako parametr pierwszy plik `file1` na podany jako parametr drugi plik `file2`. Użyj w programie funkcji dostępu do plików niskiego poziomu: `open(), read()`, `write()`, `close()`. Znajdź opis tych funkcji w systemie pomocy. Program powinien działać według następującego schematu:

1. Utwórz bufor `buf` o długości 512 bajtów (tyle wynosi długość sektora na dysku).
2. Otwórz plik `file1`.
3. Utwórz plik `file2`.
4. Czytaj 512 bajtów z pliku `file1` do bufora `buf`.
5. Zapisz liczbę rzeczywiście odczytanych bajtów z bufora `buf` do pliku `file2`.
6. Gdy z `file1` odczytałeś 512 bajtów to przejdź do kroku 5.
7. Gdy odczytałeś mniej niż 512 bajtów to zamknij pliki i zakończ program.

Zastosuj debugger `gdb` do uruchomienia powyższego programu

Zadanie 2. 4. 5 Archiwizacja i kopiowania plików

W systemie pomocy znajdź opis archiwizatora `tar`. Używając programu `tar` spakuj wszystkie pliki zawarte w katalogu bieżącym do pojedynczego archiwum o nazwie `programy.tar` (polecenie: `tar –cvf programy.tar *`). Następnie zamontuj pamięć USB i skopiuj archiwum na dyskietkę. Dalej utwórz katalog `nowy` i skopiuj do niego archiwum z dyskietki i rozpakuj do postaci pojedynczych plików (polecenie: `tar –xvf programy.tar`).

3. Komunikacja z systemem wbudowanym

3.1 Kanały komunikacji, protokoły, programy

System wbudowany nie posiada zwykle rozbudowanych interfejsów komunikacji z użytkownikiem. Nie posiada też na ogół pełnej klawiatury, monitora, myszy. Bywa że wyposażony jest w pewną ilość diod sygnalizacyjnych, kilka przycisków czy przełączników, być może posiada prosty wyświetlacz LCD. Jednak na etapie instalacji i konfiguracji systemu operacyjnego oraz na etapie tworzenia oprogramowania aplikacyjnego należy zapewnić sobie możliwość komunikacji z systemem wbudowanym. Jest to potrzebne gdyż na ogół występuje konieczność:

- Testowania prawidłowości działania sprzętu
- Instalacji i konfigurowania systemu operacyjnego
- Testowanie i uruchamianie oprogramowania aplikacyjnego

Aby przeprowadzić wymienione wyżej czynności należy zapewnić sobie możliwość wykonywania w systemie wbudowanym następujących czynności:

- Przesyłanie plików z komputera macierzystego na docelowy (a także w drugą stronę)
- Poruszania się po systemie plików
- Edycji plików, w szczególności plików konfiguracyjnych systemu i aplikacji
- Uruchamiania programów i obserwacji skutków ich działania

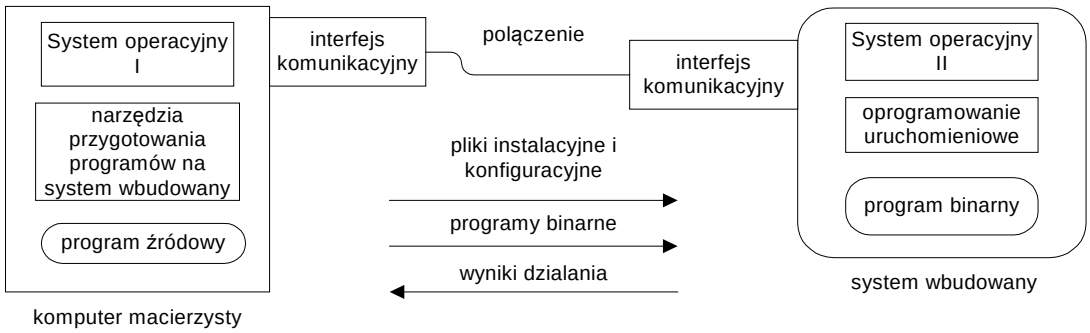
W celu zapewnienia takiej komunikacji należy określić:

- Medium komunikacyjne
- Protokół komunikacji
- Oprogramowanie działające po stronie komputera macierzystego
- Oprogramowanie działające po stronie komputera wbudowanego

Jako medium komunikacyjne najczęściej stosuje się:

- Interfejs RS232, RS484
- Sieć Ethernet
- Połączenie USB
- Połączenie JTAG

Protokół komunikacji związany jest zwykle z medium komunikacyjnym. W przypadku interfejsu RS232 stosuje się transmisję typu start/stop, dla sieci Ethernet protokół TCP/IP. Połączenie JTAG stosowane jest do zaawansowanego testowania działania systemu wbudowanego.



Rys. 3-1 Komunikacja komputera macierzystego z z systemem wbudowanym

Typową potrzebą jest możliwość uruchamiania w systemie wbudowanym programów i obserwacja rezultatów ich działania co ilustruje Rys. 3-1. Aby uruchomić program na systemie wbudowanym należy wykonać następujące kroki:

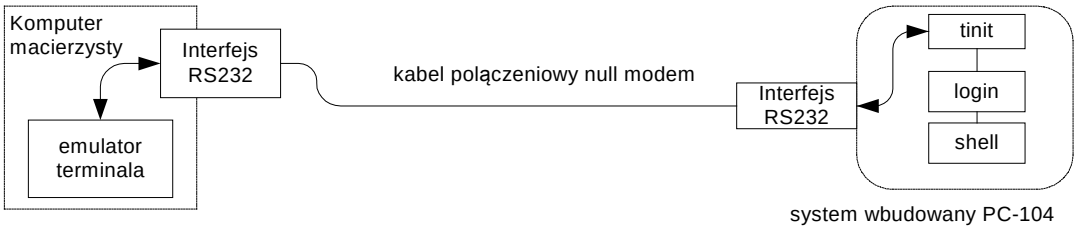
1. Utworzyć program przeznaczony na system wbudowany.
2. Przesłać program do systemu wbudowanego.
3. Spowodować jego uruchomienie
4. Obserwować wyniki jego działania

Wymienione wyżej czynności realizowane są zazwyczaj przez oddzielne aplikacje działające według specyficznych protokołów. Przygotowanie aplikacji odbywa się na komputerze macierzystym przy użyciu systemu skrośnej kompilacji co będzie dalej omówione. Oprogramowanie działające po stronie komputera macierzystego to zwykle emulator terminala tekstowego, klient programu uruchomieniowego np. `gdb GNU debugger` lub zintegrowane

środowisko uruchomieniowe jak np. Eclipse. Oprogramowanie działające po stronie komputera wbudowanego to zwykle interpreter poleceń (ang. *shell*), serwer popularnych usług przesyłania plików (FTP,SFTP,SCP), program uruchomieniowy np. `gdbserver` czy innego rodzaju agent.

3.2 Wykorzystanie interfejsu RS232 do komunikacji z systemem docelowym

Interfejs RS232 jest dogodnym kanałem komunikacji z systemem wbudowanym gdyż nie wymaga rozbudowanej infrastuktury sieciowej. Wymagane jest jednak aby komputer macierzysty posiadał interfejs RS232 co w obecnych komputerach nie jest często spotykane. Gdy brak takiego interfejsu można użyć konwertera USB-RS232.



Rys. 3-2 Połączenie komputera macierzystego z systemem wbudowanym poprzez interfejs RS232

Jeżeli komputer macierzysty pracuje w systemie Windows do komunikacji z systemem wbudowanym można użyć programu HyperTerminal lub PuTTy. Ustawienie terminala są nastepujące: szybkość transmisji 57600, bity danych 8, parzystość brak, bity stopu 1, sterowanie przepływem – brak.

Przykład 3-1 Ustawienie parametrów transmisji w programie HyperTerminal.

Należy także dokonać zmiany w konfiguracji po stronie systemu QNX. Należy zmodyfikować plik konfiguracyjny `/etc/config/ttys` programu `tinit`. Do pliku należy dodać linię:

```
ser1 "/bin/login" qansi-m on
```

Jest to informacja dla procesu `tinit` by na porcie `ser1` uruchomić program inicjalizacji konsoli login, typ terminala ma być qnansi-m [7]. Gdy zalogujemy się do komputera wbudowanego można ustalić jego adres IP za pmocą polecenia: netstat –i . Dalsza komunikacja z systemem może przebiegać przy użyciu narzędzi TCP/IP.

3.3 Wykorzystanie sieci QNET do komunikacji z systemem docelowym

Do komunikacji z systemem docelowym można użyć standardowych usług sieci QNET jak dostęp do sieciowego systemu plików i zdalne wykonywanie poleceń. W celu skomunikowania się z systemem docelowym należy:

1. Odnaleźć komputer w sieci QNET za pomocą polecenia: `ls /net`

```
# ls /net
compaq1          koziol          pc104-komp5
```

2. Zalogować się jako `root` na systemie docelowym za pomocą polecenia:

```
on -f pc104-komp5 login
```

3. Przesłać skompilowany program z komputera macierzystego na komputer docelowy. Można użyć polecenia:

```
cp test1 /net/pc104-komp4/dev/shmem
```

Można użyć polecenia `ftp` bądź Midnight Commandera

4. Przejść do konsoli systemu docelowego, zmienić katalog bieżący na `/dev/shmem` i uruchomić program test1

```
# pwd
/dev/shmem
# ls
test1
# ./test1
witamy w lab Politechnki wroclawskiej
koziol
i= 0, j=0
i= 1, j=0
i= 2, j=3
i= 3, j=6
i= 4, j=9
i= 5, j=12
i= 6, j=15
i= 7, j=18
i= 8, j=21
i= 9, j=24
koniec
```

Ekran 3-1 Wykorzystanie zdalnego terminala do uruchomienia programu w systemie docelowym

3.4 Wykorzystanie usługi telnet do komunikacji z systemem wbudowanym

Telnet jest protokołem sieciowym używanym w Internecie i sieciach lokalnych. Protokół telnet jest jednym z najstarszych protokołów sieciowych, opracowany został w 1969 i opisany w RFC 15 i RFC 854. Zapewnia on komunikację ze zdalnym systemem poprzez wirtualny terminal implementowany przez program klienta.
Istnieje wiele programów będących klientem telnetu np. PuTTY. Podstawowy program klienta telnetu o nazwie `telnet` zainstalowany jest w większości systemów, w tym w systemie QNX6 Neutrino. Można też innych programów, np. popularnego programu `PutTTY` lub `ZOC Terminal`. Uruchomienie programu telnet:

```
$telnet [adres_IP [nr_portu]]
```

Aby skomunikować się z systemem wbudowanym za pomocą telnetu, po stronie tego systemu musi być zainstalowany i uruchomiony serwer telnetu (np. telnetd). Zazwyczaj serwer telnetu uruchamiany jest przez superserwer sieciowy `inetd` lub `xinetd`.

```
$ telnet 192.168.0.3
Trying 192.168.0.3...
Connected to 192.168.0.3.
Escape character is '^]'.
login:
login: root
Wed Oct  8 03:37:06 2014 on /dev/ttyp1
Last login: Tue Oct 7 21:37:32 2014 on /dev/ttyp0
edit the file .profile if you want to change your environment.
# _
```

Przykład 3-2 Wykorzystanie usługi telnet do komunikacji z systemem wbudowanym

Polecenie ps pokazuje proces demona sieciowego `inetd` i serwer usługi telnet czyli program `in.telnetd.`

```
# ps
       PID TTY          TIME CMD
    167954 ?        00:00:00 inetd
    552981 ?        00:00:00 -sh
    417824 ?        00:00:00 in.telnetd
    421921 ?        00:00:00 -sh
```

Przykład 3-3 Proces `in.telned` jest serwerem usługi telnet

3.5 Wykorzystanie usługi ftp do przesyłanie plików pomiędzy systemem macierzystym i docelowym

Protokół FTP jest standardowym protokołem przesyłania danych w sieci wykorzystującej TCP/IP a więc także w Internecie. Umożliwia on przesyłanie plików pomiędzy komputerami z których jeden jest serwerem (udostępnia on usługę) a pozostałe komputery, zwane klientami z tej usługi korzystają. Istnieje wiele klientów protokołu ftp. Popularny jest program `ftp` który jest elementem większości systemów operacyjnych.

```
$ ftp 192.168.0.3
Connected to 192.168.0.3.
220 192.168.0.3 FTP server ready.
Name (192.168.0.3:root): juka
331 Password required for juka.
Password:
230-
    Welcome to QNX Neutrino!
230 User juka logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> _
```

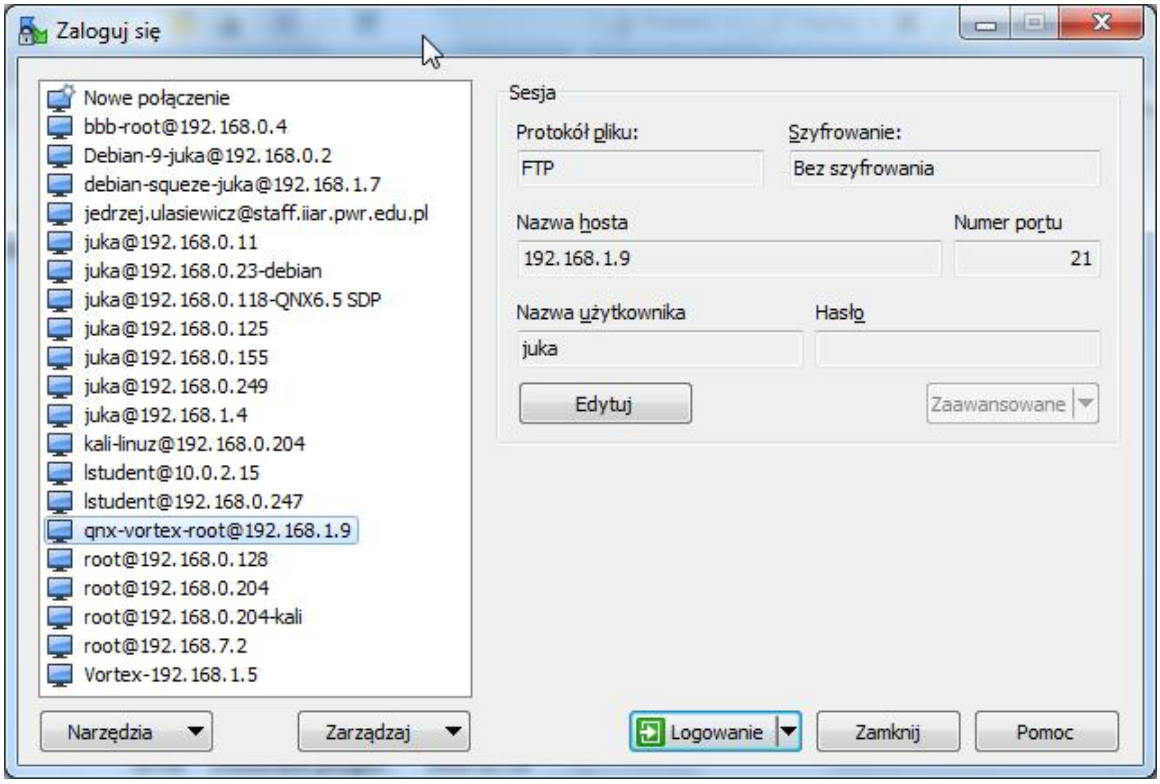
Przykład 3-4 Połączenie z systemem docelowym za pomoca programu ftp

Podstawowe polecenia programu `ftp` podane są w poniższej tabeli.

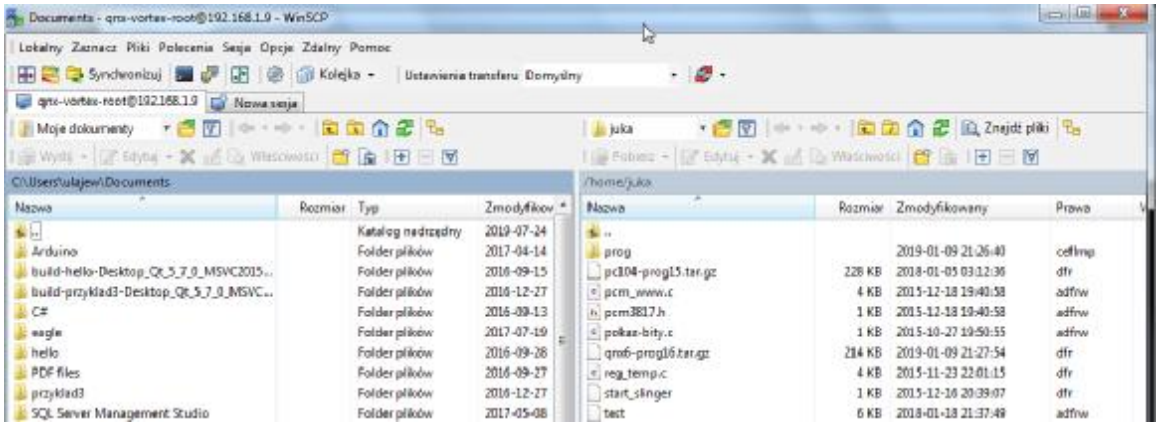
Polecenie	Opis
`help, ?`	Uzyskanie pomocy
`open`	Uzyskanie połączenia z serwerem ftp
`cd`	Zmiana bieżącego katalogu na komputerze zdalnym
`pwd`	Wyświetlenie nazwy katalogu bieżącego na serwerze
`lcd`	Zmiana katalogu bieżącego komputera lokalnego
`mget`	Pobranie wielu plików z serwera do komputera lokalnego
`put, send`	Przesłanie pliku z komputera lokalnego do serwera
`mput`	Przesłanie wielu plików z komputera lokalnego do serwera
`status`	Podanie bieżącego stanu wszystkich opcji
`by`	Rozłączenie

Tabela 3-1 Podstawowe polecenia programu `ftp`

Innym popularnym programem służącym do przesyłania plików jest WinSCP. Po uruchomieniu programu należy podać adres IP komputera wbudowanego, rodzaj protokołu i nazwę uzytkownika co pokazuje poniższy przykład.

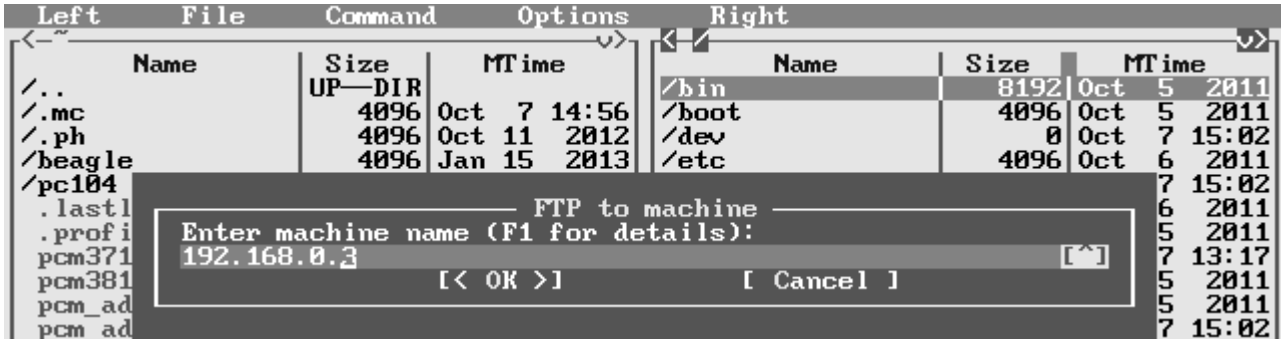


Przykład 3-1 Podawanie parametrów połączenia w programie WinSCP



Przykład 3-2 Działanie programu WinSCP

Do kopiowania plików pomiędzy systemem macierzystym i docelowym wykorzystać można program `Midnight Commander`.



Ekran 3-2 Wykorzystanie programu Midnight Commander do kopiowanie plików przez FTP – wejście w opcję FTP

3.6 Zdalne uruchamianie programu wykonywanego w systemie docelowym

Do zdalnego uruchamiania programów w systemie docelowym można użyć połączenia TCP/IP, programu `gdb` wykonywanego w systemie macierzystym i klienta pdebug uruchomionego w systemie docelowym. Dalej podano przykład jak to wykonać.

1. W systemie macierzystym np. w katalogu `/home/juka` utworzyć podany niżej program `test1.c` a następnie go skompilować używając opcji -g
   `gcc test1.c –o test1 –g`

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void) {
      int i,j;
      puts("Witamy w Lab PRW");
      system("hostname");
      for(i=0;i<10;i++) {
         j=i+10;
         printf("Krok  %d\n",i);
         sleep(1);
      }
      printf("Koniec\n");
      return EXIT_SUCCESS;
}
```

Przykład 3-5 Program `test1.c`

2. Uruchomić system docelowy (nazywa się `pc104-komp5`) i sprawdzić czy jest widziany w katalogu `/net` za pomocą polecenia:
   `ls /net`

```
# ls /net
compaq1          koziol          pc104-komp5
```

3. Zalogować się jako `root` w systemie docelowym za pomocą polecenia:

`on –f pc104-komp5 login`

Uzyskać adres IP komputera docelowego za pomocą polecenia:

`netstat –ni`

```
# on -f pc104-komp5 login
login: root
sh: j_init: tcgetpgrp() failed: Inappropriate I/O control operation
sh: warning: won't have full job control
Fri Oct  3 03:16:52 2014 on /net/koziol.dom/dev/ttyp0
Last login: Fri Oct 3 01:58:48 2014 on /dev/ttyp1
edit the file .profile if you want to change your environment.
# netstat -ni
Name  Mtu   Network       Address              Ipkts Ierrs    Opkts Oerrs Colls
lo0   33212 <Link>                                113     0      113     0     0
lo0   33212 127           127.0.0.1               113     0      113     0     0
en0   1500  <Link>        00:0b:ab:71:1c:72      3433     0      818     0     0
en0   1500  192.168       192.168.0.3            3433     0      818     0     0
# _
```

Na komputerze macierzystym sprawdzić czy jest połączenie TCP/IP z systemem docelowym za pomocą polecenia ping

```
ping 192.168.0.3
```

```
# ping 192.168.0.3
PING 192.168.0.3 (192.168.0.3): 56 data bytes
64 bytes from 192.168.0.3: icmp_seq=0 ttl=255 time=2 ms
64 bytes from 192.168.0.3: icmp_seq=1 ttl=255 time=3 ms
64 bytes from 192.168.0.3: icmp_seq=2 ttl=255 time=2 ms
```

4. Na komputerze docelowym uruchomić program pdebug

```
pdebug 8001 &
```

5. Na komputerze macierzystym uruchomić program `gdb`
Następnie ustalić adres i port komputera docelowego:

```
(gdb) target qnx 192.168.0.3:8001
```

```
(gdb) target qnx 192.168.0.3:8001
Remote debugging using 192.168.0.3:8001
(gdb) _
```

Pierwszy argument jest typem protokołu. Mogą być także async – port RS232, cisco – TCP/IP. Dalsze parametry zależą od typu protokołu. Dla protokołu qnx może to być:

```
target qnx adres_IP:port      - komunikacja przez TCP/IP
target qnx /dev/ser1          - komunikacja przez port szeregowy
```

6. Przesłać program binarny do katalogu `/tmp` komputera docelowego

```
(gdb) upload test1 /tmp/test1
```

7. Wczytać tablicę symboli:

```
(gdb) sym test1
```

9. Uruchomić program w systemie docelowym

```
(gdb) run /tmp/test1
```

10. Ustawić breakpoint na 10 linii

```
(gdb) break 10
```

10. Wylistować kod źródłowy:

```
(gdb)l
```

```
(gdb) target qnx 192.168.0.3:8001
Remote debugging using 192.168.0.3:8001
(gdb) upload test1 /tmp/test1
(gdb) sym test1
Reading symbols from test1...done.
(gdb) run /tmp/test1
Starting program:  /tmp/test1
(gdb) break 10
Breakpoint 1 at 0x804853c: file test1.c, line 10.
(gdb) l
1       #include <stdio.h>
2
3       int main(int argc, char * argv[])
4       {
5           int i,j=0;
6           printf("witamy w lab Politechnki wroclawskiej\n");
7           system("hostname");
8           for(i=0;i<10;i++) {
9               printf("i= %d, j=%d\n",i,j);
10              j=i*3;
(gdb) _
```

7. Przeprowadzić uruchomienie programu:

```
(gdb) c
```

```
(gdb) c
Continuing.
witamy w lab Politechnki wroclawskiej
pc104-komp5
i= 0, j=0

Breakpoint 1, main (argc=1, argv=0x8047df4) at test1.c:10
10              j=i*3;
(gdb) _
```

3.7 Komunikacja za pomocą TCP/IP i interfejsu graficznego Phwindows

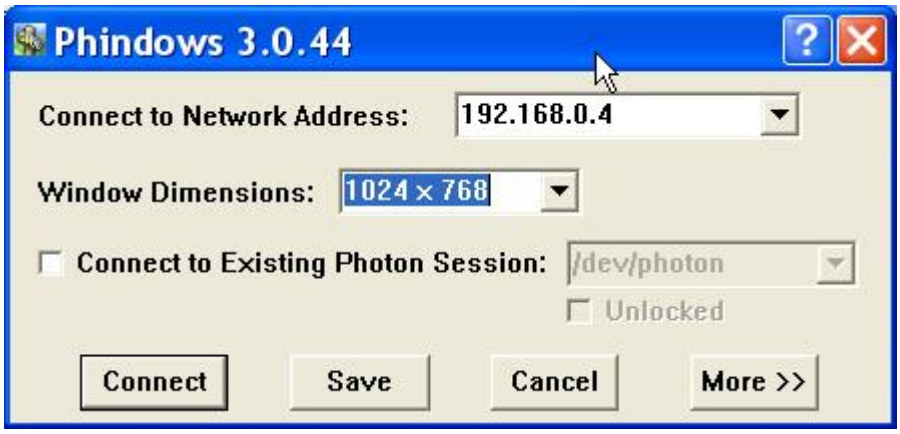
Z systemem QNX6 można komunikować się za pomocą interfejsu TCP/IP i interfejsu graficznego Photon. Na komputerze macierzystym należy uruchomić serwer o nazwie phrelay, program ten jest częścią QNX6.5 Software Development platform. Należy też w pliku `/etc/inetd.conf` odkomentować linię:

```
phrelay stream tcp nowait root /usr/photon/bin/phrelay phrelay –y
```

W pliku `/etc/services` ma być obecna linia:

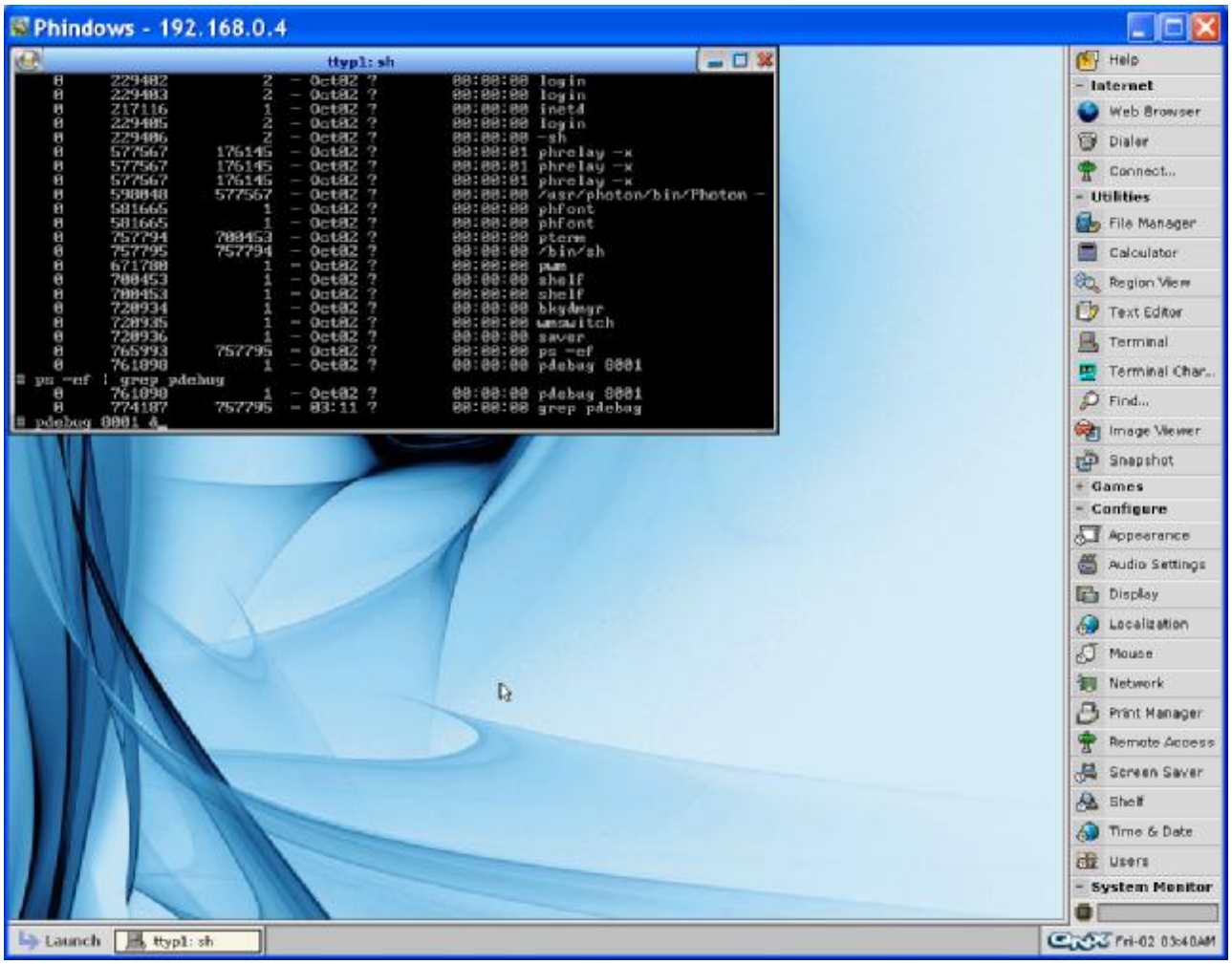
```
phrelay 4868/tcp
```

Po uruchomieniu podajemy adres IP systemu docelowego i rozdzielczość wirtualnego monitora.



Ekran 3-3 Uruchamianie klienta interfejsu Photon w systemie Windows.

Po naciśnięciu przycisku Connect pojawi się obraz ekranu wirtualnego. Po zalogowaniu zobaczymy ekran interfejsu Photon jak poniżej.



Ekran 3-4 Zgłoszenie się interfejsu Photon w systemie Windows.

3.8 Zadania

Zadanie 3. 4. 1 Komunikacja z systemem docelowym za pomocą sieci QNET

Znajdź wybrany komputer wbudowany pc104 w sieci QNET Laboratorium. Połącz się z wybranym komputerem za pomocą zdalnie wykonanego polecenia `login`. Następnie utwórz podany w Przykład 3-5 program, skompiluj go, prześlij na system wbudowany i tam uruchom.

Zadanie 3. 4. 2 Komunikacja z systemem docelowym za pomocą protokołu TCP/IP

Znajdź wybrany komputer wbudowany pc104 w sieci QNET Laboratorium. Połącz się z wybranym komputerem za pomocą zdalnie wykonanego polecenia login i ustal jego adres IP. Połącz się z systemem wbudowanym za pomocą programu telnet. Następnie utwórz podany w Przykład 3-5 program, skompiluj go, prześlij na system wbudowany za pomocą protokołu FTP i tam uruchom wykorzystując połączenie telnet.

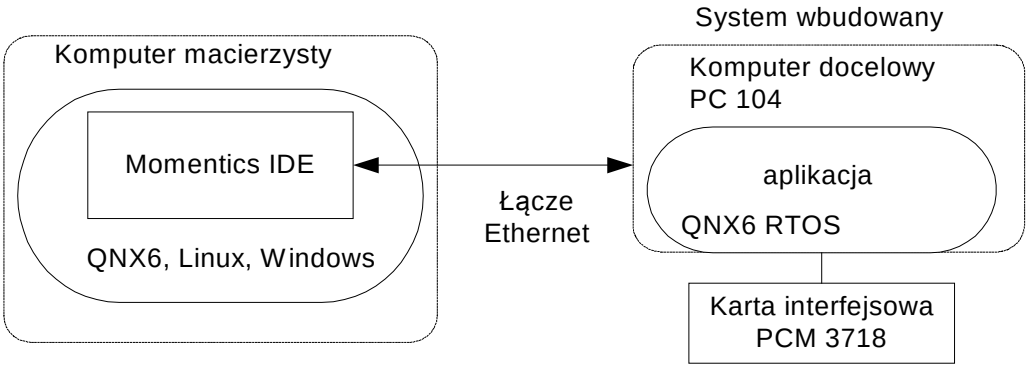
Zadanie 3. 4. 3 Zdalne uruchamianie programu za pomocą debuggera gdb

Utwórz podany w Przykład 3-5 program, skompiluj go z opcją –g , prześlij na system wbudowany (do katalogu /dev/shmem) za pomocą protokołu FTP lub sieci QNET. Połącz się z systemem wbudowanym za pomocą programu telnet, przejdź do katalogu `/dev/shmem` i uruchom tam program kliencki debuggera `pdebug nr_portu.` Następnie na komputerze macierzystym uruchom debugger gdb, połącz się z systemem docelowym i uruchom wybrany program.

4. Wykorzystanie platformy Momentics do tworzenia oprogramowania dla systemów wbudowanych

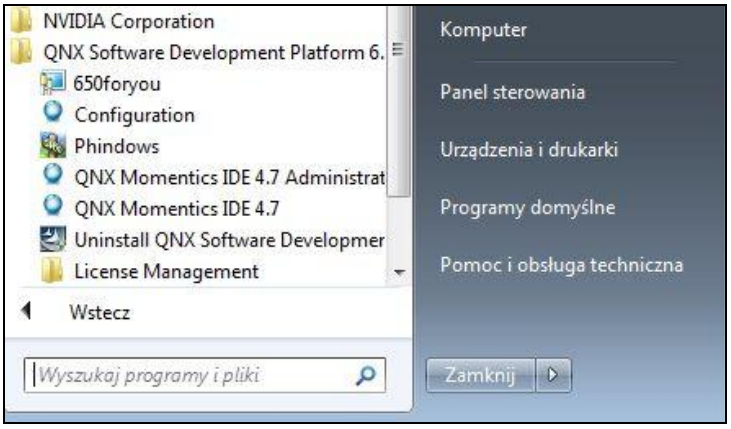
Platforma Momentics jest zintegrowanym środowiskiem programistycznym (ang. IDE) służącym do projektowania i tworzenia oprogramowania systemów wbudowanych w systemie QNX6 Neutrino. Pakiet zainstalować można w systemie QNX6, Linux i Windows. Pozwala on tworzyć, kompilować i testować i oprogramowanie dla wszystkich platform sprzętowych wspieranych przez QNX. Taki sposób rozwijania oprogramowania nazywany jest pracą skrośną (ang. *cross development*) – projekt przygotowywany i kompilowany jest na platformie innej niż docelowa, na której będzie ostatecznie uruchomiony. Pakiet Momentics zawiera następujące komponenty:

- Edytor i debuger kodu źródłowego.
- Narzędzie analizy pokrycia kodu (ang. *Code Coverage*).
- Narzędzia informacji o systemie docelowym.
- Konstruktor systemu (ang. *System Builder*) oraz pakiet wsparcia dla płyt głównych (ang. *Board Support Package*).
- Narzędzie konfiguracji aplikacji i systemu (ang. *Application Profiler*,*System Profiler*).
- Narzędzie Application Builder do budowy interfejsu graficznego opartego na interfejsie Photon
- Program Phindows przeznaczony do zdalnej pracy w systemie docelowym.
- Analizator pamięci.



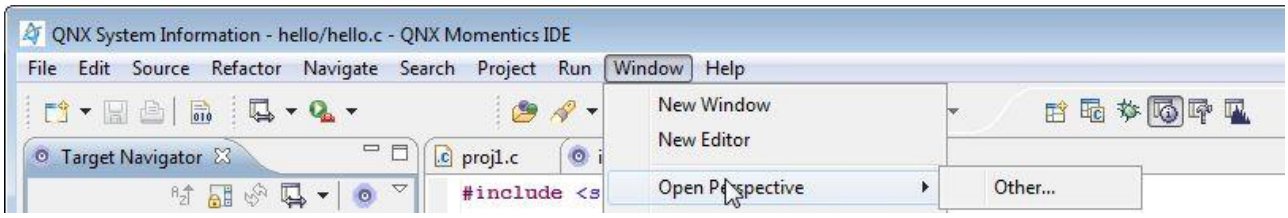
Rys. 4-1 System skrośnego rozwoju oprogramowania

Narzędzie uruchamiamy wybierając ikonę QNX Momentics IDE z menu uruchomieniowego Windows.



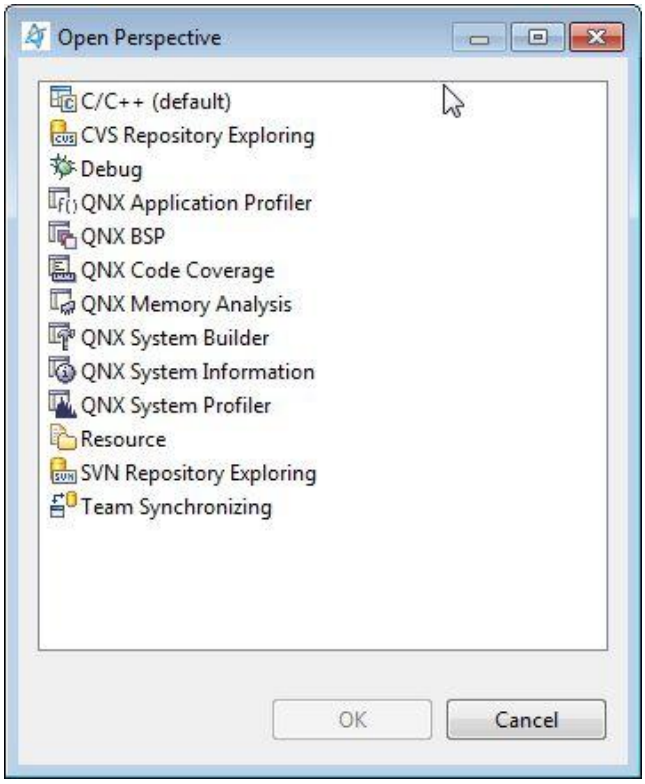
Ekran 4-1

Wybór funkcji uzyskuje się otwierając tak zwaną perspektywę. Dokonujemy tego klikając w klawisze: ***Window / Open Perspective*** a następnie wybieramy odpowiednią perspektywę co pokazano na poniższym rysunku.



Ekran 4-2 Narzędzie Momentics Development Suite - wybór perspektywy

Na początku wybierzemy perspektywę QNX System information



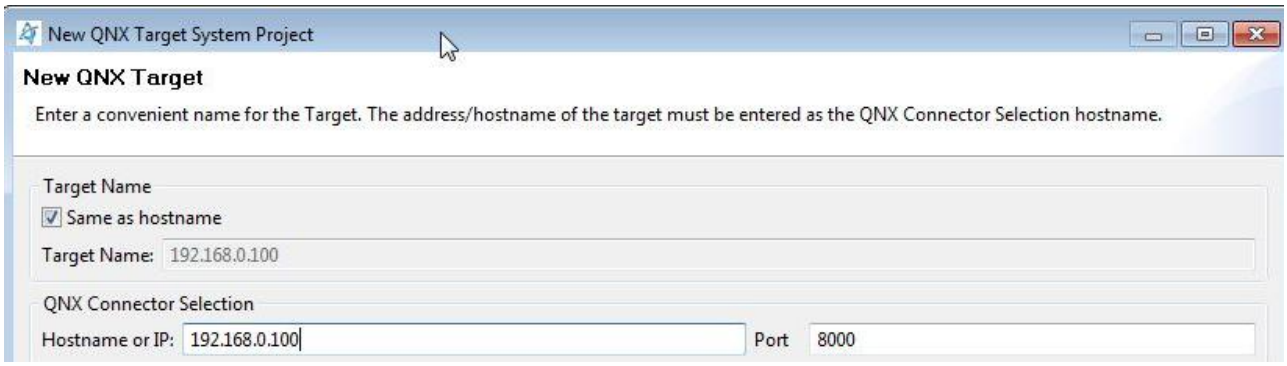
Ekran 4-3 Narzędzie Momentics Development Suite - wybór perspektywy

4.1 Połączenie z systemem docelowym

Aby połączyć się z systemem docelowym należy dołączyć obydwa komputery (macierzysty i docelowy) do sieci Ethernet. System docelowy musi pracować pod kontrolą QNX6 Neutrino i powinien mieć uruchomione:

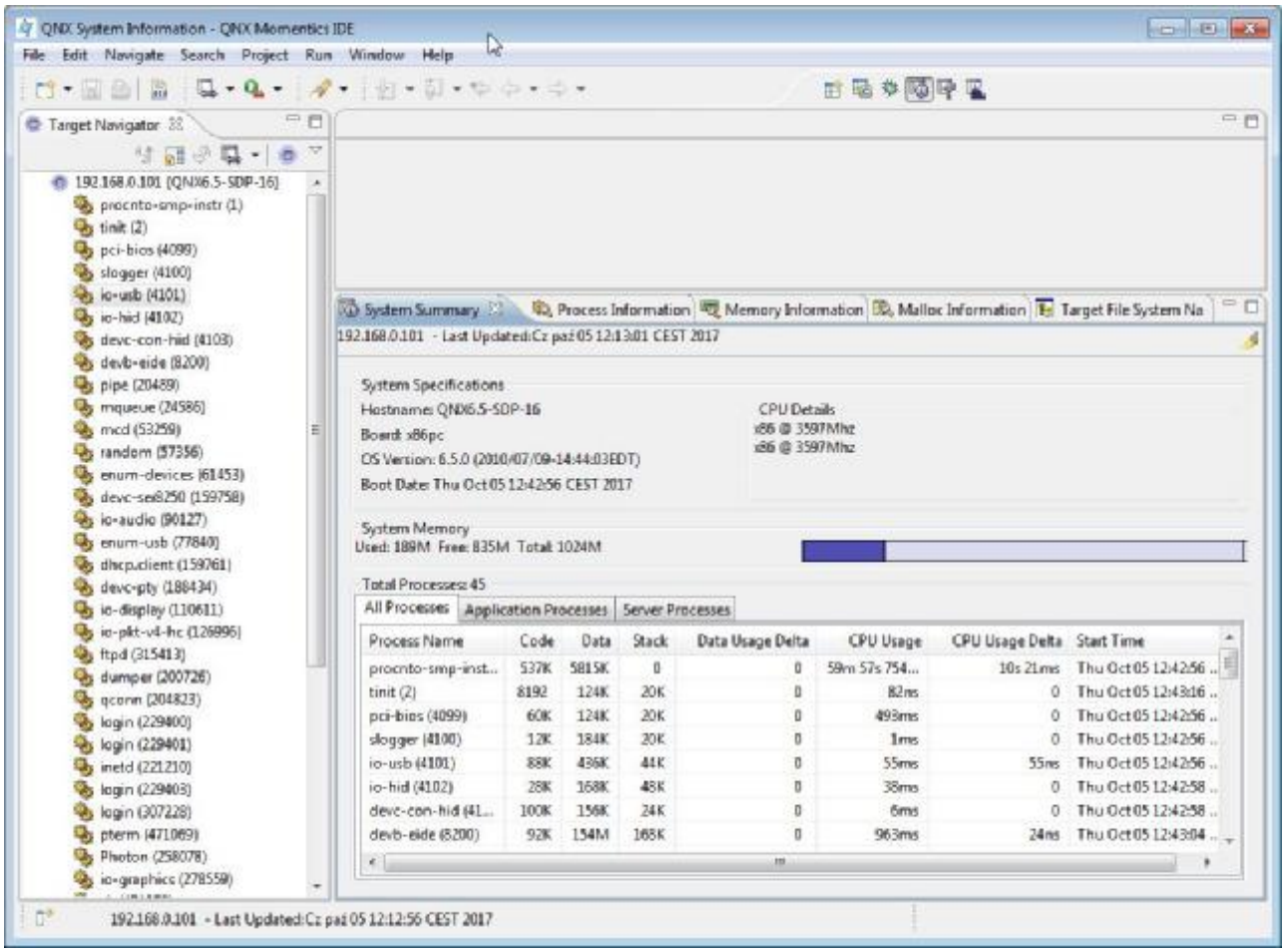
- Sieć QNET
- Interfejs TCP/IP
- Program `qconn`

Połączenie z siecią QNET możemy sprawdzić pisząc na konsoli: `$ls /net`. W folderze `/net` powinna się pojawić nazwa komputera docelowego. Następnie wybieramy opcje: *File / New / Other / QNX / QNX Target System Project* Pojawia się wtedy formatka w której należy wpisać nazwę komputera docelowego i jego adres IP tak jak podano poniżej.



Rys. 4-1 Ikona wyboru systemu docelowego

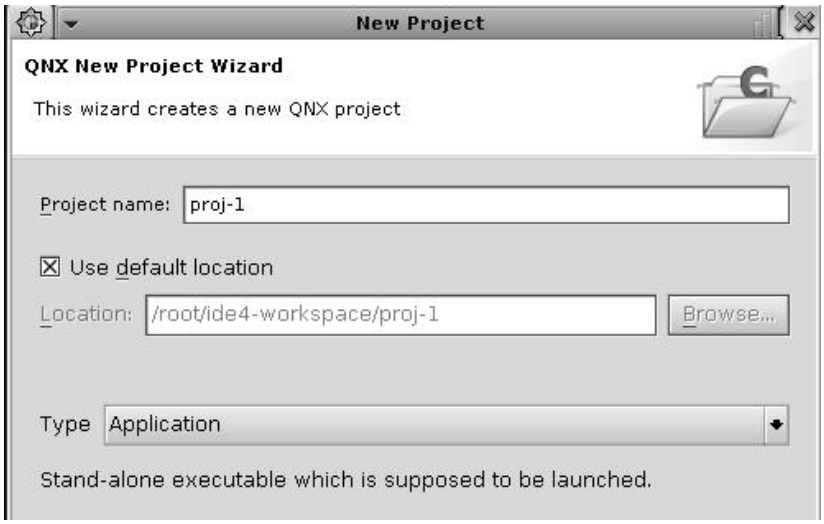
Po wciśnięciu przycisku Finish w oknie Target Navigator powinna się pojawić ikona systemu docelowego. Jeżeli wybrana jest perspektywa QNX System Information to po prawej stronie pojawią się zakładki System Summary, Process Information, Memory Information, Malloc Information jak pokazano poniżej. Zakładki te zawierają rozmaite informacje o systemie docelowym.



Ekran 4-4 Uzyskiwanie informacji o systemie docelowym

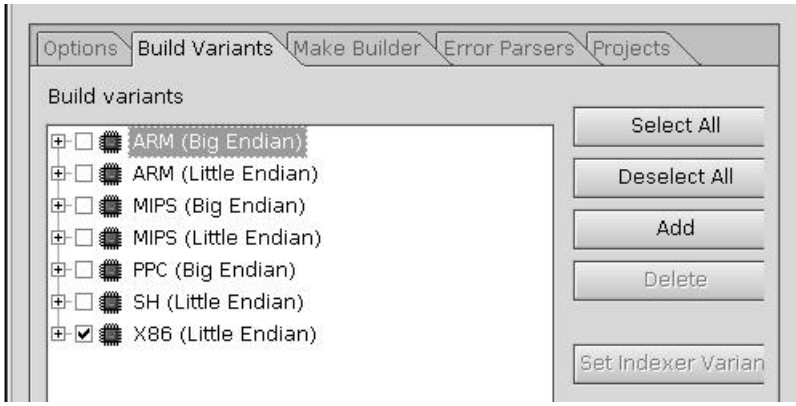
4.2 Tworzenie projektu nowego programu

Środowisko Momentics zawiera narzędzia umożliwiające tworzenie programu mającego być wykonanym w systemie docelowym. Aby utworzyć nowy projekt programu na system docelowy wybieramy opcje: New / QNX C Project po czym pojawia się formatka jak poniżej. W oknie Project Name wpisujemy nazwę projektu i klikamy w przycisk Next.



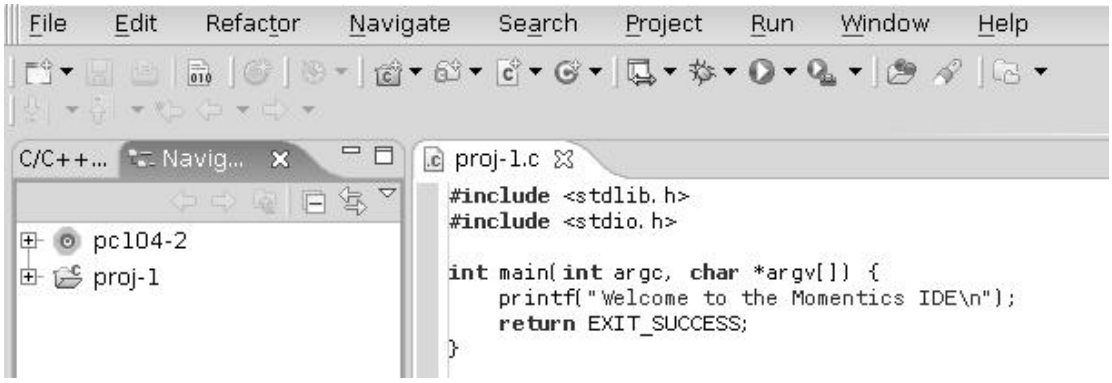
Rys. 4-2 Tworzenie nowego projektu

W kolejnej formatce podanej poniżej zaznaczamy typ systemu docelowego (w naszym przykładzie X86) i naciskamy przycisk Next.



Rys. 4-3 Wybór architektury systemu docelowego

Po naciśnięciu przycisku Finish pojawi się okno z kodem szkieletowym pokazane na poniższym rysunku



Rys. 4-4 Okno edycji kodu programu szkieletowego

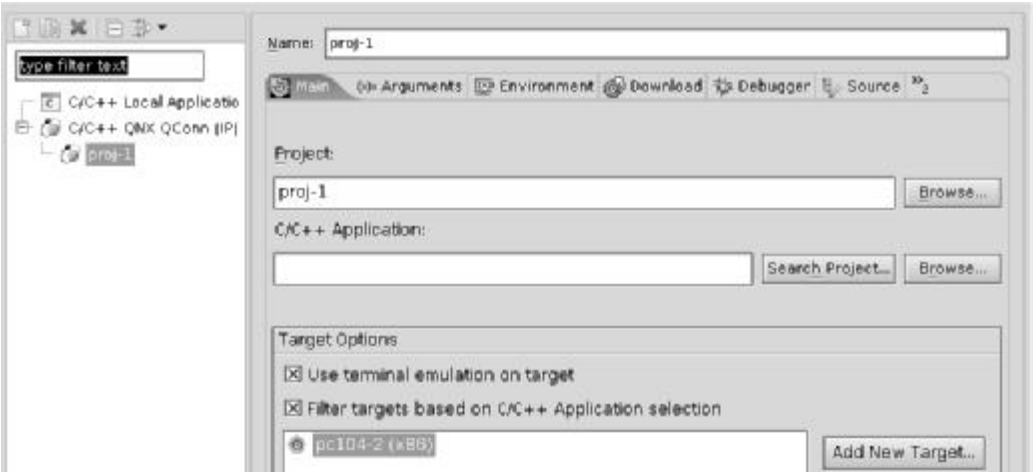
Przykładowy kod można dowolnie modyfikować. Kolejną czynnością jest kompilacja programu. Wykonujemy ją wybierając z menu opcje: Project / Build Project . Gdy w programie nie ma błędów zakładka Console zawierała będzie kod kompilacji jak pokazano poniżej.

