2 Tworzenie oprogramowania dla systemu wbudowanego

2.1 Przenośność programów

Standardowe narzędzia wytwarzania oprogramowania (ang. *toolchain*) dostępne w określonej dystrybucji Linuksa, są wykonywane na danym komputerze i generują kod na ten sam procesor który zainstalowany jest w komputerze, zwykle o architekturze x86.

Taki zbiór narzędzi nazywa się rodzimym systemem tworzenia oprogramowania (ang. *native*). Wytworzony w systemie rodzimym kod binarny, zwykle nie wykona się w systemie docelowym z następujących powodów:

- System docelowy może mieć inny procesor niż komputer macierzysty. Np. komputer macierzysty wykorzystuje procesor x86 a system docelowy ARM.
- System docelowy może mieć inny system operacyjny niż komputer macierzysty.

Nawet jeżeli, obydwa systemy wykorzystują takie same procesory i pracują pod kontrolą takiego samego systemu operacyjnego, tworzenie aplikacji nie musi przebiegać tak samo, gdyż:

- program dla systemu docelowego może być optymalizowany w inny sposób
- wykorzystywać inne urządzenia i biblioteki.

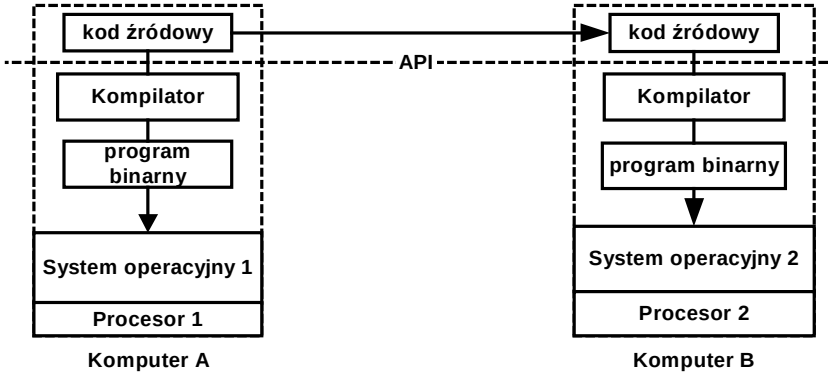
Przenośność oprogramowania

Jakie warunki należy spełnić, aby program wytworzony na komputerze i dla komputera A wykonał się na komputerze B (inny procesor, system operacyjny).

Literatura wyróżnia następujące poziomy przenośności oprogramowania:

- Przenośność na poziomie kodu źródłowego
- Przenośność na poziomie kodu pośredniego
- Przenośność na poziomie kodu wynikowego

Przenośność programu na poziomie kodu źródłowego
Aby wykonać program przygotowany dla maszyny A np. w języku C, należy skompilować powtórnie na maszynie B.



Rys. 2-1 Zgodność na poziomie kodu źródłowego API

Aby program uruchomił się na maszynie B, kompilator maszyny B, musi właściwie zinterpretować, wywołania systemowe programu. Na przykład funkcja open, w obydwu systemach, musi w taki sam sposób interpretować parametry i się wykonywać.

API (ang. Application Program Interface) - sposób komunikacji z systemem operacyjnym – zbiór funkcji języka C.

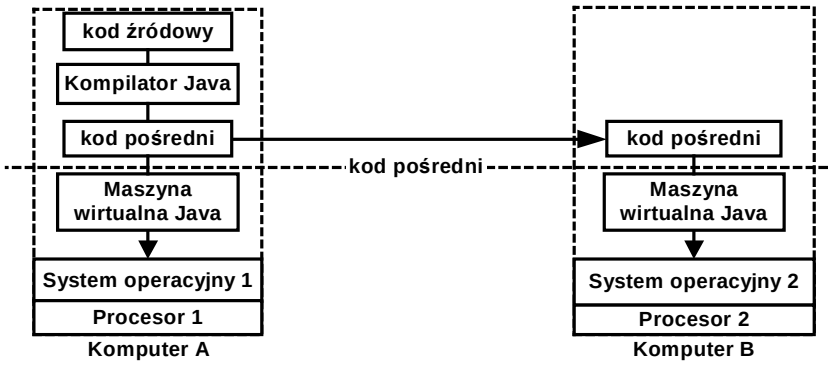
Zgodność na poziomie API, wymaga rekompilacji programów.

Aby wystąpiła zgodność na poziomie API

- Maszyna A i maszyna B mogą mieć inne procesory
- Systemy operacyjne, muszą być zgodne na poziome API (nie koniecznie identyczne).

Przenośność na poziomie kodu pośredniego

Kod pośredni (ang. *Byte Code*) stanowi fundament języka Java. Program dla maszyny A kompiluje się tam do postaci kodu pośredniego, który może być bez rekompilacji wykonany przez maszynę wirtualną Javy uruchomioną na komputerze B.



Rys. 2-2 Zgodność na poziomie kodu pośredniego

Warunek zgodności:

- Komputer A i komputer B mogą posiadać różne procesory i systemy operacyjne,
- Wymagana jest zgodność na poziomie kodu pośredniego.

Maszyny wirtualne komputerów A i B muszą w ten sam sposób interpretować kod pośredni.

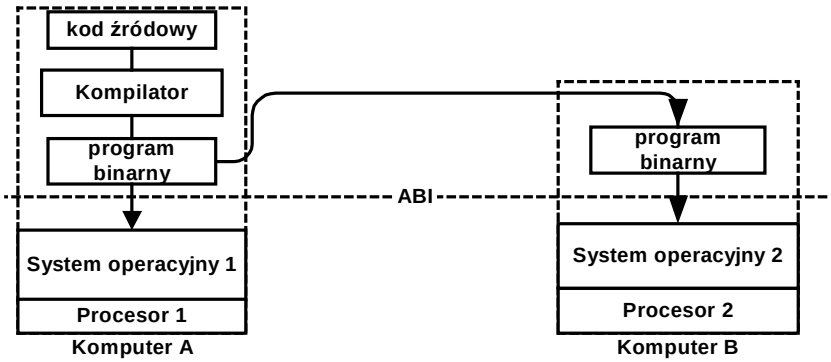
Przenośność na poziomie kodu wynikowego (ABI – *Application Binary Interface*)

Aby program skompilowany na maszynie A wykonał się na maszynie B maszyny A i B muszą co najmniej:

- Posiadać zgodną architekturę
- Posiadać taką sama listę rozkazów

Ale spełnienie powyższych warunków nie jest wystarczające. Wymagane jest spełnienie dodatkowych wymagań, określanych jako zgodność na poziomie interfejsu binarnego ABI (ang. *Application Binary Interface*). Zgodność na poziomie ABI obejmuje:

- Postać interfejsu binarnego pomiędzy aplikacją a systemem operacyjnym
- Typy, rozmiary i sposób interpretacji i przekazywania parametrów (big endian, little endian)
- Numery wywołań systemowych
- Sposób korzystania z bibliotek
- Sposób reagowania na błędy
- Formatu pliku wykonywalnego (zwykle ELF)



Rys. 2-3 Zgodność na poziomie interfejsu binarnego ABI

☞ **Przenośność na poziomie kodu binarnego ABI (*Application Binary Interface*)**

Przenośność na poziomie kodu binarnego ma miejsce, gdy program źródłowy przygotowany i skompilowany dla maszyny A daje się bez rekompilacji wykonać na maszynie B.

