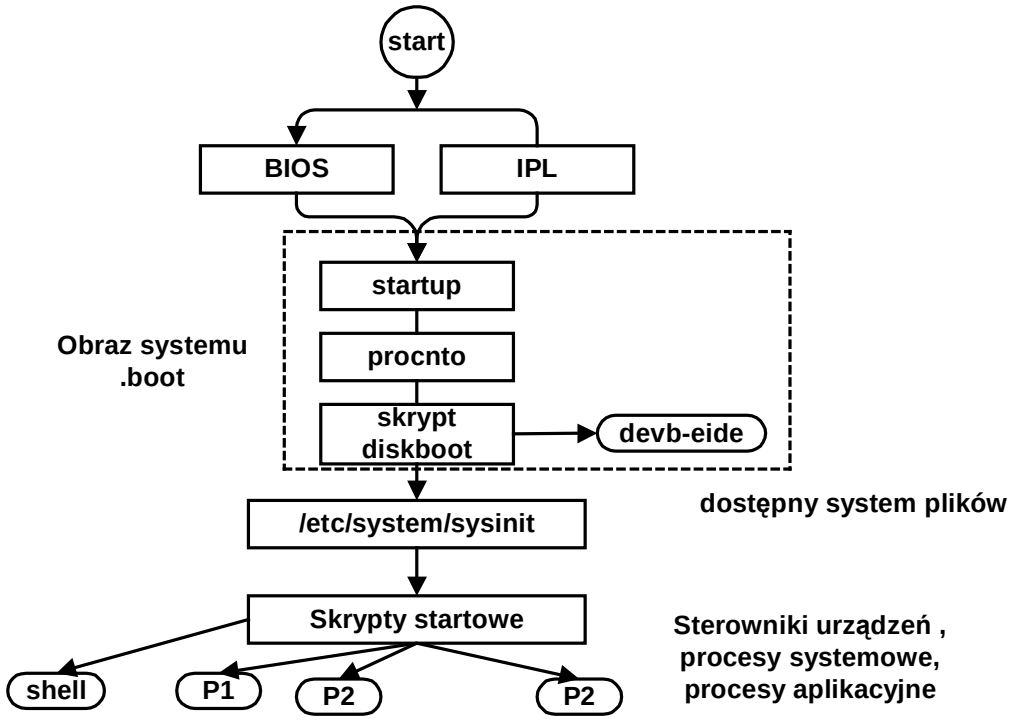
Instalacja i konfiguracja systemu operacyjnego

1. Pojęcia podstawowe, wczytywanie systemu

1.1 Program ładowania wstępnego IPL i BIOS

Gdy procesor startuje brak jeszcze systemu operacyjnego – musi on być załadowany, skonfigurowany i wystartowany. Role te pełni IPL (ang. *Initial Program Loader*). Funkcje IPL to:

- Start od obszaru kodu wskazywanego przez „reset vector”
- Konfiguracja kontrolera pamięci, kontrolera PCI i innych kontrolerów
- Konfiguracja zegara
- Kopiuje obraz systemu do pamięci operacyjnej
- Wykonuje skok do początku obrazu systemu



Rys. 1-1 Start systemu QNX6 Neutrino

Wyróżnia się dwa rodzaje sposobów pracy IPL:

- Zimny start – pierwszy program jaki się w komputerze wykonuje, składniki sprzętu nie są zainicjowane
- Ciepły start – program uruchomiony przez BIOS lub monitor zawarty w pamięci ROM, pewne składniki sprzętu są już zainicjowane.

Po zakończeniu pracy IPL wymagany jest następujący stan systemu:

- Kontroler pamięci jest skonfigurowany i umożliwia do niej dostep
- Niezbędne minimum konfiguracji sprzętu zostało wykonane
- Obraz systemu operacyjnego umieszczony jest w liniowo adresowalnej pamięci
- Początkowa część obrazu systemu operacyjnego umieszczona jest w pamięci RAM