```
Problems  Console  Properties
C-Build [proj-1]
/root/ide4-workspace/proj-1/proj-1.c
/bin/rm -f  /root/ide4-workspace/proj-1/x86/o-g/proj-1_g
/usr/qnx632/host/qnx6/x86//usr/bin/qcc -Vgcc_ntox86
-o/root/ide4-workspace/proj-1/x86/o-g/proj-1_g    proj-1.o  -L.
-L/usr/qnx632/target/qnx6/x86/lib -L/usr/qnx632/target/qnx6/x86/usr/lib    -g
make[2]: Leaving directory `/root/ide4-workspace/proj-1/x86/o-g'
make[1]: Leaving directory `/root/ide4-workspace/proj-1/x86'
```

Rys. 4-5 Raport z kompilacji programu przykładowego

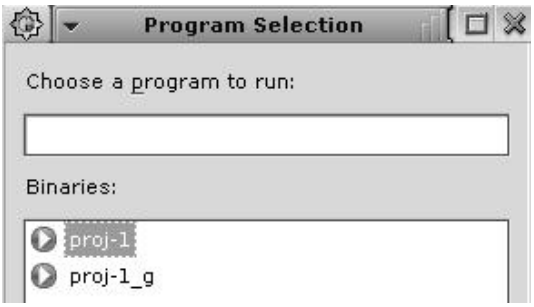
4.3 Wykonanie programu na systemie docelowym

Po poprawnej kompilacji programu możemy wykonać go na platformie docelowej. W tym celu wybieramy opcję Run / Run . Pojawi się formatka jak poniżej.



Rys. 4-6 Formatka specyfikacji wykonania programu

Dalej wybieramy wariant C/C++ QNX Qconn (IP) i klikamy w przycisk **Search Project** wybierając z okna **Binaries** wersję proj-1 lub proj-1_g (wersja przeznaczona do debugowania)



Rys. 4-7 Wybór wersji programu

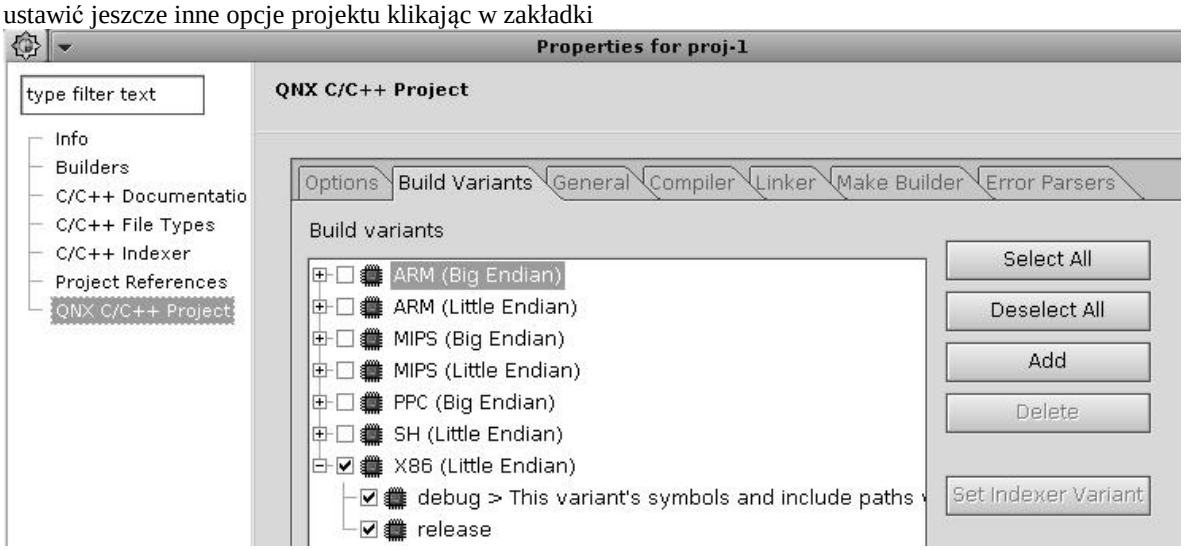
Gdy wersja zostanie określona klikamy w przycisk Run co spowoduje przesłanie programu na system docelowy (domyslnie do folderu /tmp) i jego uruchomienie. Rezultat działania programu pokazany zostanie na konsoli tak jak w poniższym przykładzie.

```
Problems  Console  Properties
<terminated> proj-1 [C/C++ QNX QConn (IP)] /tmp/proj-1root1317834426569 on
Welcome to the Momentics IDE
```

Rys. 4-8 Wykonanie programu przykładowego

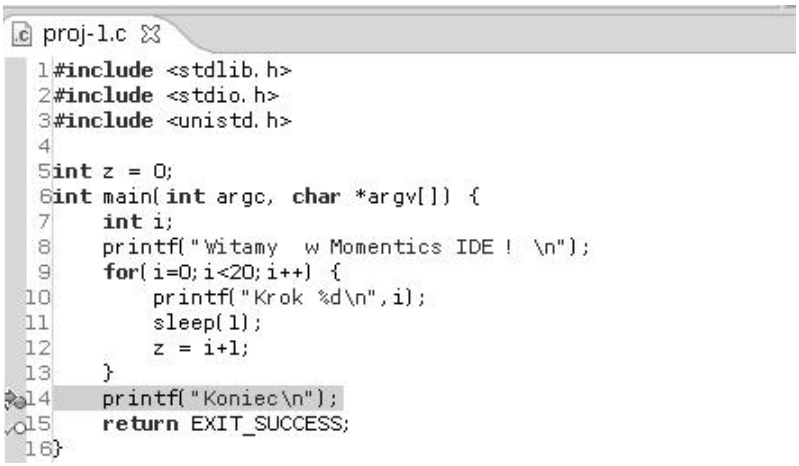
4.4 Debugowanie programu w systemie docelowym

Aby umożliwić debugowanie programu należy się upewnić czy projekt ma ustawioną opcję która to umożliwia. W tym celu należy wybrać opcję Project / Properities i kliknąć w element QNX C / C++ Project a następnie wybrać zakładkę Build Variants. Powinna się pojawić formatka jak poniżej. Należy zwrócić uwagę czy ikona debug jest zaznaczona. Gdy tak powstanie wersja programu wynikowego przeznaczona do debuggowanie. Posługujac się tą formatką można ustawić jeszcze inne opcje projektu klikając w zakładki



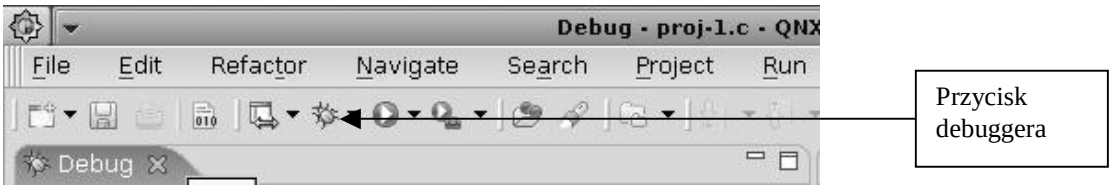
Rys. 4-9 Formatka właściwości projektu

Aby uczynić debugowanie ciekawszym zmodyfikujmy program przykładowy aby wykonywał w pętli wypisywanie zawartości licznika zadana liczbę razy.



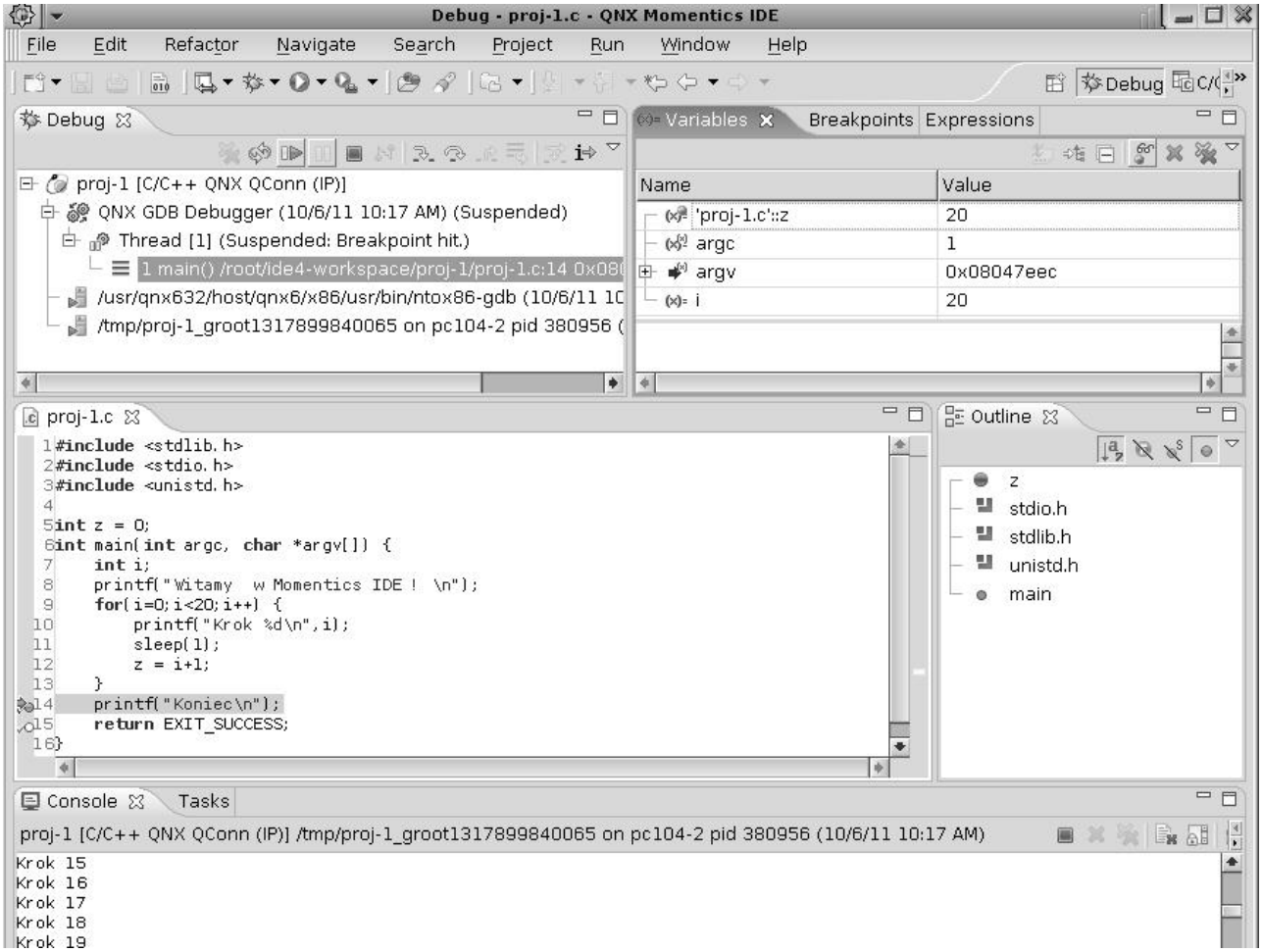
Rys. 4-10 Kod źródłowy programu testowego

Następnie skompilujmy program wybierając opcje: **Project / Build.** Do debugowania przechodzimy wciskając ikonę z pluskwą co pokazuje poniższy rysunek.



Rys. 4-11 Uruchamianie debuggera

Po uruchomieniu debuggera pokażą się okna: programu uruchomieniowego (Debuggera), kodu źródłowego (na rys proj-1.c) , Zmiennych (Variables), punktów wstrzymania (Breakpoints) i konsoli (Console). Okna te pokazuje poniższy rysunek.



Rys. 4-12 Formatka programu uruchomieniowego

Do sterowania przebiegiem uruchamiania używamy ikonek w oknie debuggera. Najważniejsze podaje poniższa tabela.

	Restart		Uruchom proces od początku
	Resume	F8	Uruchom proces od punktu bieżącego
	Terminate		Zakończ proces
	Step into	F5	Wykonaj krok programu wchodząc do funkcji
	Step over	F6	Wykonaj krok programu nie wchodząc do funkcji
	Run to return	F7	Wyjdź z funkcji
	Suspend		Zawieś proces

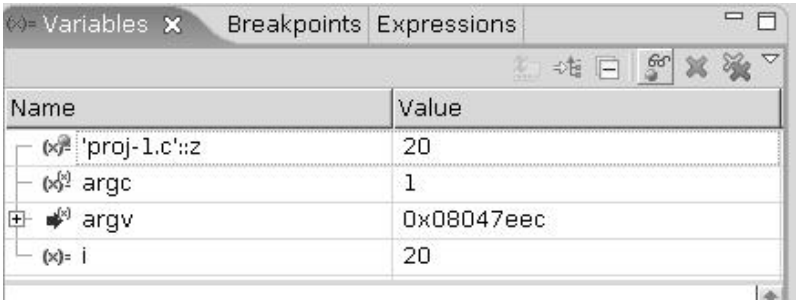
Tab. 4-1 Najważniejsze ikony okna programu uruchomieniowego

Naciskając ikonę wykonujemy jeden krok programu co pozwala na pracę krokową. Możemy również uruchomić program do najbliższego punktu wstrzymania (*ang. Breakpoint*). Punkt wstrzymania ustawiamy klikając prawym klawiszem myszy na listewkę po lewej stronie okna programu źródłowego. Gdy ustawimy punkt wstrzymania na linię 14 i naciśniemy ikonę program wykona się do linii 14. Wartości zmiennych obserwować można w oknie inspekcji zmiennych (Variables). Debugger posiada znacznie więcej możliwości. Należą do nich:

- Inspekcja i zmiana wartości zmiennych
- Użycie brakpointów i watchpointów
- Inspekcja rejestrów
- Inspekcja obszarów pamięci
- Badanie użycia bibliotek dzielonych
- Monitorowanie obsługi sygnałów

- Obserwacja komunikatów z programu (konsola)
- Użycie debuggera gdb

Opisane są one w dokumentacji.



Rys. 4-13 Okno inspekcji zmiennych

Breakpoint wstrzymuje program gdy osiągnięta jest pewna linia programu. Watchpoint wstrzymuje program gdy zmienia się wartość pewnego wyrażenia.

4.5 Zadania

Zadanie 4. 4. 1 Uruchomienie programu sterującego diodami karty pcm3218

Uruchom podani poniżej program sterujący diodami LED karty pcm3718. Posłuż się metodami:

- Użycie maszyny wirtualnej z obrazem systemu QNX5 SDP i debugera gdb
- Użycie środowiska Momentics (także w trybie debug)

```
// Karta PCM 3817H - test wyjsc cyfrowych
#include <sys/neutrino.h>
#include <hw/inout.h>

#define ADRB 0x300
#define DIOL 3
#define DIOH 11
static int base = ADRB;

main() {
   int val;
   unsigned char i = 0;
   printf("Test karty PCM3718 \n");
   system("hostname");
   ThreadCtl( _NTO_TCTL_IO, 0 );
   base = mmap_device_io(16,ADRB);
   // Test wyjsc cyfrowych ------------------------
   val = 0x01;
   for(i=0;i<7;i++) {
            out8(base + DIOH, val);
            val = val << 1;
            usleep(200000);
   }
   printf("Test wyjsc cyfrowych \n");
   for(i=0;i<3;i++) {
        out8(base + DIOH, 0xff);
        usleep(300000);
        out8(base + DIOH, 0x00);
        usleep(300000);
  }
}
```

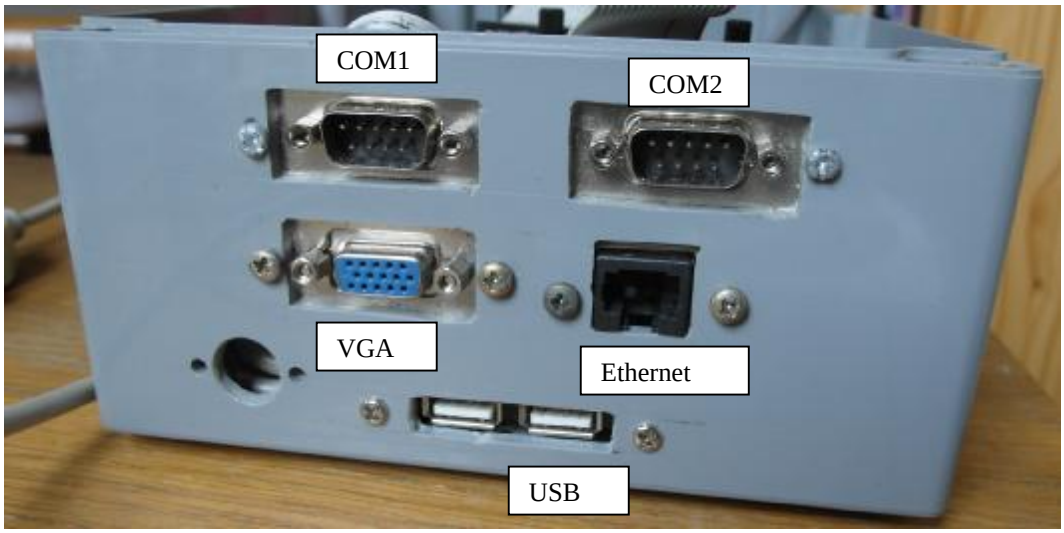
Przykład 4-1 Program sterującu diodami LED karty PCM3718

5. Budowa i wykorzystanie komputera dydaktycznego typu PC104 Vortex

5.1 Wstęp

Komputer dydaktyczny PC104 przeznaczony jest do realizacji ćwiczeń laboratoryjnych z przedmiotów Systemy Czasu Rzeczywistego i Komputerowe Systemy Sterowania. Może być także wykorzystany jako pomoc dydaktyczna w innych przedmiotach takich jak Systemy Wbudowane.

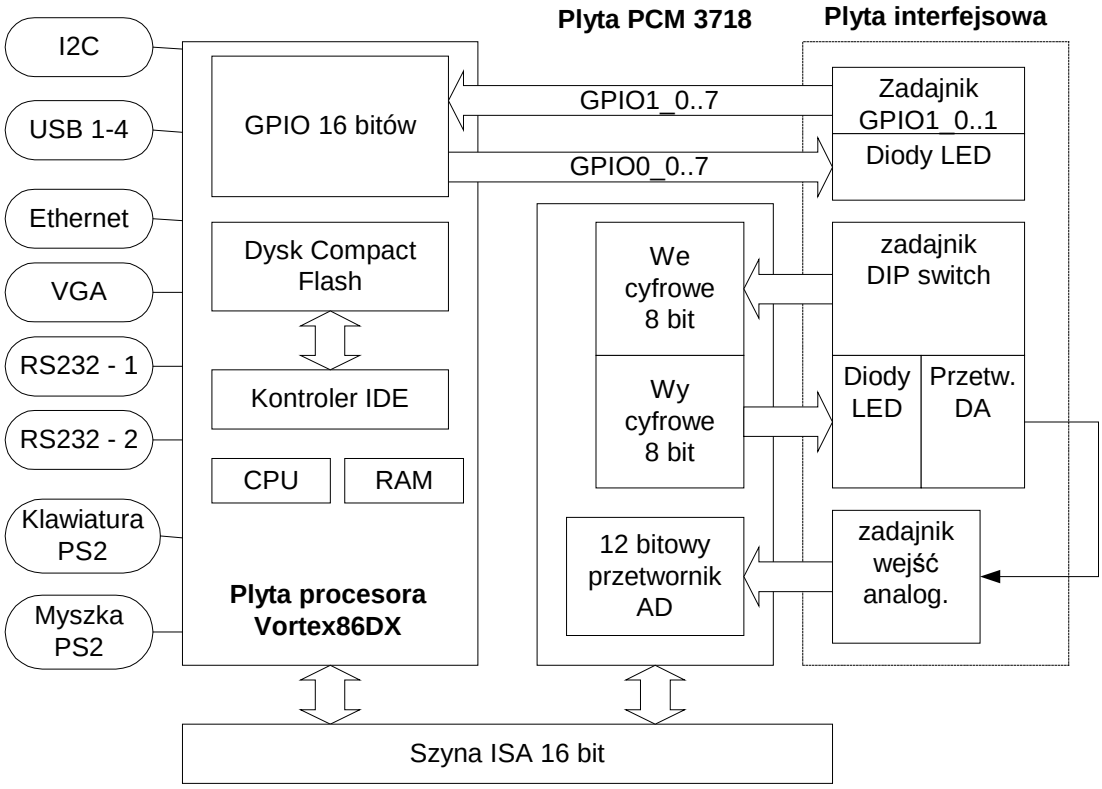
Fotografia 5-1 Komputer dydaktyczny Vortex



Fotografia 5-2 Złącza komputera dydaktycznego Vortex

Komputer składa się z następujących elementów:

- Płyta procesora Vortex86SX/DX w standardzie PC104 . System operacyjny wczytywany jest z dysku Flash
- Płyta akwizycji danych PCM3718 zawierająca konwertery AD, 8 wejść binarnych i 8 wyjść binarnych
- Płyta interfejsowa zawierająca 8 bitowy przetwornik DA, zadajniki wejść analogowych i cyfrowych oraz zespół diód LED prezentujących stan wyjść binarnych
- Zasilacza 5V 3A
- Obudowy



Rys. 5-1 Schemat komputera dydaktycznego typu PC104

5.2 Moduł procesora Vortex86DX

Jako płyty procesora wykorzystano następujące moduły procesor Vortex86DX. Posiada on następujące własności:

Procesor:

- Procesor Vortex86DX SOC 1GHz, zgodny z x86, 6 stopniowy potok instrukcji
- Jednostka zmiennoprzecinkowa
- Wbudowana pamięć podręczna L1 16K – instrukcji, 16K danych
- Wbudowana pamięć podręczna L2 256 Kb
- Pamięć operacyjna - SDRAM 256 MB rozszerzalna do 512
- Kontroler DMA
- Watchdog timer: system reset; programowalny w zakresie od 30.5µ sek do 512 sek x 2 s
- Złącze PC104 – ISA
- Zegar czasu rzeczywistego

Pamięci zewnętrzne:

- Złącze do dysku IDE – 2 dyski
- Gniazdo dysku Compact Flash
- Kontroler zewnętrznej pamięci SPI

Porty:

- Cztery porty szeregowe RS232 w tym 1 RS485
- Złącze klawiatury/myszki PS2
- Interfejs USB x 4
- Interfejs I2C
- Dwa interfejsy Fast Ethernet
- Dwa liczniki/timery zgodne z 8254
- JTAG – do debuggowania

Grafika:

- Kontroler graficzny `SMI SM712 2D`
- Pamięć Video 4 Mb
- Wyjścia VGA, LVDS, CRT+TTL

Zasilanie:

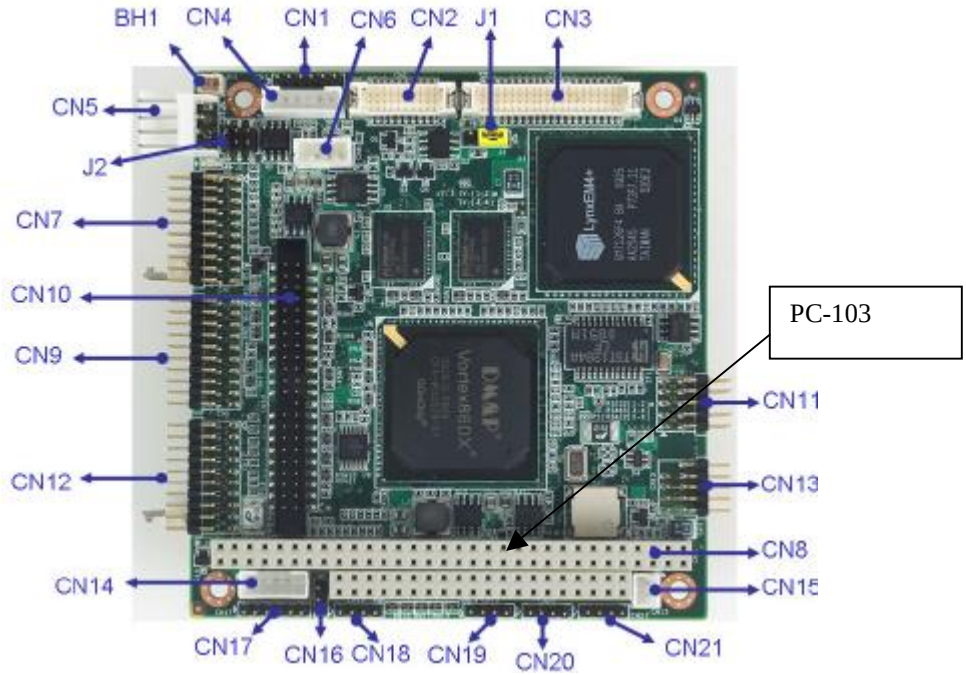
- 5V 0.55-0.85 A

Warunki pracy:

- 0-60 C

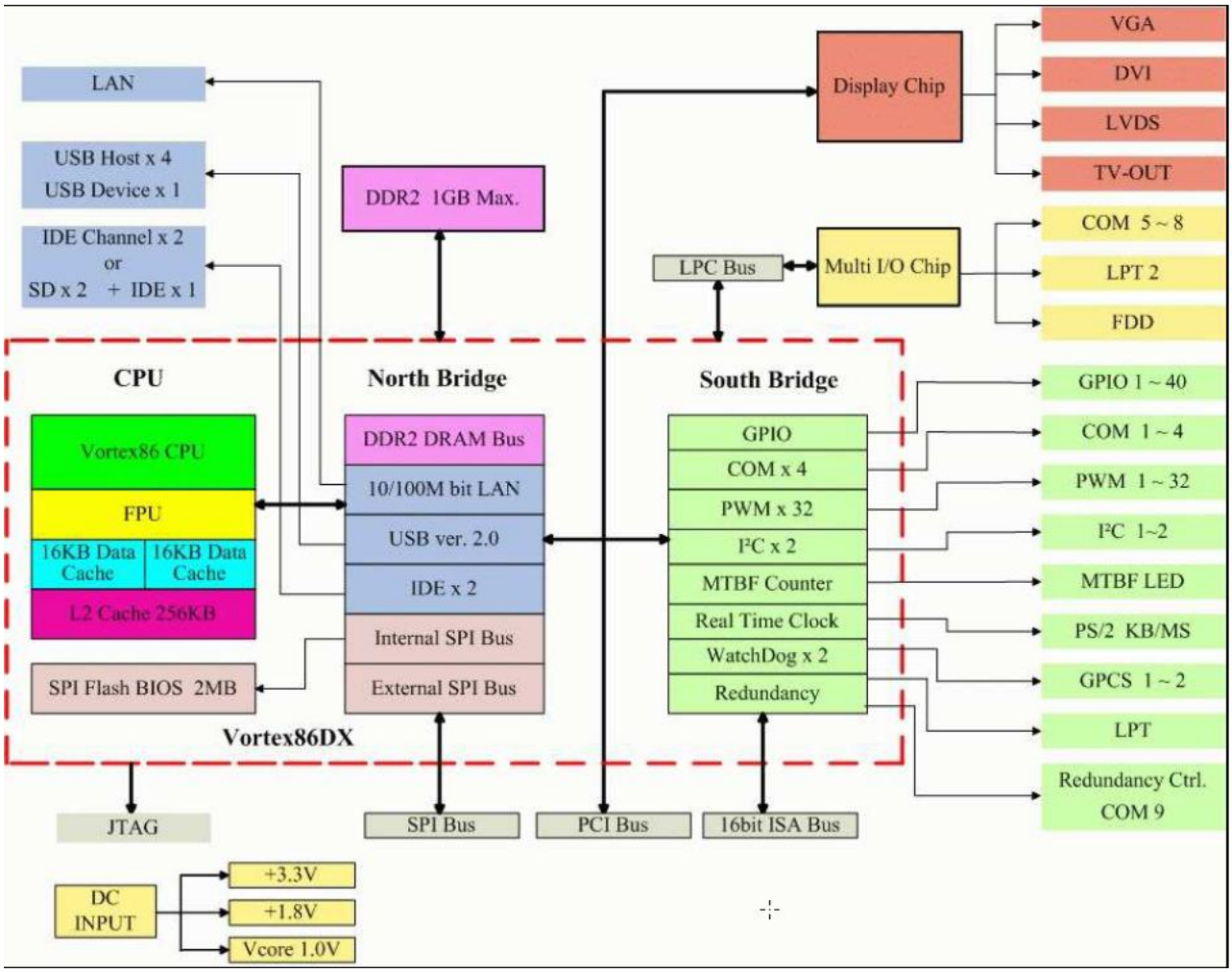
Wymiary:

- 96x90 mm
- Waga 0.097 Kg



Fotografia 5-3 Widok płyty PCM 3343 od strony elementów (góra)

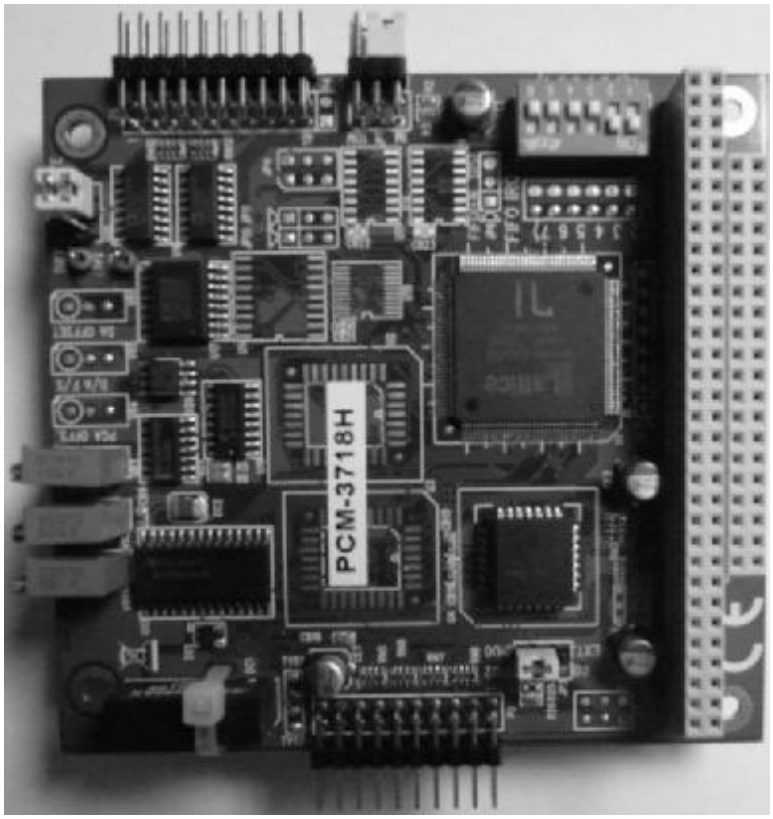
Fotografia 5-4 Widok płyty PCM 3343 od strony druku (dół)



Rys. 5-1 Schemat procesora Vortex86DX

5.3 Opis karty PCM3718

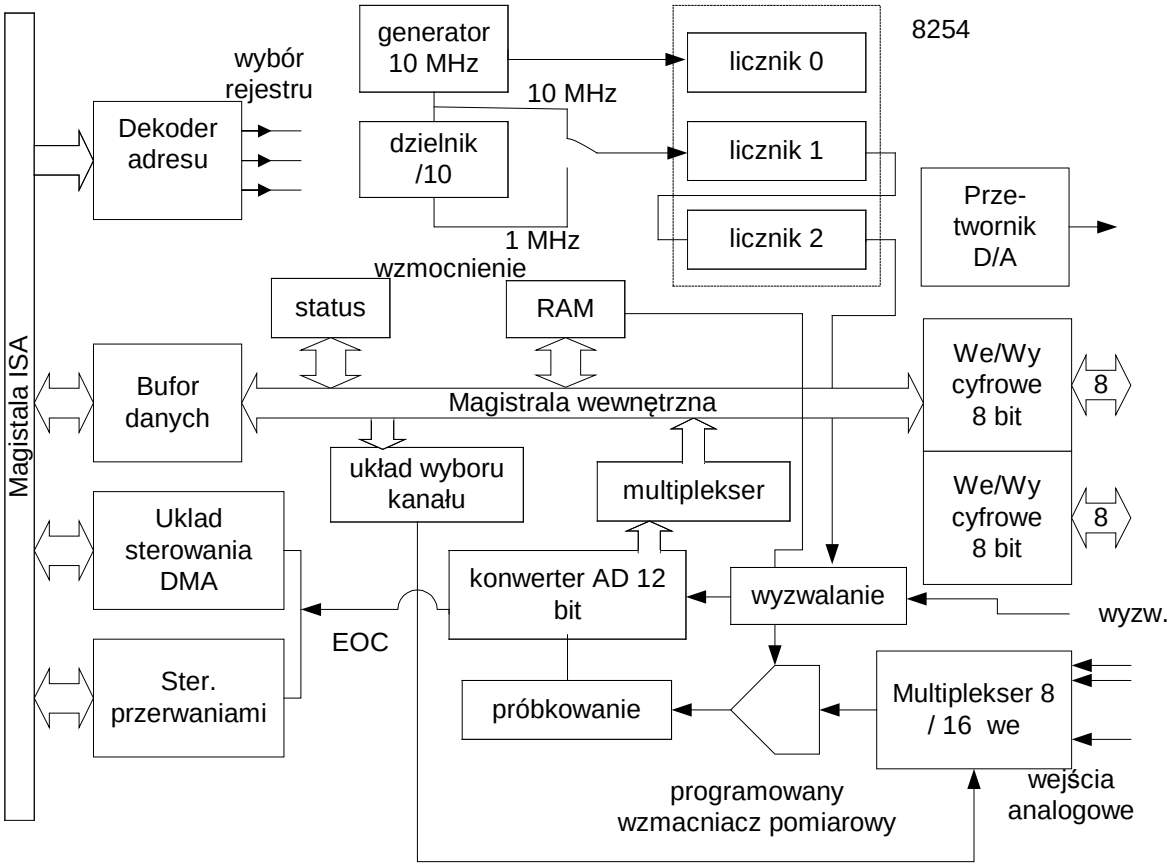
Karta PCM-3718 firmy Advantech Co. Ltd jest to typową kartą przetwornikową standardu PC104 stosowaną w celach sterowania i akwizycji danych. Jej widok pokazany jest na poniższym rysunku.



Rys. 5-2 Widok karty interfejsowej PCM3718

Karta zawiera następujące układy:

- 16 pojedynczych lub 8 różnicowych wejść analogowych AD dołączonych do multipleksera i dalej poprzez wzmacniacz pomiarowy do 12 bitowego przetwornika analogowo cyfrowego. Maksymalna częstotliwość próbkowania wynosi 60 KHz. Zakres mierzonych napięć to +/- 0.005V, +/- 0.01 V, +/-0.5V, +/-1V, +/-5V, +/-10V dla wejść różnicowych i 0.01V, 0.1V, 1V, 10V dla wejść unipolarnych.
- 16 wejść/wyjść cyfrowych DI poziomu TTL (0V, 5V).
- Układ licznika timera typu 8254 zawierający trzy 16 bitowe liczniki dołączone do generatora 1MHz / 10 MHz. Jeden licznik może być podłączony do źródła zewnętrznego a dwa wykorzystywane są do wyzwalania przetwornika AD.
- 1 kanałowy 12 bitowy przetwornik DA (tylko PCM 3718HO)



Rys. 5-3 Schemat blokowy karty interfejsowej PCM-3718

Z punktu widzenia programisty karta widziana jest jako zestaw 16 rejestrów 8 bitowych poczynając od adresu bazowego BASE.

Adres portu	Odczyt	Zapis
BASE+0	A/D bajt młodszy & kanał	Wyzwalanie programowe A/D
BASE+1	A/D bajt starszy	N/A
BASE+2	Zakres kanałów multipleksera	Zakres kanałów multipleksera
BASE+3	D/I bajt młodszy (DI0-7)	D/O bajt młodszy (DO0-7)
BASE+4	N/A	Przetwornik DA bity DA0- DA3
BASE+5	N/A	Przetwornik DA bity DA4- DA11
BASE+6	N/A	N/A
BASE+7	N/A	N/A
BASE+8	Status	Kasowanie źródła przerwania
BASE+9	Rejestr sterujący - zapis	Rejestr sterujący - odczyt
BASE+10	N/A	Konfiguracja liczników
BASE+11	D/I bajt starszy (DI8-15)	D/O bajt starszy (DO8-15)
BASE+12	Licznik 0	Licznik 0
BASE+13	Licznik 1	Licznik 1
BASE+14	Licznik 2	Licznik 2
BASE+15	N/A	Sterowanie licznikami

Tab. 5-1 Rejestry karty PCM-3718