	Zgodność na poziomie kodu źródłowego API	Zgodność na poziomie kodu pośredniego	Zgodność na poziomie kodu binarnego ABI
Maszyna wirtualna komputera A i B	-	Zgodne	-
System operacyjny komputera A i B	Zgodne	Różne	Zgodne
Procesor komputera A i B	Różne	Różne	Zgodne

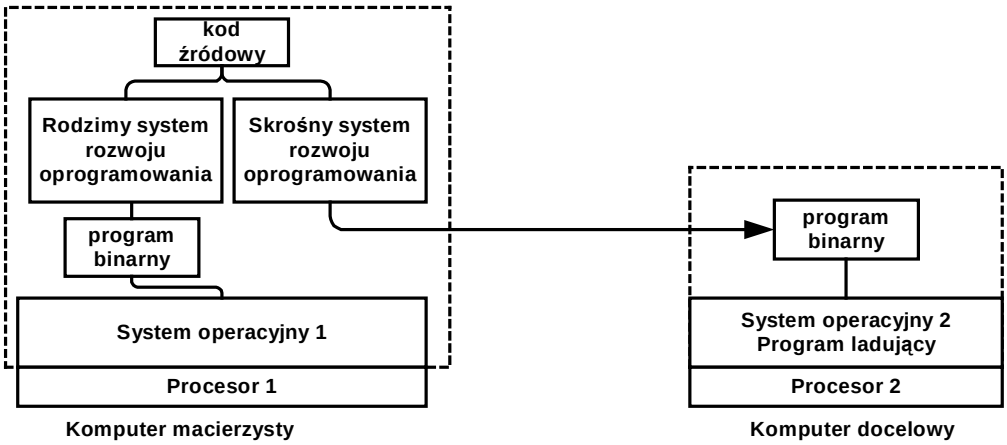
Tabela 2-1 Zestawienie różnych rodzajów zgodności programów

2.2 Systemy skrośnego tworzenia oprogramowania - podstawowe konfiguracje

Oprogramowanie dla systemów wbudowanych zazwyczaj tworzy się w taki sposób że, programy przygotowywane są na komputerze macierzystym a potem przenoszone na system wbudowany nazywany też docelowym (ang. *target*).

Takie postępowanie nazywa się skrośnym tworzeniem oprogramowania (ang. *host target development, cross-compiling toolchain*)

System skrośnego tworzenia oprogramowania
Programy tworzone są na komputerze macierzystym, przenoszone są do komputera wbudowanego i tam wykonywane.



Rys. 2-4 Koncepcja rodzimego i skrośnego systemu tworzenia oprogramowania

Zazwyczaj nie tworzy się oprogramowania na systemie docelowym gdyż:

- Komputer docelowy zwykle ma za małe zasoby (pamięć operacyjna, masowa) by pomieścić system tworzenia oprogramowania.
- Jest on do tego celu zbyt wolny.
- Nie posiada wygodnych dla programisty interfejsów (monitor, klawiatura, myszka).

Można wyróżnić trzy zasadnicze metody tworzenia oprogramowania dla systemów wbudowanych:.

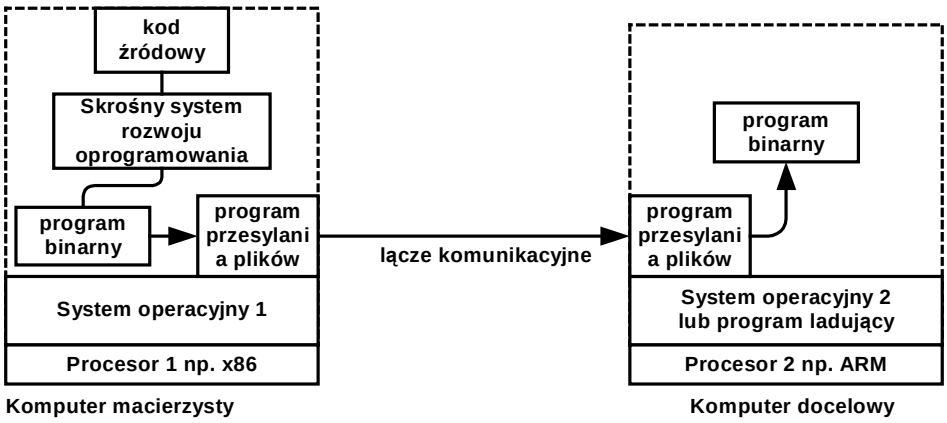
1. Oprogramowanie tworzone jest na komputerze macierzystym i przenoszone do systemu docelowego przez łącze komunikacyjne.
2. Oprogramowanie tworzone jest na komputerze macierzystym i zapisywane na jakimś nośniku w pamięci nieulotnej. Następnie nośnik z zapisanym programem przenoszony jest fizycznie do systemu docelowego.
3. Oprogramowanie w całości tworzone jest na systemie docelowym.

Metoda 1

Oprogramowanie tworzone jest na komputerze macierzystym i przenoszone do systemu docelowego

Komputer macierzysty musi posiadać zainstalowany system tworzenia oprogramowania dla systemu docelowego.

Systemem operacyjnym - Linux lub Windows, a procesorem x86.



Rys. 2-5 Program przenoszony przez łącze komunikacyjne

Warunek zastosowania - komputer wbudowany musi być zdolny do odebrania programu binarnego i jego wykonania.

Musi tam być zainstalowany system operacyjny albo co najmniej zaawansowany program ładujący (ang. *bootloader*) np. uboot .

System docelowy musi być w stanie:
- odebrać plik z programem binarnym
- obsłużyć interfejs komunikacyjny,
- obsłużyć protokół komunikacyjny,
- zarządzać pamięcią,
- sygnalizować błędy.

Łączem komunikacyjnym jest zazwyczaj:
- Ethernet
- USB
- RS232
- JTAG.

Protokołem komunikacyjnym może być FTP, SCP, SFTP albo też TFTP. Gdy stosujemy połączenie poprzez interfejs RS232 może to być Xmodem, Zmodem, Kermit.

Warunek zastosowania - obecność w systemie docelowym programu ładującego zdolnego pobrać program z komputera macierzystego i go uruchomić.

Metoda 2

Oprogramowanie tworzone jest na komputerze macierzystym i zapisywane na nośniku w pamięci nieulotnej. Następnie nośnik z zapisanym programem przenoszony jest fizycznie do systemu docelowego.

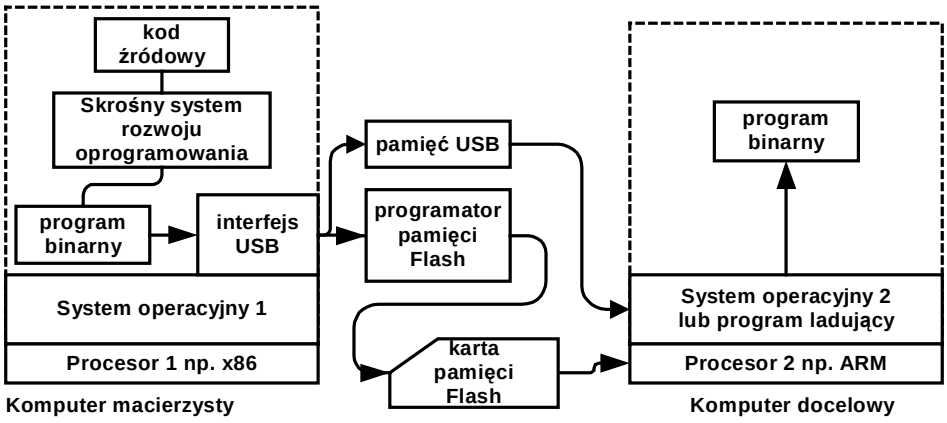
Nosnik - pamięć Flash w postaci karty Compact Flash, karty SD czy karty mikro SD.

Programator kart pamięci zazwyczaj przyłączony jest do komputera poprzez interfejs USB.

System docelowy musi posiadać zdolność odczytu takiej karty a więc musi zawierać co najmniej program ładujący.