1.2 BIOS i ładowanie systemu operacyjnego

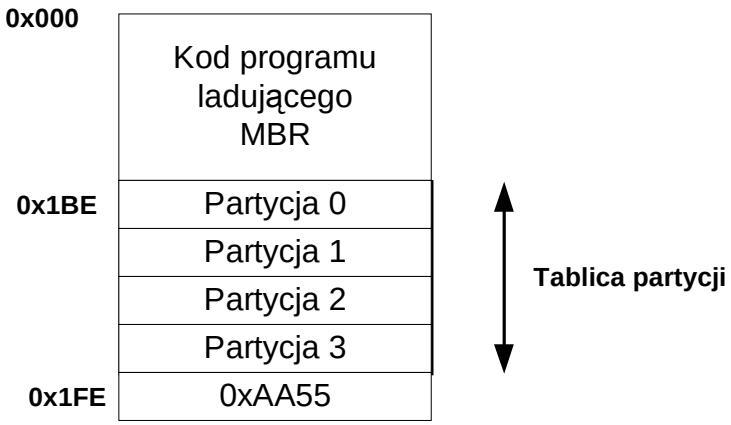
BIOS (ang. *Basic Input Output System*) jest programem który startuje jako pierwszy po włączeniu komputera. Zapewnia dostęp do podstawowych komponentów komputera. Jego funkcje to:

1. Dostarczenie interfejsu do konfiguracji systemu (BIOS Setup)
2. Wczytywanie systemu operacyjnego
3. Diagnostyka systemu

Po włączeniu zasilania BIOS wykonuje następujące czynności:

1. Sprawdzenie integralności kodu programu BIOS
2. Ustalenie rozmiaru i sprawdzenie pamięci głównej
3. Znalezienie, zainicjalizowanie i skatalogowanie wszystkich magistrali
4. Dostarczenie interfejsu do konfiguracji systemu (BIOS Setup)
5. Zidentyfikowanie urządzeń zdolnych do wczytywania systemu (dyski IDE, SATA, Floppy, CDROM, FLASH).
6. Załadowane z wybranego urządzenia (użytkownik może określić z którego urządzenia dysku należy załadować system operacyjny gdy urządzeń jest więcej) pierwszych 512 bajtów do pamięci operacyjnej pod adres 0000:7C00. Bajty ładowane są z obszaru znajdującego się na samym początku dysku (sektor 0, cylinder 0, powierzchnia 0). W obszarze ty znajduje się pierwotny program ładujący MBR (ang. *Master Boot Record*) i tablica partycji dysku.
7. Następuje przekazanie sterowania do programu ładującego zawartego w MBR (rozkazu znajdującego się pod adresem 0000:7C00). Program ten stwierdza która partycja jest aktywna i ładuje do pamięci operacyjnej zawarty na jej początku program ładujący (ang. *Boot Record*). Program ten jest specyficzny dla systemu operacyjnego i wie jak go załadować. Dalsze ładowanie systemu przeprowadza program ładujący.

Znane są programy ładujące które mogą wybrać inną partycję dysku z której ma być załadowany system operacyjny. Przykładem jest LILO lub GRUB, U-boot.



Rys. 1-2 Struktura sektora ładującego MBR

1	Flaga aktywności
2	Początek partycji
3	Typ partycji
4	Koniec partycji
5	Sektor początkowy partycji
6	Liczba sektorów partycji

Rys. 1-3 **Zawartość tablicy partycji**

Jedna partycja oznaczona jest jako aktywna. Zawiera ona program ładujący.

1.3 Program ładujący i obraz systemu

Obraz systemu jest plikiem o nazwie **`/.boot`** a zapasowym plikiem **`/.altboot`**. Zawiera on:

- Program startowy (ang. *startup program*)
- Mikrojądro i administrator procesów
- Programy i skrypty startowe