```
#include <sys/neutrino.h>
#include <hw/inout.h>
#define A0       0  // Przetw. AD bajt lo & kanał
#define A1       1  // Przetw. AD bajt starszy
#define MUXR     2  // Rejestrt multipleksera
#define DOUT1    3  // Wyjścia cyfrowe DO 1
#define DINP1    3  // Wejścia cyfrowe DI 1
#define CONTR    9  // Rejestr ster. przetwornika AD
#define STATR    8  // Rejestr stat. przetwornika AD
#define TIMR    10  // Konfiguracja liczników
#define DINP2   11  // Wejścia cyfrowe DO 2
#define DOUT2   11  // Wyjścia cyfrowe DO 2
#define COUNT0  12  // Sterowanie licznikiem 0
#define COUNT1  13  // Sterowanie licznikiem 1
#define COUNT2  14  // Sterowanie licznikiem 2
#define COUNTC  15  // Sterowanie liczn. układu 8254
```

Przykład 5-1 Plik nagłówkowy karty PCM-3718

5.4 Dostęp do portów we/wy:

Urządzenia wejścia wyjścia posiadają zestawy rejestrów które program może odczytywać i zapisywać. W rejestrach umieszczane są dane konfigurujące urządzenie, statusy, dane przeznaczone do wysłania i odbioru itd.

☞ Próba wykonania w trybie użytkownika operacji wejścia wyjścia lub innej niebezpiecznej operacji spowoduje wygenerowanie wyjątku, wywołanie systemu operacyjnego i w konsekwencji zakończenie procesu.

Procesy które wykonują operacje wejścia wyjścia, muszą należeć do użytkownika `root` (UID=0). Dodatkowo proces powinien wykonać wywołanie funkcji `ThreadCtl(_NTO_TCTL_IO,0)`. W pewnych architekturach urządzenia wejścia wyjścia mogą znajdować się w przestrzeni pamięci wprowadza się ich odwzorowanie na przestrzeń wejścia wyjścia poprzez wykonanie funkcji `mmap_device_io.`

Uzyskanie dostępu do rejestrów urządzenia	
`int mmap_device_io(int len, uint64_t io)`	
`len`	Liczba bajtów urządzenia która ma być udostępniona
`io`	Adres początkowy udostępnianego obszaru

Funkcja zwraca wartość będącą argumentem dla funkcji `in*()` i `out*()` czytających i piszących dane do rejestrów lub stałą `MAP_DEVICE_FAILED` gdy nie można uzyskać dostępu do urządzenia.

Odczyt bajtu z portu zewnętrznego	
`unsigned char in8(uintptr_t port)`	
`port`	Adres portu w przestrzeni wejścia/wyjścia

Wysłanie bajtu na port zewnętrzny	
`void out8(int port, unsigned char val)`	
`port`	Adres portu w przestrzeni wejścia wyjścia.
`val`	Bajt wyprowadzany na port.

```
#include <hw.inout.h>
...
uintptr_t port;
unsigned char x,y;
ThreadCtl( _NTO_TCTL_IO, 0 );
port = mmap_device_io(1, 0x300);
x = in8(port);
out8(port,y);
```

Przykład 5-2 Odczyt i zapis bajtu z portu urządzenia

5.5 Wejścia i wyjścia cyfrowe

Karta PCM-3718 oferuje dwa 8 bitowe cyfrowe kanały wejściowe lub wyjściowe. Kanały te używają portów o adresach `BASE+3` i `BASE+11`, które służą zarówno do czytania jak i zapisywania. Zapis na port `BASE+3` lub `BASE+11` bajtu `DO` powoduje ustawienie linii wyjściowych portu zgodnie z zawartością bitową bajtu `DO`. Odczyt z portu `BASE+3` lub `BASE+11` powoduje zwrócenie wartości `DI` odpowiadających poziomom logicznym na przyłączeniach portu.

	B7	B6	B5	B4	B3	B2	B1	B0
Odczyt	DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0
Zapis	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0

Tabela 5-1 Działanie portów wejść i wyjść cyfrowych DI i DO o adresach BASE+3 i BASE+11

```
// Karta PCM 3817H - test we/wy cyfrowych -------------------------------
// Odczyt portu DI1 i wypisanie na DO2 kolejnych liczb 0-255
#include "pcm3817.h"
#define ADRB 0x300
static int base = ADRB;
#define WE 1
#define WY 2

void dout(int num, unsigned char val) {
// Sterowanie portem wyjsc cyfrowych
// num - port (1,2) , val - wartosc (0 - 255)
  if(num <= 1) {
    out8(base + DIOL, val);
  } else {
    out8(base + DIOH, val);
  }
 }

  unsigned char dinp(int num) {
  // Odczyt portu wejsc cyfrowych
  // // num - port (1,2)
  if(num <= 1) {
    return( in8(base + DIOL));
  } else {
   return( in8(base + DIOH));
  }
 }

main() {
   int val1,val2, chn ;
   unsigned char i = 0;
   ThreadCtl( _NTO_TCTL_IO, 0 );
   base = mmap_device_io(16,ADRB);
   printf("Test wejsc/wyjsc cyfrowych\n");
   // Test wejsc  cyfrowych
   i= 0;
   do {
      val1 = dinp(1);
      printf(" wejscie1 %02X \n",val1);
      usleep(50000);
      i=i+1%256;
      dout(2,i);
   } while(1);

}
```

Przykład 5-3 Zapis / odczyt portu wyjść / wyjść cyfrowych

5.6 Obsługa przetwornika AD

Aby dokonać pomiaru wielkości analogowej za pomoce przetwornika AD należy określić:

- Zakres pomiarowy każdego z wejść
- Zakres pracy multipleksera przełączającego wejścia
- Sposób wyzwalania przetwornika,
- Sposób rozpoznawania końca pomiaru
- Sposób przesyłania wyniku pomiaru.

Ustalanie zakresu pomiarowego przetwornika
Każdy z kanałów przetwornika posiada indywidualnie ustawiany zakres pomiarowy. Aby ustalić zakresu pomiarowy przetwornika należy:

1. Wpisać do rejestru BASE+2 numer ustawianego kanału (bity 0-3)
2. Wpisać do rejestru BASE+1 zakres pomiarowy (bity 0-3) zgodnie z poniższa tabelą

Zakres	Unipolar/ bipolar	G3	G2	G1	G0
+/-5	B	0	0	0	0
+/-2.5	B	0	0	0	1
+/-1.25	B	0	0	0	1
+/-0.625	B	0	0	1	1
0-10	U	0	1	0	0
0-5	U	0	1	0	1
0-2.5	U	0	1	1	0
0-1.25	U	0	1	1	1

Tab. 5-2 Zakresy pomiarowe przetwornika AD

Ustalanie zakresu pracy multipleksera
Karta posiada 8/16 wejść analogowych przełączanych multiplekserem. Aby przygotować układ do pracy należy zaprogramować numer najniższego CL i numer najwyższego mierzonego kanału CH. Układ zaczyna pomiar od kanału CL. Po dokonaniu pomiaru przechodzi do kolejnego kanału aż do CH po czym powraca do CL. Programowanie zakresu kanałów odbywa się poprzez wpis do rejestru BASE+2 numerów CL (bity 0-3) i CH (bity 4-7).

	B7	B6	B5	B4	B3	B2	B1	B0
BASE+2	CH3	CH2	CH1	CH0	CL3	CL2	CL1	CL0

Tabela 5-2 Rejestr sterowania multiplekserem karty PCM-3718

Ustalanie źródła wyzwalania, sygnalizacji zakończenia pomiaru i sposobu przesyłania wyniku.
Przetwornik AD pracować może w wielu trybach. Tryby te dotyczą:

- Wyzwalania przetwornika,
- Rozpoznawania końca pomiaru
- Przesyłania wyniku pomiaru.

Przetwornik może być wyzwalany:

- programowo
- przez impulsy z umieszczonych na karcie układów

Koniec pomiaru może być:

- Odczytany w rejestrze statusowym
- Sygnalizowany przerwaniem.

Wyniki konwersji mogą być:

- Odczytywane z portów układu
- Zapisywane do pamięci operacyjnej poprzez układ DMA.

O trybie pracy przetwornika decydują wpisy dokonane do rejestru sterującego (adres BASE+9).

	B7	B6	B5	B4	B3	B2	B1	B0
BASE+9	INTE	I2	I1	I0	-	DMAE	ST1	ST0

Tab. 5-3 Rejestr sterujący karty PCM-3718

Bit INTE steruje generowaniem przerwań przez kartę.

- Gdy INTE = 0 generowanie przerwań jest zablokowane.
- Gdy INTE = 1 oraz DMAE = 0 oznacza to , że przerwanie jest generowane gdy konwersja AD zostanie zakończona.
- Gdy INTE = 1 oraz DMAE = 1 oznacza to, że przerwanie jest generowane gdy z kontrolera DMA przyjdzie impuls T/C wskazujący zakończenie transferu DMA.

Bity `I2`, `I1`, `I0` służą do wyboru poziomu przerwania zgodnie z Tabela 5-3.

I2	I1	I0	Poziom przerwania
0	0	0	-
0	0	1	-
0	1	0	IRQ2
0	1	1	IRQ3
1	0	0	IRQ4
1	0	1	IRQ5
1	1	0	IRQ6
1	1	1	IRQ7

Tabela 5-3 Poziomy przerwania karty PCL-718

Bity `ST0`, `ST1` - określenie źródła wyzwalania konwersji przetwornika

ST1	ST0	Źródło wyzwalania
0	X	Programowe
1	0	Zewnętrzne
1	1	Z licznika układu 8254

Tabela 5-4 Specyfikacja źródeł wyzwalanie karty PCL-718

- Wyzwalanie programowe - zapis dowolnej wartości pod adres `BASE+0`.
- Wyzwalanie zewnętrzne - pobudzenie linii sterującej `TRIG0` umieszczonej na łączówce karty.
- Przez liczniki układu 8254 (licznik 1 i licznik 2).

Liczniki te dołączone są do generatora kwarcowego o częstotliwości *Fzeg* 10MHz lub 1MHz.
Wyjście licznika 2 może powodować wyzwolenie konwersji. Stopień podziału licznika L_1 i L_2 można zaprogramować i otrzymać żądaną częstotliwość dokonywania konwersji.

$$f = \frac{Fzeg}{L_1 * L_2}$$

Konwersja AD odbywa się metodą sukcesywnej aproksymacji i trwa około 15 µs. Zakończenie konwersji może być wykryte poprzez odczyt rejestru statusowego przetwornika AD lub przez przerwanie.

	B7	B6	B5	B4	B3	B2	B1	B0
BASE+8	EOC	UNI	MUX	INT	CN3	CN2	CN1	CN0

Tabela 5-5 Rejestr statusowy przetwornika AD

Operacja zapisu do tego rejestru powoduje wyzerowanie bitu `INT` nie zmieniając pozostałych bitów czyli skasowanie przerwania.

EOC	Wskaźnik zakończenia konwersji. 0 gdy przetwornik jest gotowy a 1 gdy konwersja jeszcze się nie zakończyła .
UNI	Wskaźnik trybu: 0 – tryb bipolarny, 1 – tryb unipolarny.
MUX	Wskaźnik trybu: 0 – 8 kanałów różnicowych, 1 - 16 kanałów pojedynczych z wspólną masą.
INT	Wskaźnik przerwania: 0 – konwersja AD nie jest zakończona od ostatniego wyzerowania tego bitu, 1 – konwersja AD została zakończona i przetwornik jest gotowy do następnego przetwarzania. Jeżeli bit `INTE` rejestru kontrolnego (`BASE+9`) jest ustawiony, wówczas, wraz z ustawieniem bitu `INT` pojawi się przerwanie `IRQ`.
CN0-CH3	Numer kanału, który jest przeznaczony do następnego przetworzenia w przetworniku AD.

Tabela 5-6 Znaczenie bitów rejestru statusowego przetwornika AD

- Gdy konwersja się zakończy jej wynik może być odczytany z rejestrów danych przetwornika AD.

- Rejestry danych AD służą tylko do czytania i używają adresów `BASE+0` i `BASE+1`.
- Zapis do rejestru spod adresu `BASE+0` powoduje wyzwolenie programowe przetwornika AD (start przetwarzania).

	B7	B6	B5	B4	B3	B2	B1	B0
BASE+0	AD3	AD2	AD1	AD0	C3	C2	C1	C0
BASE+1	AD11	AD10	AD9	AD8	AD7	AD6	AD5	AD4

Tabela 5-7 Rejestry przetwornika AD karty PCL-718

- Bity `AD11-AD0`-12 bitowa wartość wynikową podawaną przez przetwornik
- Bity `C3-C0` numer kanału AD z którego pochodzi dana wartość.

```
#include "pcl718.h"
#define ADRB 0x300
static int base = ADRB;

card_init(int from, int to, unsigned char zakres) {
// Inicjalizacja karty
// from - kanal poczatkowy, to kanal koncowy
// zakresy pomiarowe:  0-10 V -> 4, 0-5V -> 5,
// 0-2.5 -> 6, 0-1.25 -> 7
   unsigned char val,i;
   printf("inicjacja kanaly od %d  do %d\n",from,to);
   out8(base + CONTR,  0x00);
   val =  in8(base + CONTR);
   if(val != 0x00) {
     printf("Blad inicjalizacji\n");
     exit(0);
   }
   // Ustawienie kan. pocz i konc
   out8(base + MUXR,  (to << 4) | from);
   out8(base + TIMR,  0x00);
   // Odczyt rejestru MUX ----
   val =  in8(base + MUXR);
   // Ustawienie zakresu pomiarowego kanalow
   for(i = from; i<= to; i++) {
      out8(base + MUXR,  i);
      out8(base + RANGE,zakres);
   }
   // Ustawienie kan. pocz i konc
   out8(base + MUXR,  (to << 4) | from);
   val =  in8(base + MUXR);
}
```

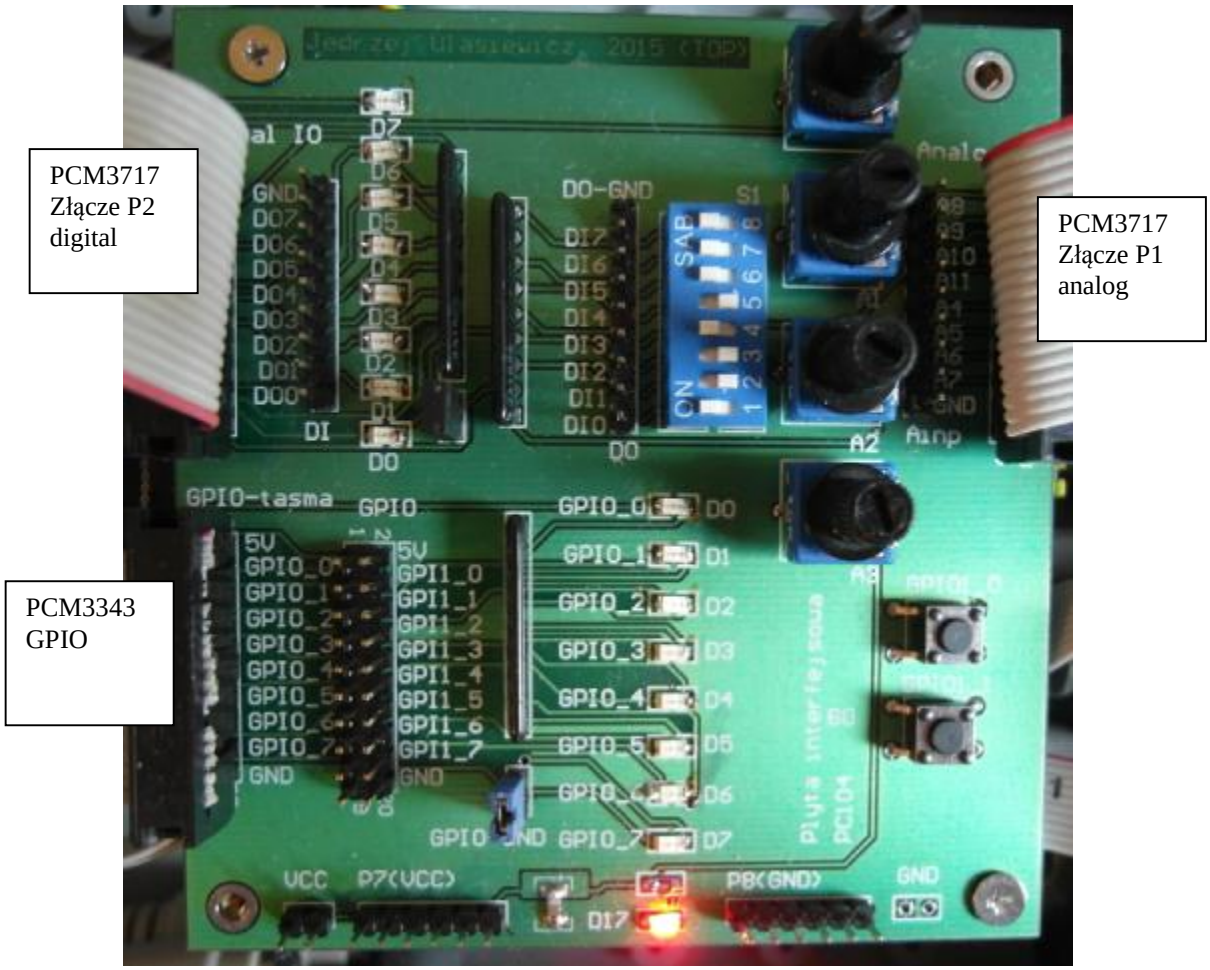
```
int aread(unsigned int *chan) {
  unsigned int stat,al,ah;
  unsigned int x,xh,xl;
  int i = 0;
  // Start konwersji
  ...
   do {
       // Odczyt statusu EOC
       stat =  …
       i++;
       if(i >= 0xFFFF) return(-1);
   } while( … );
   al =  …
   ah =  …
   *chan = …
    x = …
    return(x);
}

main() {
   int val,val2, chn,j ;
   unsigned char d1,d2, i = 0;
   printf("Program startuje \n");
   ThreadCtl( _NTO_TCTL_IO, 0 );
   base = mmap_device_io(16,ADRB);
   card_init(0,3,5);
  do {
      for(j=0; j<7; j++) {
           val = aread(&chn);
           printf(" %d - %d ",chn,val);
      }
      d1 = dinp(1);
      printf("we1 %2X  \n",d1);
      usleep(500000);
      if(chn == 0) { // Regulacja
       …
      }
    } while(1);
}
```

Przykład 5-4 Obsługa przetwornika AD karty PCM-3718 w trybie odpytywania

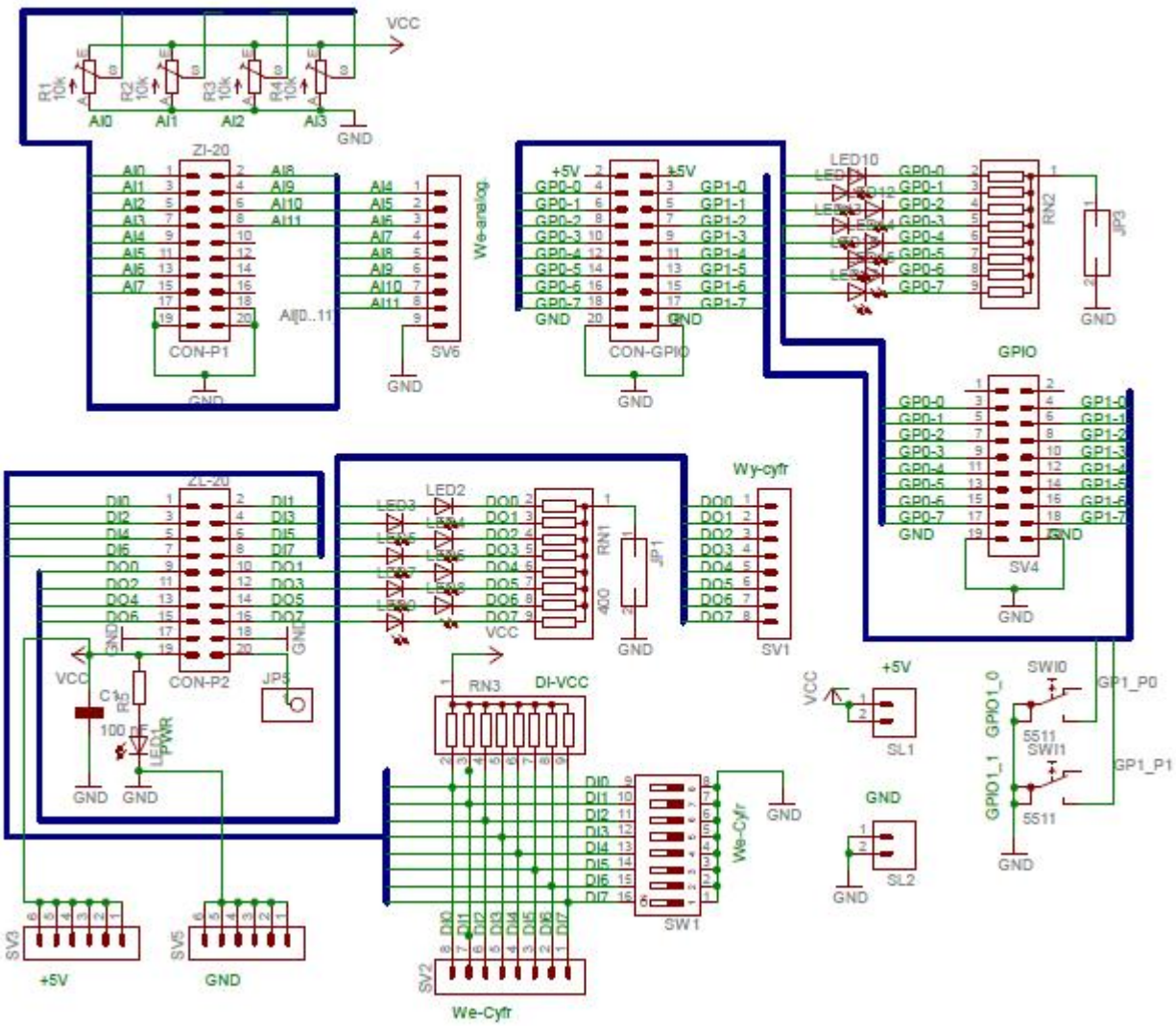
5.7 Płyta Interfejsowa

W celu umożliwienia obserwacji wyjść i zadawania parametrów wejściowych opracowana została płyta interfejsowa. Umożliwia ona:

- Zadawanie wejść cyfrowych portu DINP1 (we/wy 1) za pomocą przełącznika DIP Switch S1 i dołączanie wejść zewnętrznych przez złącze DI
- Obserwację wyjść cyfrowych portu DOUT2 (we/wy 2) za pomocą diod LED1do LED8 i dołączanie wyjść zewnętrznych przez złącze DO
- Zadawanie wielkości analogowych kanałów A0, A1, A2, A3 za pomocą potencjometrów.
- Dołączanie zewnętrznych wejść analogowych A4,..A11
- Obserwację wyjść cyfrowych interfejsu GPIO0, dołączanie układów zewnętrznych przez złącze GPIO
- Zadawanie wejść cyfrowych interfejsu GPIO1 (GPIO1_0 i GPIO1_1)
- Dołączanie układów zewnętrznych przez złącze GPIO



Fotografia 5-5 Płyta interfejsowa, widok



Rys. 5-4 Schemat płyty interfejsowej

5.8 Interfejs GPIO

Komputer PCM3343 zawiera dwa 8 bitowe układy wejść wyjść cyfrowych nazywane GPIO (ang. *General Purpose Input Output*). Każda linia GPIO może być skonfigurowana jako wejście lub jako wyjście. Kierunek ten zależy od wpisu w rejestrze kierunku co pokazuje poniższa tabela.

	GPIO0	GPIO1
Rejestr danych	0x78	0x79
Rejestr kierunku	0x98	0x99

Tab. 5-4 Rejestry GPIO

Gdy w danym bicie rejestru kierunku jest 0 to bit ten jest wejściem a gdy 1 to wyjściem. Przykładowo:

- Wysłanie na port 0x98 bajtu 0x00 powoduje że bity [7..0] portu GPIO0 pracują w trybie wejścia
- Wysłanie na port 0x98 bajtu 0xFF powoduje że bity [7..0] portu GPIO0 pracują w trybie wyjścia
- Wysłanie na port 0x98 bajtu 0x0F powoduje że bity [7..4] portu GPIO0 pracują w trybie wejścia a bity [3..0] w trybie wyjścia.
- Wysłanie na port 0x98 bajtu 0x03 powoduje że bity [7..2] portu GPIO0 pracują w trybie wejścia a bity [2..0] w trybie wyjścia.

Rejestry danych służą do wpisywania lub odczytywania danych.

```
// Test wejsc i wyjsc cyfrowych GPIO
#include <stdlib.h>
#include <stdio.h>
#define GPIO0_DATA 0x78
#define GPIO1_DATA 0x79
#define GPIO0_DIR 0x98
#define GPIO1_DIR 0x99
#define WE 0x00
#define WY 0xFF

static int base = GPIO0_DATA;

main() {
   int val,chn,j ;
   unsigned char i = 0;
   printf("Vortex - test  GPIO \n");
   ThreadCtl( _NTO_TCTL_IO, 0 );
   base = mmap_device_io(20,GPIO0_DATA);
   // Inicjacja karty ----------------
   // GPIO0 - wyjscie
   out8(GPIO0_DIR, 0xFF);
  // GPIO1 - wejscie
   out8(GPIO1_DIR, 0x00);
   // Test wyjsc - przesowna jedynka
   val = 1;
    for(i=0;i<8;i++) {
            printf("%2X  ",val);
            out8(GPIO0_DATA,val);
            usleep(500000);
            val = val << 1;
    }

     // Test wejsc cyfrowych GPIO - 1 ------------------------
     printf("test wejsc GPIO1 \n");
     for(i=0;i<10;i++) {
            val = in8(GPIO1_DATA);
            printf("wejscie: %2X\n",val);
            out8(GPIO0_DATA,~val);
            sleep(1);
     }
 }
```

Przykład 5-1 Obsługa GPIO – zapis i odczyt

5.9 Komunikacja z komputerem macierzystym

Aby ustanowić komunikację pomiędzy komputerem Vortex a macierzystym komputerem PC należy wykonać następujące kroki:

1. Połączyć kablem typu null modem złącze komunikacyjne COM1 komputera Vortex ze złączem RS232 komputera PC, połączyć kablem Ethernet złącze sieciowe komputera Vortex z siecią A (ta sama podsieć co komputer PC), dołączyć zasilanie.
2. Na komputerze PC uruchomić Hyper Terminal (lub inny program terminalowy RS232). Ustawić parametry komunikacji jak poniżej.

Ekran 5-1 Parametry komunikacji RS232 z komputerem Vortex

Po chwili powinno się pokazać zgłoszenie konsoli portu szeregowego komutera Vortex

```
login: root
Wed Sep 19 19:34:37 2018 on /dev/ser1
Last login: Thu Jan 18 21:27:19 2018 on /dev/ttyp0
edit the file .profile if you want to change your environment.
To start the Photon windowing environment, type "ph".
# _
```

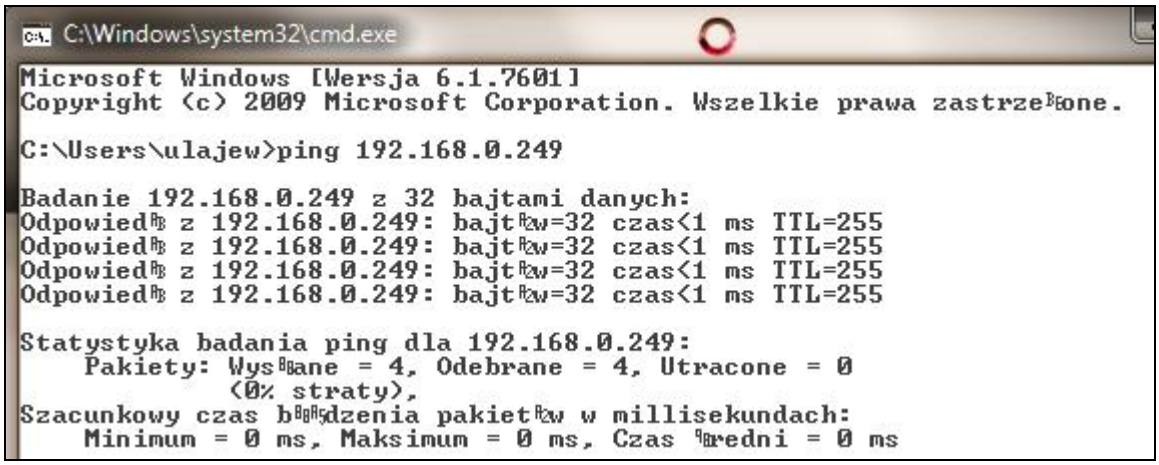
3. Uruchomić program `dhcp.client` w celu wysłania zgłoszenia do serwera DHCP z żądaniem nadania adresu IP.

   `# dhcp.client`

4. Następnie sprawdzić adres IP za pomocą polecenia `netstat -i`