Rozwiązanie to jest stosowane w przypadkach gdy:

- uruchamiamy system operacyjny
- aplikacja ma być wykonywana w środowisku pozbawionego systemu operacyjnego



Rys. 2-6 Program przenoszony przez pamięć Flash lub dysk USB

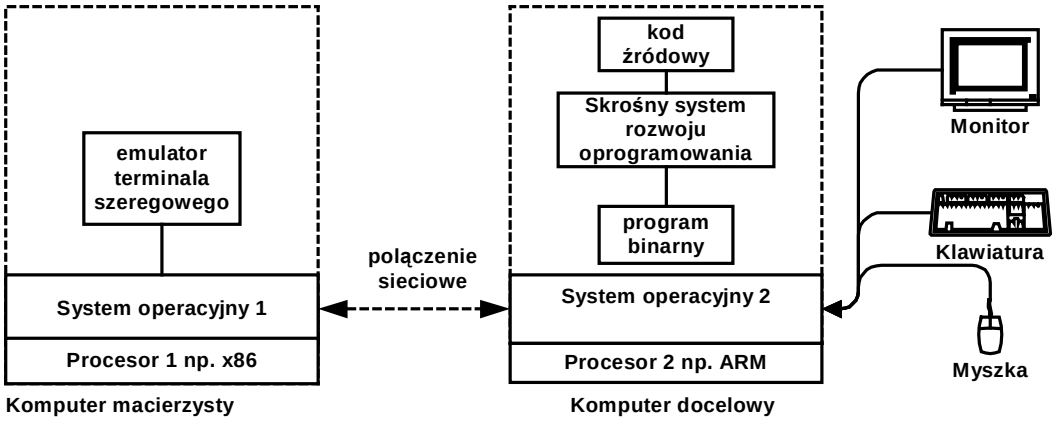
Metoda 3
Oprogramowanie tworzone jest na komputerze docelowym.

Komputer musi być wyposażony w:
- działający system operacyjny,
- urządzenie pamięciowe mieszczące system plików
- stosunkowo bogaty zestaw interfejsów.

Natomiast nie zawiera on zwykle: monitora, klawiatury i myszki. Dołącza się je poprzez zawarte w urządzeniu interfejsy (HDMI, VGA, USB).

Można się komunikować z urządzeniem poprzez komputer macierzysty z wykorzystaniem emulatora terminala szeregowego.

Rozwiązanie stosuje się w: BeagleBoard, BeagleBone Black, Raspberry Pi, komputerach z magistralą PC104



Rys. 2-7 System tworzenia oprogramowania wykonywany na komputerze docelowym

Rodzaj konfiguracji	Zastosowanie	Przykłady
Program przenoszony przez łącze komunikacyjne	Tworzenie programów aplikacyjnych na system docelowy z systemem operacyjnym	BeagleBone, Raspberry Pi
Program przenoszony przez pamięć nieulotną	1. Uruchamianie systemu operacyjnego na komputerze docelowym. 2. Uruchamianie programów w środowisku bez systemu operacyjnego	Mikrosterowniki bez systemu operacyjnego
Program wytwarzany w systemie docelowym	System docelowy jest oparty o rozbudowany komputer z systemem operacyjnym i licznymi interfejsami.	BeagleBone, PC104 QNX

Tabela 2-2 Porównanie metod tworzenia oprogramowania dla systemów wbudowanych

2.3 Budowa systemu skrośnego tworzenia oprogramowania

Zadaniem skrośnego systemu tworzenia oprogramowania (ang. *toolchain*) jest takie przetworzenie kodu źródłowego aplikacji, aby wytworzyć program wynikowy (binarny), który może być uruchomiony w systemie docelowym.

System docelowy może wykorzystywać inny procesor niż komputer macierzysty.

Typowo komputer macierzysty wykorzystuje procesor zgodny z x86, podczas gdy komputer docelowy pracuje na procesorze ARM.

Nie istnieje ogólny system tworzenia oprogramowania dla systemów wbudowanych. System taki przeznaczony jest zwykle dla:

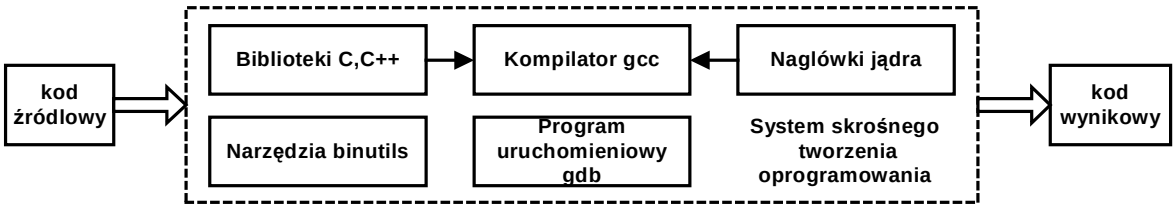
- określonego systemu operacyjnego
- wybranego typu procesora, dysponującego określoną architekturą i listą rozkazów.

Są dwie metody uzyskania systemu tworzenia oprogramowania:

- Wykorzystać gotowy system utworzony przez inne osoby.
- Samemu utworzyć taki system wykorzystując oprogramowanie GNU.

System tworzenia oprogramowania składa się z następujących narzędzi:

- Kompilator gcc
- Biblioteki C, C++
- Nagłówki jądra (ang. *kernel headers*)
- Narzędzia binutils
- Programu uruchomieniowego GDB (ang. *debugger*)



Rys. 2-8 Składniki skrośnego systemu tworzenia oprogramowania

Kompilator gcc

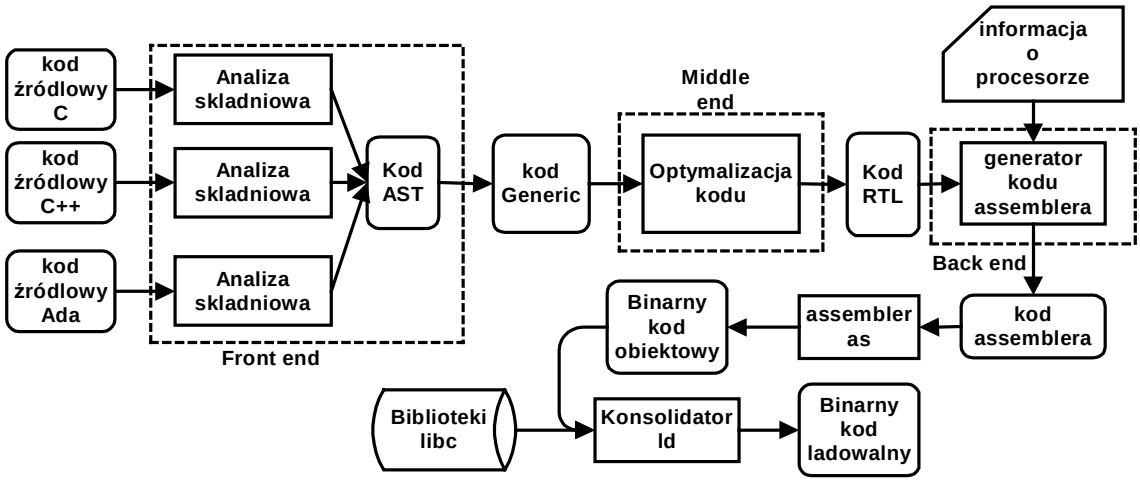
Do tworzenia oprogramowania dla systemów wbudowanych wykorzystuje się zazwyczaj kompilator GCC (ang. *Gnu Compiler Collection*). Jest to zbiór kompilatorów i innych narzędzi jak assemblery i linker służących do przekształcenia kody źródłowego w kod binarny.

W skład GCC wchodza kompilatory takich języków jak Ada, C, C++, Java, Fortran, Objective C.

Program gcc wywoływany jest z linii poleceń analizuje argumenty i stosownie do nich wywołuje kompilator odpowiedni dla danego języka.

Dla języka C będzie to program preprocesor **`cpp`**, kompilator **`cc1`**, assembler **`as`** i program łączący **`ld`**.

Kompilator GCC składa się z części frontowej (ang. *Front End*), środkowej (ang. *Middle End*) i końcowej (ang. *Back End*).



Rys. 2-9 Schemat koncepcyjny procesu kompilacji w kompilatorze GCC

Część frontowa
Indywidualna dla każdego języka programowania. Kod źródłowy poddawany jest procesowi analizy składniowej. Na tym etapie generowana jest większość komunikatów diagnostycznych, tworzone są struktury danych i zmienne. W wyniku działania pierwszego etapu powstaje reprezentacja programu w niezależnym od języka kodzie AST (ang. *Abstract Syntax Tree*) a dalej w tak zwanym kodzie generycznym.

Część środkowa
Część środkowa dokonuje różnych optymalizacji zamieniając kod AST na kod wyjściowy w języku RTL (ang. *Register Transfer Language*).

Część końcowa
Generuje ona kod dla zadanej architektury i procesora w języku assemblera danego procesora.GCC może generować kod dla szerokiego zestawu procesorów: Alpha, ARM, AVR, MIPS, Motorola 68000, Power PC, SPARC, x86, x86-64 i wiele innych.

Kod asemblerowy jest następnie tłumaczony przez program assemblera na binarny kod ładowalny danego procesora zapisywany w formacie ELF.

Narzędzia binutils

Narzędzia binutils jest to zbiór programów narzędziowych stosowanych do wytwarzania i programów wykonywalnych przeznaczonych na daną architekturę procesora.

as	Assembler – generuje kod binarny na daną architekturę procesora
ld	Linker – łączy moduły programowe
addr2line	Przekształca adres w komunikacie o błędzie na nazwy plików i numery linii. Przydatny w uruchamianiu programów.
ar	Program do tworzenia i obsługo archiwów. Łączy kilka skompilowanych plików obiektowych, zwykle zawierających biblioteki, w pojedynczy plik o rozszerzeniu a.
c++filt	Zamiana identyfikatorów występujących w plikach binarnych c++ w identyfikatory tekstowe
dlltool	Tworzy pliki DLL
gprof	Wyświetla informację o częstości użycia występujących w programie funkcji (tak zwane profilowanie programów).
nlmconv	Konwersja plików obiektowych na NLM
nm	Wypisuje symbole z plików obiektowych
objcopy	Kopiuje i tłumaczy pliki obiektowe
objdump	Wyświetla informację o plikach obiektowych
ranlib	Tworzy indeks do pliku archiwalnego.
size	Wyświetla długości sekcji plików obiektowych i archiwalnych
strings	Wyświetla łańcuchy tekstowe zawarte w plikach binarnych.
strip	Program do usuwania niekoniecznych części programu w celu jego optymalizacji.
windres	Kompilator plików zasobów systemu Windows

Tabela 2-3 Zestawienie programów wchodzących w skład binutils

Nagłówki jądra

Program aplikacyjny jak i biblioteki, aby dostarczyć żądanych funkcjonalności muszą odwoływać się do usług jądra. Usługi te dostępne są w postaci funkcji bibliotecznych, ale te są tylko opakowaniem do wywołań systemowych. Przyporządkowanie numerów wywołań do określonych funkcji dane jest w plikach nagłówkowych jądra. Zbiór wywołań systemowych może zależeć od wersji jądra.

Biblioteki

Standardowa biblioteka języka C nazywa się glibc i jej konstrukcja zgodna jest z różnymi standardami (np. POSIX1.j, Single Unix Specification). Biblioteka ta jest dobrze przetestowana ale nie była konstruowana z myślą o systemach wbudowanych stąd zoptymalizowana została pod względem szybkości działania a nie rozmiaru. Stąd wymaga dużego obszaru pamięci RAM co najmniej 2 MB.

Istnieją biblioteki do języka C specjalnie skonstruowane dla systemów wbudowanych. Wymienić tu można takie biblioteki jak embedded C library, uClib, **`diet, libc`**.

2.4 Tworzenie systemu tworzenia oprogramowania skrośnego

Skąd bierze się system tworzenia oprogramowania (ang. *toolchain*)?

Założono że taki system został już wcześniej przez kogoś utworzony i możemy z niego skorzystać.

Należy uwzględnić jeszcze etap wytworzenia systemu tworzenia oprogramowania skrośnego. (ang. Build).

Należy wyróżnić trzy maszyny:

- Maszynę na której tworzony jest system tworzenia oprogramowania skrośnego (ang. *Build computer*).
- Maszynę na której tworzone jest oprogramowania na system docelowy (ang. *Host computer*).
- Maszynę na której wykonywany jest program aplikacyjny (ang. *Target computer*).

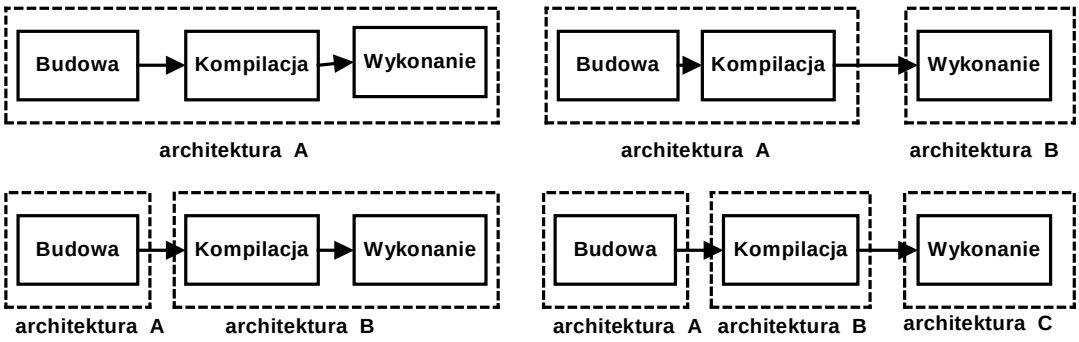
Maszyny te mogą być takie same lub też się od siebie różnić.

Można rozróżnić trzy etapy tworzenia takiego oprogramowania skrośnego:

- Etap I - w którym tworzony jest system do tworzenia oprogramowania skrośnego (ang. *Build*)
- Etap II - w którym uruchamiany jest system tworzenia oprogramowania (ang. *Host*).
- Etap III - w którym wykonywany jest otrzymany na poprzednim etapie program (ang. *Target*)

Załóżmy że mamy do dyspozycji trzy maszyny o różnych architekturach, które oznaczymy jako A, B, C. Stąd możliwe są cztery konfiguracje przydziałów etapów na architektury:

1. Wszystkie etapy I, II i III wykonywane są na maszynie o jednej architekturze A.Ten sposób wykorzystywany jest głównie dla maszyn x86 pracujących w tym samym systemie operacyjnym.
2. Etap I i II wykonywany na komputerze o architekturze A, a program binarny przesyłany jest do systemu docelowego o architekturze B. Jest to najbardziej rozpowszechniona metoda tworzenia oprogramowania dla systemów wbudowanych.
3. Etap I wykonywany jest na komputerze o architekturze A, podczas gdy etapy II i III wykonywane są na komputerze o architekturze B.
4. Wszystkie etapy I, II i III wykonywane są na innych komputerach o architekturach A,B i C. Ta konfiguracja nazywana jest konfiguracją Kanadyjską (ang. *Canadian build*).



Rys. 2-10 Możliwe konfiguracje budowy systemu kompilacyjnego, kompilacji i wykonania programu

W ogólności są dwie możliwości uzyskania systemu tworzenia oprogramowania:

- Skorzystać z gotowego systemu tworzenia oprogramowania
- Samemu taki zbudować korzystają z dostępnego oprogramowania GNU

Gotowe systemy tworzenia oprogramowania są zwykle dostarczane przez producentów sprzętu (procesorów, płytek) lub też są produktem społecznościowym.