```
# netstat -i
Name  Mtu   Network       Address              Ipkts Ierrs    Opkts Oerrs Colls
lo0   33192 <Link>                                 0     0        0     0     0
lo0   33192 127/8         localhost.localdo        0     0        0     0     0
en0   1500  <Link>        00:0b:ab:84:3a:ee      294     0       39     0     0
en0   1500  192.168.0/24  192.168.0.249          294     0       39     0     0
```

5. Za pomocą narzędzia ping sprawdzić czy koputer Vortex odpowiada.



6. Gdy komputer Vortex otrzyma adres IP można się z nim łączyć za pomocą narzędzi opisanych w poprzednim rozdziale:

- Phwindows
- Momentics
- Telnet
- FTP

5.10 Zadania

Zadanie 5. 4. 1 Obsługa wejść / wyjść cyfrowych – Komputer PC104

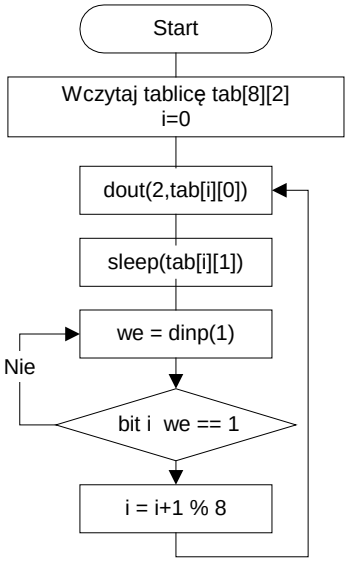
Korzystając z podanego w rozdziale 5.5 przykładu napisz i uruchom w systemie docelowym program który:

- Odczytuje stan mikro przełączników z wejść cyfrowych i wyświetla je na konsoli
- Zapala kolejno diody LED sterowane wyjściami cyfrowymi

Zadanie 5. 4. 2 Sterowanie sekwencyjne – Komputer PC104

Sterowanie sekwencyjne polega na załączaniu urządzeń dwustanowych (reprezentowanych przez wyjścia cyfrowe) zgodnie z zadaną wcześniej sekwencją. Przejście do kolejnej sekwencji uzależnione jest od czasu i może być uzależnione od czynników zewnętrznych np. od stanu wejść cyfrowych. Sterowanie sekwencyjne może być opisane za pomocą sekwencji par `(wy(i), T(i)) i=0,1,...7` gdzie i jest numerem kroku, `wy(i)` jest stanem wyjść cyfrowych a `T(i)` opóźnieniem przejścia do stanu `i=(i+1)%8`. Dodatkowym warunkiem przejścia do stanu i+1 będzie wymaganie aby wartość bitu i wejścia cyfrowego `we(i)` była równa 1.

Napisz program który realizuje sterowanie sekwencyjne z wykorzystaniem karty PCM3718. Z tablicy `tab[8][2]` program pobiera pary: wartość wyjścia cyfrowego `wy(i),` opóźnienie w sekundach T(i). Następnie wyprowadza na wyjścia cyfrowe żądany bajt wyjściowy wykonując funkcję: `dout(2,wy(i)).` Po odczekaniu Ti sekund odczytuje wartość wejść cyfrowych `we(i)=dinp(1)` i sprawdza czy bit i równy jest 1. Gdy tak przechodzi do kroku i+1, gdy nie czeka. Po wyczerpaniu tablicy sterującej (gdy i ==7) proces zaczyna się od początku.



Rys. 5-5 Przebieg sterowania sekwencyjnego

Zadanie 5. 4. 3 Wyświetlanie stanu wejść analogowych i cyfrowych

Napisz program który cyklicznie wyświetla stan wejść analogowych A0-A3 wraz z numerem kanału oraz stan wejść cyfrowych w postaci szesnastkowej. Program powinien działać w pętli z 0.5 sekundowym opóźnieniem (funkcja usleep). Stan kanału A0 powinien być wyświetlany w postaci linijki diodowej korzystając z wyjść cyfrowych DOUT2.

Np:

```
A0 = 0    A1 = 2346  A2 = 3321  A3 = 4095    DINP = 10011100
```

Aby to zrealizować należy:

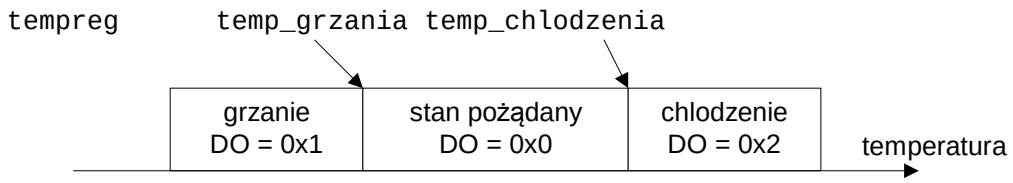
1. Zainicjować przetwornik AD aby odczytywał kanały od 0 do 3 a jako zakres pomiarowy przyjąć 0-5 V. Funkcje te wykonuje procedura `card_init(int from, int to, unsigned char zakres)`.
2. Odczyt wejść analogowych ma być realizowany przez procedurę `int aread(unsigned int *chan)`. Procedura zwraca mierzona wartość i nadaje zmiennej chan wartość aktualnie zmierzonego kanału.
3. W pętli wykonywać odczyt kanałów A0 do A3 poprzez 4 krotne wywołanie funkcji `aread` i wyświetlić numer i stan tych kanałów na konsoli.
4. Odczytać i wyświetlić stan wejść cyfrowych.
5. Wyświetlić stan kanału A0 w postaci linijki diodowej

Użyć GPIO_0 do wyświetlenia stanu wejść cyfrowych DINP0.

Zadanie 5. 4. 4 Obsługa wejść analogowych – regulator trój położeniowy

Napisz program realizujący funkcje trójpołożeniowego regulatora (np. temperatury). Wywołanie programu:

```
$ tempreg temp_grzania temp_chlodzenia
```



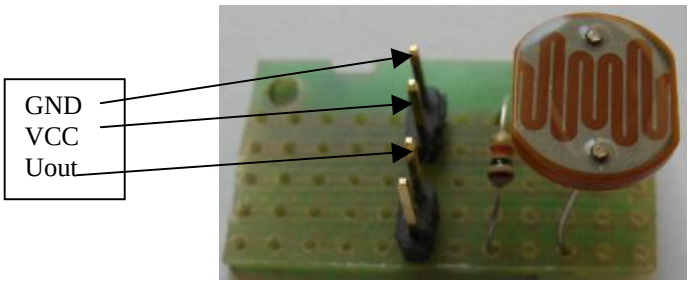
Na GPIO_0 proszę wyświetlić w postaci linijki diodowej stan kanału A0.
Aby to zrealizować należy:

1. Zainicjować przetwornik AD aby odczytywał kanał A0 a jako zakres pomiarowy przyjąć 0-5 V. Funkcje te wykonuje procedura `card_init(int from, int to, unsigned char zakres)`.
2. W pętli wykonywać odczyt kanału A0 poprzez e wywołanie funkcji `aread` a następnie w zależności od zmierzonej wartości uruchomić grzanie (grzaniem steruje bit 0 wyjścia cyfrowego) lub chłodzenie (chlodzeniem steruje bit 1 wyjścia cyfrowego). Używając GPIO0 wyświetlić w postaci liniki diodowej stan kanału A0.
3. Odczytać i wyświetlić stan wejść cyfrowych.

6. Obsługa czujników pomiarowych

6.1 Czujnik oświetlenia na fotorezystorze

Fotorezystor (fotoopornik) jest przyrządem półprzewodnikowym który zmienia swą rezystancję w zależności od padającego na niego promieniowania. Zachowanie jego nie zależy od kierunku przepływu prądu. Różnica prądu płynącego przez oświetlony fotorezystor i prądu płynącego przez fotorezystor przy braku oświetlenia, nazywamy prądem fotoelektrycznym. Jego wartość zależy od natężenia oświetlenia. Fotorezystory wykonuje się najczęściej w postaci półprzewodnikowych warstw monokrystalicznych lub polikrystalicznych naniesionych na podłoże izolacyjne (szklane lub ceramiczne). Warstwa światłoczuła umieszczona jest pomiędzy dwiema metalowymi elektrodami, najczęściej mającymi postać grzebienia, do których dołączone są wyprowadzenia co pokazuje poniższa fotografia. Zaletą fotorezystora jest prostota, wadą zaś wrażliwość na temperaturę i znaczna bezwładność. Fotorezystory stosuje się do pomiaru oświetlenia pomieszczeń, wyłącznikach zmierzchowych.

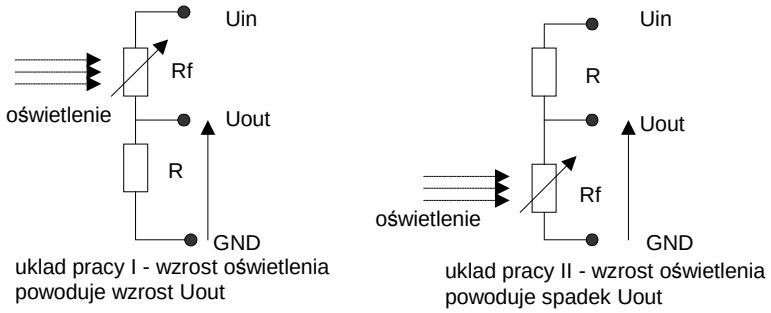


Fotografia 6-1 Fotorezystor w układzie dzielnika napięcia

Najważniejsze parametry fotorezystora to:

- Rezystancja ciemna R_d – rezystancja gdy element jest zaciemniony
- Czułość widmowa – zależność rezystancji od oświetlenia
- Zakres widmowy – zakres długości fali przy której czułość nie spada więcej niż 10% wartości maksymalnej
- Współczynnik n – stosunek rezystancji ciemnej R_d do rezystancji R_{50}przy oświetleniu 50 luksów $n = R_d / R_{50}$.

Podstawowe układy pracy fotorezystora podaje poniższy rysunek.



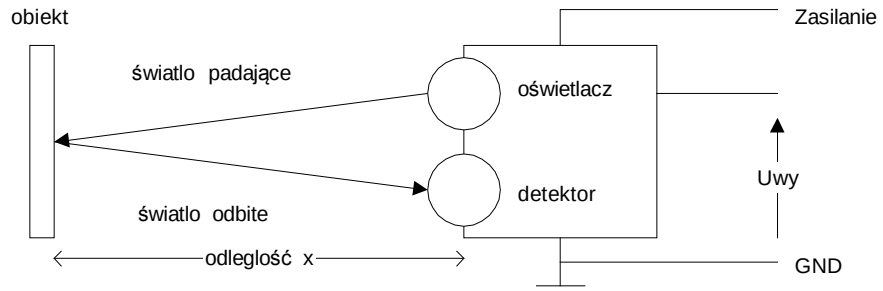
Rys. 6-1 Układy pracy fotorezystora

Jak widać fotorezystor Rf jest elementem dzielnika napięciowego zasilanego napięciem Uin. Szeregowo z fotorezystorem dołączony jest stały rezystor R a Uout jest napięciem wyjściowym dołączonym do przetwornika analogowo cyfrowego. W układzie I wzrost oświetlenia powoduje spadek rezystancji Rf a zatem wzrost napięcia Uout wyrażonego wzorem Uout = R * Uin/(R + Rf). W układzie II wzrost oświetlenia powoduje spadek rezystancji Rf a zatem spadek napięcia Uout wyrażonego wzorem Uout = Rf * Uin/(R + Rf). Doboru rezystancji R dokonać można rozwiązując układ równań dla różnych wartości rezystancji Rf. Należy zmierzyć rezystancję ciemną R_d i rezystancję R_{max} odpowiadającą maksymalnemu oświetleniu. Należy jednak założyć że prąd płynący przez rezystory nie może być zbyt duży bo spowoduje nagrzewanie fotorezystora i nadmierny pobór mocy. W praktyce przyjmuje się R równe około 10 kΩ. Autor wykonał eksperyment z posiadanym fotorezystorem pracującym w układzie I i zasilanym napięciem 5V. Wartość rezystora R wynosiła 10 kΩ.. Przy braku oświetlenia napięcie Uout wynosiło 0.85V a przy oświetleniu pełnym Uout wynosiło 4.86 V.

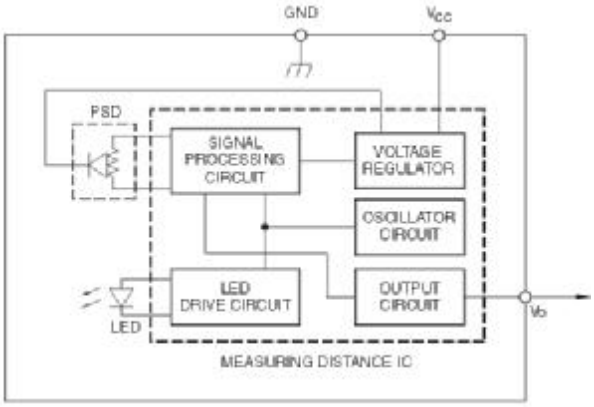
6.2 Czujnik odległość na podczerwień z wyjściem analogowym

Jedną z szeroko stosowanych metod pomiaru odległości jest wykorzystanie czujnika na podczerwień z wyjściem analogowym. Do określania odległości wykorzystuje się pomiar siły światła odbitego od obiektu. Urządzenie składa się

z oświetlającej diody LED pracującej w podczerwieni i detektora mierzącego natężenie światła odbitego. Pomiar polega na wykorzystaniu zależności natężenia odbitego światła od odległości od obiektu.

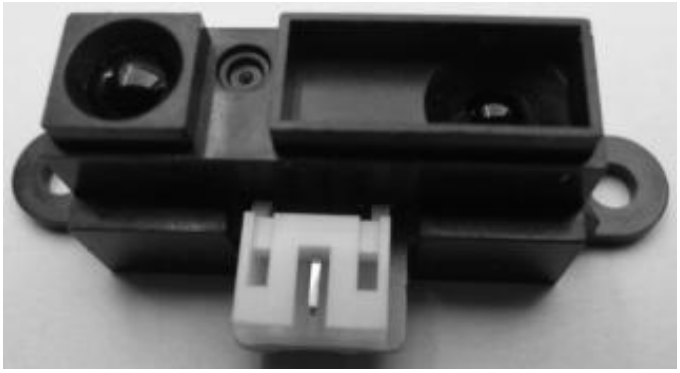


Rys. 6-2 Zasada działania czujnika odległości na podczerwień

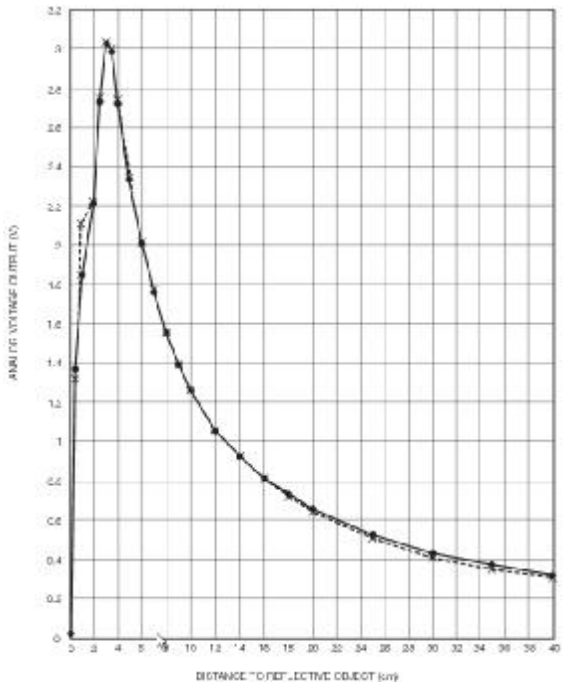


Rys. 6-3 Budowa czujnika odległości na podczerwień

W czujniku takim istnieje zależność Uwy = f(x) pomiędzy odległością a napięciem wyjściowym. Zależność ta zwykle nie jest ani liniowa ani nawet monotoniczna. Przykładem takiego urządzenia jest czujnik SHARP GP2Y0A41SK0F.



Fotografia 6-2 Wygląd czujnika SHARP GP2Y0A41SK0F lub 2D I 120X F

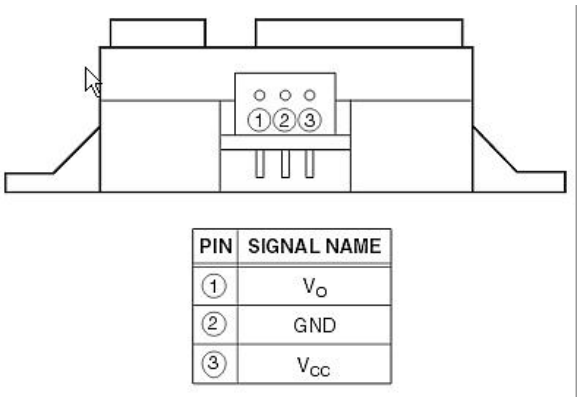


Rys. 6-4 Zależność napięcia wyjściowego od odległości dla czujnika 2D I 120X

Czujnik ten zasilany jest napięciem 0.3 – 7V i wykorzystuje długość fali λ=870nm. Posiada on trzy łączówki: zasilanie, napięcie wyjściowe i ziemia co pokazuje poniższa tabela.

Numer złącza	Symbol	Opis	Typowa wartość	Kolor przewodu
1	Uwy	Napięcie wyjściowe	0-Vcc	Żółty
2	GND	Ziemia	0	Czarny
3	Vcc	Zasilanie	3-5 V	Czerwony

Tabela 6-1 Opis łączówek czujnika odległości GP2Y0A41SK



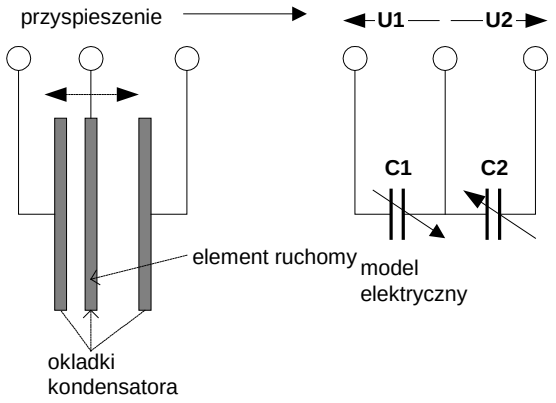
Rys. 6-5 Wyprowadzenia układu do pomiaru odległości SHARP 2D I 120X F

Według danych katalogowych zakres pomiaru wynosi od 4 do 30 cm ale w praktyce jest on większy (do 70-100 cm).

6.3 Czujnik przyspieszenia

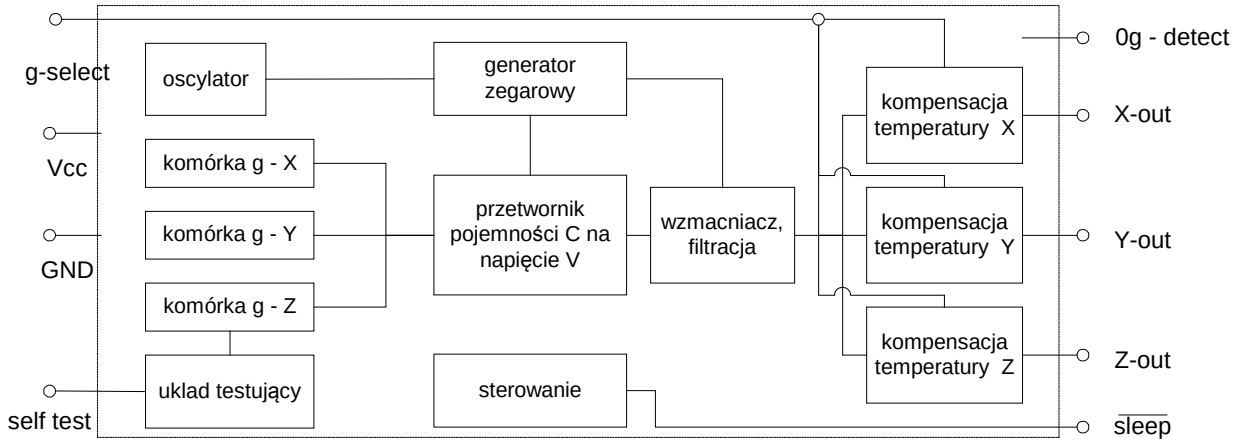
Czujnik przyspieszenia – akcelerometr, mierzy przyspieszenie w określonym kierunku. Mierzone jest przyspieszenie statyczne (ziemskie) i dynamiczne. Często w jednym urządzeniu zintegrowane są trzy czujniki mierzące przyspieszenie w trzech kierunkach X,Y,Z. Jako że mierzone jest przyspieszenie ziemskie akcelerometr pozwala na określenie orientacji urządzenia względem pionu. Pomiar może odbywać się na różnych zasadach ale najczęściej wykorzystywane są czujniki pojemnościowe i piezoelektryczne. Czujniki przyspieszenia używane są w urządzeniach mobilnych np. telefonach (do określenia położenia urządzenia), w robotyce do wykrywania ruchu, stałych dyskach (do

wykrywania upadku), nawigacji (krokomierz), elektronice samochodowej (pasy bezpieczeństwa) i innych urządzeniach.
Przykładem akcelerometru jest czujnik MMA7361L firmy Freescale Semiconductor. Układ ten posiada trzy zintegrowane czujniki mierzące przyspieszenia w prostopadłych wzajemnie osiach X,Y,Z, wykrywanie stanu zerowego przyspieszenia (upadek), dwa zakresy czułości oraz układ samo testowania. Układ wykorzystuje zjawisko zmiany pojemności pomiędzy okładkami kondensatora utworzonego z kilku warstw materiału o różnych elektrycznych i mechanicznych własnościach co pokazuje poniższy rysunek.



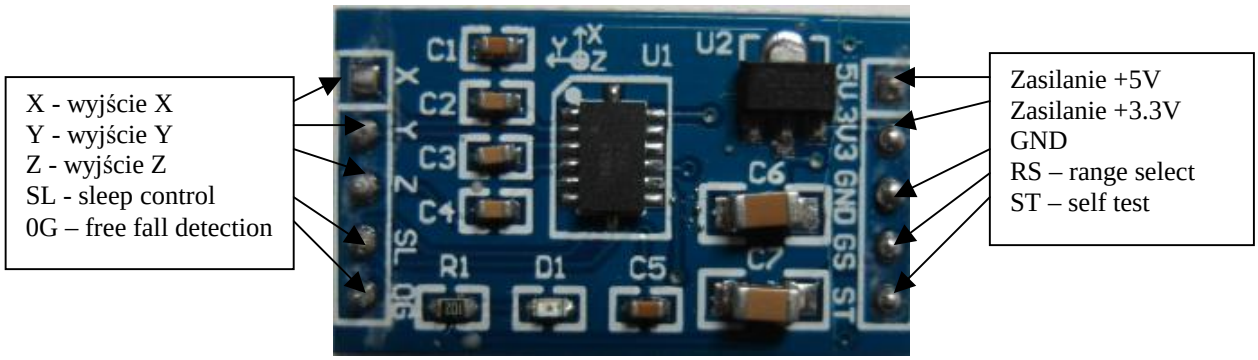
Rys. 6-6 Zasada działania akcelerometru pojemnościowego

Układ składa się z elementu ruchomego umieszczonego pomiędzy nieruchomymi okładkami kondensatora. Pod wpływem przyspieszenia, dzięki swojej masie, przemieszcza się on w przeciwnym do wektora przyspieszenia kierunku przez co odległości pomiędzy pojemności elementem środkowym a skrajnymi ulegają zmianie. Zatem i pojemności C1 i C2 pomiędzy okładkami takiego podwójnego kondensatora ulegają zmianie. Wpływa to na zmianę stosunku napięć U1 i U2 co może być wykryte przez odpowiednie układy elektroniczne. Opisany układ tworzy tak zwaną komórkę G (ang. g-cell). Układ składa się z trzech takich komórek zorientowanych w wzajemnie prostopadłych kierunkach X,Y,Z tworzących kartezjański układ współrzędnych. Budowę wewnętrzną układu MMA7361L pokazuje poniższy rysunek.



Rys. 6-7 Uproszczony schemat akcelerometru MMA7361L

Składa się on z trzech komórek G podłączonych do przetwornika pojemność – napięcie, układów filtrujących i wzmacniających układów sterujących i kompensacji temperatury. Układ powinien być zasilany napięciem 2.2 – 3.6 V. Posiada on wyjścia typu analogowego, napięcia wyjściowe X-out, Y-out, Z-out są proporcjonalne do przyspieszenia w danym kierunku. Dla przyspieszenia 0g napięcie wyjściowe wynosi 1.65V. Zmiany napięcia są uzależnione od wybranego zakresu pomiarowego, do dyspozycji są dwa zakresy pomiarowe ± 1.5g i ± 6g które wybierane są zewnętrznym sygnałem g-select (0 zakres ± 1.5g 1 zakres ± 6g). Czułość układu wynosi 800 mV/g dla zakresu 1.5g. Układ posiada system wykrywania swobodnego spadku, gdy przyspieszenia we wszystkich kierunkach wynoszą 0g, aktywowane jest wtedy wyjście 0g-detect. W oparciu o układ MMA7361L zbudowano moduł akcelerometru (LC Technology – www.lctech-inc.com). Moduł zawiera regulator napięcia z 5V do 3,3V, diodę sygnalizacyjną i wyprowadzenia. Wygląd zewnętrzny pokazano poniżej.



Rys. 6-8 Moduł akcelerometru z układem MMA7361L

Opis wyprowadzeń układu podaje poniższa tabela.

Oznaczenie	Ang.	Typ	Opis	Podłączenie testowe
X	Xout	wy analog.	Napięcie wyjściowe osi X	We analog. A4
Y	Yout	wy analog.	Napięcie wyjściowe osi Y	We analog. A5
Z	Zout	wy analog.	Napięcie wyjściowe osi Z	We analog. A6
SL	Sleep control	wejście	Gdy podany poziom wysoki – układ pracuje, gdy podany poziom niski – układ uśpiony	Vcc
0G	0 g	wyjście	Gdy wykryto spadek swobodny generowany jest poziom wysoki	
5V	-	-	Zasilanie 5V	Vcc 5V
3V3	-	-	Zasilanie 3.3V	
GND	Ground	-	Ziemia	GND
GS	Range select	wejście	Gdy poziom niski – zakres pomiarowy ±1.6g, gdy poziom wysoki - zakres pomiarowy ± 6g.	GND
ST	Self test	wejście	Gdy poziom wysoki - ryb samo testowania	

Tabela 6-2 Opis sygnałów wyjściowych akcelerometru

6.4 Sonar

Moduł HC-SR04 jest ultradzwiękowym sensorem służącym do wyznaczania odległości do przedmiotu w zakresie os 2 cm do 4.5 m.

Fotografia 6-3 Moduł HC-SR04

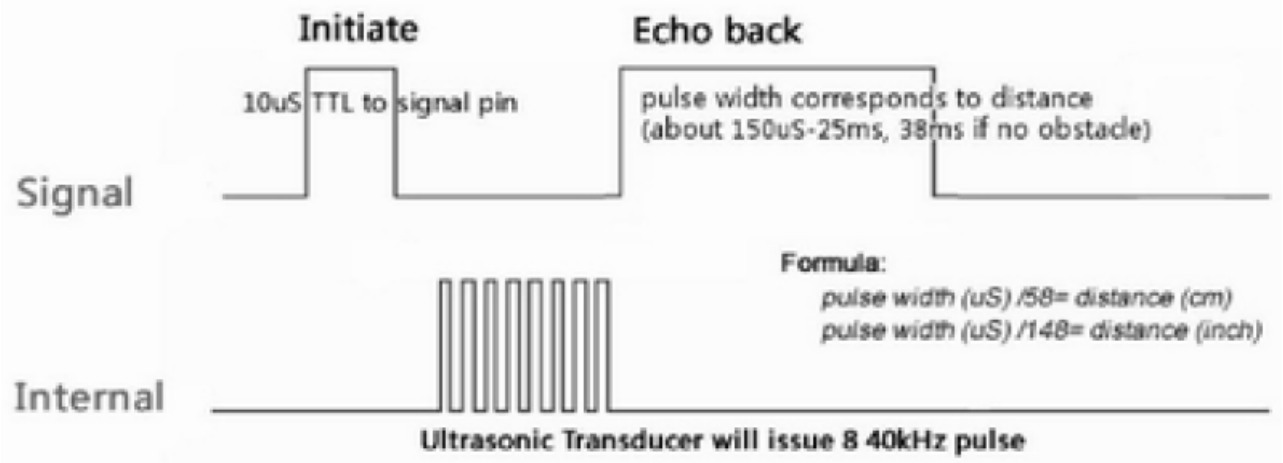
Własności modułu podaje poniższa specyfikacja a opis podaje [16].

- Power Supply :+5V DC
- Quiescent Current : <2mA
- Working Current: 15mA
- Effectual Angle: <15°
- Ranging Distance : 2cm – 400 cm/1" - 13ft
- Resolution : 0.3 cm
- Measuring Angle: 30 degree
- Trigger Input Pulse width: 10uS
- Dimension: 45mm x 20mm x 15mm

Opis połączeń:

Trig	- wyzwalanie (wejście)	- połączyć do DI0
Echo	- wyjście echa	- połączyć do DO0
GND	- ziemia	- połączyć do GND
VCC	- 5V	- połączyć do VCC

Aby wystartować pomiar należy do wejścia Trig podać impuls długości min 10 us. Następnie na wyjściu Echo należy zmierzyć czas pomiędzy zboczem narastającym a opadającym jak pokazuje poniższy rysunek. Odległość w centymetrach wylicza się wylicza się dzieląc długość impulsu w mikrosekundach przez 58.



Rys. 6-9 Pomiar odległości za pomocą czujnika HC-SR04

```
// Karta PCM 3718H - ultradzw. czujnika odleglosci  HC-SR04
//
// Podlaczenie czujnika:
// CZUJNIK -- KARTA ----------------------
// TRIG        P2_0      OUTPUT
// ECHO        P1_0        INPUT
// VCC         +5V
// GND         GND
// ---------------------------------------------
// Uwaga - przelacznik S1 ma byc na 0
//
#include "pcm3817.h"
#include <sys/syspage.h>
#include <inttypes.h>
#include <sys/neutrino.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define ADRB 0x300
#define WE 1
#define WY 2
#define SIZE 10000

static int base = ADRB;
unsigned char tab[SIZE];
```

```
void dout(int num, unsigned char val) {
// Sterowanie portem wyjsc cyfrowych
// num - port (1,2) , val - wartosc (0 - 255)
  if(num <= 1) {
    out8(base + DIOL, val);
  } else {
    out8(base + DIOH, val);
  }
 }


  unsigned char dinp(int num) {
  // Odczyt portu wejsc cyfrowych
  // // num - port (1,2)
  if(num <= 1) {
    return( in8(base + DIOL));
  } else {
   return( in8(base + DIOH));
  }
 }


int main(int argc, char *argv[]) {
   int i, val1,val2, chn, kroki,cnt = 0, cnt1 = 0,j,cnt2 ;
   unsigned char val;
   uint64_t start,stop,cps,czas_imp, biez,rozn;
   double msec;
   struct timespec res;
   kroki = 9999;
   printf("Program startuje - kroki %d \n",kroki);
   ThreadCtl( _NTO_TCTL_IO, 0 );
   base = mmap_device_io(16,ADRB);
   cps = SYSPAGE_ENTRY(qtime)->cycles_per_sec;
   if(clock_getres(CLOCK_REALTIME,&res) == -1) {
        perror("getres");
        return EXIT_FAILURE;
   }
   printf("Cykli na sek %ld rozdzielczosc zegara %ld %ld nanosek\n",cps,res.tv_sec,res.tv_nsec);
   printf("Czujnik odleglosci HC-SRO4 max krokow: %d\n",kroki);
   dout(1,0x00);
   usleep(100);
   i= 0;
   printf("Start \n",val);
   start = ClockCycles();

   // Wyzwolenie ----
   dout(2,0xFF);
   usleep(10);
   dout(2,0x00);
   cnt = 0;

   do {
       val = dinp(1);
       val = val & 0x01;
       tab[cnt] = val;
       cnt++;
       if(cnt == SIZE) {
               printf("Koniec \n");
               return 0;
        }
     } while(cnt< kroki);
     stop = ClockCycles();
     printf("Cykli procesora %ld\n",stop - start);

     for(i=0;i<kroki;i++) printf("%01X ",tab[i]);
}
```

Przykład 6-1 Program `czuj_odl_1.c` odczytujący dane z czujnika odległości

```
Program startuje - kroki 9999
Cykli na sek 800264200 rozdzielczosc zegara 0 0 nanosek
Czujnik odleglosci HC-SRO4 max krokow: 9999
Start
Cykli procesora 13110468
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Przykład 6-2 Wyniki działania programu czuj_odl_1

6.5 Zadania

Zadanie 6. 4. 1 Pomiar odległości za pomocą czujnika odległości HC-SR04

Zapoznać się z dokumentacją czujnika. Należy dołączyć czujnik do płyty interfejsowej karty PCM3718 tak aby jego wyzwalanie Trig podłączone było do wyjść cyfrowych `dout(2,...)` a wyjście Echo do wejść cyfrowych `val = dinp(1)`

podłączając także ziemię i zasilanie 5V.

- Napisać program odczytujący odległość podawaną przez czujnik.
- Zademonstrować pomiar odległości za pomocą czujnika. Odległość ma być wyświetlana na konsoli i linijce diodowej zrealizowanej na GPIO0.

Do pomiaru czasu proszę wykorzystać licznik cykli procesora czyli funkcję `cykle = ClockCycles`(). Ilość cykli na sekundę podaje funkcja:
`cps = SYSPAGE_ENTRY(qtime)->cycles_per_sec`;

Zadanie 6. 4. 2 Pomiar oświetlania za pomocą fotorezystora

Podłącz czujnik z fotorezystorem do kanału analogowego Analog input A4 zgodnie z poniższą tabelą. Położenie wyprowadzeń pokazano na Fotografia 6-1.

Łączówka fotorezystora	Łączówka płyty interfejsowej
GND	P8(GND)
VCC	P7(VCC)
Uout	Ainp A4

Tab. 6-1 Połączenie układu pomiaru oświetlenia z płytą interfejsową

Napisz program mierzący napięcie w kanałach A0-A4 i wyświetl wartość kanału A4 na linijce diodowej podłaczonej do GPIO0.

Zadanie 6. 4. 3 Pomiar odległości za pomocą czujnika odległości GP2Y0A41SK

Zapoznać się z dokumentacją czujnika GP2Y0A41SK. Należy dołączyć czujnik do płyty interfejsowej karty PCM3718 tak aby jego napięcie wyjściowe Uwy podłączone było do wejścia analogowego Ain7 podłączając także ziemię i zasilanie 5V.

- Napisać program odczytujący poziom napięcia wyjściowego generowanego przez czujnik. Dobrać zakres pomiarowy przetwornika AD.
- Zdjąć 8 punktową charakterystykę czujnika określającą zależność pomiędzy napięciem wyjściowym a odległością od obiektu i przedstawić ja w postaci wykresu.
- Opracować funkcję aproksymującą zależność x = F(Uwy) i zaimplementować ją programowo.
- Zademonstrować pomiar odległości za pomocą czujnika. Odległość ma być wyświetlana na konsoli i linijce diodowej podłaczonej do GPIO0.

Zadanie 6. 4. 4 Pomiar przyspieszenia za pomocą akcelerometru MMA7361L

Podłącz moduł akcelerometru do wejść analogowych A4, A5, A6 karty interfejsowej PCM3718. Najpierw podłączyć ziemię i zasilanie +5V a potem pozostałe sygnały. Aby układ pracował poprawnie należy wejście SL połączyć z sygnałem poziomu wysokiego.

- Napisać program wyświetlający na konsoli napięcia odpowiadające sygnałom A4 - X, A4 - Y, A6 - Z.
- Uzupełnić program o sygnalizację na diodach D0-D7 położenia osi X a na wyjściu GPIO0 położenia osi Y

7. Tworzenie procesów w systemach RTS

7.1 Wstęp

Do tworzenia nowych procesów wykorzystuje się funkcję `fork`. Proces bieżący przekształca się w inny proces za pomocą funkcji `exec`. Funkcja `exit` służy do zakończenia procesu bieżącego, natomiast funkcji `wait` używa się do oczekiwania na zakończenie procesu potomnego i do uzyskania jego statusu.

- Funkcja `int fork()` - powoduje utworzenie nowego procesu będącego kopią procesu macierzystego. Segment kodu jest taki sam w obu zadaniach. Proces potomny posiada własny segment danych i stosu. Funkcja zwraca 0 w kodzie procesu potomnego a PID nowego procesu w procesie macierzystym (lub -1 gdy nowy proces nie może być utworzony).

- Fukcja execl(fname,arg1,arg2,...,NULL) przekształca bieżący proces w proces o kodzie zawartym w pliku wykonywalnym fname, przekazując mu parametry arg1,arg2, itd.

- Funkcja pid = wait(&status) powoduje zablokowanie procesu bieżącego do czasu zakończenia się jednego zadania potomnego. Gdy zad. potomne wykona funkcję exit(status); funkcja zwróci PID procesu potomnego i nada wartość zmiennej status. Gdy nie ma procesów potomnych funkcja wait zwróci -1.

- Funkcja exit(int stat) powoduje zakończenie procesu bieżącego i przekazanie kodu powrotu stat do procesu macierzystego.

Podstawowy schemat tworzenia nowego procesu podany jest poniżej.

```
#include <stdio.h>
#include <process.h>
#include <unistd.h>

void main(void){
  int pid,status;
  if((pid = fork()) == 0) { /* Proces potomny ---*/
      printf(" Potomny = %d \n",getpid());
      sleep(30);
      exit(0);
  }
 /* Proces macierzysty */
 printf("Macierzysty = %d \n",getpid());
 pid = wait(&status);
 printf("Proces %d zakończony status: %d\n",pid,status);
}
```

Przykład 7-1 Podstawowy wzorzec tworzenia procesu potomnego

7.2 Schemat użycia funkcji spawn.

Funkcja `spawnl` używana jest do tworzenia nowych procesów.

Funkcja 7-1 `spawnl` – utworzenie nowego procesu	
`pid_t spawnl(int mode, char * path, arg0,arg1,..., argN,NULL)`	
`mode`	Tryb wykonania procesu : P_WAIT, P_NOWAIT, P_OVERLAY, P_NOWAITO.
`path`	Ścieżka z nazwą pliku wykonywalnego.
`arg0`	Argument 0 przekazywany do funkcji `main` tworzonego procesu. Powinna być to nazwa pliku wykonywalnego ale bez ścieżki.
…	…
`argN`	Argument N przekazywany do funkcji `main` tworzonego procesu .

Funkcja zwraca PID nowo utworzonego procesu lub -1 gdy błąd.
Tryby wykonania:

```
P_WAIT    - proces czeka na zakończenie procesu potomnego
P_NOWAIT  - proces potomny wykonywany jest współbieżnie
```

```
P_OVERLAY - proces bieżący zastępowany jest przez proces potomny
```

Przykład programu tworzącego proces potomny za pomocą funkcji `spawnl` podano poniżej.

```
// Ilustracja działania funkcji spawnl – uruchomienie programu potomny
#include <stdio.h>
#include <process.h>
#include <sys/wait.h>
main(void){
  int pid,i,res,status;
  char buf[10];
  res = spawnl(P_NOWAIT,"potomny","potomny",”10”,NULL);
  if(res < 0) {
     perror("SPAWN"); exit(0);
  }
  for(i=1;i < 10;i++) {
       printf("Macierzysty - krok %d \n",i);
       sleep(1);
  }
  pid = wait(&status);
  printf("Proces %d zakonczony, status %d\n",pid,WEXITSTATUS(status));
}
```

Przykład 7-2 Przykład utworzenia procesu potomnego za pomocą funkcji `spawnl`

```
#include <stdlib.h>
main(int argc,char *argv[]) {
  int id, i ;
  for(i=1;i <= atoi(argv[1]);i++) {
       printf("Potomny krok: %d \n",i);
       sleep(1);
  }
  exit(i);
}
```

Przykład 7-3 Kod procesu potomnego

7.3 Zadania

Zadanie 7. 4. 1 Tworzenie procesów za pomocą funkcji fork - struktura 1 poziomowa.

Proces macierzysty o nazwie procm1 powinien utworzyć zadaną liczbę procesów potomnych PP-1, PP-2,..., PP-N za pomocą funkcji fork a następnie czekać na ich zakończenie. Zarówno proces macierzysty jak i procesy potomne powinny w pętli wyświetlać na konsoli swój numer identyfikacyjny jak i numer kroku w odstępach 1 sekundowych. Numer identyfikacyjny procesu potomnego NR wynika z kolejności jego utworzenia. Np. proces o numerze 3 wykonujący N kroków powinien wyświetlać napisy:

```
Proces 3 krok 1
Proces 3 krok 2
...............
Proces 3 krok N
```

Aby wykonać zadanie do procesu procm1 należy przekazać informacje:
- Ile ma być procesów potomnych
- Ile kroków ma wykonać każdy z procesów potomnych.