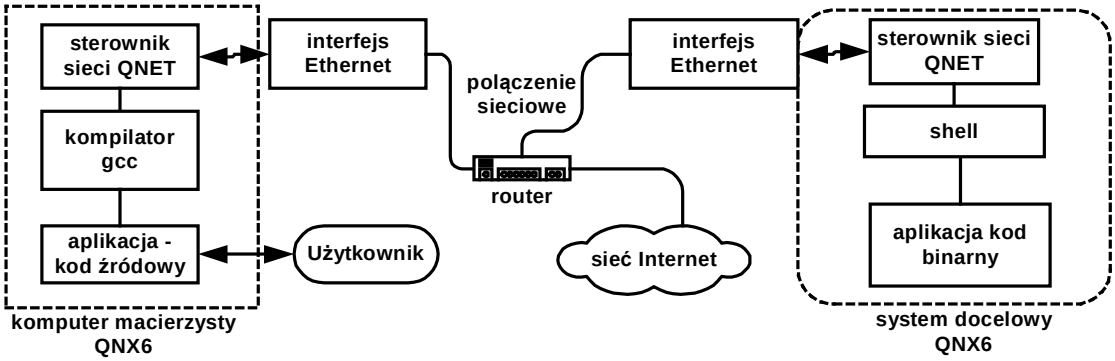
Możliwe jest także samodzielne utworzenie systemu tworzenia oprogramowania skrośnego dla danej platformy sprzętowej. Informacje - Linux from Scratch.

Gdy zdecydujemy się na samodzielne tworzenie systemu oprogramowania skrośnego zalecane jest skorzystanie z ułatwiających ten proces narzędzi.

- Crosstool
- Crosstool-ng
- Buildroot
- OpenWRT
- Scratchbox
- Ptxdist
- Linaro
- CodeSourcery

2.5 Przykład 1 – System macierzysty i docelowy - QNX6 Neutrino, komunikacja sieć QNET

W tym przykładzie komputer macierzysty i docelowy pracują w tym samym systemie operacyjnym QNX6 Neutrino. Komunikacja poprzez sieć QNET. Komputer macierzysty i docelowy połączone siecią Ethernet.

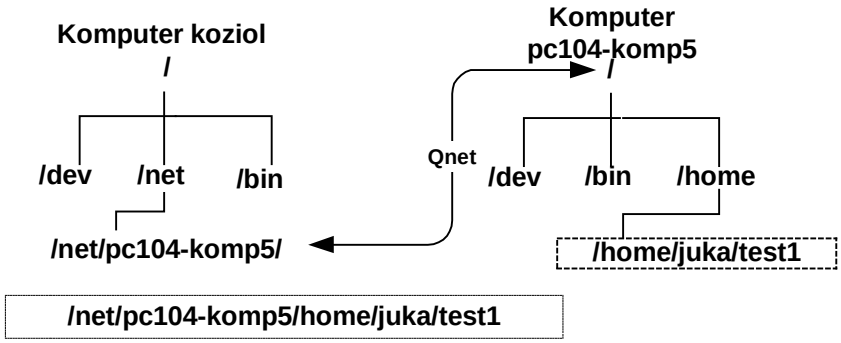


Rys. 2-11 Komunikacja z systemem wbudowanym za pośrednictwem sieci QNET

Wykorzystujemy:

1. Sieciowy system plików QNET. Systemy plików stacji dołączonych do sieci widziane w katalogu /net
2. Możliwość zdalnego wykonania programu na innym węźle sieci QNET.

System QNX6 implementuje sieć QNET. Gdy uruchomiona jest sieć Qnet, do lokalnej przestrzeni nazw zaczynającej się od znaku **/** dodawany jest katalog **/net** zawierający podkatalogi odpowiadające nazwom dołączonych do sieci węzłów.



Rys. 2-1 Odwzorowanie przestrzeni nazw komputera pc104-komp5 w katalogu /net/pc104-komp5/ komputera koziol

W podanym wyżej przykładzie plik /home/juka/test1 umieszczony na komputerze pc104-komp5 jest z komputera koziol widziany jako /net/cp104-komp5/home/juka/test1.

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 domyślnym 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 2-1 Lokalne i zdalne wykonanie (na komputerze pc104-komp5) polecenia hostname podającego nazwę węzła bieżącego.

1. W systemie macierzystym utworzyć podany niżej program a następnie go skompilować
   gcc test.c –o test

```
#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 2-2 Program test.c

2. Uruchomić system docelowy i sprawdzić czy jest widziany w katalogu **/net** za pomocą polecenia: **ls /net**

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

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

`on -f pc104-komp5 login`

4. 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

5. Na komputerze docelowym uruchomić program **`test1`** używając pseudoterminala.

```
# 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
Thu Oct  9 01:43:18 2014 on /net/koziol.dom/dev/ttyp0
Last login: Thu Oct 9 01:42:15 2014 on /net/koziol.dom/dev/ttyp0
edit the file .profile if you want to change your environment.
# pwd
/root
# cd /dev/shmem
# ls
test1
# ./test1_
```

Ekran 2-1 Uruchomienie programu test1 w systemie docelowym za pomocą pseudoterminala

Inna możliwość – zalogowanie się do systemu docelowego poprzez telnet.

```
# telnet 192.168.0.4
Trying 192.168.0.4...
Connected to 192.168.0.4.
Escape character is '^]'.


QNX Neutrino (vortex-5) (ttyp0)

login: root
Fri Oct  2 03:06:32 2015 on /dev/ttyp0
Last login: Fri Oct 2 03:03:08 2015 on /dev/ttyp0
edit the file .profile if you want to change your environment.
To start the Photon windowing environment, type "ph".
# cd /tmp
# ./hello
jestem
QNX vortex-5 6.5.0 2010/07/09-14:44:03EDT x86pc x86
# _
```

Ekran 2-2 Uruchomienie programu hello w systemie docelowym za pomocą pseudoterminala telnet

2.6 Przykład 2 – System macierzysty i docelowy - QNX6 Neutrino, komunikacja TCP/IP

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 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 2-3 Program **`test1.c`**

2. Uruchomić system docelowy 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`

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) _
```

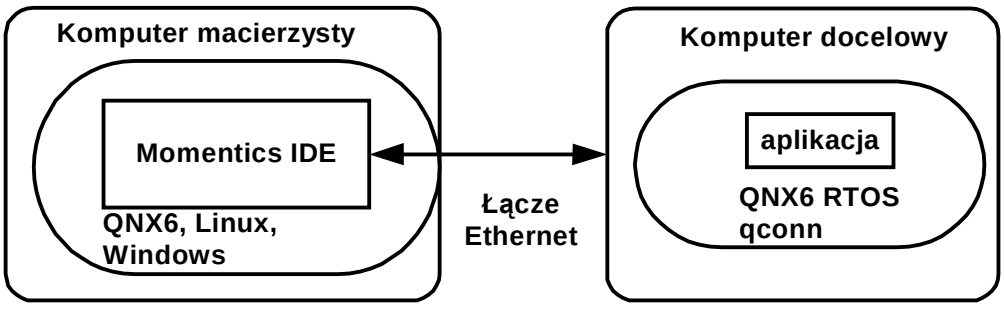
2.7 Przykład 3 – System macierzysty i docelowy - QNX6 Neutrino, środowisko Momentics

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:

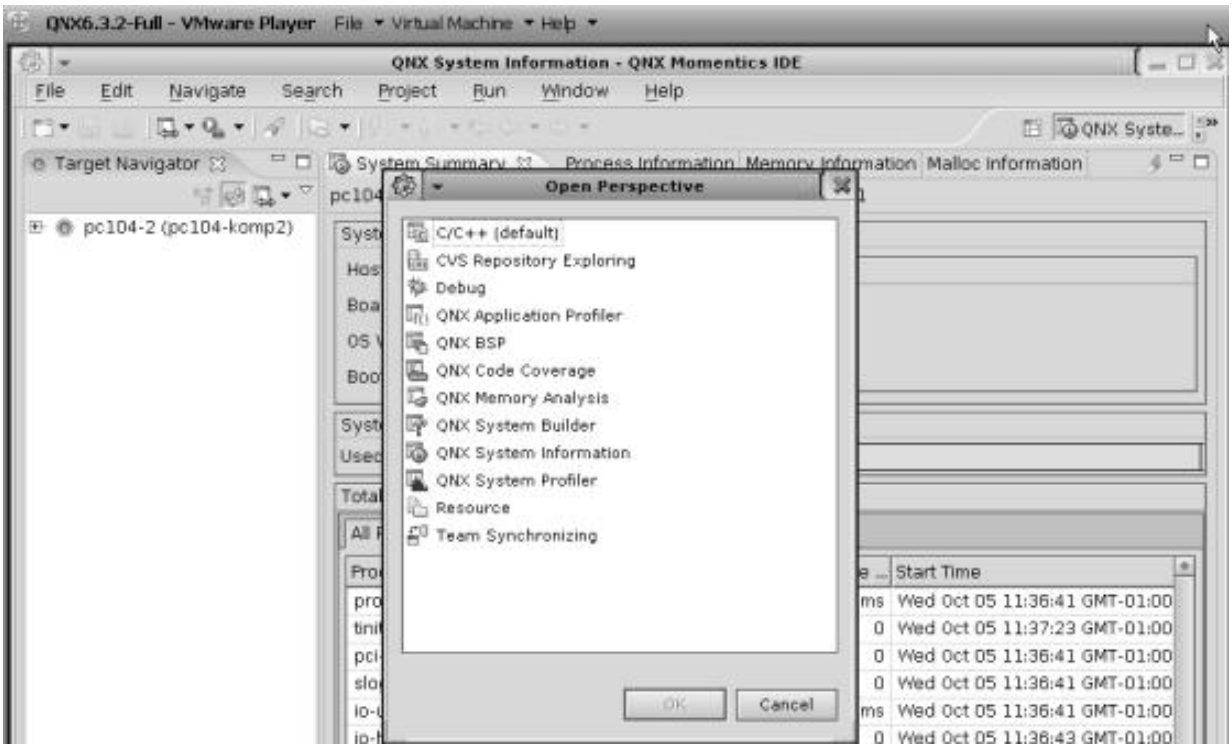
- Edytor i debuger kodu źródłowego.
- Narzędzie analizy pokrycia kodu (ang. *Code Coverage*).
- Narzędzia informacji o systemie docelowym.
- Konstruktor systemu operacyjnego (ang. *System Builder*) oraz pakiet wsparcia dla płyt głównych (ang. *Board Support Package*).
- Narzędzie konfiguracji aplikacji 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. 2-2 System skrośnego rozwoju oprogramowania

2.8 Uruchomienie narzędzia Momentics, wybór perspektywy

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.



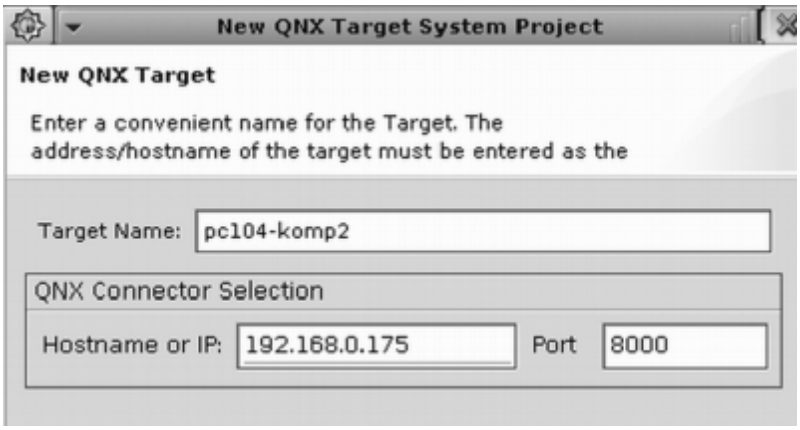
Rys. 2-3 Narzędzie Momentics Development Suite - wybór perspektywy

2.9 Połączenie z systemem docelowym

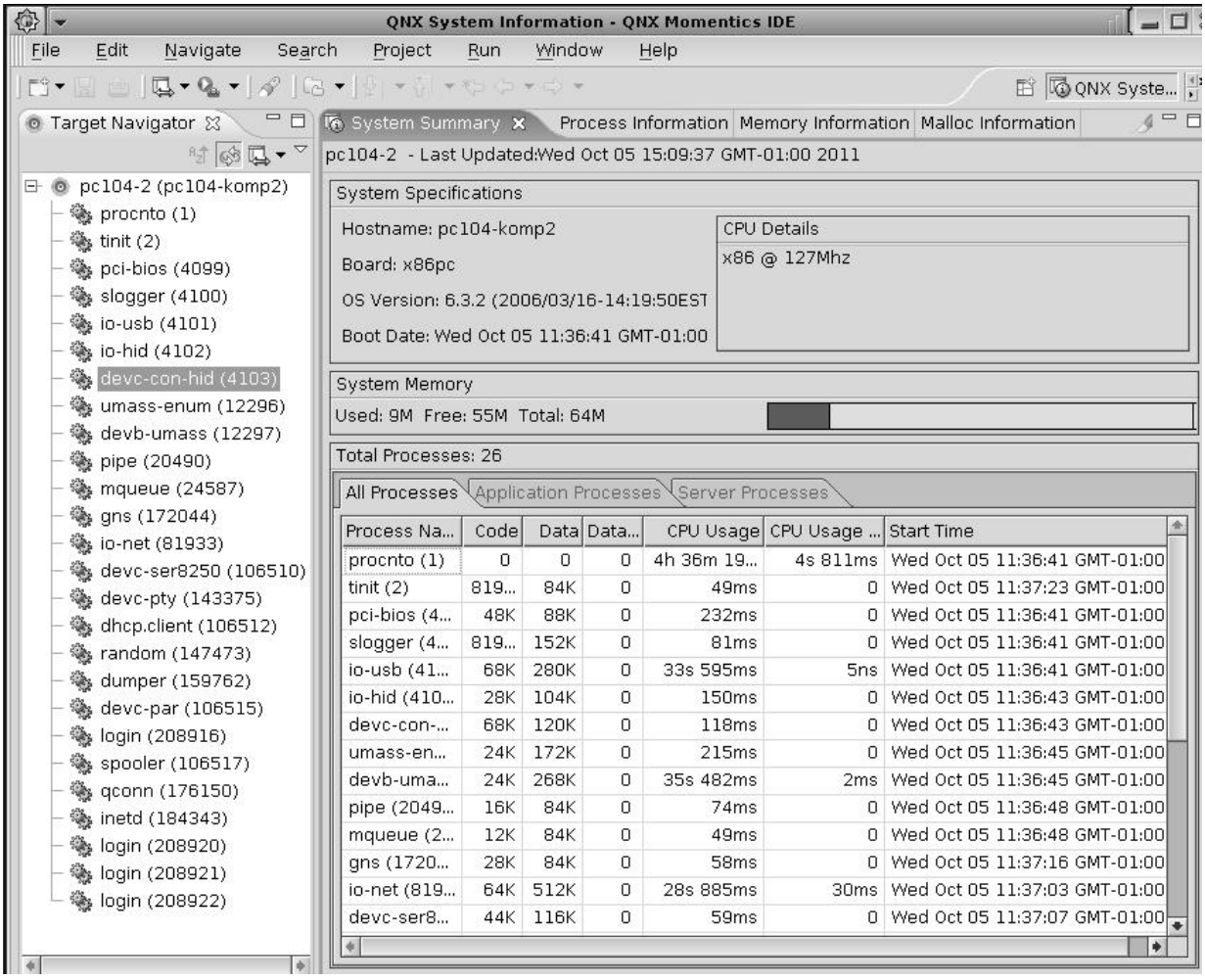
Aby połączyć się z systemem docelowym należy dołączyć obydwa komputery (macierzysty i docelowy) do sieci Ethernet. System docelowy musi pracować pod kontrolą QNX6 Neutrino i powinien mieć uruchomione:

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

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. 2-4 Ikona wyboru systemu docelowego



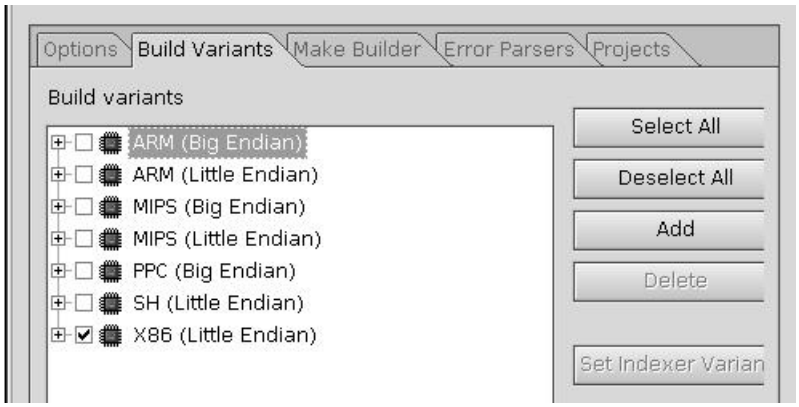
Rys. 2-5 Uzyskiwanie informacji o systemie docelowym

2.10 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.