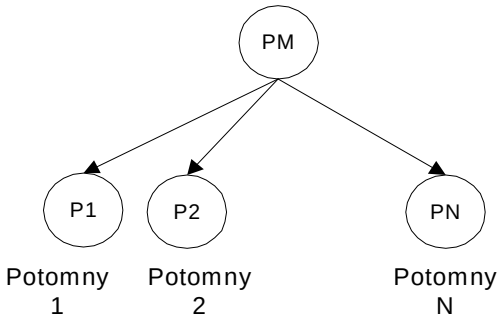
Informacje te przekazuje się jako parametry programu `procm1` z linii poleceń.

`procm1 K0 K1 K2 .... KN` gdzie:

`K0` - liczba kroków procesu macierzystego
`K1` - liczba kroków procesu potomnego `P1`

...
KN - liczba kroków procesu potomnego PN

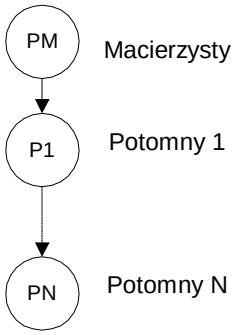
Np. wywołanie procm1 10 11 12 oznacza że należy utworzyć 2 procesy potomne i maja one wykonac 11 i 12 kroków. Proces macierzysty ma wykonać 10 kroków. Na zakończenie procesu potomnego powinien on wykonać funkcję `exit(NR)` przekazując jako kod powrotu swój numer identyfikacyjny procesowi macierzystemu. Proces macierzysty powinien czekać na zakończenie się procesów potomnych (funkcja `pid = wait(&status)`) i dla każdego zakończonego procesu wyświetlić: `pid` i kod powrotu. W tej wersji programu procesy potomne nie posiadają swoich procesów potomnych.



Rysunek 7-1 Proces macierzysty i procesy potomne – struktura dwupoziomowa

Zadanie 7. 4. 2 Tworzenie procesów za pomocą funkcji fork - struktura N poziomowa.

Zadanie to jest analogiczne do poprzedniego ale struktura tworzonych procesów ma być N poziomowa. Znaczy to że zarówno proces macierzysty jak i każdy proces potomny (z wyjątkiem ostatniego procesu N) tworzy dokładnie jeden proces potomny. Drzewo procesów będzie wyglądało jak poniżej.



Zadanie należy rozwiązać stosując funkcję rekurencyjną tworz(int poziom, char *argv[]). W funkcji tej argument poziom oznacza zmniejszany przy każdym kolejnym wykonaniu poziom wywołania funkcji a argument argv zadaną liczbę kroków.

Rysunek 7-2 Proces macierzysty i procesy potomne – struktura pionowa

Zadanie 7. 4. 3 Tworzenie procesów za pomocą funkcji fork i transformacja ich w inny proces.

Zadanie to jest podobne do zadania 1. Różnica jest taka ze procesy potomne powinny być przekształcone w inne procesy o nazwie `proc_pot` za pomocą funkcji `execl`.
Tak wiec:
- Proces macierzysty uruchamia się poleceniem `procm3 K0 K1 K2 .... KN`
- Proces macierzysty `procm3` powinien utworzyć zadaną liczbę procesów potomnych `PP-1, PP-2,...,PP-N` za pomocą funkcji `fork` a następnie wyprowadzić na konsole komunikaty:

Proces macierzysty krok 1
Proces macierzysty krok 2
.....
Proces macierzysty krok K0

Następnie proces macierzysty ma czekać na zakończenie się procesów potomnych. Dla każdego zakończonego procesu potomnego należy wyświetlić jego `pid` i kod powrotu.

- Procesy potomne przekształcają się w procesy `proc_pot` z których każdy ma wyświetlać w pętli na konsoli swój numer identyfikacyjny i numer kroku w odstępach 1 sekundowych. Numer identyfikacyjny i liczba kroków do wykonania ma być przekazana z procesu macierzystego jako parametr. Na zakończenie procesu potomnego powinien on wykonać funkcje `exit(NR)` przekazując jako kod powrotu swój numer identyfikacyjny procesowi macierzystemu.

- Procesy `proc_pot` należy utworzyć edytorem w postaci oddzielnych plików, skompilować, uruchomić i przetestować. Uruchomienie procesu `proc_pot` jako `proc_pot 4 6` powinno spowodować wyprowadzenie komunikatów:

```
Proces 4 krok 1
Proces 4 krok 2
...............
Proces 4 krok 6
Proces 4 zakończony
```

Zadanie 7. 4. 4 Tworzenie N współbieżnych procesów potomnych na jednym węźle

Proces macierzysty `spawnp` powinien utworzyć zadaną liczbę procesów potomnych PP-1, PP-2,...,PP-N za pomocą funkcji `spawnl`. Liczba procesów potomnych jest zadana przez liczbę parametrów podawanych z linii poleceń. Np. wywołanie `spawnp` 10 11 12 oznacza że należy utworzyć 2 procesy potomne i maja one wykonać 11 i 12 kroków. Proces macierzysty ma wykonać 10 kroków. Zarówno proces macierzysty jak i procesy potomne powinny w pętli wyświetlać na konsoli swój numer identyfikacyjny jak i numer kroku w odstępach 1 sekundowych. Numer identyfikacyjny procesu potomnego NR wynika z kolejności jego utworzenia.

Uwaga 1
Proces macierzysty uruchamia się poleceniem `spawnp` K0 K1 K2 .... KN

Uwaga 2
Proces macierzysty spawn powinien utworzyć N procesów potomnych za pomocą funkcji `spawnl` i wyprowadzić na konsole komunikaty:

```
Proces macierzysty krok 1
Proces macierzysty krok 2
...............
Proces macierzysty krok K0
```

Następnie proces macierzysty ma czekać na zakończenie się procesów potomnych. Dla każdego zakończonego procesu potomnego należy wyświetlić jego pid i kod powrotu (patrz makro WEXITSTATUS).

Uwaga 3
Każdy z procesów potomnych `proc_pot` ma wyświetlać na konsoli w pętli swój numer identyfikacyjny i numer kroku w odstępach 1 sekundowych. Np. proces o numerze 3 wykonujący N kroków powinien wyświetlać napisy:

```
Proces 3 krok 1
...............
Proces 3 krok N
```

Numer identyfikacyjny i liczba kroków do wykonania ma być przekazana z procesu macierzystego jako parametr. Na zakończenie procesu potomnego powinien on wykonać funkcję `exit(NR)` przekazując jako kod powrotu swój numer identyfikacyjny procesowi macierzystemu.

Uwaga ! Procesy `proc_pot` należy utworzyć edytorem w postaci oddzielnych plików, skompilować, uruchomić i przetestować. Proces macierzysty powinien czekać na procesy potomne wykonując odpowiednią liczbę razy funkcję `wait(&status)`. Należy wypisać każdorazowo komunikat o zakończeniu procesu potomnego i jego status (patrz makro `WEXITSTATUS`).

Zadanie 7. 4. 5 Tworzenie procesów na komputerze PC104

Podobnie jak w zadaniu pierwszym proces macierzysty o nazwie `czasy` powinien utworzyć zadaną liczbę procesów potomnych PP-1, PP-2,..., PP-N za pomocą funkcji `fork` a następnie czekać na ich zakończenie. Zarówno proces macierzysty jak i procesy potomne powinny w pętli wyświetlać na konsoli swój numer identyfikacyjny jak i numer kroku w odstępach zgodnych z ich opóźnieniem. Opóźnienia w sekundach procesów powinny być zadawane z linii poleceń tak jak poniżej.

```
czasy C0 C1 C2 ... Cn
```

Czas C0 jest czasem zakończenia całego programu a Ci opóźnieniem i-tego procesu potomnego. Uruchom zadanie na komputerze PC104. Niech każdy z procesów co 1 krok zmienia stan diody Di na płycie interfejsu komputera.

8. Tworzenie procesów w systemach RTS - procesy zdalne, priorytety procesów, ograniczenia na użycie zasobów

8.1 Wykonanie polecenia systemowego i uruchamianie procesów na zdalnych węzłach

Jeszcze inną metodą utworzenia nowego procesu jest użycie funkcji `system()`. Prototyp tej funkcji podany jest poniżej.

Funkcja 8-1 `system` – wykonanie polecenia systemowego	
`int system(char * polecenie)`	
`polecenie`	Łańcuch zawierający polecenie do wykonania

Funkcja zwraca:

`-1`	Gdy nie udało się uruchomić interpretera poleceń sh
`>0`	Kod powrotu przekazany przez zakończony interpreter poleceń sh

Funkcja system powoduje uruchomienie interpretera poleceń `sh` i przekazanie mu do wykonania łańcucha `polecenie`. Tym sposobem uruchomione mogą być programy, polecenia systemu lub skrypty. W szczegolności za pomocą polecenia system uruchamiać można procesy na innym węźle. Do tego celu wykorzystać można polecenie `on`:

```
on [–n | -f] nazwa_węzła polecenie
```

Komenda on powoduje, że na węźle o danej wykonane zostanie wyspecyfikowane jako parametr polecenie. Tak więc, możliwe jest zdalne wykonywanie poleceń.

```
#include <stdlib.h>
int main( void )   {
    int res;
    res = system("on –n pc104-komp1 my_prog" );
    return 0;
}
```

Przykład 8-1 Uruchomienie procesu `my_prog` na zdalnym węźle pc104-komp1

```
#include <stdio.h>
#include <sys/utsname.h>
int main(int argc, char * argv[]) {
  int i, res;
  char c;
  char node[40];
  struct utsname info;
  printf("My_prog - liczba argumentow: %d\n",argc);
  res = uname(&info);
  strcpy(node,info.nodename);
  printf("wezel: %s\n",node);
  for(i=0; i<10;i++) {
     printf("%s - my_prog, krok %d\n",node,i);
     sleep(1);
  }
  return 0;
}
```

Przykład 8-2 Kod procesu `my_prog` - uzyskiwanie informacji o węźle

8.2 Priorytety i strategie szeregowania

W systemie QNX6 priorytet procesu jest liczba z zakresu 0-255. Wyższym liczbom odpowiada wyższy priorytet. Zwykły użytkownik może nadawać swym procesom priorytety z zakresu od 1 do 63. Priorytet procesu dziedziczony jest z procesu macierzystego. Priorytet uzyskać można za pomocą funkcji `getprio()` lub `sched_getparam()` a ustawiać za pomocą funkcji `setprio()` lub `sched_setparam()`. System implementuje następujące strategie szeregowania: FIFO, karuzelową i sporadyczną.

Numer	Symbol	Opis
1	`SCHED_FIFO`	Szeregowanie FIFO
2	`SCHED_RR`	Szeregowanie karuzelowe
4	`SCHED_SPORADIC`	Szeregowanie sporadyczne

Tabela 8-1 Strategie szeregowania systemie QNX6 Neutrino

Strategię szeregowania testuje się za pomocą polecenia `sched_getscheduler()` a ustawia za pomocą polecenia `sched_setscheduler()`.

Atrybut procesu	Testowanie	Ustawianie
Priorytet procesu	`getprio()`, `sched_getparam()`	`setprio()`, `sched_setparam()`
Strategia szeregowania	`sched_getparam()` `sched_getscheduler()`	`sched_setparam()` `sched_setscheduler()`

Tabela 8-2 Funkcje operujące na priorytetach

8.3 Ustanawianie ograniczeń na zużycie zasobów

System QNX6 Neutrino posiada mechanizmy pozwalające na ustanowienie limitu na takie zasoby jak:

- czas procesora,
- pamięć operacyjna,
- wielkość pamięci pobranej ze sterty,
- wielkość segmentu stosu,
- maksymalna liczba deskryptorów plików,
- maksymalna wielkość pliku utworzonego przez proces
- maksymalna liczba procesów potomnych tworzonych przez proces.

Dla każdego z tych zasobów istnieje:

- ograniczenie miękkie (*ang. soft limit*)
- ograniczenie twarde (*ang. hard limit*).

Ograniczenie miękkie może być zmieniane przez proces bieżący ale nie może przekroczyć twardego. Ograniczenie twarde może być zmieniane przez proces o statusie administratora. Ustanawianie i testowanie limitów zasobów może być ustanawiane z poziomu shella przy użyciu polecenie ulimit. Do testowania i ustawiania poziomu zużycia zasobu przez uzytkownika służy polecenie **`ulimit`**

```
ulimit [-acdfHlmnpsStuv] [limit]
```

Opcje:

`-S`	Zmień lub pokaż miękkie ograniczenie
`-H`	Zmień lub pokaż twarde ograniczenie
`-a`	Pokaż wszystkie ograniczenia
`-c`	Maksymalna wielkość pamięci operacyjnej
`-d`	Maksymalna wielkość segmentu danych
`-f`	Maksymalna wielkość tworzonego pliku
`-l`	Maksymalna wielkość pamięci operacyjnej która może być zablokowana
`-n`	Maksymalna liczba deskryptorów plików
`-s`	Maksymalna wielkość stosu
`-t`	Maksymalna wielkość jednostek czasu procesora
`-p`	Maksymalna liczba tworzonych przez użytkownika procesów
`-v`	Maksymalna wielkość pamięci wirtualnej dla procesu

Tab. 8-1 Parametry polecenia ulimit

Limity zasobów przyznane użytkownikowi można wylistować pisząc polecenie:

```
$ulimit -a
```

```
ulimit -a
time(cpu-seconds)    unlimited
file(blocks)         unlimited
coredump(blocks)     unlimited
data(kbytes)         unlimited
stack(kbytes)        unlimited
lockedmem(kbytes)    unlimited
memory(kbytes)       unlimited
nofiles(descriptors) 1000
processes            10
vmemory(kbytes)      unlimited
```

Przykład 8-3 Listowanie limitu zasobów

Ustawiane limity zmieniane są w jednostkach 1024 bajtowych z wyjątkiem:

- –t – sekundy,
- -p – bloki 512 bajtów
- -n – sztuki
- -u – sztuki,

Limity zasobów można ustanowić dla:

- Systemu - w pliku **`/etc/rc.d/rc.local`**
- Użytkownika – w pliku **`/home/nazwa_użytkownika/.profile`**
- Procesów potomnych

Do testowania limitów zasobów służy funkcja `getrlimit()`,do ustawiania limitów zasobów funkcja `setrlimit()`.

Oznaczenie	Opis	Akcja przy przekroczeniu
`RLIMIT_CORE`	Pamięć operacyjna	Zakończenie procesu.
`RLIMIT_CPU`	Czas procesora	Wysłanie sygnału SIGXCPU.
`RLIMIT_DATA`	Wielkość pamięci pobranej ze sterty.	Funkcja kończy się błędem.
`RLIMIT_FSIZE`	Maksymalna wielkość pliku	Wysłanie sygnału SIGXFSZ do
`RLIMIT_NOFILE`	Maksymalna liczba deskryptorów plików	Funkcja kończy się błędem.
`RLIMIT_STACK`	Maksymalny rozmiar stosu	Wysłanie sygnału SIGSEGV.
`RLIMIT_NPROC`	Maksymalna liczba procesów potomnych	Procesy przekraczające limit nie będą utworzone

Tabela 8-3 Podlegające ograniczeniu zasoby przydzielane procesowi i jego procesom potomnym

8.4 Zadania

Zadanie 8. 4. 1 Ustawianie i testowanie priorytetów procesu

Napisz program `prior` ilustrujący działanie priorytetów. Program ten ma być uruchamiany z parametrami określającymi priorytety procesów potomnych. Na przykład uruchomienie programu `prior` z parametrami `7 8 9 10` znaczy że proces macierzysty tworzy cztery procesy potomne o priorytetach 7, 8, 9 i 10. Priorytet ma być ustawiony funkcją `setprio()` a testowany funkcją `getprio()`. Procesy potomne mają wykonywać zadaną liczbę kroków pętli w której wykonywane są intensywne obliczenia bez żadnych operacji blokujących. Należy dokonać pomiaru czasu wykonania poszczególnych procesów. Czas wykonania można zmierzyć wykorzystując funkcję `time(NULL)` która zwraca liczbę sekund która upłynęła od 01/01/1970. Poprawność ustawienia priorytetów można sprawdzić wykorzystując polecenie `ps –l` lub `pidin`.

Zadanie 8. 4. 2 Ustawianie i testowanie priorytetów procesu – komputer PC104

Napisz program `prior2` ilustrujący działanie priorytetów analogiczny do poprzedniego ale wykonywany na komputerze PC104. Procesy powinny cyklicznie stany wyjść cyfrowych. Proces Pi zmienia stan diody o numerze i.

Zadanie 8. 4. 3 Ustawianie i testowanie priorytetów i strategii szeregowania procesu

Uzupełnij program z poprzedniego ćwiczenia o ustawianie strategii szeregowania.

Zadanie 8. 4. 4 Ustawianie i testowanie limitów na zużycie zasobów systemowych

Utwórz nowego uzytkownika test i ustaw mu limity zasobów na przykład liczbę procesów. Ograniczenie wpisz do pliku `/home/test/.profile`. Skonstruuj programy testowe aby ograniczenia te zostały przekroczone i zaobserwuj reakcje systemu.

9. Zastosowanie plików do zapisu wyników i komunikacji między komputerami

9.1 Wstęp

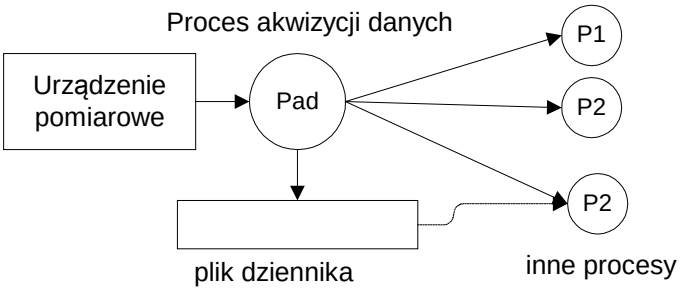
Pliki pełnią w systemach sterowania ważną rolę. Służą do:

- Dostępu do urządzeń (porty COM, I2C, SPI)
- Zapamiętywania danych (pamięć nieulotna)
- Komunikacji między procesami i komputerami

Standard POSIX definiuje dwa rodzaje dostępu do plików:

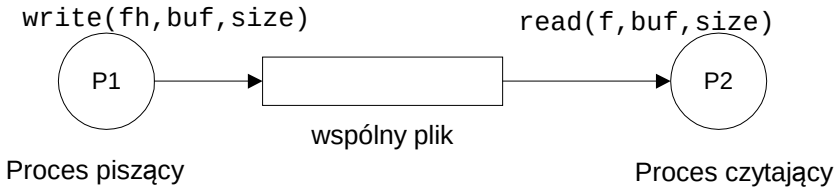
- Interfejs niskiego poziomu - plik identyfikowany jest przez uchwyt (liczba całkowita).
- Strumienie zdefiniowane w standardowej bibliotece wejścia/wyjścia - plik identyfikowany jest przez wskaźnik. Są to funkcje zaczynające się na f np. fread, fopen...)

Podstawową funkcją pliku jest trwałe zapamiętywanie informacji. Ma to zastosowanie w systemach sterowania i akwizycji danych. Zwykle procesy pozyskujące dane zapisują je w pliku dziennika (ang. *log file*).



Rys. 9-1 Schemat procesu akwizycji danych

Inna funkcją plików może być komunikacja międzyprocesowa lub nawet między komputerami gdy w dyspozycji jest sieciowy system plików (np. NFS). Komunikacja poprzez wspólne pliki jest prostym mechanizmem komunikacji międzyprocesowej działającym tak w obrębie jednego węzła jak i w sieci Qnet. Mechanizm ten może być zastosowany w systemach akwizycji danych.



Rysunek 9-1 Procesy P1 i P2 komunikują się za pomocą wspólnego pliku

W przypadku gdy procesy komunikuja się przez wspólne pliki należy rozwiązać nastepujące problemy:

- Problem współbieżnego dostępu wielu procesów do jednego pliku – należy zastosować blokady
- Problem znalezienia ostatniego zapisu
- Problem przycinania rosnącego pliku

9.2 Funkcje niskiego poziomu

Niskopoziomowe funkcje dostępu do plików zapewniają dostęp do:

- plików regularnych
- katalogów
- łącz nazwanych
- łącz nie nazwanych
- gniazdek
- urządzeń (porty szeregowe, równoległe).

Ważniejsze niskopoziomowe funkcje dostępu do plików podane są poniżej.

Nr	Funkcja	Opis
1	`open`	Otwarcie lub utworzenie pliku
2	`creat`	Tworzy pusty plik
3	`read`	Odczyt z pliku
4	`write`	Zapis do pliku
5	`lseek`	Pozycjonowanie bieżącej pozycji pliku
6	`fcntl`	Ustawianie i testowanie różnorodnych atrybutów pliku
7	`fstat`	Testowanie statusu pliku
8	`close`	Zamknięcie pliku
9	`unlink, remove`	Usuwa plik

Tab. 9-1 Ważniejsze niskopoziomowe funkcje dostępu do plików

Poniżej podano przykład odczytu pliku.

```
main(void) {
  int fd;
  char buf[80];
  fd = open(„/home/jan/mój_plik.txt”,O_RDONLY);
  do {
     rd = read(fd,buf,80);
     printf("Odczytano %d bajtów\n",rd);
  } while(rd > 0);
  close(fd);
}
```

Przykład 9-1 Przykład odczytu pliku

9.3 Standardowa biblioteka wejścia wyjścia

Biblioteka rozszerza możliwości funkcji niskopoziomowych:

- Zapewnia wiele rozbudowanych funkcji ułatwiających formatowanie wyjścia i skanowania wejścia
- Obsługuje buforowanie
- Funkcje zadeklarowane są w pliku nagłówkowym stdio.h
- Odpowiednikiem uchwytu jest strumień (ang. stream) widziany w programie jako FILE*

Funkcje wysokiego poziomu, zawarte w standardowej bibliotece wejścia wyjścia pokazuje poniższa tabela.

Funkcja	Opis
`fopen, fclose`	Otwarcie lub utworzenie pliku, zamknięcie pliku
`fread`	Odczyt z pliku
`fwrite`	Zapis do pliku
`fseek`	Pozycjonowanie bieżącej pozycji pliku
`fgetc,getc, getchar`	Odczyt znaku
`fputc, putc, putchar`	Zapis znaku
`frintf, fprintf, sprintf`	Formatowane wyjście
`scanf, fscanf, sscanf`	Skanowanie wejścia
`fflush`	Zapis danych na nośnik

Tab. 9-2 Ważniejsze funkcje wysokiego poziomu dostępu do plików

Poniżej podano przykład odczytu pliku przy wykorzystaniu funkcji opartych o strumienie.

```
#include <stdio.h>
#define SIZE 80

int main() {
   int ile;
   FILE *f;
   char buf[SIZE];
   f = fopen("fread.c","r");
   if(f == NULL) { perror("fopen"); exit(0);}
   do {
    ile = fread(&buf,sizeof(buf),1,f);
    printf("%s\n",buf);
   } while(ile == 1);
   fclose(f);
   return 0;
}
```

Przykład 9-2 Przykład odczytu pliku funkcją `fread`

Za pomocą funkcji `fileno` można uzyskać uchwyt pliku znając jego identyfikator strumieniowy. `int fileno(FILE * stream)`. Podobnie znając uchwyt można uzyskać strumień korzystając z funkcji fdopen. `FILE * stream fdopen(int fdes, char * mode)`.

9.4 Blokowanie plików

W sytuacji gdy więcej procesów korzysta współbieżnie z pliku a przynajmniej jeden z nich pisze może dojść do błędnego odczytu jego zawartości. Podobnie dwa (lub więcej) procesy nie powinny jednocześnie pisać do pliku. Aby zapewnić wzajemne wykluczanie w dostępie do pliku stosuje się blokady plików.

Funkcja 9-1 `lockf – zablokowanie dostepu do pliku`	
`int lockf(int fd, int funkcja, int zakres)`	
`fd`	Uchwyt pliku
`funkcja`	Specyfikacja operacji: F_LOCK, F_ULOCK, F_TEST, F_TLOCK
`ile`	Zakres blokowanie

Funkcja zwraca:

`-1`	Gdy błąd
`>0`	Sukces

F_LOCK	Zablokuj dostęp do pliku na długości `zakres` od pozycji bieżącej
F_ULOCK	Zwolnij dostęp do pliku.
F_TEST	Testuj czy fragment pliku jest zablokowany przez inny proces
F_TLOCK	Testuj czy fragment pliku jest zablokowany przez inny proces. Gdy nie to zajmij plik

```
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(int argc, char * argv[]) {
 int fd,ile,res, num = 444;
 char buf[8];
 fd = open("plik.txt",O_WRONLY | O_CREAT,  S_IRUSR | S_IWUSR );
 if(fd < 0) {
      perror("open");
      exit(0);
 }
```

```
  printf("Zalozenie blokady \n");
  res = lockf(fd,F_LOCK,0);
  ile = write(fd,&num,0);
  res = lockf(fd,F_ULOCK,0);
  close(fd);
}
```

Przykład 9-3 Blokowanie pliku do zapisu

9.5 Zadania

Zadanie 9. 4. 1 Akwizycja danych i zapamiętywanie danych w pliku

Napisz program `filew` który cyklicznie pobiera stan wejść analogowych A0-A3 oraz stan wejść cyfrowych DI w postaci szesnastkowej. Pobrane dane powinny być zapisane w pliku w katalogu `/dev/shmem`. W każdej linii pliku powinien się znaleźć:

Czas bieżący (w formacie `hh/mm/ss`) A0 A1 A2 A3 DI

```
17:01:41  - 1865 1453    7    7   54
```

Program powinien działać w pętli z 0.5 sekundowym opóźnieniem (funkcja usleep) i w każdym kroku zwiększać licznik `cnt` modulo 256. Stan licznika powinien być wyświetlany w postaci binarnej na diodach połączonych do wyjść cyfrowych DOUT2.

Aby to zrealizować należy:

1. Zainicjować przetwornik AD aby odczytywał kanały od 0 do 3 a jako zakres pomiarowy przyjąć 0-5 V.
2. Odczyt wejść analogowych ma być realizowany przez procedurę `int aread(unsigned int *chan)`.
3. Utworzyć plik `F = fopen(”/dev/shmem/wynik.txt”,”w”);`
4. W pętli wykonywać odczyt czasu , odczyt kanałów A0 do A3 odczyt stanu wejść cyfrowych.
5. Zapisać czas, kanały do pliku F.
6. Zwiększyć stan licznika pętli `cnt = (cnt +1) % 256` i zapisać na wyjścia cyfrowe DOUT2
7. Gdy licznik nie jest większy od 20 przejść do kroku 4.
8. Zamknąć plik F

Czas pobieramy przy pomocy sekwencji:

```
tim = time(NULL);
strftime(czas,sizeof(czas),"%T",localtime( &tim));
```

Program uruchamiamy logując się jako root na komputer PC104:

```
$on –n pc104-komp1 login
#cd  /dev/shmem
#cp /net/komp_lokalny/home/kat_lokalny/filer .
#./filew
```

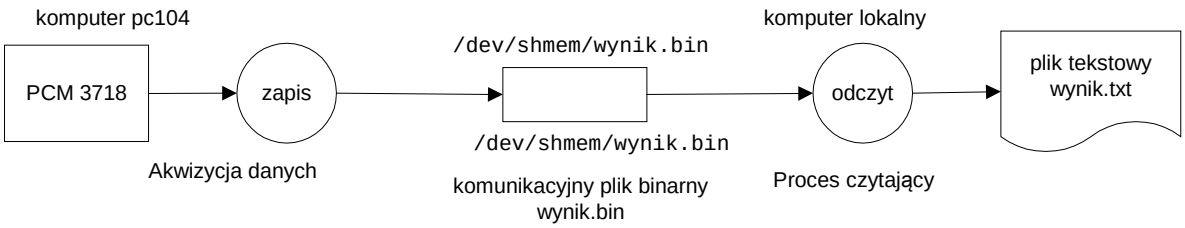
Zadanie 9. 4. 2 Komunikacja sieciowa przez wspólny plik

Gdy dysponujemy sieciowym systemem plików, a tak jest w QNX6 neutrino, pliki mogą być wykorzystane do komunikacji sieciowej między procesami a więc także w rozproszonym systemie akwizycji danych. Należy jednak rozwiązać:

- Problem współbieżnego dostępu wielu procesów do jednego pliku – należy zastosować blokady
- Problem znalezienia ostatniego zapisu
- Problem przycinania rosnącego pliku

W tym zadaniu aplikacja będzie się składała z 2 programów:

- Programu `zapis` który wykonywany jest na komputerze wbudowanym PC104.Pobiera on dane z karty interfejsowej i wpisuje do lokalnego pliku `/dev/shmem/wynik.bin`
- Programu `odczyt` który wykonywany jest na komputerze PC i pobiera dane z utworzonego przez poprzedni program pliku `/dev/shmem/wynik.bin`.



Rys. 9-2 Sieciowy system akwizycji danych – komunikacja poprzez plik wynik.bin

```
typedef struct {
  time_t czas;  // Czas pomiaru
  int An[4];    // Wejścia analogowe
  int Do;       // Wejścia binarne
} buf_t
```

Program `zapis` powinien wykonać następujące kroki:

1. Utworzyć za pomocą funkcji open plik binarny `/dev/shmem/wynik.bin`
   `fh = open("/dev/shmem/wynik.bin",O_WRONLY | O_CREAT,0666);`
2. Cyklicznie odczytywać czas, stany wejść analogowych `an[0] do an[3]` i wejść binarnych. Zapisać je w strukturze `buf_t buf.`
3. Strukturę `buf` zapisać do pliku `fh`:
   `res = write(fh,&buf,sizeof(buf));`

Program `odczyt` powinien wykonać następujące kroki:

1. Utworzyć za pomocą funkcji open plik binarny `/net/pc104-komp1/dev/shmem/wynik.bin`
   `fh = open("/net/pc104-komp1/dev/shmem/wynik.bin",O_WRONLY | O_CREAT,0666);`
2. Utworzyć za pomocą funkcji `fopen` plik tekstowy wynik.txt
   `pl = fopen("wynik.txt","w");`
3. Cyklicznie czytać dane z pliku `fh` , wyświetlać je na konsoli i zapisywać je do pliku `pl`.
   Odczyt wykonujemy funkcją: `res = read(fh,&buf,sizeof(buf))`. Do zapisu można użyć funkcji `fprintf`.

Format danych w pliku tekstowym powinien być jak poniżej:

```
12/11/2014 9:45:23 1334 222  443   2234  10010011
```

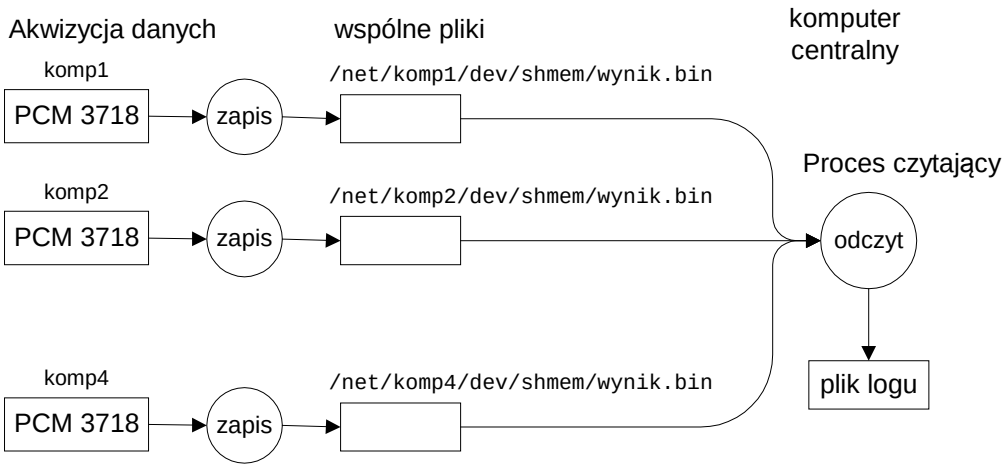
Uwaga!

1. Program czytający powinien czytać z pliku komunikacyjnego najbardziej aktualne dane. W związku z tym można ustawić bieżącą pozycję pliku `fh` za pomocą funkcji `lseek(fh,-sizeof(buf),SEEK_END);`
2. Należy **zawsze** sprawdzać czy otwarcie, zapis i odczyt pliku się udał.
3. W pliku tekstowym czas z postaci binarnej do ascii ozyskujemy za pomocą funkcji `strftime`
4. Jako że dwa procesy maja dostęp do pliku wynik.bin należy zastosować blokady, należy użyć funkcji `lockf`.

Zadanie 9. 4. 3 Rozproszony system akwizycji danych – komunikacja poprzez pliki

Wykorzystaj programy z poprzedniego ćwiczenia do implementacji rozproszonego systemu akwizycji danych. Na komputerach pomiarowych PC104 uruchomione mają być procesy `zapis` czytające dane z interfejsów karty pcm3718 i piszące je do lokalnych plików `/dev/shmem/wynik.bin`. Proces `odczyt` uruchomiony na stacji roboczej ma pobierać dane z plików lokalnych i zapisywać je w logu. Program odczyt uruchamiamy z parametrami będącymi nazwami komputerów pomiarowych.

```
odczyt komp1 komp2 ... kompN
```



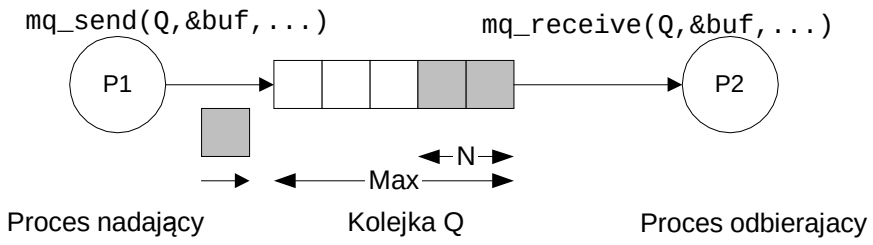
Rys. 9-3 Rozproszony system akwizycji danych – komunikacja poprzez pliki

Uwaga!
Użytkownik nie będący `root` nie jest w stanie uruchomić procesu korzystającego z systemu we/wy. Może to zrobić tylko `root`. W związku z tym należy poprosić prowadzącego aby przełączył użytkownika na `root`.

10. Zastosowanie kolejek komunikatów POSIX do komunikacji między procesami w systemach akwizycji danych

10.1 Wstęp

Kolejki komunikatów POSIX są wygodnym mechanizmem komunikacji międzyprocesowej działającym tak w obrębie jednego węzła jak i w sieci Qnet.



Rysunek 10-1 Procesy P1 i P2 komunikują się za pomocą kolejki Q

Aby możliwe było użycie kolejki komunikatów musi być uruchomiony proces `Mqueue`. Kolejkę komunikatów tworzy się za pomocą polecenia `mq_open()`.

Funkcja 10-1 `mq_open` – tworzenie kolejki komunikatów	
`mqd_t mq_open(char *name,int oflag, int mode, mq_attr *attr)`	
`name`	Nazwa kolejki komunikatów
`oflag`	Tryb tworzenia kolejki
`mode`	Prawa dostępu (`r` - odczyt, `w` - zapis) dla właściciela pliku, grupy i innych, analogiczne jak w przypadku plików regularnych.
`attr`	Atrybuty kolejki komunikatów lub NULL gdy domyślne

Gdy nazwa kolejki zaczyna się od znaku / wtedy kolejka tworzona jest w katalogu `/dev/mqueue` co można sprawdzić poleceniem: `ls –l /dev/mqueue`. Gdy nazwa kolejki zaczyna się od znaku innego niż / wtedy kolejka tworzona jest w katalogu domowym użytkownika. Aby użyć kolejki komunikatów należy zadeklarować zmienną typu `mqd_t` (kolejka komunikatów) i zmienną typu `mq_attr` (atrybuty kolejki komunikatów pokazane w Tabela 10-2). Następnie należy otworzyć kolejkę komunikatów używając funkcji `mq_open`. Nazwa kolejki może być podana dla węzła lokalnego lub zdalnego. Wtedy należy użyć pełnej nazwy zdalnego węzła `/net/nazwa_wezla/dev/mqueue/nazwa_kolejki`. Z kolejkami komunikatów związane są następujące funkcje:

`mq_open()`	Tworzenie lub otwieranie kolejki komunikatów
`mq_send()`	Zapis do kolejki komunikatów
`mq_receive()`	Odczyt z kolejki komunikatów
`mq_getattr()`	Pobranie atrybutów i statusu kolejki.
`mq_setattr()`	Ustawienie atrybutów kolejki
`mq_notify()`	Ustalenie trybu zawiadamiania o zdarzeniach w kolejce.
`mq_close()`	Zamykanie kolejki komunikatów.
`mq_unlink()`	Kasowanie kolejki komunikatów

Tabela 10-1 Funkcje obsługi kolejek komunikatów

`long mq_maxmsg`	Maksymalna liczba komunikatów w kolejce.
`long mq_msgsize`	Maksymalna wielkość pojedynczego komunikatu.
`long mq_curmsg`	Aktualna liczba komunikatów w kolejce.
`long mq_flags`	Flagi
`long mq_sendwait`	Liczba procesów zablokowanych na operacji zapisu.
`long mq_recvwait`	Liczba procesów zablokowanych na operacji odczytu.

Tabela 10-2 Atrybuty `mq_attr` kolejki komunikatów

Aby użyć kolejek komunikatów należy do programu dołączyć plik nagłówkowy `<mqueue.h>`. Poniżej pokazany został przykład użycia kolejki komunikatów. Program `odbior` tworzy kolejkę komunikatów o nazwie `Kolejka` i czeka na komunikaty. Po uruchomieniu programu można sprawdzić czy kolejka została utworzona pisząc na konsoli polecenie:

```
$ odbior &
$ ls –l
nrw-rw---- 1 juka   juka 0 Apr 27 15:45 Kolejka
```

Kody źródłowy programów odbierania i wysyłania pokazano poniżej.

```
// Kompilacja: cc wyslij.c -o wyslij
#include <stdio.h>
#include <mqueue.h>
main(int argc, char *argv[]) {
     int kom, res;
     mqd_t  mq;
     struct mq_attr attr;
     /* Utworzenie kolejki komunikatow -----------------*/
     attr.mq_msgsize = sizeof(kom);
     attr.mq_maxmsg = 1;
     attr.mq_flags = 0;
     mq=mq_open("Kolejka", O_RDWR | O_CREAT , 0660, &attr );
     kom = atoi(argv[1]);
     res = mq_send(mq,&kom,sizeof(kom),10);
     printf("Wyslano komunikat: %d\n",kom);
     mq_close(mq);
}
```

Przykład 10-1 Proces wysyłający komunikaty do kolejki

```
#include <stdio.h>
#include <mqueue.h>
main(int argc, char *argv[]) {
     int i, kom, res;
     unsigned int prior;
     mqd_t  mq;
     struct mq_attr attr;
     /* Utworzenie kolejki komunikatow -----------------*/
     attr.mq_msgsize = sizeof(kom);
     attr.mq_maxmsg = 1;
     attr.mq_flags = 0;
     mq=mq_open("Kolejka", O_RDWR | O_CREAT , 0660, &attr );
     res = mq_receive(mq,&kom,sizeof(kom),&prior);
     printf("Otrzymano komunikat: %d\n",kom);
     mq_close(mq);
}
```

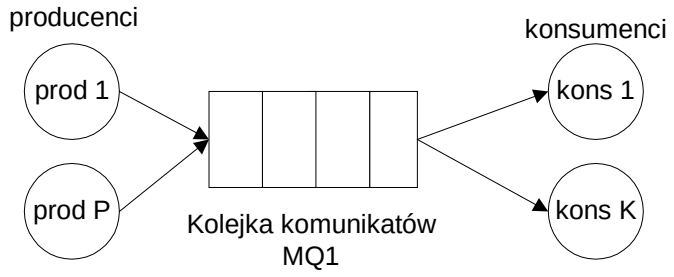
Przykład 10-2 Proces odbierający komunikaty z kolejki

10.2Zadania

Zadanie 10. 4. 1 Rozwiązanie problemu producenta i konsumenta za pomocą kolejek komunikatów

Należy rozwiązać problem producenta i konsumenta używając mechanizmu kolejek komunikatów. Użyte będą procesy:

`init` – proces tworzy kolejkę komuniktów
`prod` - producent komunikatów, może być wiele kopii tego procesu
`kons` - konsument komunikatów, może być wiele kopii tego procesu



Rysunek 10-2 Problem producenta i konsumenta

Postać przesyłanego komunikatu powinna być dana strukturą jak poniżej:

```
typedef struct  {
        int  type;       /* typ procesu:  1 PROD, 2 KONS  */
        int  pnr ;           /* numer procesu     */
        char text[SIZE];     /* tekst  komunikatu */
    } ms_type;
```

Definicja struktury powinna być zawarta w pliku nagłówkowym `common.h`

Proces init powinien skasować poprzednią kolejkę o ile istnieje i utworzyć kolejkę komunikatów. Należy uzyc funkcji:

```
mq=mq_open("Kolejka", O_RDWR | O_CREAT , 0660, &attr );
```

Procesy producenta `prod` powinien być napisany według poniższego wzoru:

- Pobrać z linii poleceń swój numer nr i liczbę kroków
- Otworzyć kolejkę komunikatów.
- Wykonywać w pętli następującą sekwencję instrukcji:

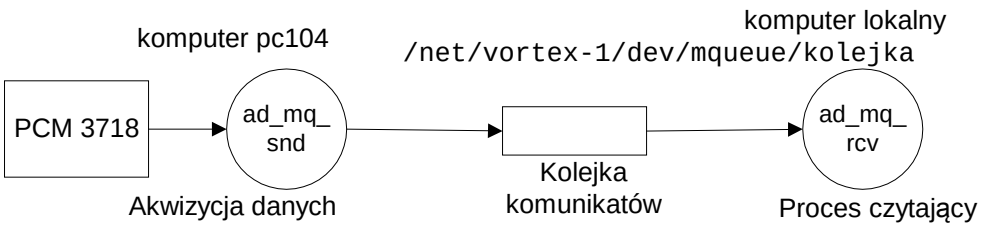
```
...
for(i=0;i<liczba_krokow;i++) {
    msg.pnr = nr;
    msg.type = PROD;
    sprintf(msg.text,"Producent %d krok %d",nr,i);
    // Przeslanie komunikatu do kolejki
    res = mq_send(mq,&msg,sizeof(msg),priority);
    ......
    sleep(1);
}
```

Proces konsumenta `kons` należy napisać podobnie jak proces producenta. Zamiast funkcji `mq_send` należy użyć funkcji `mq_receive`.

Uruchomić program w wersji sieciowej.

Zadanie 10. 4. 2 System akwizycji danych– komunikacja poprzez kolejkę komunikatów

Napisz programy `ad_mq_snd` i `ad_mq_rcv`. Program `ad_mq_snd` czyta dane pomiarowe i umieszcza je w kolejce komunikatów komputera PC104. Dane te są następnie pobierane przez program `ad_mq_rcv`_działający na najpierw na tym samym komputerez a potem na komputerze lokalnym.



Rys. 10-1 Akwizycja danych z karty PCM3718 - komunikacja poprzez kolejkę komunikatów

```
struct {
       int kanal[SIZE]; // Kanaly AD
       int digital;     // Kanal we cyfr.
       time_t czas;     // Czas wykonania pomiaru
} msg;
```

Program `ad_mq_snd` wysyłający dane

Program `ad_mq_snd` cyklicznie pobiera stan wejść analogowych A0-A3 oraz stan wejść cyfrowych DI a następnie umieszcza je w strukturze `msg` i wysyła do kolejki komunikatów. W strukturze powinien się znaleźć czas wykonania pomiaru. Pobrane dane powinny być zapisane w kolejce komunikatów w katalogu `/dev/mqueue`.

Aby to zrealizować należy:

1. Zainicjować przetwornik AD aby odczytywał kanały od 0 do 3 a jako zakres pomiarowy przyjąć 0-5 V.
2. Utworzyć kolejkę komunikatów
   `mq = mq_open(”/kolejka”, O_RDWR | O_CREAT , 0660, &attr )`
3. Odczyt wejść analogowych ma być realizowany przez procedurę `int aread(unsigned int *chan)`.
4. W pętli wykonywać odczyt czasu , odczyt kanałów A0 do A3 przetwornika AD, odczyt stanu wejść cyfrowych.
5. Zapisać czas, kanały do kolejki `mq`.
6. Zwiększyć stan licznika pętli cnt = (cnt +1) % 256 i zapisać na wyjścia cyfrowe DOUT2
7. Gdy licznik nie jest większy od 100 przejść do kroku 4.
8. Zamknąć kolejkę mq.

Czas pobieramy przy pomocy sekwencji: `tim = time(NULL);`

Program `ad_mq_rcv` odbierający dane

Program `ad_mq_rcv` odbierający dane powinien działać na komputerze lokalnym i odbierać dane z komputera pomiarowego PC104. Aby to zrealizować należy:

1. Otworzyć kolejkę komunikatów
   `mq = mq_open(”/net/vortex-1/dev/mqueue/kolejka”,O_RDWR | O_CREAT,0660,&attr);`
2. W pętli wykonywać odczyt struktur `msg` z kolejki `mq`.
3. Wypisać na konsoli wartości kanałów A0 do A3, stan wejść cyfrowych i czas wykonania pomiaru

Czas wykonania pomiaru można uzyskać za pomoca funkcji
`strftime(czas,sizeof(czas),"%T",localtime( &tim));`

Rozszerzenia:

A) Uogólnić system na wiele komputerów pomiarowych
B) Zrealizować zdalne uruchamianie programów `ad_mq_snd` wysyłających dane pomiarowe.

Zadanie 10. 4. 3 System akwizycji danych– komunikacja poprzez kolejkę komunikatów, dwa źródła danych

Uzupełnij program z poprzedniego przykładu o proces `czuj_odl_mq` pobierający dane z sonaru. Proces pobiera dane o odległości do przedmiotu z sonaru (patrz 6.4). Dalej otrzymane dane zapisywane są do kolejki komunikatów w postaci struktury `msg2_t`.

```
#define SIZE 8
#define SONAR          2
```

```
#define PRZETWORNIK    1

typedef struct {
      int typ;              // Typ komunikatu
      float odleglosc;      // odleglosc przedmiotu
 } msg2_t;
```

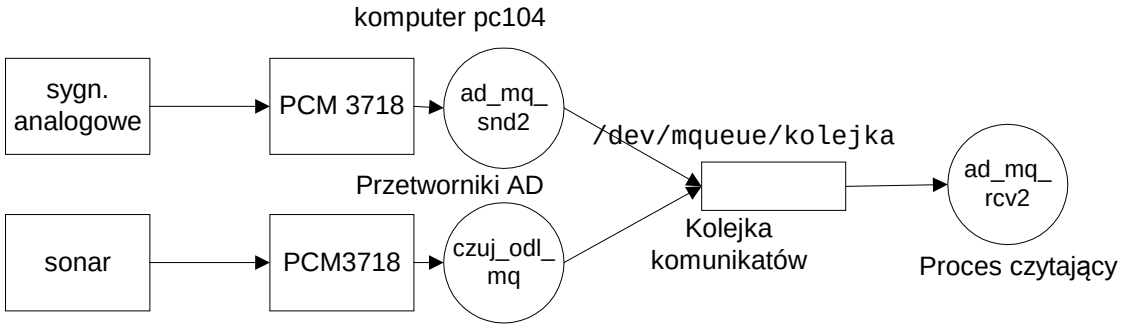
Dane z przetworników AD i wejść cyfrowych karty PCM3718 wpisywane są przez proces `ad_mq_snd2` do kolejki komunikatów w postaci struktury:

```
typedef struct {
        int typ;              // Typ komunikatu
        int kanal[SIZE];      // Kanaly analog
        int digital;          // Kan wejsc cyfr
  } msg1_t;
```

Proces `ad_mq_rcv2` czyta komunikaty z kolejki posix w postaci unii:

```
union {
   int typ;
   msg1_t msg1;
   msg2_t msg2;
} msg;
```

Następnie w zależności od typu komunikatu interpretuzje je albo jako komunikat `msg1_t (gdy typ==1)` albo jako `msg2_t (gdy typ == 2).` Program `ad_mq_snd2` czyta dane pomiarowe z przetworników AD i umieszcza je w kolejce komunikatów. Dane te są następnie pobierane przez program `ad_mq_rcv2` działający na komputerze lokalnym lub komputerze PC104.



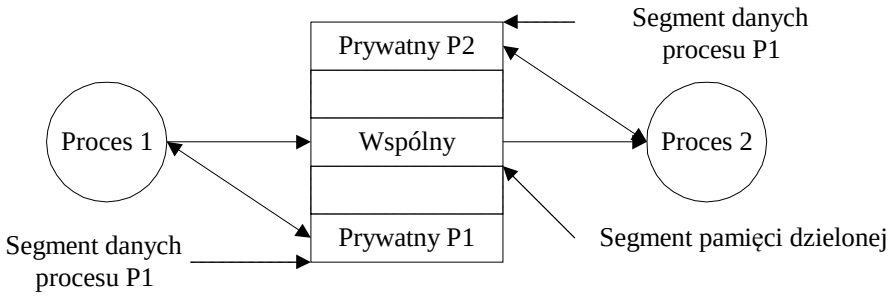
Rys. 10-2 Akwizycja danych z karty PCM3718 i sonaru - komunikacja poprzez kolejkę komunikatów

11. Wykorzystanie pamięci dzielonej i semaforów w synchronizacji dostępu do wspólnych danych.

11.1Pamięć dzielona

Jedną z możliwość komunikowania się procesów jest komunikacja przez pamięć dzieloną. Ta metoda komunikacji może być użyta gdy procesy wykonywane są na maszynie jednoprocesorowej lub wieloprocesorowej ze wspólną pamięcią. Nie ma natomiast zastosowania przy innych architekturach.
Aby procesy mogły mieć wspólny dostęp do tych samych danych należy:

1. Utworzyć oddzielny segment pamięci.
2. Udostępnić dostęp do segmentu zainteresowanym procesom.



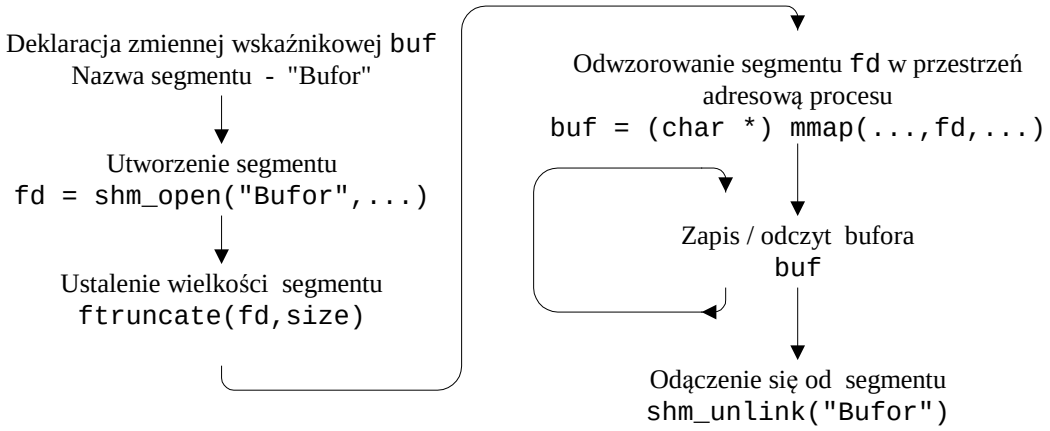
Rys. 11-1 Procesy P1 i P2 komunikuję się poprzez wspólny obszar pamięci

Standard Posix 1003.4 definiuje funkcje pozwalające na tworzenie i udostępnianie segmentów pamięci. Są to funkcje `shm_open()`, `ftruncate()`, `mmap()`, `munmap()` ,`mprotect()`, `shm_unlink`. Najważniejsze z funkcji podane są poniżej.

Opis	Funkcja
Tworzenie segmentu pamięci dzielonej	`shm_open()`
Ustalanie rozmiaru segmentu pamięci	`ftruncate()`
Odwzorowanie segmentu pamięci dzielonej w obszar procesu	`mmap()`
Odłączenie się od segmentu pamięci	`shm_unlink()`

Tabela 11-1 Funkcje operowania na pamięci dzielonej

Schemat utworzenia i udostępnienia segmentu pamięci dzielonej podano na poniższym rysunku.



Rys. 11-2 Schemat użycia segmentu pamięci dzielonej

Podany dalej Przykład 11-1 ilustruje sposób użycia segmentu pamięci dzielonej do wymiany danych pomiędzy procesami.

```
// Komunikacja przez pamiec wspolna
#include <stdlib.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <sys/wait.h>
#define LSIZE     80   //  Dlugosc linii

typedef struct {
    char buf[LSIZE];
    int cnt;
} bufor_t;

main(int argc,char *argv[]) {
 int i,stat,k, pid, size, fd, res, n;
 bufor_t  *wbuf ;

 // Utworzenie segmentu ---------------------------
 shm_unlink("bufor");

 fd=shm_open("bufor", O_RDWR|O_CREAT , 0774);
 if (fd < -0) { perror("open"); exit(-1); }
 printf("fd: %d\n",fd);
 size = ftruncate(fd, sizeof(bufor_t));
 if(size < 0) {perror("ftrunc"); exit(-1); }
 // Odwzorowanie segmentu fd w obszar pamieci procesow
 wbuf = ( bufor_t *)mmap(0,sizeof(bufor_t), PROT_READ|PROT_WRITE,  MAP_SHARED,
          fd, 0);
 if(wbuf == NULL) {perror("map");  exit(-1); }
 printf("Bufor utworzony\n");

 // Inicjacja obszaru -------------------------------
 wbuf-> cnt = 0;

 if(fork() == 0) {
   printf("Start - Producent\n");
   for(i=0;i<10;i++) {
        sprintf(wbuf->buf,"Komunikat %d",i);
        wbuf->cnt ++; sleep(1);
   }
   shm_unlink("bufor");
   exit(n);
 }

 if(fork() == 0) { // Konsument
   printf("Start - Konsument\n");
   for(i=0;i<10;i++) {
       printf("Konsument: %d odebrano %s\n",i,wbuf->buf);
       wbuf-> cnt --; sleep(1);
   }
   shm_unlink("bufor");
   exit(n);
 }
 // Czekamy na procesy potomne -----
 while (wait(&stat) != -1);
 return 0;
}
```

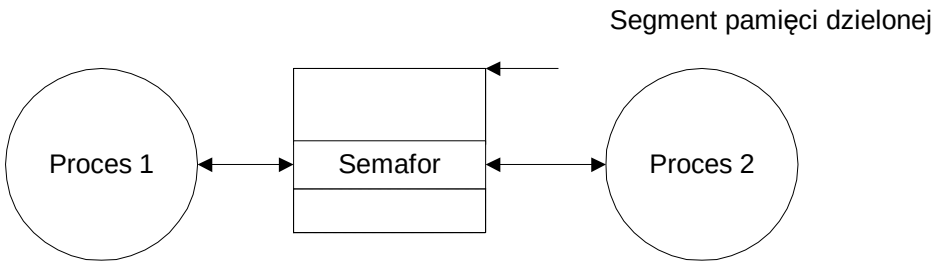
Przykład 11-1 Komunikacja przez pamięć wspólną – dwa procesy piszą do wspólnego bufora

11.2Semafory

Standard POSIX definiuje dwa typy semaforów:

- Semafory nienazwane
- Semafory nazwane

Dostęp do semafora nienazwanego następuje po adresie semafora. Może on być użyty do synchronizacji procesów o ile jest umieszczony w pamięci dzielonej. Stąd nazwa semafor nienazwany. Inny typ semafora to semafor nazwany. Dostęp do niego następuje po nazwie.



Przed użyciem semafora nienazwanego musi on być zadeklarowany jako obiekt typu `sem_t` a pamięć używana przez ten semafor musi zostać mu jawnie przydzielona. O ile semafor nienazwany ma być użyty w różnych procesach powinien być umieszczony w wcześniej zaalokowanej pamięci dzielonej. Funkcje operujące na semaforach podaje Tabela 11-2.

Działanie	Funkcja
Utworzenie semafora nazwanego	`sem_open()`
Inicjacja semafora nienazwanego	`sem_init()`
Czekanie na semaforze	`sem_wait()`
Sygnalizacja na semaforze	`sem_post()`
Sygnalizacja warunkowa	`sem trywait()`
Zamknięcie semafora nazwanego i nienazwanego	`sem_close()`
Zamknięcie semafora nazwanego	`sem_unlink()`
Skasowanie semafora nienazwanego	`sem_destroy()`

Tabela 11-2 Operacje na semaforach w standardzie POSIX 1003.1

Przed użyciem semafora nienazwanego trzeba:

1. Utworzyć segment pamięci za pomocą funkcji `shm_open()`.
2. Określić wymiar segmentu używając funkcji `ftruncate()`.
3. Odwzorować obszar pamięci wspólnej w przestrzeni danych procesu – `mmap()`.
4. Zainicjować semafor za pomocą funkcji `sem_init()`.

Aby użyć semafora nazwanego należy go otworzyć lub utworzyć (o ile nie istnieje) do czego wykorzystuje się funkcję `sem_open()`. Pobieranie i zwrot jednostek abstrakcyjnego zasobu następuje przez wykonanie funkcji semaforowych `sem_wait()` i `sem_post()`.

11.3Ćwiczenia

Zadanie 11. 4. 1 Zadanie 1 – Proces piszący i czytający są niezależne

W Przykład 11-1 proces czytający do segmentu pamięci wspólnej i proces piszący są procesami potomnymi jednego procesu. Dokonaj takiej modyfikacji przykładu Przykład 11-1 aby proces piszący i czytający były niezależne – uruchamiane oddzielnie z konsoli. Rozważ możliwość synchronizacji procesów: wstrzymanie zapisu gdy bufor pełny i wstrzymanie odczytu gdy bufor pusty.

Zadanie 11. 4. 2 Bufor cykliczny w pamięci dzielonej

Rozszerz Przykład 11-1 tak aby zaimplementować bufor cykliczny. Bufor cykliczny powinien posiadać BSIZE elementów długości LSIZE oraz wskaźniki `cnt`, `head`, `tail`.

```
typedef struct {
    int head; // Tutaj producent wstawia nowy element
    int tail; // Stąd pobiera element konsument
    int cnt;  // Liczba elementów w buforze
    char buf[BSIZE][LSIZE];
} bufor_t
```

Zadanie 11. 4. 3 Problem producenta i konsumenta

W poprzednim zadaniu implementowano bufor cykliczny położony we wspólnym segmencie pamięci. W rozwiązaniu tym brakowało mechanizmu wstrzymywania producenta (gdy bufor był pełny) i konsumenta (gdy bufor był pusty). Należy wprowadzić taki mechanizm za pomocą semaforów nienazwanych co prowadzi do prawidłowego rozwiązania problemu. Semafory powinny być położone w tym segmencie pamięci w którym umieszczony jest bufor.

```
typedef struct {
    int head; // Tutaj producent wstawia nowy element
    int tail; // Stąd pobiera element konsument
    int cnt;  // Liczba elementów w buforze
    sem_t mutex; // Semafor chroniący sekcję krytyczną
    sem_t empty; // Semafor wstrzymujący producenta
    sem_t full;  // Semafor wstrzymujący konsumenta
    char buf[BSIZE][LSIZE];
} bufor_t
```

Program testowy powinien za pomocą funkcji `fork` tworzyć `prod` procesów producenta i `kons` procesów konsumenta z których każdy ma wykonać zadana liczbę kroków tak jak poniżej podanym przykładzie:

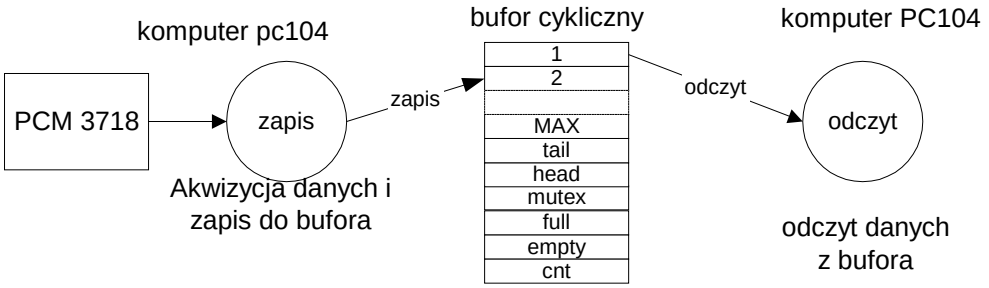
```
$prodkons prod kroki_prod kons kroki_kons
```

Zadanie 11. 4. 4 Problem producenta i konsumenta – procesy niezależne

W poprzednim zadaniu procesy konsumenta i producenta były procesami potomnymi jednego procesu. Dokonaj takiej rozszerzenia zadania aby proces producenta i proces konsumenta były niezależne – uruchamiane oddzielnie z konsoli. W taki przypadku tylko jeden proces może tworzyć segment pamięci dzielonej, inicjować semafory i liczniki.

Zadanie 11. 4. 5 Obsługa karty interfejsowej PCM3718 – komunikacja przez pamięć dzieloną

Wykorzystując napisane wcześniej programy napisz program zapis obsługujący odczyty z karty PCM3718 i zapisujący je do bufora w pamięci dzielonej.



Rys. 11-3 Akwizycja danych z karty PCM3718 - komunikacja poprzez bufor cykliczny

Dane uzyskiwane z karty zapisywane są przez program zapis w podanej niżej strukturze `adc_t` i dalej wpisywane do bufora położonego w pamięci dzielonej. Struktura bufora `bufor_t` podana jest ponizej. Dane z bufora pobiera program odczyt i wypisuje je na konsoli.

Program `zapis` wykonuje następujące kroki:

1. Inicjuje kartę interfejsową
2. Kasuje obszar pamięci dzielonej o ile był utworzony
3. Tworzy segment pamięci dzielonej – funkcja `shm_open`
4. Ustala jego rozmiar – funkcja `ftrunc`
5. Mapuje do lokalnej przestrzeni adresowej inicjując zmienną `bufor_t *buf` – funkcja `mmap`
6. Inicjuje liczniki i semafory struktury `*buf`
7. Odczytuje kanały wejść analogowych a0 – a3 , wejścia cyfrowe i czas bieżący i zapisuje je w zmiennej `adc_t pomiar`.
8. Zapisuje zmienna `pomiar` w buforze `*buf` na pozycji `head`.
9. Przechodzi do kroku 6.

Program `odczyt` wykonuje następujące kroki:

1. Otwiera segment pamięci dzielonej – funkcja `shm_open`
2. Mapuje do lokalnej przestrzeni adresowej inicjując zmienną `bufor_t *buf` – funkcja `mmap`
3. Odczytuje zmienną `pomiar` z pozycji `tail` bufora `*buf`
4. Wypisuje kanały wejść analogowych a0 – a3 , wejścia cyfrowe i czas bieżący
5. Przechodzi do kroku 4.

```
#define BSIZE      8   // Rozmiar bufora
#define MAX        4  // Ilosc kanalow DA

typedef struct {     // Struktura rekordu z urzadzenia
   int analog[MAX];  // Kanaly analogowe
   int dinput;       // Wejscia cyfrowe
   time_t czas;      // Czas wykonania pomiaru
} adc_t;

typedef struct {      // Struktura pamieci dzielonej
    int head;    // Tu wpisujemy nowe rekordy
    int tail;    // Stąd pobieramy rekordy
    int cnt;     // Licznik rekordów
    sem_t mutex; // Semafor wzajemnego wykluczania
    sem_t empty; // Semafor wstrzymujący producenta
    sem_t full;  // Semafor wstrzymujący konsumenta
    adc_t pom[BSIZE];   // Pomiary
} bufor_t;

bufor_t *buf;
```

Przykład 11-2 Struktura bufora cyklicznego

12. Sygnały i ich obsługa.

12.1Wstęp

Sygnały są reprezentacja asynchronicznych i zwykle awaryjnych zdarzeń zachodzących w systemie w systemie. Listę obsługiwanych sygnałów można uzyskać pisząc na konsoli:

```
$kill -l
```

System obsługuje następujące sygnały:

SIGNULL	SIGHUP	SIGINT	SIGQUIT	SIGILL	SIGTRAP
SIGIOT	SIGABRT	SIGEMT	SIGFPE	SIGKILL	SIGBUS
SIGSEGV	SIGSYS	SIGPIPE	SIGALRM	SIGTERM	SIGUSR1
SIGUSR2	SIGCHLD	SIGPWR	SIGWINCH	SIGURG	SIGPOLL
SIGSTOP	SIGTSTP	SIGCONT	SIGDEV	SIGTTIN	SIGTTOU

Sygnały mogą być generowane:

- przez system operacyjny, gdy wystąpi zdarzenie awaryjne,
- w programie za pomocą funkcji kill(), alarm(), raise() oraz timery
- z konsoli za pomocą polecenia kill.

Sygnał może być obsłużony przez program aplikacyjny. Funkcja systemowa signal pozwala na zainstalowanie procedury obsługi sygnału.

```
signal(int sig, void(*funct) (int))
```

Gdzie:

`sig` Numer sygnału

`funct` Nazwa funkcji obsługującej sygnał

Procedura void funct(int) wykonana będzie gdy pojawi się sygnał sig. W systemie pierwotnie zdefiniowane są dwie funkcje obsługi sygnałów:

SIG_DFL - akcja domyślna, powoduje zwykle zakończenie procesu,

SIG_IGN - zignorowanie sygnału (nie zawsze jest to możliwe).

Prosty program przechwytujący sygnał SIGINT generowany przy naciśnięciu klawiszy (Ctrl+Break) podano poniżej.

```
#include <signal.h>
#include <stdlib.h>
#include <setjmp.h>
int sigcnt = 0;
int sig = 0;

void sighandler(int signum) {
/* Funkcja obslugi sygnalu */
  sigcnt++;
  sig = signum;
}

void main(void) {
   int i =0;
   printf("Program wystartował \n");
   signal(SIGINT, sighandler);
   do {
     printf(" %d %d %d  \n",i,sigcnt,sig);
     sleep(1); i++;
   } while(1);
}
```

Przykład 12-1 Obsługa sygnału SGINT

12.2Zadania

Zadanie 12. 4. 1 Obsługa sygnału SIGINT

Uruchom podany w Przykładzie 1 program. Sprawdź co stanie się przy próbie jego przerwania poprzez jednoczesne naciśnięcie klawiszy (Ctrl+Break).

Zadanie 12. 4. 2 Wprowadzanie hasła

Napisz program który wykonuje w pętli następujące czynności:
1. Ustawia czasomierz (funkcja alarm) na generacje sygnału za 5 sekund.
2. Wypisuje komunikat „Podaj hasło:” i próbuje wczytać łańcuch z klawiatury.
3. Gdy uda się wprowadzić hasło przed upływem 5 sekund, alarm jest kasowany i następuje wyjście z pętli.
4. Gdy nie uda się wprowadzić hasła w ciągu 5 sekund należy wyprowadzić napis: „Ponów próbe” i przejść do kroku 2.

Zadanie 12. 4. 3 Przesyłanie sygnałów pomiędzy procesami

Napisz dwa procesy – macierzysty i potomny. Proces macierzysty czeka w pętli na sygnał. Proces potomny generuje cykliczne sygnały (za pomocą funkcji kill).

Zadanie 12. 4. 4 Restarty procesu

Napisz program wykonujący restart po każdorazowej próbie jego przerwania wykonanej poprzez naciśnięcie klawiszy (Ctrl+Break). Naciśnięcie tej kombinacji klawiszy powoduje wygenerowanie sygnału SIGINT. W programie skorzystaj z funkcji setjmp i longjmp.

Zadanie 12. 4. 5 Implementacja funkcji alarm

Dokonaj próby samodzielnej funkcji `myalarm(int t)`. Wykonanie tej funkcji spowoduje wygenerowanie sygnału SIGUSR1 po upływie t sekund. Wykonanie tej funkcji z parametrem 0 ma spowodować odwołanie alarmu.

Zadanie 12. 4. 6 Implementacja przeterminowanie wysyłania komunikatu

W zadaniu Zadanie 16. 4. 1 proces klienta wysyłał komunikaty do procesu serwera. Dokonaj modyfikacji procesu klienta aby narzucić przeterminowanie T na wysłanie komunikatu. Rozwiąż zadanie dla przypadków:

a) Z użyciem funkcji alarm (T >= 1 sek).
b) Z użyciem timera (T – dowolny) .

13. Wątki w systemach RTS

13.1Tworzenie wątków

Aby wykorzystać możliwości wątków należy dysponować funkcjami umożliwiającymi administrowania wątkami. Zestaw operujących na wątkach funkcji zdefiniowany jest w pochodzącej z normy POSIX 1003 bibliotece `pthreads` (*ang. posix threads*) dostępnej w systemie QNX6 Neutrino. Prototypy operujących na wątkach funkcji zawarte są w pliku nagłówkowym `<pthread.h>`. Biblioteka `pthreads` zawiera następujące grupy funkcji:

1. Tworzenie wątków.
2. Operowanie na atrybutach wątków (ustawianie i testowanie).
3. Kończenie wątków.
4. Zapewnianie wzajemnego wykluczania.
5. Synchronizacja wątków.

Pierwsze trzy grupy funkcji zawierają mechanizmy do tworzenia wątków, ustalania ich własności, identyfikacji, kończenia oraz oczekiwania na zakończenie. Ważniejsze funkcje z tej grupy podaje Tabela 13-1.

Tworzenie wątku	`pthread_create()`
Uzyskanie identyfikatora wątku bieżącego	`Pthread_self()`
Kończenie wątku bieżącego	`pthread_exit()`
Oczekiwanie na zakończenie innego wątku	`pthread_join()`
Kończenie innego wątku	`pthread_cancel()`
Wywołanie procedury szeregującej	`pthread_yield()`

Tabela 13-1 Ważniejsze funkcje systemowe dotyczące tworzenia i kończenia wątków

Prosty program tworzący watki podaje Przykład 13-1.

```
#include <pthread.h>
#include <stdlib.h>
#define NUM_THREADS 2
#define KROKOW      4
pthread_t tid[NUM_THREADS]; // Tablica identyfikatorow watkow

void * kod(void *arg) {
   int numer = (int) arg;
   int i;
   for(i=0;i<KROKOW;i++) {
      printf("Watek: %d krok: %d \n",numer,i);
      sleep(1);
   }
   return (void*) numer;
}

int main(int argc, char *argv[]) {
   int i, status;
   void * statp;
   for (i = 0; i < NUM_THREADS; i++)
       pthread_create(&tid[i], NULL, kod,(void *)(i+1));
   for (i = 0; i < NUM_THREADS; i++) {
             pthread_join(tid[i], (void *) &status);
             printf("Watek %d zakonczony\n",status );
   }
   return  0;
}
```

Przykład 13-1 Program tworzący wątki

13.2Priorytety wątków

Priorytet wątku najlepiej ustawiać przed jego utworzeniem poprzez odpowiednią modyfikację struktury atrybutów wątku. Ważniejsze funkcje służące temu celowi podaje poniższa tabela.

Testowanie parametrów szeregowania	`pthread_getschedparam`
Ustawianie parametrów szeregowania	`pthread_setschedparam`
Inicjacja atrybutów wątku	`pthread_attr_init`
Umożliwienie dowolnego ustawienia parametrów szeregowania	`pthread_attr_setinheritsched`
Ustawienie atrybutów parametrów szeregowania	`pthread_attr_setchedparam`
Testowanie atrybutów parametrów szeregowania	`pthread_attr_getchedparam`
Ustawienie atrybutów strategii szeregowania	`pthread_attr_setchedpolicy`
Testowanie atrybutów strategii szeregowania	`pthread_attr_getchedpolicy`

Tabela 13-2 Ważniejsze funkcje operowania na priorytetach wątków

W normalnej sytuacji priorytet wątku i strategia szeregowania dziedziczone są z wątku macierzystego. Aby jednak priorytet i strategia szeregowania była pobrana z jego atrybutów a nie z wątku macierzystego należy ustawić flagę `PTHREAD_EXPLICITY_SCHED`. Flagę te ustawia się za pomocą funkcji `pthread_attr_setinheritsched()`. Poniżej podano przykład programu w którym tworzy się wątek o zadanym priorytecie.

```
// Kompilacja gcc priorytet.c -o priorytet -lm
#include <pthread.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
// Dobrac do szybkosci procesora
#define KROKI  100000000
#define PRIORYTET 8

int tid;

void * kod(void *arg) {
   float y,z;
   int ss, t2, i = 1;
   struct sched_param spar;
   pthread_getschedparam(0,&ss,&spar);
   printf("Start watku - strategia: %d priorytet: %d\n",ss,spar.sched_priority);
   while(i < KROKI) {
        z = sqrt(i*i); z = sqrt(z);
        if(i%100000 == 0) {
                printf("Licznik %d\n",i);
        }
        i++;
   }
   printf("Watek  zakonczony, \n");
   return (NULL);
}

int main(int argc, char *argv[]) {
 int res, i, sched, kon,num = 0;
 pthread_attr_t attr;
 struct sched_param p,p2;
 sched_getparam(0,&p);
 sched = sched_getscheduler(0);
 printf("Watek glowny priorytet: %d strategia: %d\n", p.sched_priority,sched);
 pthread_attr_init(&attr);
 pthread_attr_setinheritsched(&attr,PTHREAD_EXPLICIT_SCHED);
 p.sched_priority = PRIORYTET;
```

```
 pthread_attr_setschedparam(&attr,&p);
 pthread_attr_getschedparam(&attr,&p2);
 pthread_attr_setschedpolicy(&attr,SCHED_RR);
 printf("Priorytet watku: %d wynosi: %d\n",i, p2.sched_priority);
 pthread_create(&tid, &attr, kod,NULL);
 pthread_join(tid, NULL);
}
```

Przykład 13-2 Program demonstrujący ustawianie priorytetu wątku

Za pomocą polecenia `pidin` można sprawdzić czy nowy wątek ma prawidłowy priorytet i strategię szeregowania.

```
$pidin -p priorytet
     pid tid name                 prio STATE       Blocked
 1859636   1 ./priorytet           10r JOIN        2
 1859636   2 ./priorytet            8r RUNNING
```

Z kolei za pomocą polecenia `hogs` można sprawdzić ile czasu procesora zabiera wykonanie danego wątku. Gdy w jednej konsoli uruchomimy program priorytet a w drugiej napiszemy polecenie `hogs -%5 -n` otrzymamy dane o zajętości procesora przez proces `priorytet`.

13.3 Synchronizacja wątków

Współbieżny dostęp do danych może naruszyć ich integralność. Aby zapewnić integralność należy zapewnić wzajemne wykluczanie w dostęp do wspólnych danych. Do zapewnienia wyłączności dostępu do danych stosuje się mechanizm muteksu (*ang. mutex*). Najważniejsze operacje na muteksach podaje Tabela 13-3.

Inicjacja muteksu	`pthread_mutex_init()`
Zajęcie muteksu	`pthread_mutex_lock()`
Zajęcie muteksu z przeterminowaniem	`pthread_mutex_timedlock()`
Próba zajęcia muteksu	`pthread_mutex_trylock()`
Zwolnienie muteksu	`pthread_mutex_unlock()`
Skasowanie muteksu	`pthread_mutex_destroy()`
Ustalanie protokołu zajmowania muteksu	`pthread_mutexattr_setprotocol()`
Ustalanie pułapu priorytetu	`pthread_mutexattr_setprioceiling()`

Tabela 13-3 Ważniejsze funkcje operowania na muteksach

Do synchronizacji wątków stosuje się zmienne warunkowe. Najważniejsze operacje wykonywane na zmiennych warunkowych podaje Tabela 13-4.

Inicjacja zmiennej warunkowej	`pthread_cond_init()`
Zawieszenie wątku w kolejce	`pthread_cond_wait()`
Zawieszenie wątku w kolejce zmiennej warunkowej i czekanie z limitem czasowym	`pthread_cond_timedwait()`
Wznowienie wątku zawieszonego w kolejce	`pthread_cond_signal()`
Wznowienie wszystkich wątków zawieszonych w kolejce zmiennej warunkowej	`pthread_cond_broadcast()`
Skasowanie zmiennej warunkowej	`pthread_cond_destroy()`

Tabela 13-4 Najważniejsze operacje na zmiennych warunkowych

13.4 Zadania

Zadanie 13. 4. 1 Ustawianie priorytetów i strategii szeregowania wątków

Wzorując się na podanym wcześniej przykładzie napisz program priorytet służący do obserwacji wpływu priorytetu wątków na sposób ich wykonania. Program uruchamiamy podając priorytety kolejnych wątków:

```
priorytet pr1 pr2 ... prN
```

W programie należy zadeklarować następujące zmienne globalne:

`pthread_attr_t attr[WMAX]`	atrybuty wątków
`int tid[WMAX]`	identyfikatory wątków
`int licznik[WMAX]`	liczniki zwiększane przez wątki
`int stop[WMAX]`	stop[i] = 1 gdy wątek i się zakończył

Wątek główny wykonuje następujące kroki:
1. Inicjuje tablice `licznik, stop i attr`.
2. Ustawia atrybuty wątków `attr` tak aby ich priorytety zgodne były z parametrami `pr1 pr2 ... prN`
3. Tworzy N = `argc – 1` wątków o atrybutach podanych w tablicy `attr`
4. Z cyklem 1 sekundowym wypisuje zawartość liczników obrazujących postęp.
5. Gdy wszystkie wątki się zakończą wykonuje funkcję `pthread_join(...)`.

Wątek roboczy i wykonuje następujące kroki:
1. Wypisuje swój priorytet i strategię szeregowania
2. Wykonuje kroki zajmujące czas procesora jak w Przykład 13-2 i co jakiś czas aktualizuje tablicę `licznik[i]` wpisując tam numer aktualnego kroku.
3. Gdy zakończy obliczenia wpisuje do tablicy `stop[i] = 1`.