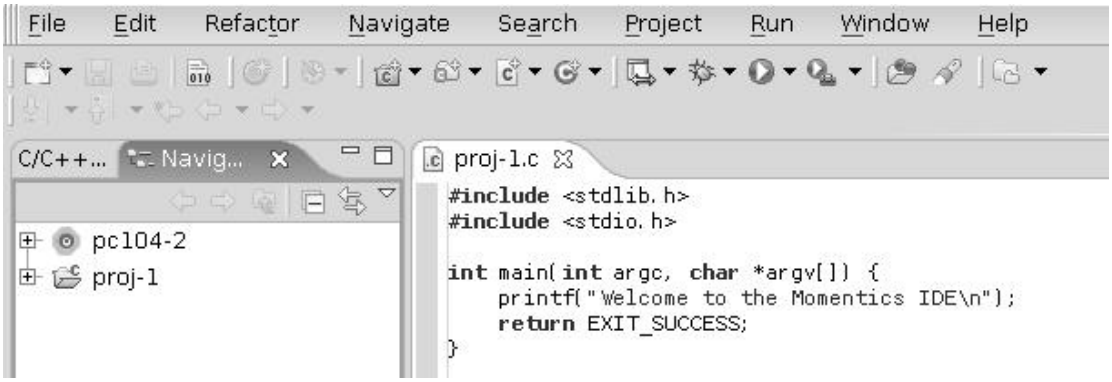
Rys. 2-6 Tworzenie nowego projektu

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



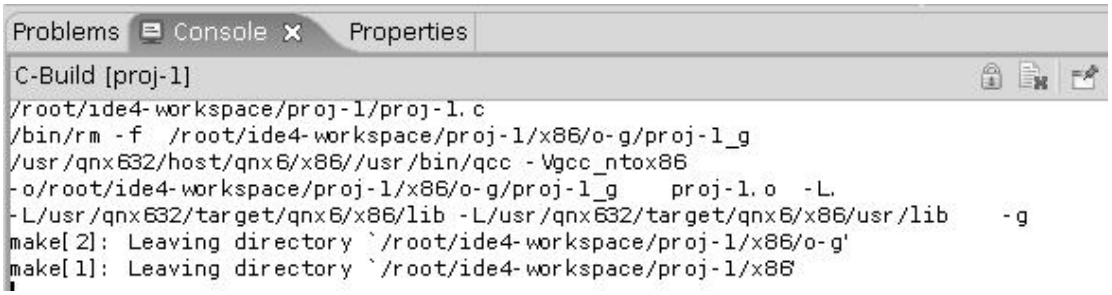
Rys. 2-7 Wybór architektury systemu docelowego

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



Rys. 2-8 Okno edycji kodu programu szkieletowego

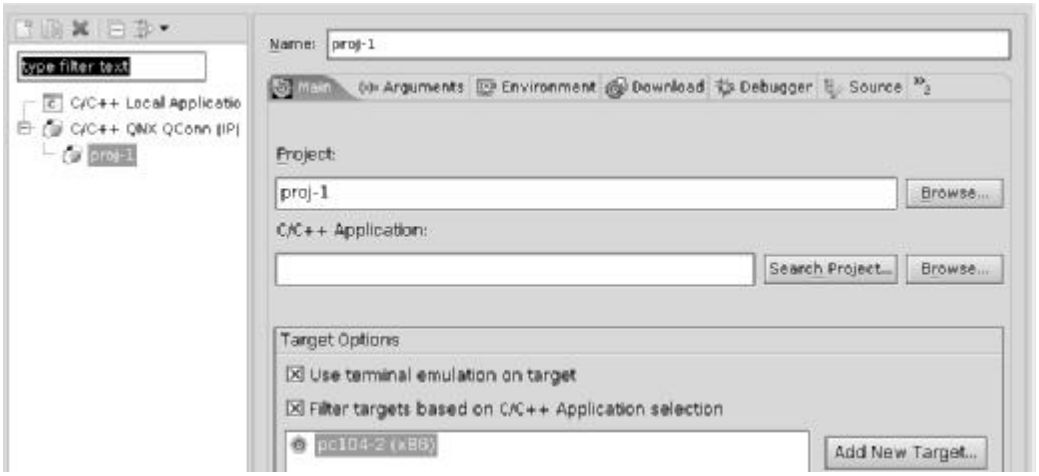
Przykładowy kod można dowolnie modyfikować. Kolejną czynnością jest kompilacja programu. Wykonujemy ją wybierając z menu opcje: **Project / Build Project** . Gdy w programie nie ma błędów zakładka Console zawierała będzie kod kompilacji jak pokazano poniżej.



Rys. 2-9 Raport z kompilacji programu przykładowego

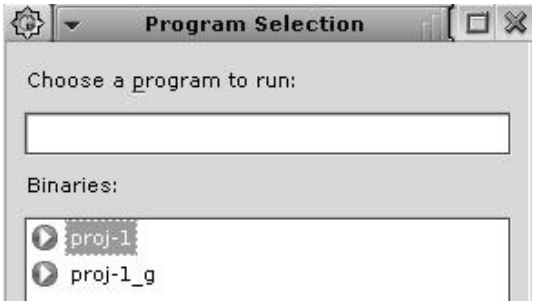
2.11 Wykonanie programu na systemie docelowym

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



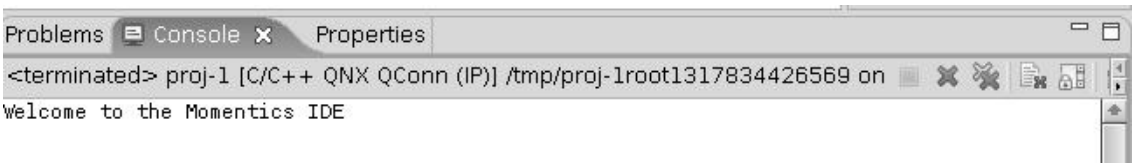
Rys. 2-10 Formatka specyfikacji wykonania programu

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. 2-11 Wybór wersji programu