Z uruchomionym programem należy wykonać następujące ćwiczenia:
1. Uruchomić program na komputerze pc104 z różnymi parametrami.
2. Zaobserwować działanie programu na komputerze pc104 za pomocą polecenia `pidin`
3. Zaobserwować działanie programu na komputerze pc104 za pomocą polecenia `hogs`
4. Zaobserwować działanie programu na komputerze pc104 za pomocą analizatora systemu z platformy *Momentics*
5. Zmienić strategię szeregowania na FIFO i zaobserwować działanie wątków.
6. Zmienić strategię szeregowania na sporadyczną i zaobserwować działanie wątków

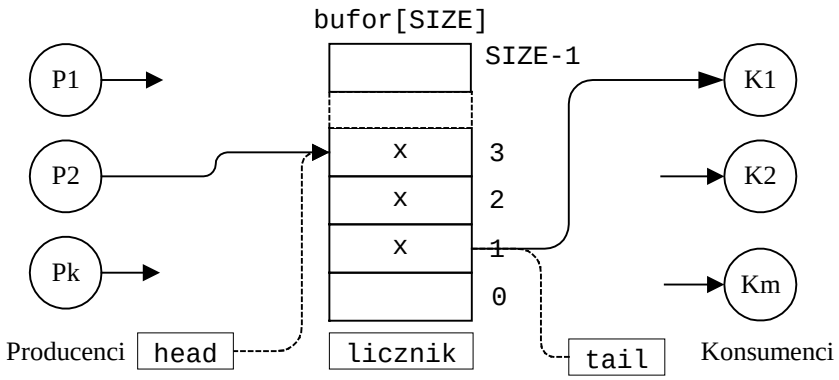
```
#Priorytet 2 4 6 8
Watek glowny priorytet: 10 strategia: 2
Licznik -  0 ,   0 ,   0 ,   0 ,
Start watku: 3 Strategia: 2 priorytet: 8
Licznik -  0 ,   0 ,   0 ,   1000 ,
...
Licznik -  0 ,   0 ,   0 ,   8000 ,
Watek: 3 zakonczony, czas: 7
Start watku: 2 Strategia: 2 priorytet: 6
Licznik -  0 ,   0 ,   0 ,   10000 ,
...
Licznik -  0 ,   0 ,   9000 ,   10000 ,
Watek: 2 zakonczony, czas: 14
Start watku: 1 Strategia: 2 priorytet: 4
Licznik -  0 ,   0 ,   10000 ,   10000 ,
...
Licznik -  0 ,   8000 ,   10000 ,   10000 ,
Watek: 1 zakonczony, czas: 21
Start watku: 0 Strategia: 2 priorytet: 2
Licznik -  0 ,   10000 ,   10000 ,   10000 ,
...
Licznik -  8000 ,   10000 ,   10000 ,   10000 ,
Watek: 0 zakonczony, czas: 28
Licznik -  10000 ,   10000 ,   10000 ,   10000 ,
```

Przykład 13-3 Wyniki działania programu priorytet

Zadanie 13. 4. 2 Zadanie 1 Problem producenta i konsumenta

Rozwiąż pokazany na Rys. 13-1 problem producenta i konsumenta posługując się mechanizmem wątków. Bufor ma być tablicą `char bufor[SIZE][LSIZE]` zawierająca napisy oraz wskaźniki `head` i `tail` oraz zmienną `licznik`. Wątek producenta ma wpisywać do bufora łańcuchy: „Producent: i krok: k” które mają być następnie pobierane przez konsumenta. Wpis następuje na pozycji `head`. Konsument pobiera zawartość bufora z pozycji `tail` i wyświetla ją na konsoli. Do synchronizacji użyj muteksu `mutex` oraz zmiennych warunkowych `puste` i `pelne`.

Program należy uruchamiać podając liczbę producentów i konsumentów: `prodkons liczba_prod liczba_kons`.



Rys. 13-1 Problem producenta i konsumenta

14. Timery i ich wykorzystanie w systemach RTS

14.1 Funkcje i programowanie timerów

Jedną z najczęściej spotykanych funkcji systemu czasu rzeczywistego jest generowanie zdarzeń które w ustalonym czasie uruchomić mają określone akcje systemu. System operacyjny zawiera specjalnie do tego celu utworzone obiekty nazywane timerami (*ang. timers*). Aby użyć timera należy:

- Utworzyć timer – podaje się specyfikację generowanego zdarzenia
- Nastawić timer – podaje się specyfikację czasu wyzwolenia

W systemie QNX6 Neutrino timery generować mogą następujące typy zdarzeń:

- impulsy,
- sygnały,
- utworzenie nowego wątku.

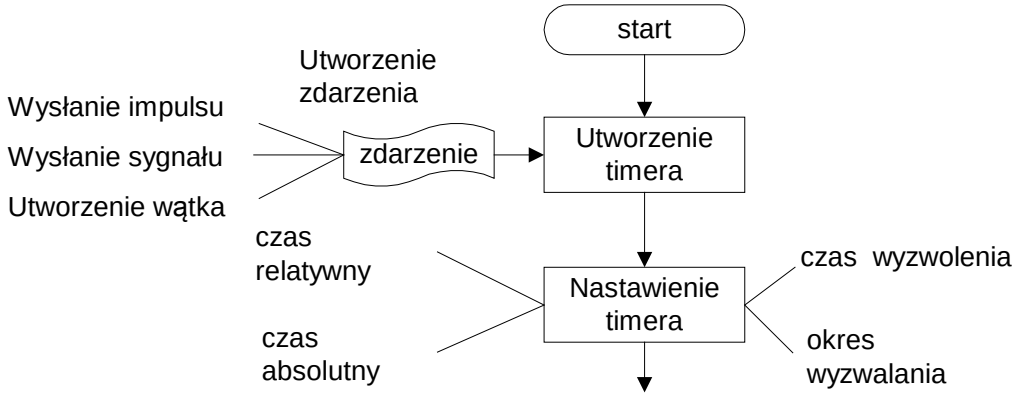
Ustawienie timera polega na przekazaniu mu informacji o

- planowanym czasie wyzwolenia,
- sposobie określenia tego czasu
- trybie pracy timera.

Czas można określać w sposób:
- absolutny - czas UTC lub lokalny
- relatywny - przesunięcie czasowe począwszy od chwili bieżącej

Timer może pracować w dwóch trybach:

1. Wyzwolenie jednorazowe (*ang. one shot*)
2. Wyzwalanie cykliczne (*ang. periodical*)



Rys. 14-1 Etapy przygotowania timera do pracy

14.2 Zdarzenia

System QNX6 Neutrino posiada uniwersalny a zarazem jednolity system powiadamiania o zdarzeniach (ang. *event*). Zdarzenie może być:
- impulsem,
- sygnałem
- zdarzeniem które uruchamia wątek.

W zawiadomieniach używa się struktury `sigevent` zdefiniowanej w pliku nagłówkowym `<sys/siginfo.h>` której pole `sigev_notify` decyduje o typie zdarzenia.

Wysyłanie impulsów

W przypadku wysyłania impulsów pole `sigev_notify` przyjmuje wartość `SIGEV_PULSE`. Z każdym impulsem wiąże się kod i wartość . Impuls wysyłany jest do kanału za pośrednictwem połączenia i zgodnie z pewnym priorytetem. Atrybuty impulsu zawarte są w polach struktury `sigevent` .

Pole	Zawartość
`sigev_coid`	Pole zawiera identyfikator połączenia coid z kanałem do którego ma być wysłany impuls
`sigev_value`	32 bitowa wartość związana z impulsem
`sigev_code`	8 bitowy kod związany z impulsem
`sigev_priority`	Priorytet impulsu, wartość 0 nie jest dopuszczalna

Tabela 14-1 Pola struktury sigevent gdy wysyłany jest impuls

Makro do inicjowania pól struktury `sigevent` na wysłanie impulsu:

```
SIGEV_PULSE_INIT(&event,coid,priority,code,value )
```

Wysyłanie sygnałów

Wysyłanie sygnałów zachodzi gdy pole `sigev_notify` przyjmie wartość `SIGEV_SIGNAL.` Do inicjowania struktury można użyć makra:

```
SIGEV_SIGNAL_INIT( &event, signal )
```

`signal` - numer sygnału.

Uruchamianie wątków

Gdy pole `sigev_notify` struktury `sigevent` przyjmie wartość `SIGEV_THREAD` to zdarzenie polegało będzie na uruchomieniu wątku.

Pole	Zawartość
`sigev_notify_function`	Pole zawiera adres funkcji `(void *) func(void *value)`. Z funkcji tej będzie utworzony wątek gdy zajdzie zdarzenie.
`sigev_value`	Pole zawiera parametr value przekazywany do funkcji `func()`.
`sigev_notify_attributes`	Pole zawiera strukturę z atrybutami wątku który ma być utworzony

Tabela 14-2 Pola struktury `sigevent` gdy uruchamiany jest wątek

Makro do inicjowania elementów struktury:

```
SIGEV_THREAD_INIT( &event, func, value, attributes )
```

14.3Tworzenie i ustawianie timerów

Timer jest obiektem tworzonym przez system operacyjny a jego funkcją jest generowanie zdarzeń w precyzyjnie określonych chwilach czasu. Aby użyć timera należy wykonać następujące czynności:

1. Zdecydować jaki typ zawiadomień ma generować timer (impulsy, sygnały, uruchomienie wątku) i utworzyć strukturę typu `sigevent`.
2. Utworzyć timer.
3. Zdecydować o rodzaju określenia czasu (absolutny lub relatywny).
4. Zdecydować o trybie pracy (timer jednorazowy lub cykliczny)
5. Nastawić go czyli określić tryb pracy i czas zadziałania.

Opis	Funkcja
Utworzenia timera	`timer_create()`
Nastawienie timera	`timer_settime()`
Uzyskanie ustawień timera	`timer_gettime()`
Kasowanie timera	`timer_delete()`

Tabela 14-3 Funkcje operujące na timerach

Tworzenie timera

Timer tworzy się za pomocą funkcji `timer_create()`.

```
int timer_create(clockid_t clock, struct sigevent *evn, timer_t
*timerid)
```

`clock` Identyfikator zegara użytego do odmierzania czasu obecnie `CLOCK_REALTIME`

`evn` Struktura typu sigevent zawierająca specyfikację generowanego zdarzenia.

`timerid` Wskaźnik do struktury zawierającej nowo tworzony timer

Typ powiadomienia określa struktura `sigevent`

- impuls - `SIGEV_PULSE_INIT`
- sygnał - `SIGEV_SIGNAL_INIT`
- odblokowanie wątku - `SIGEV_THREAD_INIT`.

Ustawianie timera

Ustawienie timera polega na określeniu: sposobu określenia czasu, czasu wyzwolenia, okresu repetycji. Do ustawiania timera służy funkcja `timer_settime()`

`int timer_settime(timer_t *timerid,int flag, struct itimerspec *val, struct itimerspec *oldval)`	
`timerid`	Identyfikator timera zainicjowany przez funkcję `timer_create`
`flag`	Flagi specyfikujące sposób określenia czasu 0 – czas relatywny `TIMER_ABSTIME` – czas absolutny
`val`	Specyfikacja nowego czasu aktywacji
`oldval`	Specyfikacja poprzedniego czasu aktywacji

```
struct itimerspec {
 struct timespec it_value;       // pierwsza aktywacja
 struct timespec it_interval;    // interwał
}
struct timespec {
  long tv_sec;   // sekundy
  long tv_nsec;  // nanosekundy
}
```

`it_value` - Czas pierwszego uruchomienie
`it_interval` - Okres repetycji

`it_value`	`it_interval`	Typ zdarzeń
`v > 0`	`x > 0`	Cykliczne generowanie zdarzeń co `x` począwszy od `v`
`v > 0`	`0`	Jednorazowa generacja zdarzenia w `v`
`0`	dowolny	Timer zablokowany
`x > 0`	`x > 0`	Cykliczne generowanie zdarzeń co `x`

Tabela 14-4 Ustawianie trybu pracy timera

Przykład 2 - timer cykliczny po upływie 2.5 sekundy będzie generował zdarzenia cyklicznie co 1 sekundę.

```
it_value.tv_sec = 2;
it_value.tv_nsec = 500 000 000;
it_interval.tv_sec = 1
it_interval.tv_nsec = 0;
```

```
// Serwer odbierajacy komunikaty
// Komunikaty wysyla proces send
// uruchomiony timer wysylajacy impulsy
// -------------------------------------------------------
  ...
 #define SIZE 256
 #define MOJA_NAZWA "seweryn"
 #define FLAGA 0
 struct  {
     int  type;        // typ komunikatu
     char text[SIZE];  // tekst  komunikatu
 } msg, rmsg;

 main(int argc, char *argv[]) {
```

```
  int  pid, con,i ,coid,id,priority;
  struct _msg_info info;
  name_attach_t  *nazwa_s;
  timer_t timid;
  struct sigevent evn;
  struct itimerspec t;

 // Utworzenie i rejestracja nazwy -----------------
  nazwa_s = name_attach(NULL,MOJA_NAZWA, FLAGA);

  // Tworzenie polaczenia ”do samego siebie”
  coid = ConnectAttach(0,0,nazwa_s->chid,0,0);

  // Inicjacja struktury evn -------------
  priority = getprio(0);
  SIGEV_PULSE_INIT(&evn,coid,priority,1,0);

  // Utworzenie timera -------------------
  id = timer_create(CLOCK_REALTIME,&evn,&timid);

  // Nastawienie timera ------------------
  t.it_value.tv_sec  = 2;
  t.it_value.tv_nsec = 0;
  t.it_interval.tv_sec = 2;
  t.it_interval.tv_sec = 0;
  timer_settime(timid,0,&t,NULL);

  // Odbior komunikatow ----------------------------------
  for(i=0;; i++) {
   pid = MsgReceive(nazwa_s->chid,&msg,sizeof(msg),&info);
   if(pid == -1) {
      printf("Blad:  %d\n",errno); continue;
   }
   if(pid == 0)  { // Impuls
       printf("Odebrany impuls \n");
   } else {        // Komunikat
       printf("Kom: %d text: %s \n",msg.type, msg.text);
       sprintf(msg.text,"potwierdzenie %d",i+1);
       MsgReply(pid,0,&msg,sizeof(msg));
   }
  }
}
```

Przykład 14-1 Proces serwera z timerem wysyłającym impulsy

14.4Zadania

Zadanie 14. 4. 1 Cykliczna generacja impulsów sygnałów i uruchamiane wątków

Zmodyfikuj podany wcześniej Przykład 14-1. Program powinien utrzymywać trzy liczniki:licznik impulsów, licznik sygnałów i licznik uruchomień wątków. Zmodyfikowany program powinien:

- Co 1 sekundę generował impuls. Sekcja obsługi impulsu powinna wyświetlać licznik komunikatów, impulsów, sygnałów i uruchomień wątków oraz zwiększać licznik impulsów.
- Co 3 sekundy generować sygnał. Funkcja obsługi sygnału powinna zwiększać licznik impulsów.
- Co 5 sekund uruchamiać wątek. Funkcja wykonawcza wątku powinna wyświetlać stan liczników i zwiększać licznik wątków.

Zadanie 14. 4. 2 Sterowanie sekwencyjne z użyciem czasomierzy – Komputer PC104

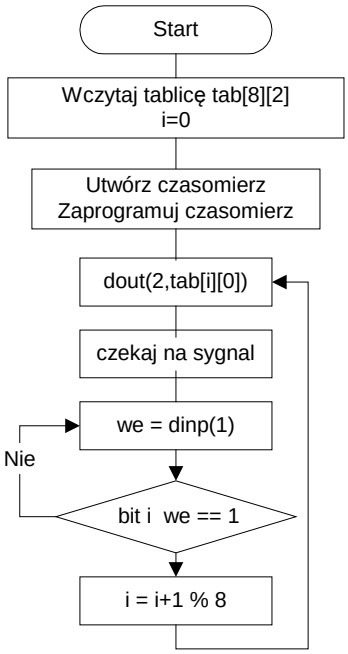
Ćwiczenie to jest modyfikacją wcześniejszego ale do odmierzania czasu zostanie użyty czasomierz (ang. *timer*). Sterowanie sekwencyjne polega na załączaniu urządzeń dwustanowych (reprezentowanych przez wyjścia cyfrowe) zgodnie z zadaną wcześniej sekwencją. Przejście do kolejnej sekwencji uzależnione jest od czasu i może być

uzależnione od czynników zewnętrznych np. od stanu wejść cyfrowych. Sterowanie sekwencyjne może być opisane za pomocą sekwencji struktur

```
typedef  struct  {
              int nr; // numer kroku
              struct timespec czas;      //  Opoznienie
              int wy;                    // Co na wyjsciu
              int we;                    // Co na wejsciu
 } krok_t;

krok_t prog[SIZE]   = { ...}
```

`prog[i] i=0,1,...7` gdzie i jest numerem kroku, `wy(i)` jest stanem wyjść cyfrowych a `prog[i].czas` jest opóźnieniem przejścia do stanu `i=(i+1)%SIZE`. Dodatkowym warunkiem przejścia do stanu i+1 będzie wymaganie aby wartość bitu i wejścia cyfrowego `prog[i].we` była równa 1.

Napisz program który realizuje sterowanie sekwencyjne z wykorzystaniem karty PCM3718. Z tablicy `tab[8]` program pobiera: wartość wyjścia cyfrowego `wy,` opóźnienie czas. Następnie wyprowadza na wyjścia cyfrowe żądany bajt wyjściowy wykonując funkcję: `dout(2,wy).` Po odczekaniu Ti sekund odczytuje wartość wejść cyfrowych `we(i)=dinp(1)` i sprawdza czy bit i równy jest 1. Gdy tak przechodzi do kroku i+1, gdy nie czeka. Po wyczerpaniu tablicy sterującej (gdy i ==7) proces zaczyna się od początku.



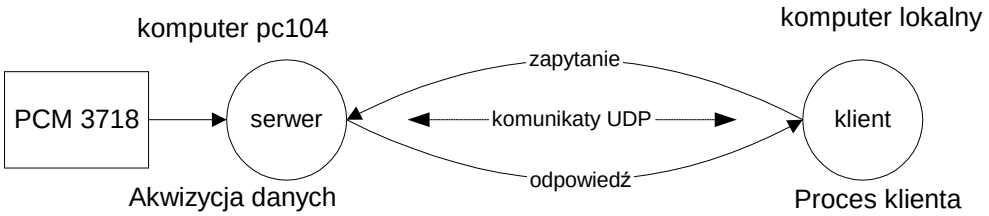
Rys. 14-2 Przebieg sterowania sekwencyjnego

Do odmierzania czasu użyj czasomierza (funkcje `timer_create`, `timer_settime`) który generuje sygnał SIGUSR1. Do czekania na sygnał użyj funkcji `sigpause`.

15. Rozproszony system sterowania i akwizycji danych, komunikacja UDP

Jeżeli system akwizycji danych składa się z komputerów pracujących pod kontrolą różnych systemów operacyjnych (Windows, Linux, QNX) nie jest możliwe wykorzystanie protokołu QNET. W takim przypadku do komunikacji wykorzystany może być protokół UDP. Tematem ćwiczenia jest napisanie programu serwera i klienta obsługującego kartę PCM3718. Program serwera wykonywany jest na komputerze PC104 a program klienta na komputerze lokalnym. Program serwera ma wykonywać zlecenia klienta i wykonywać podstawowe funkcje dotyczące obsługi karty PCM3718 takich jak:

- Podanie wartości kanałów analogowych
- Podanie wartości wejść cyfrowych
- Ustawienie wyjść cyfrowych



Rys. 15-1 Akwizycja danych z karty PCM3718 - komunikacja poprzez datagramy UDP

15.1 Adresy gniazd i komunikacja bezpołączeniowa UDP

Jeżeli mające się komunikować procesy znajdują się na różnych komputerach do komunikacji może być użyty protokół TCP/IP wraz z interfejsem gniazdek BSD. W komunikacji UDP każdy komunikat adresowany jest oddzielnie a ponadto zachowywane są granice przesyłanych komunikatów. W komunikacji bezpołączeniowej stosowane są następujące funkcje:

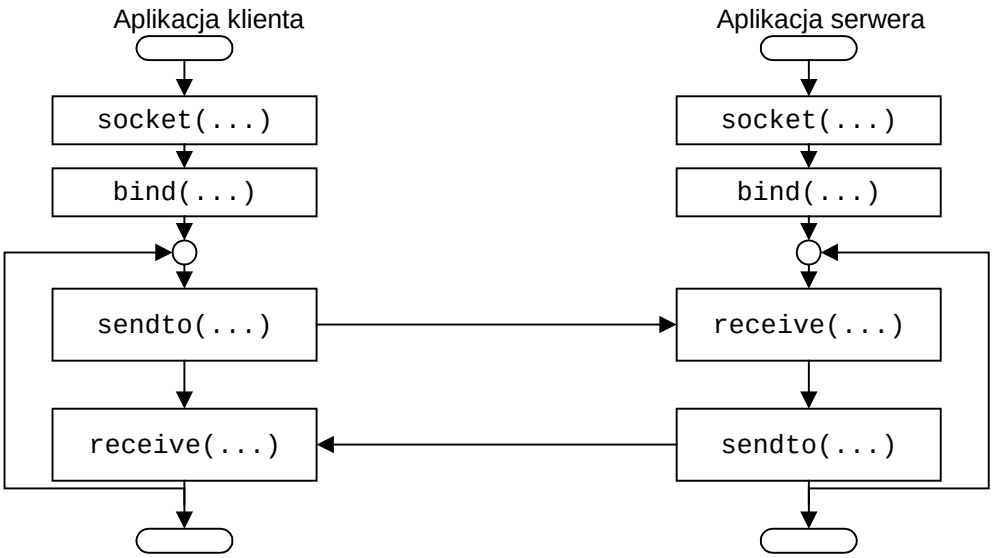
`socket`	Utworzenie gniazdka
`bind`	Powiązanie z gniazdkiem adresu IP
`sendto`	Wysłanie datagramu do odbiorcy
`recfrom`	Odbiór datagramu
`htons, htonl`	Konwersja formatu lokalnego liczby do formatu sieciowego (s – liczba krótka, s – liczba długa)
`ntohs, ntohl`	Konwersja formatu sieciowego liczby do formatu lokalnego (s – liczba krótka, s – liczba długa)
`close`	Zamknięcie gniazdka
`gethostbyname`	Uzyskanie adresu IP komputera na podstawie jego nazwy
`inet_aton`	Zamiana kropkowego formatu zapisu adresu IP na format binarny

Tabela 15-1 Ważniejsze funkcje używane w interfejsie gniazdek – komunikacja bezpołączeniowa

Sprawdź w podręczniku ich parametry i znaczenia. Kolejność działań podejmowanych przez klienta i serwera podana jest poniżej a ich współpracę pokazuje Rys. 15-2.

<u>Klient:</u>
Tworzy gniazdko - `socket`
Nadaje gniazdku adres - `bind`
Nadaje lub odbiera dane - `sendto, recfrom,`

<u>Serwer:</u>
Tworzy gniazdko - `socket`
Nadaje gniazdku adres - `bind`
Nadaje lub odbiera dane - `sendto, recfrom`



Rys. 15-2 Przebieg komunikacji bezpołączeniowej

Dane poniżej przykłady mogą być użyte jako wzorce do budowania programów korzystających z komunikacji bezpołączeniowej.

```
 // Proces odbierajacy komunikaty - wysyla udp_cli
 // kompilacja gcc udp_serw.c -lsocket -o udp_serw
 #include <arpa/inet.h>
 #include <netinet/in.h>
 #include <stdio.h>
 #include <sys/types.h>
 #include <sys/socket.h>
 #include <unistd.h>
 #define BUFLEN 80
 #define KROKI 10
 #define PORT 9950

 typedef struct {
  int typ;
  char buf[BUFLEN];
} msgt;

  void blad(char *s) {
     perror(s);
     exit(1);
   }

int main(void) {
   struct sockaddr_in adr_moj, adr_cli;
   int s, i, slen=sizeof(adr_cli),snd, rec, blen=sizeof(msgt);
   char buf[BUFLEN];
   msgt msg;

   gethostname(buf,sizeof(buf));
   printf("Host: %s\n",buf);

   s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
   if(s < 0) blad("socket");
   printf("Gniazdko %d utworzone\n",s);
   // Ustalenie adresu IP nadawcy
   memset((char *) &adr_moj, 0, sizeof(adr_moj));
   adr_moj.sin_family = AF_INET;
```

```
   adr_moj.sin_port = htons(PORT);
   adr_moj.sin_addr.s_addr = htonl(INADDR_ANY);
   if (bind(s, &adr_moj, sizeof(adr_moj))==-1)
       blad("bind");

   // Odbior komunikatow ------------
   for (i=0; i<KROKI; i++) {
      rec = recvfrom(s, &msg, blen, 0, &adr_cli, &slen);
      if(rec < 0) blad("recvfrom()");
      printf("Odebrano komunikat z %s:%d res %d\n  Typ: %d %s\n",
            inet_ntoa(adr_cli.sin_addr), ntohs(adr_cli.sin_port),
            rec,msg.typ,msg.buf);
      // Odpowiedz -----
      sprintf(msg.buf,"Odpowiedz %d",i);
      snd = sendto(s, &msg, blen, 0, &adr_cli, slen);
      if(snd < 0) blad("sendto()");
   }

   close(s);
   return 0;
}
```

Przykład 15-1 Proces odbierający komunikaty - serwer

```
// Proces wysylajacy komunikaty - wysyla udp_serwi
 // kompilacja gcc udp_cli.c -lsocket -o udp_cli
 #include <netinet/in.h>
 #include <stdio.h>
 #include <sys/types.h>
 #include <sys/socket.h>
 #include <unistd.h>
 #define BUFLEN 80
 #define KROKI 10
 #define PORT 9950
 #define SRV_IP "127.0.0.1"

typedef struct {
  int typ;
  char buf[BUFLEN];
} msgt;

  void blad(char *s) {
     perror(s);
     exit(1);
   }

int main(int argc, char * argv[]) {
   struct sockaddr_in adr_moj, adr_serw, adr_x;
   int s, i, slen=sizeof(adr_serw), snd, blen=sizeof(msgt),rec;
   char buf[BUFLEN];
   msgt msg;

   s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
   if(s < 0) blad("socket");
   printf("Gniazdko %d utworzone\n",s);
   memset((char *) &adr_serw, 0, sizeof(adr_serw));
   adr_serw.sin_family = AF_INET;
   adr_serw.sin_port = htons(PORT);
   if (inet_aton(argv[1], &adr_serw.sin_addr)==0) {
```

```
        fprintf(stderr, "inet_aton() failed\n");
        exit(1);
   }

   for (i=0; i<KROKI; i++) {
      printf("Sending packet %d\n", i);
      msg.typ = 1;
      sprintf(msg.buf, "Komunikat %d", i);
      snd = sendto(s, &msg, blen, 0, &adr_serw, slen);
      if(snd < 0) blad("sendto()");
      printf("Wyslano komunikat res: %d\n", snd);
      rec = recvfrom(s, &msg, blen, 0, &adr_x, &slen);
      if(rec < 0) blad("recvfrom()");
      sleep(1);
   }
   close(s);
   return 0;
}
```

Przykład 15-2 Proces wysyłający komunikaty UDP – klient.

Przykłady należy skompilować a następnie uruchomić w oddzielnych oknach tego samego komputera lub też na różnych komputerach. Program klienta uruchomić podając adres IP komputera na którym wykonywany jest program serwera.

```
$./udp_cli adres_ip_serwera
```

Przykładowo gdy mamy dwa komputery o adresach IP: komputer klienta IP=192.168.0.158, komputer serwera IP=192.168.0.160 to najpierw na komputerze serwera uruchamiamy program serwera pisząc

```
$./udp_serw
```

Następnie na komputerze klienta uruchamiamy program

```
$./udp_cli 192.168.0.160
```

15.2 Specyfikacja komunikacji klient - serwer

Na komputerze typu PC104 wyposażonym w kartę interfejsową PCM3718 wykonywany ma być program serwera obsługującego tę kartę. Serwer realizuje następujące funkcje:

- Odczyt wybranego kanału z przetwornika AD
- Zapis stanu wyjść cyfrowych
- Odczyt stanu wejść cyfrowych

Komunikacja z serwerem zachodzi za pośrednictwem następujących komunikatów w podanym niżej formacie.

```
#define ADREAD   1    // Odczyt z przetwornika AD
#define DIREAD   2    // Odczyt z wejsc cyfrowych DINP1
#define DOWRITE  4    // Zapis na wyjscia cyfrowe DOUT2

typedef struct {
       int typ;    // Typ polecenia
       int value;  // Wartosc
       int chan;   // Kanal
       int result; // -1 gdy blad
} pcl_t;
```

Przykład 15-3 Format komunikatu aplikacji współpracy z kartą PCM3718

15.3 Zadania

Zadanie 15. 4. 1 Program klienta i serwera

Napisz program klienta odczytujący i wyświetlający na konsoli:

- Zawartość kanałów AD0 – AD3

- Stan wejść cyfrowych w postaci binarnej

```
Utworzyć gniazdko
Nadać gniazdku adres
do {
      wysłać komunikaty ADREAD dla odczytu kanałów AD0 – AD3
      wyświetlić na konsoli kanały AD0 – AD3
      wysłać komunikaty DIREAD dla odczytu wejść cyfrowych
      wyświetlić na konsoli stan wejść cyfrowych
} while(true)
```

Przykład 15-4 Szkic programu klienta

Napisz własny program serwera obsługujący komunikaty wysyłane przez klienta. Wywołanie programu: Zwróć uwagę na to że serwer może być uruchomiony tylko na komputerze PC104 i wtedy gdy program ten ma EUID – 0 czyli wykonywany jest przez użytkownika `root`.

```
// Utworzenie gniadka
s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if(s < 0) blad("socket");

// Nadanie adresu gniazdku ---------
...
if (bind(s, (struct sockaddr *)&adr_moj, sizeof(adr_moj))==-1)
       blad("bind");
...

// Inicjacja układów karty
card_init(0,MAX,5);

do {
  rec = recvfrom(s, (void *)&msg, blen, 0,(struct sockaddr *) &adr_cli, &slen);
  if(rec < 0) blad("recvfrom()");
  printf("Odebrano komunikat z %s:%d res %d\n  typ: %d \n",
          inet_ntoa(adr_cli.sin_addr), ntohs(adr_cli.sin_port), rec,msg.typ);
   // Obsluga zlecen klientow
   switch(msg.typ) {
        case ADREAD:   // Odczyt kanalu analogowego ---------
                       ...
                       msg.value = ?;
                       break;
        case DIREAD:   msg.value = ?;
                       break;
        case DOWRITE:   ? ;
                       break;
        default:       msg.result = -1;
      }  // switch

    // Odpowiedz --------------
   snd = sendto(s, (void *)&msg, blen, 0, (struct sockaddr *)&adr_cli, slen);
   if(snd < 0) blad("sendto()");
} while(true)
```

Przykład 15-5 Szkic programu serwera UDP

,

Zadanie 15. 4. 2 Program klienta i serwera - wykorzystanie pamięci dzielonej

W poprzednim zadaniu występował problem z odczytem kanałów analogowych wynikający z niemożności odczytu zadanego kanału. Należało kanały odczytywać po kolei aż do napotkania żądanego kanału, co zmniejszało szybkość działania systemu. W tym zadaniu dane z przetwornika AD i wejścia cyfrowe mają być odczytywane przez oddzielny

proces o nazwie `odczyt` który komunikuje się z procesem głównym przez pamięć dzieloną. Tym sposobem proces odczy wpisuje bieżące pomiary do pamięci dzielonej skąd odczytuje je proces główny i nie musi on czekać na dane pomiarowe z właściwego kanału.

```
typedef struct {
  int advalue[MAX]; // dane kanalow analogowych
  int digit;        // kanal wejsc cyfrowych
  sem_t mutex;      // semafor wzajemnego wykluczania
} buf_t;
```

Program `serwer` wykonuje następujące kroki:

1. Inicjuje kartę interfejsową
2. Kasuje obszar pamięci dzielonej o ile był utworzony
3. Tworzy segment pamięci dzielonej – funkcja `shm_open`
4. Tworzy gniazdko
5. Ustala jego rozmiar – funkcja `ftrunc`
6. Mapuje do lokalnej przestrzeni adresowej inicjując zmienną `bufor_t *buf` – funkcja `mmap`
7. Inicjuje liczniki i semafory struktury `*buf`
8. Czeka na komunikat i określa jego typ
9. Odczytuje kanał analogowy lub cyfrowy z segmentu pamięci dzielonej
10. Przechodzi do kroku 8.

Program `odczyt` wykonuje następujące kroki:

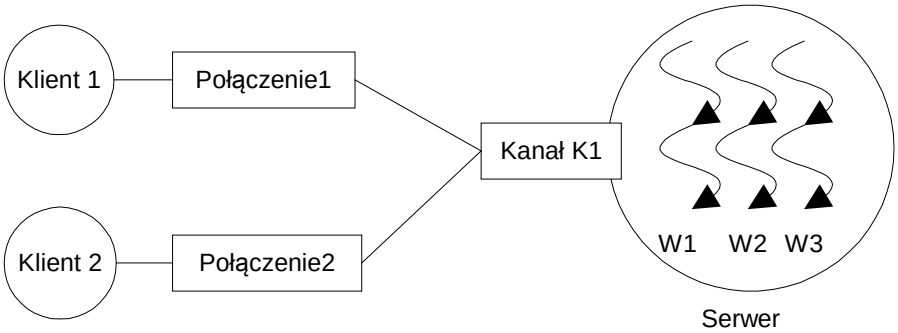
1. Odczytuje kolejny kanał analogowy i
2. Wypisuje kanał `i` do bufora w pamięci dzielonej do tablicy `advalue[i]` na pozycji `i`
3. Odczytuje wejścia cyfrowe i wpisuje do bufora w elemencie `digit`
4. Czeka 1 ms
5. Przechodzi do kroku 1.

Należy zabezpieczyć dostęp do segmentu pamięci dzielonej semaforem.

16. Wykorzystanie komunikatów do budowy systemów rozproszonych , aplikacje klient-serwer

16.1Tworzenie kanałów i połączeń

Komunikaty przesyłane są pomiędzy procesem klienta a serwera za pośrednictwem połączeń i kanałów.
Serwer tworzy kanały i oczekuje w nich komunikatów od klienta. Klient tworzy połączenie do kanału i wysyła do połączenia komunikaty.



Rys. 16-1 Połączenia klientami a kanałem serwera

Działanie	Funkcja
Tworzenie kanału	`ChannelCreate()`
Kasowanie kanału	`ChannelDestroy()`
Tworzenie połączenia	`ConnectAttach()`
Kasowanie połączenia	`ConnectDetach()`

Tabela 16-1 Funkcje operujące na kanałach i połączeniach

Tworzenie kanału

Pierwszą czynnością którą należy wykonać przy implementacji serwera jest utworzenie kanału do którego może się łączyć wielu klientów. Kanał jest własnością procesu w którym tworzący go wątek jest zawarty. Wątki które chcą się skomunikować z danym kanałem mogą:

- Być zawarte w tym samym procesie,
- Mogą należeć do innego procesu na tym samym węźle
- Mogą należeć do innego procesu na innym węźle sieci.

Tworzenie połączenia

Gdy wątek chce wysłać komunikat do jakiegoś kanału musi utworzyć z kanałem tym połączenie. Połączenie tworzone jest przez funkcję `ConnectAttach()`. Jako argument należy podać:

- identyfikator węzła NID (*ang. Node Descriptor*) z którym się komunikujemy (0 gdy jest to ten sam węzeł),
- PID procesu do którego należy kanał
- identyfikator kanału CHID .

Problem uzyskania przez proces klienta parametrów (CHID, PID, NID) może być rozwiązany w następujący sposób:

1. Użyć zmiennych globalnych zawierających ND,PID,CHID. Podejście to wykorzystane być może gdy połączenie do procesu macierzystego tworzą procesy bądź wątki potomne.
2. W ustalonym z góry miejscu serwer może utworzyć plik tekstowy który zawierał będzie ND, PID i CHID zapisane w postaci łańcucha znaków ASCII. Plik ten może być odczytany przez proces klienta nawet gdy będzie on na innym węźle sieci.
3. Użyć mechanizm globalnych nazw GNS (*ang. Global Names Service*) który zawiera funkcje `name_attach()`, `name_detach()`, `name_open()` i `name_close()`.

W przypadku przyjęcia rozwiązania 2 należy umieć przekształcić nazwę węzła w jego numer wymagany w funkcji `ConnectAttach()`. Przekształcenie nazwy węzła w jego numer wykonuje się za pomocą funkcji `netmgr_strtond()`.

Funkcja 16-1 `netmgr_strtond()` – przekształcenie nazwy węzła w numer	
`int netmgr_strtond(char *nodename, NULL)`	
`nodename`	Nazwa węzła

16.2 Wysyłanie, odbiór i potwierdzanie komunikatów

Pojedyncza transakcja komunikacyjna składa się z trzech faz:

1. Wysłanie komunikatu przez klienta do serwera
2. Odbiór komunikatu przez serwer
3. Przesłanie przez serwer odpowiedzi do klienta

Funkcje wysyłania, odbioru i potwierdzania komunikatu podaje Tabela 16-2.

Znaczenie	Funkcja
Wysłanie komunikatu	`MsgSend()`
Odbiór komunikatu	`MsgReceive(), MsgReceivePulse()`
Odpowiedź na komunikat	`MsgReply()`

Tabela 16-2 Funkcje operujące na komunikatach w systemie QNX6 Neutrino

Należy zauważyć ze funkcja `MsgSend()` służy jednocześnie do wysłania komunikatu i odbioru odpowiedzi.

16.3 Przesyłanie komunikatów przez sieć, usługa GNS

Wygodnie jest tworzyć połączenie do serwera tylko na podstawie jego nazwy. Usługa taka zapewniana jest przez administrator nazw GNS (*ang. Global Names Service*). Procesy mogą rejestrować w serwerze GNS swoje nazwy. Nzwa może być zarejestrowana lokalnie lub globalnie.

- Rejestracja lokalna: nazwa widoczna w katalogu `/dev/name/local` komputera lokalnego
- Rejestracja globalna: nazwa widoczna w katalogach `/dev/name/global` wszystkich komputerów połączonych siecią Qnet.

Uwaga – tylko programy wykonywane przez użytkownika `root` mogą rejestrować swoje nazwy globalnie. Do korzystania z mechanizmu GNS używane są następujące funkcje zestawione w Tabela 16-3.

Działanie	Funkcja
Rejestracja nazwy w serwerze GNS	`name_attach()`
Wyrejestrowanie nazwy	`name_detach()`
Lokalizacja nazwy i tworzenie połączenia	`name_open()`
Kasowanie połączenia	`name_close()`

Tabela 16-3 Funkcje obsługi nazw w systemie QNX6 Neutrino

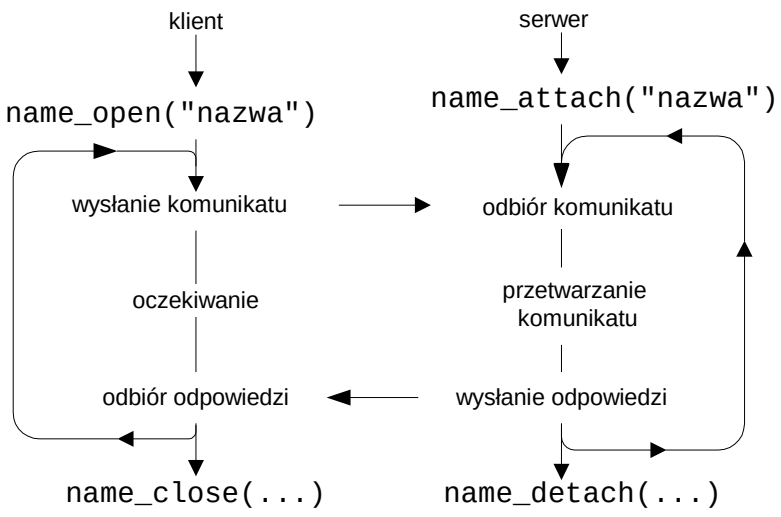
Korzystanie z mechanizmu GNS odbywa się według danego poniżej schematu który wyjaśniony jest na Rys. 16-2.

Proces serwera powinien:

- Zarejestrować nazwę – funkcja: `name_attach()`,
- Utworzyć kanał – funkcja: `ChannelCreate()`
- Przejść do odbierania komunikatów z tego kanału – funkcja: `MsgReceive()`

Proces klienta powinien:

- Zlokalizować serwer i utworzyć do niego połączenie – funkcja: `name_open()`.
- Przejść do wysyłania / odbierania komunikatów do tego połączenia – funkcja: `MsgSend()`.