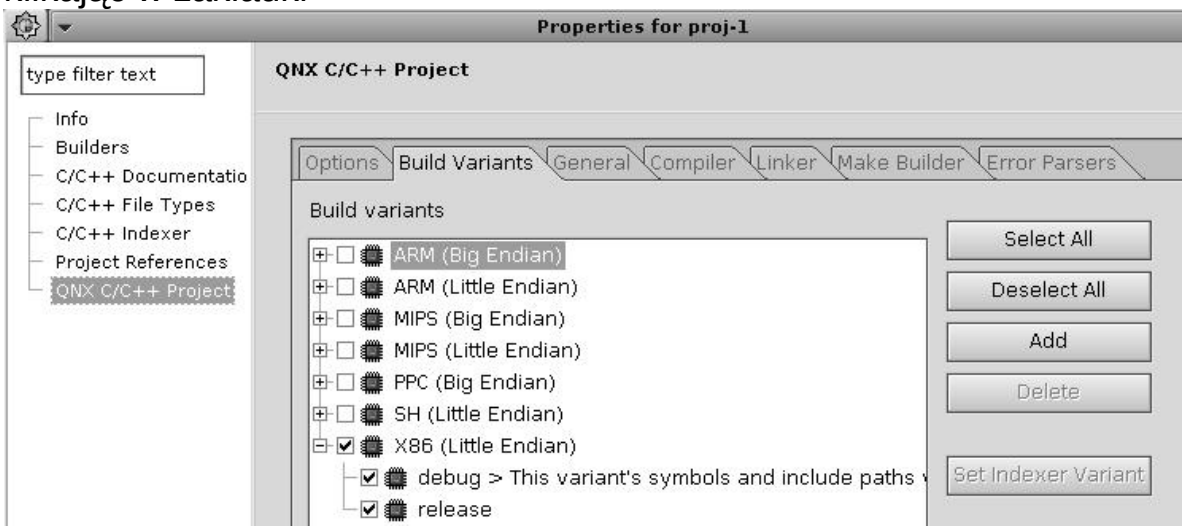
Gdy wersja zostanie określona klikamy w przycisk Run co spowoduje przesłanie programu na system docelowy (domyslnie do folderu /tmp) i jego uruchomienie. Rezultat działania programu pokazany zostanie na konsoli tak jak w poniższym przykładzie.



Rys. 2-12 Wykonanie programu przykładowego

2.12 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. 2-13 Formatka właściwości projektu

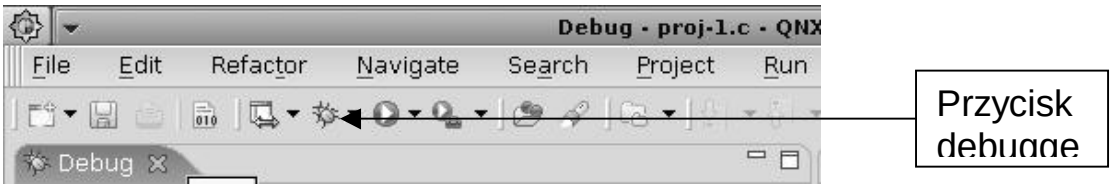
Aby uczynić debugowanie ciekawszym zmodyfikujmy program przykładowy aby wykonywał w pętli wypisywanie zawartości licznika zadana liczbę razy.

proj-1.c

```
1#include <stdlib.h>
2#include <stdio.h>
3#include <unistd.h>
4
5int z = 0;
6int main(int argc, char *argv[]) {
7    int i;
8    printf("Witamy  w Momentics IDE !  \n");
9    for(i=0;i<20;i++) {
10       printf("Krok %d\n",i);
11       sleep(1);
12       z = i+1;
13   }
14   printf("Koniec\n");
15   return EXIT_SUCCESS;
16}
```

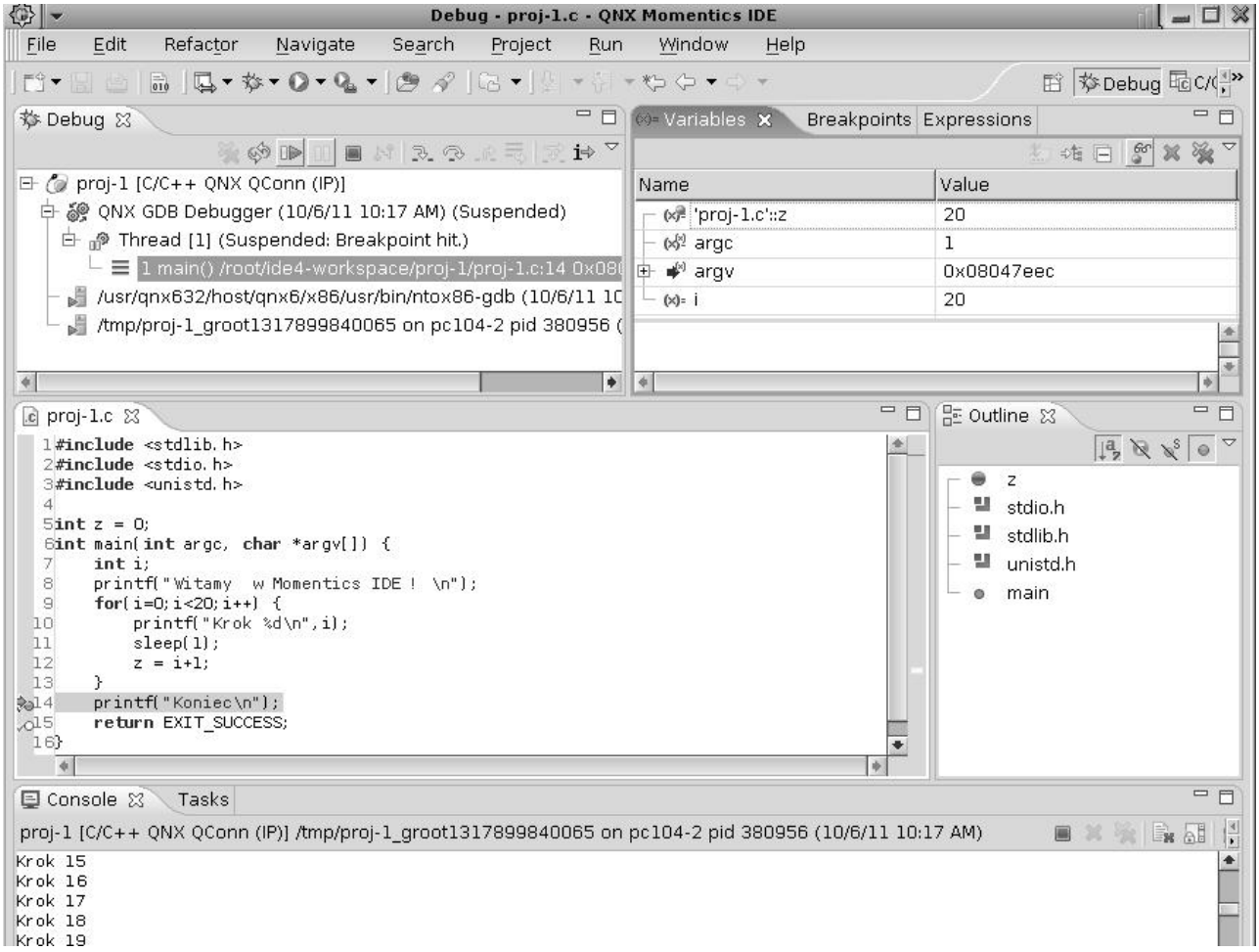
Rys. 2-14 Kod źródłowy programu testowego

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



Rys. 2-15 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. 2-16 Formatka programu uruchomieniowego

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

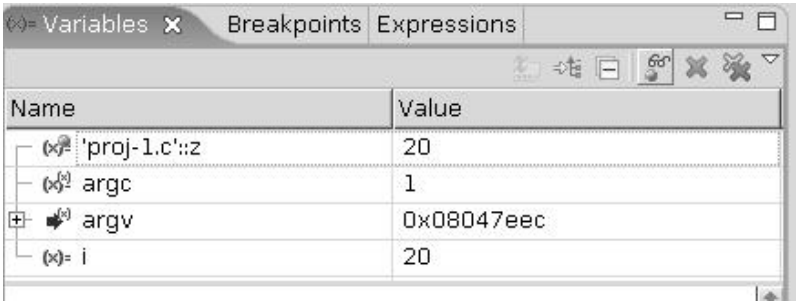
	Restart		Uruchom proces od początku
	Resume	F8	Uruchom proces od punktu bieżącego
	Terminate		Zakończ proces
	Step into	F5	Wykonaj krok programu wchodząc do funkcji
	Step over	F6	Wykonaj krok programu nie wchodząc do funkcji
	Run to return	F7	Wyjdź z funkcji
	Suspend		Zawieś proces

Tab. 2-1 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. 2-17 Okno inspekcji zmiennych

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