Rys. 16-2 Klient tworzy połączenie do serwera na podstawie jego nazwy

Podane poniżej Przykład 16-1 i Przykład 16-2 ilustrują sposób użycia mechanizmu GNS. Podane przykłady dotyczą komunikacji lokalnej. Aby uzyskać globalny zasięg nazw należy jako wartość zmiennej `flag` zamiast zera podstawić NAME_FLAG_ATTACH_GLOBAL. W tym przypadku jednak program serwera powinien posiadać przywilej administratora.

```
// ---------------------------------------------------------------------------
// QNX 6.3.2 - Neutrino                                (C) J. Ulasiewicz 2008
// Serwer - proces odbierajacy komunikaty, wysyla send.c
// ---------------------------------------------------------------------------
 #include <stdlib.h>
 #include <sys/neutrino.h>
 #include <sys/dispatch.h>
 #include <errno.h>
 #define SIZE 256
 #define MOJA_NAZWA "seweryn"

 struct  {
     int  type;        // typ komunikatu
     char text[SIZE];  // tekst  komunikatu
 } msg, rmsg;

 main(int argc, char *argv[]) {
  int  pid, con,fl,i,flag = 0;
  struct _msg_info info;
  name_attach_t  *nazwa_s;
  // flag =  NAME_FLAG_ATTACH_GLOBAL
  if((nazwa_s = name_attach(NULL,MOJA_NAZWA, flag)) == NULL) {
      perror("Rejestracja"); exit(1);
  }
  printf("Nazwa: %s zarejestrowana, kanal: %d\n",MOJA_NAZWA,nazwa_s->chid);
```

```
  // Kod serwera ----------------------------------------------------
  printf("Serwer startuje \n");
  for(i=0;; i++) {
         pid = MsgReceive(nazwa_s->chid,&msg,sizeof(msg),&info);
         if(pid == -1) { perror("receive");    continue;  }
         if(pid == 0)  { printf("Pulse \n");   continue;  }
         printf("Odebrane typ: %d text: %s \n",msg.type, msg.text);
         sprintf(msg.text,"potwierdzenie %d",i+1);
         MsgReply(pid,0,&msg,sizeof(msg));
  }
}
```

Przykład 16-1 Przesyłanie komunikatów z wykorzystaniem mechanizmu GNS – program serwera

```
//-----------------------------------------------------------------------------
//  QNX 6.3.2 - Neutrino                    (C) J. Ulasiewicz 2008
//  Proces msg_send1. wysylajacy komunikaty
//-----------------------------------------------------------------------------
 #include <stdlib.h>
 #include <sys/neutrino.h>
 #include <sys/dispatch.h>
 #include <errno.h>

 #define SIZE 256
 #define MOJA_NAZWA "seweryn"

 struct  {
    int  type;        // typ komunikatu
    char text[SIZE];  // tekst  komunikatu
 } msg, rmsg;

 main(int argc, char *argv[]) {
  int  i, fd, fl, flag = 0;
  char nazwa[40];
  struct _msg_info info;
  // flag =  NAME_FLAG_ATTACH_GLOBAL
  fd = name_open(MOJA_NAZWA, flag);
  if(fd  == -1) {
        perror("Lokalizacja"); exit(0);
  }
  printf("Lokalizacja ok fd: %d\n",fd);
  for (i=0; i < 5; i++) {
     sprintf(msg.text,"Komunikat: %d",i);
     printf("Wysylam: %s\n" , msg.text);
     if (MsgSend(fd, &msg, sizeof(msg), &rmsg, sizeof(rmsg)) == -1) {
           perror("send"); exit(0);
     }
     sleep(1);
     printf("Odebralem: %s\n" , rmsg.text);
   }
}
```

Przykład 16-2 Przesyłanie komunikatów z wykorzystaniem mechanizmu GNS – program klienta

16.4Zadania

Zadanie 16. 4. 1 Przesyłanie komunikatów pomiędzy procesem macierzystym i potomnym

Proces macierzysty o nazwie `komunikaty1` powinien:

1. Utworzyć kanał za pomocą funkcji `ChannelCreate()`.
2. Utworzyć zadaną liczbę procesów potomnych P1, P2,...,PN (np. za pomocą funkcji `fork()`). Liczba procesów potomnych jest równa liczbie parametrów podawanej z linii poleceń.
3. Odbierać i potwierdzać komunikaty od procesów potomnych.
4. Wykryć kiedy zakończą się procesy potomne.
5. Aby odebrać kody powrotu procesów potomnych wykonać stosowna liczbę funkcji `wait(&status)`. Pozwala to także na zlikwidowanie procesów „zombie”.

Proces potomny posiada swój numer (1,2,...) nadawany mu przez proces macierzysty. Proces potomny powinien wysyłać do procesu macierzystego komunikaty w formacie:

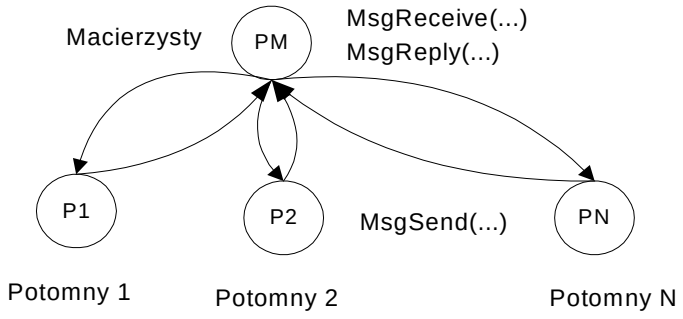
```
 struct {
        int  typ;             // Typ komunikatu
        int  od;              // Numer procesu
        char tekst[SIZE];     // Tekst komunikatu
 } kom_t;
```

Proces potomny powinien:

1. Utworzyć połączenie do procesu macierzystego za pomocą funkcji `ConnectAttach()`.

Wysyłać do procesu macierzystego komunikaty postaci:`"Komunikat: k od procesu i ", typ = 1`

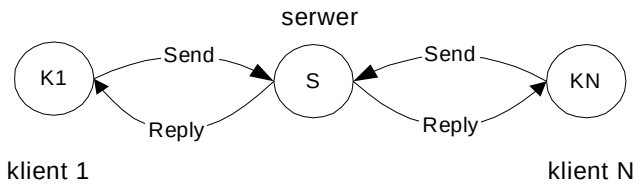
2. Po wysłaniu zadanej przez parametry programu liczby komunikatów wysłać do procesu macierzystego komunikat o zakończeniu (`typ = 0`).
3. Kończąc się powinien wykonać funkcje `exit(Numer_procesu)`



Rysunek 16-1 Przesyłanie komunikatów pomiędzy procesami zależnymi

Zadanie 16. 4. 2 Przesyłanie komunikatów pomiędzy niezależnymi procesami – zamiana małych liter na duże

W poprzednim ćwiczeniu zadania mogły się komunikować gdyż znane były identyfikatory procesów potomnych. Jest to jednak sytuacja nieczęsta. Na ogół procesy klienta potrzebują specjalnego mechanizmu do identyfikacji serwera. Napisz proces klienta, który tworzy połączenie do procesu serwera na podstawie danych (CHID,PID) zapisanych do pliku sieciowego. Klient następnie przesyła do serwera komunikaty zawierające wprowadzany z konsoli tekst.

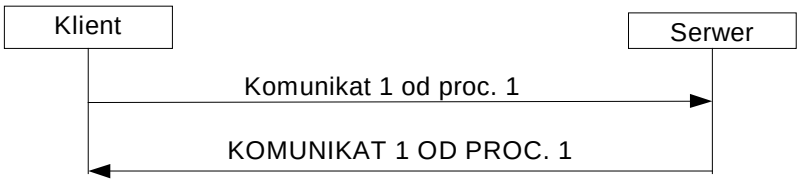


Rysunek 16-2 Klient i serwer są niezależnymi procesami

Serwer odbiera komunikaty wysyłane przez klientów i odsyła napisy otrzymane w polu `text` ale zamienia małe litery na duże. Procesy klienta i serwera uruchamiane są niezależnie z linii poleceń.

```
typedef struct  {
   int  typ;        // typ komunikatu
   int  from;       // nr procesu który wysłał komunikat
   int  ile;        // ile było malych liter
```

```
   char text[SIZE]; // tekst komunikatu
 } mss_t;
```



Rysunek 16-3 Współpraca klienta i serwera

Proces serwera

Serwer wykonuje następujące kroki:

1. Utworzenie kanału `chid = ChannelCreate(...)`.
2. Zapis do pliku info.txt `chid i pid.`
3. Odbiór zleceń klientów.
4. Odpowiedź na zlecenia klientów polegająca na zamianie małych liter na duże. W polu ile należy umieścić liczbę zamienionych liter.
5. Co 10 sekund serwer ma wyświetlać informację o liczbie otrzymanych dotychczas komunikatów.

Proces klienta

Proces klienta uruchamiany jest z parametrem: nazwa węzła na którym uruchomiony jest klient (np. `klient wezel`). Klient wykonuje następujące kroki:

1. Odczyt zdalnego pliku i pobranie z niego `chid i pid`
2. Pobranie z linii poleceń nazwy węzła na którym wykonuje się proces serwera i przekształcenie nazwy na numer węzła `nid` za pomocą funkcji `netmgr_strtond()`.
3. Utworzenie połączenia do procesu serwera za pomocą funkcji `ConnectAttach()`..
4. Wysyłanie komunikatów do serwera za pomocą funkcji `MsgSend()`. Pole `type` ma zawierać 1, pole `from` numer procesu, pole `text` łańcuch wprowadzany z konsoli
5. Odbiór i wyświetlanie odpowiedzi serwera.

Zadanie 16. 4. 3 Klient i serwer usługi FTP

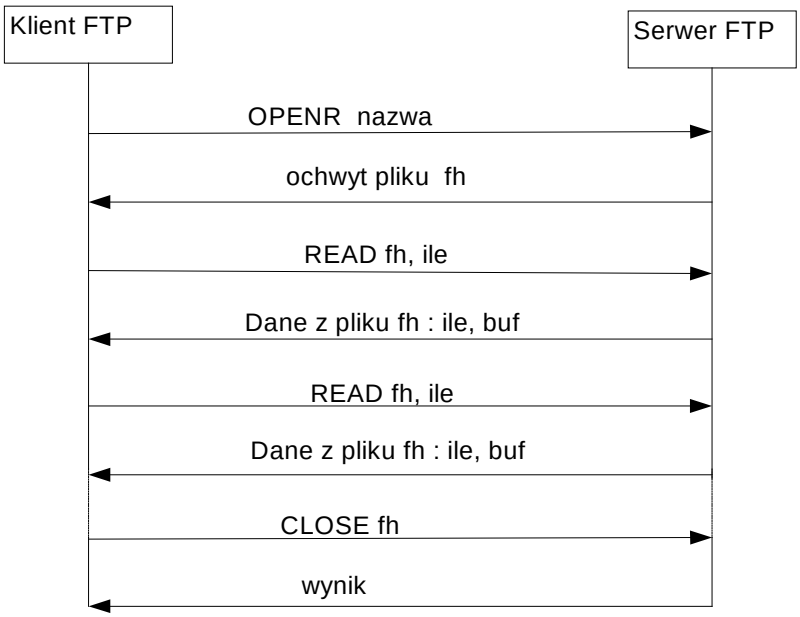
Wykorzystując usługę GNS napisz proces klienta i proces serwera realizujących przesyłanie plików. Serwer rejestruje swoją nazwę za pomocą funkcji `name_attach()`. Klient tworzy połączenie do serwera za pomocą funkcji `name_open()`. Od klienta do serwera przesyłane następujące rodzaje komunikatów:

```
#define OPENR  1  // Otwarcie pliku do odczytu
#define OPENW  2  // Otwarcie pliku do zapisu
#define READ   3  // Odczyt fragmentu pliku
#define CLOSE  4  // Zamkniecie pliku
#define WRITE  5  // Zapis fragmentu pliku
#define STOP  10  // Zatrzymanie serwera
```

Format komunikatu przesyłanego pomiędzy klientem a serwerem jest następujący:

```
#define SIZE = 512 bajtów.
typedef struct  {
    int  typ;       // typ zlecenia
    int  ile;       // liczba bajtow
    int  fh;        // uchwyt pliku
    char buf[SIZE]; // bufor
} mms;
```

Serwer odbiera komunikaty wysyłane przez klienta i realizuje je. W poleceniu OPENR klient żąda podania pliku którego nazwa umieszczona jest w polu `buf`. Serwer otwiera ten plik umieszczając jego uchwyt w polu fh. Następnie klient żąda podania porcji pliku fh w buforze buf w ilości ile = SIZE. Plik sprowadzany jest fragmentami o długości SIZE. W polu ile ma być umieszczona liczba przesyłanych bajtów. Klient może wykryć koniec pliku gdy liczba rzeczywiście przesłanych bajtów ile jest mniejsza od żądanej. Po zakończeniu przesyłania pliku klient wysyła polecenie CLOSE `fh`.



Rysunek 16-4 Współpraca klienta i serwera FTP

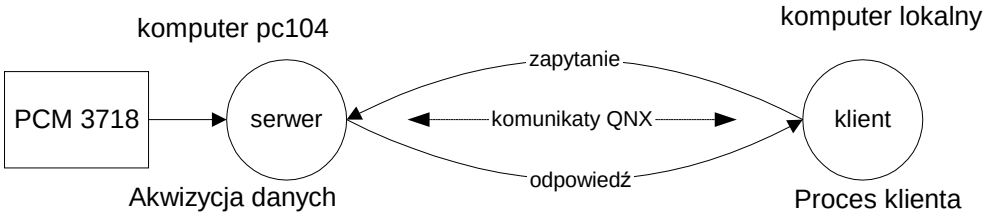
Procesy klienta i serwera uruchamiane są niezależnie z linii poleceń. Jako argumenty programów podajemy nazwę pod którą rejestruje się serwer. Przedstawiony wyżej serwer jest iteracyjnym serwerem stanowym.

Rozszerzenia:

- Dodaj funkcję zapisu pliku który przesyłany jest od klienta do serwera
- Dodaj funkcję podawania zawartości katalogu którego nazwa podawana jest przez klienta
- Dodaj funkcję zmiany katalogu bieżącego
- Zrealizuj serwer jako serwer współbieżny

17. Rozproszony system sterowania i akwizycji danych – komunikacja przez sieć QNET

Rozproszony system sterowanie czy akwizycji danych składa się z pewnej liczby stacji akwizycji danych (stacyjek sterowanie, rejestratorów, regulatorów, sterowników) połączonych z komputerami operatorskimi i serwerami. Jeżeli wszystkie lub niektóre z nich pracują pod kontrolą systemu QNX6 mogą się one łatwo komunikować poprzez sieć QNET. W takim przypadku urządzenia pomiarowe i wykonawcze są serwerami (odpowiadają na żądanie) a komputery operatorskie serwerami.



Rys. 17-1 Akwizycja danych z karty PCM3718 - komunikacja poprzez komunikaty w sieci QNET

17.1Specyfikacja komunikacji klient – serwer w sieci QNET

Na komputerze typu PC104 wyposażonym w kartę interfejsową PCM3718 wykonywany ma być program serwera obsługującego tę kartę. Serwer realizuje następujące funkcje:

- Odczyt wybranego kanału z przetwornika AD
- Zapis stanu wyjść cyfrowych
- Odczyt stanu wejść cyfrowych

Komunikacja z serwerem zachodzi za pośrednictwem następujących komunikatów w podanym niżej formacie.

```
#define ADREAD   1    // Odczyt z przetwornika AD
#define DIREAD   2    // Odczyt z wejsc cyfrowych DINP1
#define DOWRITE  4    // Zapis na wyjscia cyfrowe DOUT2

typedef struct {
       int typ;    // Typ polecenia
       int value;  // Wartosc
       int chan;   // Kanal
       int node;   // Numer wezla
       int result; // -1 gdy blad
       int debug;  // Gdy 1 wyswietlanie polecen
 } pcl_t;
```

Przykład 17-1 Format komunikatu aplikacji współpracy z kartą PCM3718

17.2 Zadania

Zadanie 17. 4. 1 Program klienta

Napisz program klienta odczytujący i wyświetlający na konsoli:

- Zawartość kanałów AD0 – AD3
- Stan wejść cyfrowych w postaci binarnej

Skorzystaj z gotowego serwera `pcm_serw` uruchomionego na jednym z komputerów typy PC104. Wywołanie programu serwera: `pcm_serw NazwaSerwera`. Wywołanie programu klienta: `pcm_cli NazwaSerwera`.

```
// Lokalizacja proces serwera poprzez wykonanie funkcji
pid = name_open(NazwaSerwera,FLAGA);
do {
      wysłać komunikaty ADREAD dla odczytu kanałów AD0 – AD3
      wyświetlić na konsoli kanały AD0 – AD3
      wysłać komunikaty DIREAD dla odczytu wejść cyfrowych
      wyświetlić na konsoli stan wejść cyfrowych
} while(true)
```

Przykład 17-2 Szkic programu klienta

Zadanie 17. 4. 2 Program serwera

Napisz własny program serwera obsługujący komunikaty z Przykład 15-3 . Wywołanie programu: `pcm_my_serw NazwaSerwera`. Przetestuj serwer z klientem opracowanym w poprzednim przykładzie. Zwróć uwagę na to że serwer może być uruchomiony tylko wtedy program ten ma EUID – 0.
Program serwera powinien pełnić następujące funkcje:

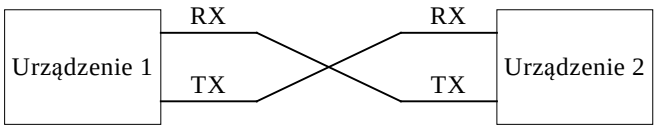
```
#define MAX 3
int tab[MAX+1]; // Tablica z aktualnymi wartościami kanałów AD

// Rejestracja procesu serwera
nazwa_s = name_attach(NULL,NazwaSerwera,FLAGA);
...
// Inicjacja układów karty
card_init(0,MAX,5);

// Utworzenie polaczenia -----------------------------
coid = ConnectAttach(0,0,nazwa_s->chid,0,0);
priority = getprio(0);

// Inicjacja zdarzenia --------------------------
SIGEV_PULSE_INIT(&event,coid,priority,1,0);

// Utworzenie timera --------------------
id = timer_create(CLOCK_REALTIME,&event,&timid);
// Ustawienie timera --------------------
timer.it_value.tv_sec     = 1;
timer.it_value.tv_nsec    = 0L;
timer.it_interval.tv_sec  = 0;
timer.it_interval.tv_nsec = 100000000L;
timer_settime(timid,0,&timer,NULL);
do {
  pid = MsgReceive(nazwa_s->chid,&msg,sizeof(msg),&info);
  if(pid == 0) {
         // Odczyt kanalów AD
         val = aread(&chan);
         tab[chan] = val;
         if(chan == MAX) { // odczyt DI
            x1 = dinp(1);  ;
         }
   }
   // Obsluga zlecen klientow
   switch(msg.typ) {
        case ADREAD:   // Odczyt kanalu analogowego ---------
                       ...
                       msg.value = tab[msg.chan];
                       break;
        case DIREAD:   msg.value = dinp(1);
                       break;
        case DOWRITE: dout(2,msg.value);
                       break;
        default:       msg.result = -1;
      }  // switch

      // Odpowiedz --------------
      MsgReply(pid,0,&msg,sizeof(msg));

} while(true)
```

Przykład 17-3 Szkic programu serwera

18. Komunikacja szeregowa – interfejs RS-232C, protokół MODBUS

18.1 Podstawy obsługi interfejsu szeregowego

Interfejs RS232C jest często wykorzystywany w automatyce. Za jego pośrednictwem łączy się z komputerem takie urządzenia jak regulatory, przyrządy pomiarowe, wyświetlacze, czytniki mediów, drukarki i inne urządzenia. Standard RS232C jest się punktem wyścia dla całej rodziny bardziej zaawansowanych protokołów przemysłowych (standard RS485, PROFIBUS). Dane przekazywane są szeregowo po dwóch liniach logicznych. Są to dane odbierane oznaczane jako RX (ang. *Received*) i dane wysyłane oznaczane jako TX (*ang. Transmitted*). Aby dwa urządzenia mogły się skomunikować linia RX jednego urządzenia należy połączyć się z linią TX drugiego co pokazuje Rys. 18-1.



Rys. 18-1 Dwa urządzenia połączone interfejsem RS232C

Przy pomocy połączenia w standardzie RS232C połączyć można tyko dwa urządzenia chociaż będący jego rozwinięciem standard RS485 pozwala na połączenie wielu stacji. Połączenie wykonuje się specjalnym kablem nazywanym null modem. Transmisja pomiędzy urządzeniami może być jednokierunkowa lub dwukierunkowa i prowadzona jest w trybie asynchronicznym.

Porty transmisji szeregowej reprezentowane są jako pliki specjalne `ser1` i `ser2` widziane w katalogu `/dev`. Tak więc programy mogą się odwoływać do tych portów poprzez nazwy `/dev/ser1` i `/dev/ser2`. Przykładowo można wyprowadzić na konsolę zawartość portu szeregowego poleceniem `cat /dev/ser1` lub wyprowadzić na port plik używając polecenia `cat plik.txt > /dev/ser1`. Systemowy program `stty` umożliwia ustalanie i testowanie parametrów transmisji portu szeregowego. Bieżące ustawienia portu szeregowego `ser1` uzyskuje się za pomocą polecenia:

```
$stty < /dev/ser1
```

Ustawianie parametrów portu szeregowego odbywa się za pomocą polecenia:

```
$stty [operandy] > /dev/ser1
```

Ważniejsze operandy podaje poniższa tabela.

Operand	Znaczenie	Wartości parametrów
`baud`	Szybkość transmisji	1 do 115200
`par`	Parzystość	o - nieparzystość e - parzystość n – brak bitu parzystości
`bits`	Liczba bitów w znaku	5,6,7,8
`stopb`	Liczba bitów stopu	1 lub 2

Tabela 18-1 Niektóre parametry polecenia `stty`

Przykładowo aby dla portu 1 ustawić szybkość transmisji na 2400, bit parzystości na parzystość, liczbę bitów w znaku na 8 i jeden bit stopu należy wykonać poniższe polecenie:

```
$ stty baud=2400 par=e  bits=8 stopb=1 > /dev/ser1
```

Porty transmisji szeregowej widziane są jako znakowe pliki specjalne `ser1` i `ser2` obecne w katalogu `/dev`. Programy mogą się odwoływać do tych portów, traktując je jak pliki na których można przeprowadzać operacje czytania i pisania. Zestawienie ważniejszych funkcji dotyczących osługi portów szeregowych podaje poniższa tabela.

Funkcja	Opis
open	Otwarcie portu
read	Odczyt z portu szeregowego
write	Zapis na port szeregowy
close	Zamknięcie portu
tcgetattr	Odczyt parametrów terminala
tcsetattr	Zapis parametrów terminala
cfsetispeed	Ustawianie prędkości transmisji - wejście
cfsetospeed	Ustawianie prędkości transmisji - wyjście
tcischars	Testowanie liczby znaków w buforze
tcflush	Kasowanie zawartości bufora
tcdrain	Oczekiwanie na zapis bufora
readcond	Zaawansowany odbiór znaków

Tabela 18-2 Ważniejsze funkcje używane w interfejsie gniazdek – komunikacja bezpołączeniowa

Poniżej podane są przykładowe programy do odbioru (`ser_odb.c`) i wysyłania znaków (`ser_nad.c`). Skompiluj te programu a następnie uruchom na dwóch komputerach w których porty RS232 /dev/ser1 połączone są kablem typu NULL modem.

```
// Uruchomienie:  ser_nad /dev/ser1
#include <stdlib.h>
#include <fcntl.h>
#include <stdio.h>
#define SIZE 80

main(int argc, char *argv[]) {
  char bufor[SIZE];
  int fd,i,res;
  if(argc < 2) {
      printf("uzycie: ser_nadl /dev/serx \n");
      exit(0);
  }
  printf("Port: %s\n",argv[1]);
  fd = open(argv[1],O_RDWR);
  for(i=0;i<10;i++) {
    sprintf(bufor,"Krok %2d\n",i);
    res = write(fd,bufor,strlen(bufor));
    printf("%s  wyslano %d\n",bufor,res);
    sleep(1);
  }
  close(fd);
}
```

Przykład 18-1 Proces wysyłający znaki na port szeregowy – `ser_nad.c`

```
// Proces odbiera znaki z portu szeregowego (C) J. Ulasiewicz 2011
// wysyla ser_nad na innym komputerze
// Uruchonienie:  ser_odb  /dev/ser1
#include <stdlib.h>
#include <fcntl.h>
#include <stdio.h>
#define SIZE 80

main(int argc, char *argv[]) {
  char bufor[SIZE];
  int fd,i,res;
  if(argc < 2) {
      printf("uzycie: ser_nadl /dev/serx \n");
      exit(0);
  }
```

```
  printf("Port: %s\n",argv[1]);
  fd = open(argv[1],O_RDWR);
  for(i=0;i<10;i++) {
    res = read(fd,bufor,8);
    bufor[res] = 0;
    printf("%s",bufor);
    sleep(1);
  }
  close(fd);
}
```

Przykład 18-2 Proces odbierający znaki z portu szeregowego - `ser_odb.c`

18.2 Zadania

Zadanie 18. 4. 1 Testowanie połączenia szeregowego przy pomocy programu terminalowego, kontrola szybkości transmisji

Na dwóch komputerach w których porty szeregowe /dev/ser1 lub /dev/ser2 połączone są kablem typu NULL modem uruchom w oknach terminala program qtalk (polecenie: `qtalk –m /dev/ser`$_i$). Pisząc w oknach terminala obserwuj przebieg komunikacji. Zapisz przebieg komunikacji w pliku. Zmień parametry transmisji za pomocą polecenia stty i przetestuj poprawność komunikacji.

Zadanie 18. 4. 2 Testowanie przykładowych programów komunikacji szeregowej

Uruchom i przetestuj podane wyżej programy komunikacji szeregowej. Najpierw uruchom każdy z programów wykorzystując do testowania program terminalowy `qtalk`. Następnie przetestuj komunikację pomiędzy programem `ser_nad` i `ser_odb`.

Zadanie 18. 4. 3 Implementacja protokołu MODBUS w trybie ASCII

Opracowany w firmie Modicon protokół MODBUS jest popularnym protokołem używanym do komunikacji ze sterownikami. Ramka wersji ASCII tego protokołu dana jest poniżej a opisana w [11] , [12]. Opracuj implementujące protokół funkcje `SendFrm` i `RecFrm` oraz opracuj program testowy wysyłający i odbierający ramki tego protokołu.

Znacznik początku ramki	Adres	Funkcja	Dane	LRC	Znacznik końca
1 znak :	2 znaki HEX	2 znaki HEX	n znaków w postaci HEX	2 znaki HEX	2 znaki CR LF

Tabela 18-3 Struktura ramki protokołu MODBUS ASCII

Wykorzystaj dany niżej szkic programu testowego. W programie tym należy dopisać funkcje:

```
int SendFrm(int dev, int Adr, int Command, int Size, char *Data)
```

`dev`	Uchwyt pliku portu szeregowego
`Adr`	Adres docelowy
`Command`	Numer polecenia
`Size`	Długość w bajtach pola danych
`Data`	Adres pola danych

Funkcja zwraca liczbę wysłanych znaków lub –1 gdy błąd. Funkcja ma wysyłać ramkę MODBUS z polami jak powyżej. Aby prawidłowo wysłać ramkę należy:

1. Zbudować ramkę binarną zawierająca pola: znacznik początku, adres, polecenie, dane
2. Obliczyć LRC dla ramki binarnej i dodać tę wartość na końcu (po danych)
3. Zamienić ramkę binarną na postać HEX – ilość znaków się podwoi.
4. Dodać na końcu znaki CR (0xD) i LF (0xA)
5. Wysłać znaki na port szeregowy

```
int RecFrm(int dev, int *Adr, int *Command, int *Size, char *Data)
```

`dev` Uchwyt pliku portu szeregowego
`Adr` Adres docelowy
`Command` Numer polecenia
`Size` Długość w bajtach pola danych
`Data` Adres pola danych

Funkcja zwraca liczbę odebranych znaków lub –1 gdy błąd. Funkcja ma wysyłać ramkę MODBUS z polami jak powyżej. Aby prawidłowo wysłać ramkę należy:

1. Czekać na znacznik początku ramki. Gdy nie pojawi się w czasie T funkcja powinna zwrócić –1.
2. Odbierać znaki do bufora aż: nie pojawi się znak LF, zostanie przekroczony timeout T lub liczba odebranych znaków będzie większa od długości bufora. Gdy nie pojawi się w czasie T znak końca ramki funkcja zwraca –1.
3. Zamienić znaki HEX na znaki binarne i skopiować je do bufora Data.
4. Obliczyć LRC – w wyniku ma być 0.
5. Określić wartości pól Adr i Size
6. Zwrócić liczbę odebranych znaków.

Do odbioru ramki można wykorzystać funkcję warunkowego odbioru znaków `readcond`.

Zadanie 18. 4. 4 Odbiór ramek protokołu MODBUS w trybie RTU

W protokole MODBUS RTU separatorem pomiędzy ramkami są przerwy w transmisji o długości 4 znaków.

Znacznik początku ramki	Adres	Funkcja	Dane	CRC	Znacznik końca
T1-T2-T3-T4	8 bitów	8 bitów	n x 8 bitów	16 bitów	T1-T2-T3-T4

Tabela 18-4 Struktura ramki protokołu MODBUS RTU

Podobnie jak w poprzednim zadaniu opracuj implementujące protokół funkcje `SendFrm` i `RecFrm` oraz opracuj program testowy wysyłający i odbierający ramki tego protokołu.

Zadanie 18. 4. 5 Wykorzystanie protokołu MODBUS do komunikacji ze sterownikiem typu PC104 zawierającym kartę akwizycji danych PCM3718

Wykorzystaj opracowane w poprzednim zadaniu funkcje `SendFrm` i `RecFrm` do odczytu kanału wejść analogowych AN0. Kanał ma być odczytywany przy pomocy danej niżej funkcji 4 protokołu aplikacyjnego.

Polecenie:

	Długość	Wartość
Kod funkcji	1 bajt	0x04
Adres startowy bloku	2 bajty	0x0000 do 0xFFFF
Liczba rejestrów	2 bajty	0x0001 do 0x007D

Odpowiedź:

	Długość	Wartość
Kod funkcji	1 bajt	0x04
Liczba bajtów	1 bajt	2 * N
Zawartość rejestrów	2 * N bajtów	

N – liczba rejestrów

Żądanie		Odpowiedź	
Nazwa pola	Zawartość	Nazwa pola	Zawartość
Funkcja	0x04	Funkcja	0x04
Adres – Hi	0x00	Liczba bajtów – Hi	0x02
Adres – Lo	0x00	Zawartość rej. – Hi	0x00
Liczba rejestrów – Hi	0x00	Zawartość rej. – Lo	0x0A
Liczba rejestrów – Lo	0x01		

Przykład 18-3 Odczyt jednego rejestru o adresie 0, zawartość rejestru 0x000A

Wykorzystaj dany niżej fragment kodu. W pierwszym etapie uruchamiania programu można odczyt kanałów przetwornika AD zasymulować zmieniając w kolejnych krokach wartość zmiennej val.

```
char Data[DataSize]; /* bufor roboczy  0  */
// Przykladowa odpowiedz na funkcje 4, wartosc danych 1
//                : 01 04 02  00 01 F8 <CR> <NL>
char odp4[] = {0x3A, 0x30, 0x31, 0x30, 0x34, 0x30, 0x32, 0x30, 0x30, 0x30, 0x31,
               0x46, 0x38, 0x0D, 0x0A};

unsigned char lrcgen(char *buf, int size)
/* Wyznaczanie LCR dla bufora buf i zawierajacego size znakow */
{
 unsigned char sum = 0;
 int i;
 for(i=0;i<size;i++) sum = sum + buf[i];
 return(~sum + 1);
}

int main(int argc, char * argv[]) {
  int fd,speed, timeout,rd,res,i;
  int Adr,Cmd,Size;
  short int val = 0;
  struct termios term;
  char c;
  fd = open(PORT_COM1,O_RDWR);
  /* ustawienia predkosci transmisji */
  speed=9600;
  timeout = 20; // 2 sek
  tcgetattr(fd,&term);
  term.c_cc[VFWD] = LF;
  tcsetattr( fd, TCSANOW, &term);
  tcflush(fd,TCIFLUSH);
  tcflush(fd,TCOFLUSH);
  do {
      res = RecFrm(fd,&Adr,&Cmd,&Size,Data);
      if(res == 0)  {
          printf("Timeout\n");
         continue;
      }   Data[Size] = 0;
      printf("Adres: %d polecenie: %d rozmiar: %d %s \n",Adr,Cmd,Size,Data);
      switch(Cmd) {
      case 4: // Odczyt kanalu DA ----------
                  val++;
                  Data[0] = 0x02; // Liczba bajtow
                  Data[1] = val >>8;
                  Data[2] = val & 0x00FF;
                  printf("Dane: %02X %02X \n",Data[1],Data[2]);
                  Size = 3;
                  break;
     default: // Nie zaimplementowane --
                 printf("Funkcja nie zaimplementowana\n");
     }
     SendFrm(fd,Adr,Cmd,Size,Data);
  } while(1);
  close(fd);
  return 0;
}
```

Przykład 18-4 Program typu SLAVE do odbioru ramek MODBUS ASCII (fragment)

19. Dodatek 1 - QNX6 Neutrino, konsola na porcie szeregowym

W celu instalacji konsoli na porcie szeregowym należy:

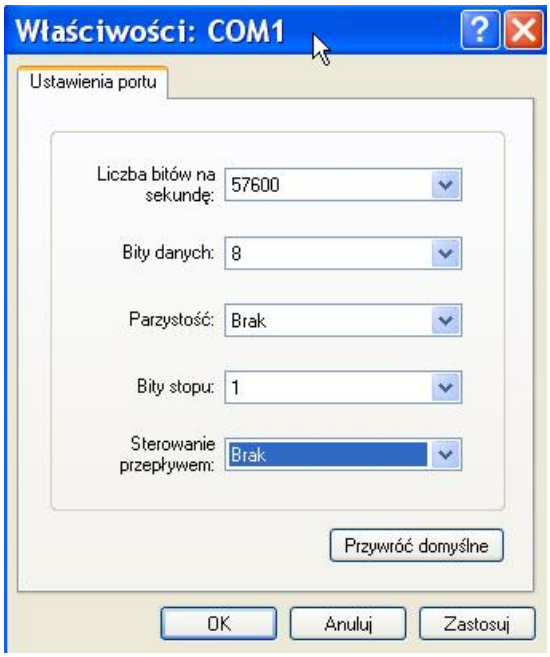
1. Upewnić się że pewien port szeregowy, np. `/dev/ser1`, jest aktywny. Można to sprawdzić za pomocą testera
2. Poddać edycji plik `/etc/config/ttys`. Dla konsoli na porcie ser1 Ma on wyglądać następująco:

```
con1 "/bin/login" qansi-m on
con2 "/bin/login" qansi-m on
con3 "/bin/login" qansi-m on
ser1 "/bin/login" qansi-m on
```

3. W systemie QNX6 Sprawdzić parametry portu szeregowego:

```
# stty < /dev/ser1
Name:  /dev/ser1
Type:  serial +ihflow +ohflow
...
par=none bits=8 stopb=1 baud=57600 rows=0,0
```

4. Następnie należy zrestartować komputer wbudowany
5. Połączyć port szeregowy ser1 komputera wbudowanego kablem null modem z portem szeregowym innego komputera np. w systemie Windows.
6. Uruchomić tam program terminalowy, np. Hiperterminal. Podać parametry transmisji



Ekran 19-1Po wprowadzeniu parametrów transmisji powinien zgłosić się shell

20. Dodatek 2 - Operacje na bitach w języku C

W języku C zdefiniowane są operacje na bitach. Operacje dwuargumentowe:

- koniunkcja bitowa ("&"),
- alternatywa bitowa ("|") i
- alternatywa rozłączna (XOR)

Operacje jednoargumentowe:

- negacja bitowa ("~"),
- przesunięcie bitowe

bit a	bit b	a&b
0	0	0
0	1	0
1	0	0
1	1	1

bitowe AND &

bit a	bit b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

bitowe OR |

bit a	bit b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

bitowe XOR ^

Przykład bitowego AND

```
  11001110
 &10011000
= 10001000
```

Przykład bitowego OR

```
 11001110
|10011000
=10001000
```

Przykład bitowego XOR

```
 11001110
^10011000
=01010110
```

bit a	~a
0	1
1	0

Negacja bitowa

```
~11001110
 00110001
```

<u>Przesunięcie w prawo >></u>
Przesuniecie w prawo [zmienna] >> [liczba miejsc] jest operacją dwuargumentową A >> n. Przesuwa ona bity operandu A o n pozycji w prawo. Na miejsce najstarszych bitów wchodzi 0
Gdy x=11100101 to x >> 1 daje w wyniku 01110010, x >> 2 daje w wyniku 00111001

<u>Przesunięcie w lewo <<</u>
Przesuniecie w lewo [zmienna] << [liczba miejsc] jest operacją dwuargumentową A << n. Przesuwa ona bity operandu A o n pozycji w lewo. Na miejsce najmłodszych bitów wchodzi 0
Gdy x=11100101 to x << 1 daje w wyniku 11001010, x << 2 daje w wyniku 10010100

```
#include <stdio.h>
 void pokaz_bity(unsigned int x){
        int i;
        for(i=(sizeof(int)*8)-1; i>=0; i--)
            (x&(1<<i))?putchar('1'):putchar('0');
      printf("\n");
 }

 int main(int argc,char *argv[]) {
        int j, m, n;
        j = atoi(argv[1]);
        printf("Dziesietne: %d hex: %08X binarnie - ",j, j);
        pokaz_bity(j);
        return 0;
 }
```

Przykład 20-1 Wypisywanie liczby w postaci HEX i binarnej

Literatura

[1] K. Haviland, D. Gray, B. Salama; UNIX Programowanie systemowe, RM Warszawa 1999.
[1] PCM-3718 Series PC/104 12-bit DAS Module with Programmable Gain User Manual. Advantech 2010 http://origindownload.advantech.com//productFile/1-GF2P4A/PCM-3718_Manual_V4.pdf
[2] Matthew N. Stones R. Linux Programowanie,Wyd. RM Warszawa 1999.
[3] QNX Neutrino RTOS V6.3, System Architecture, QNX Software Systems , Canada 2004.
[4] QNX Neutrino RTOS V6.3, Programmes Guide, QNX Software Systems , Canada 2004.
[5] QNX Neutrino RTOS V6.3, Users Guide for rel. 6.3, QNX Software Systems , Canada 2004.
[6] QNX Neutrino RTOS V6.3, Library Reference for rel. 6.3, QNX Software Systems , Canada 2004.
[7] QNX Neutrino RTOS V6.3, Utilities Reference for rel. 6.3, QNX Software Systems, Canada 2004.
[8] J. Ułasiewicz, Systemy czasu rzeczywistego QNX6 Neutrino, wyd. BTC Warszawa 2007
[9] Kernigan B, Ritchie D. Język ANSI C, WNT Warszawa 2002.
[10] Stevens Richard W. ,Programowanie zastosowań sieciowych w systemie UNIX, WNT Warszawa 1996.
[11] Mielczarek Wojciech, Szeregowe interfejsy cyfrowe, HELION 1993.
[12] Modbus protocol specification, witryna org. Modbus IDA, http://www.modbus.org.
[13] The Gnu make manual, http://www.gnu.org/software/make/manual/make.html
[14] Matthew N. Stones R. Linux Programowanie,Wyd. RM Warszawa 1999.
[15] GDB: The GNU Project Debugger, http://www.gnu.org/software/gdb/
[16] Opis czujnika HC-SR04 http://www.micropik.com/PDF/HCSR04.pdf

Doc. dr inż. Jędrzej Ułasiewicz

Katedra Informatyki Technicznej, Wydział Elektroniki Politechniki Wrocławskiej
ul. Janiszewskiego 11/17
50-372 Wrocław

Niniejszy raport otrzymują		Egz.
1	CWINT	1
2	Kierownik Katedry	1
4	Autor	4
RAZEM		6