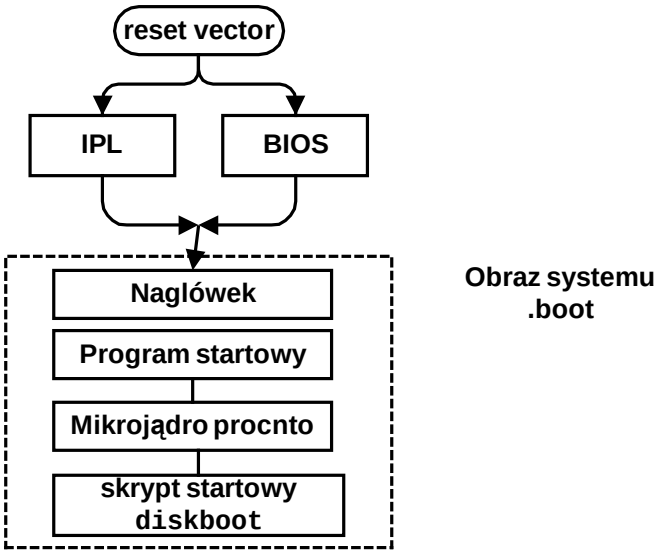
Funkcje wykonywane przez program startowy to:

- Inicjalizacja sprzętu
- Wypełnia „stronę systemową ” zawierającą informacje o sprzęcie
- Kopiowanie do pamięci operacyjnej dalszą część obrazu systemu (mikrojądro (ang. *microkernel*) i administrator procesów **`procnto`**) i jego dekompresja (gdy konieczna)
- Startuje procesy zawarte w **`boot scripts`** które dokonują dalszego ustanawiania środowiska sprzętowego i programowego poprzez wykonanie skryptów i uruchomienie programów w C

Inicjalizacja sprzętu

Ustanowienie wywołań systemowych do mikrojądra

- Interfejs do debugowania
- Interfejs do zegara i czasomierza
- Interfejs kontrolera przerwań
- Interfejs kontrolera pamięci podręcznej
- Interfejs zarządzania zasilaniem



Rys. 1-4 Pierwsza faza startu systemu

Skrypt startowy **diskboot**

Typowy skrypt **diskboot** wykonuje następujące funkcje:

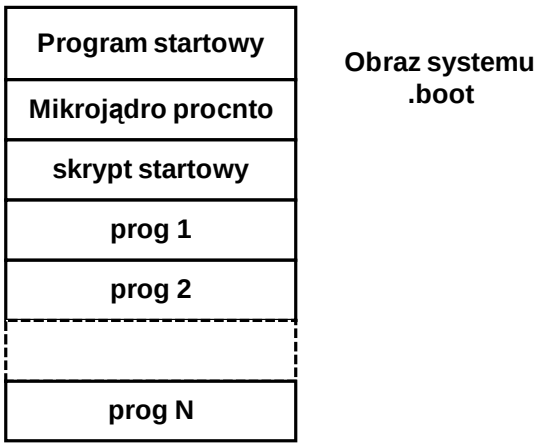
1. Startuje proces logujący **slogger** zapisujący zdarzenia przy starcie systemu. Zawartość tego pliku może być odczytana poleceniem **sloginfo**.
2. Uruchamia program **seedres** który odczytuje ustawienia PnP BIOS i przekazuje je do **procnto**.
3. Uruchamia program **pci-bios**
4. Startuje driver dysku **dev-eide** (lub inny driver dysku)
5. Montuje systemy plików : CD-ROM, DOS, Ext2, QNX4. Systemy plików montowane są w katalogu **/fs**.
6. Opcjonalnie uruchamia shell **fesh** dla systemów wbudowanych.
7. Staruje driver konsoli **devc-con-hid**
8. Uruchamia skrypt **/etc/system/sysinit**

2. Obraz systemu i jego tworzenie

Obraz systemu .**boot** jest plikiem zawierającym programy wykonywalne i dane. Może być rozpatrywany jako mały system plików zawierający foldery i pliki. W szczególności obraz systemu zawiera:

- Program ładujący
- Mikrojądro
- Administrator procesów **procnto**

Inne programy i skrypty startowe



Obraz systemu może być ładowalny (ang. *bootable*) i nie ładowalny (ang. *non bootable*). Obraz ładowalny zawierać musi program ładujący. Obraz systemu tworzony jest za pomocą narzędzia **mkifs**.

Zawartość pliku **boot** może być sprawdzona za pomocą polecenia:

ls /proc/boot

```
#ls /proc/boot
.script     ping     cat    data1        pidin
ksh         ls       ftp    procnto      devc-
ser8250-ixp2400
```

Przykład 2-1 Sprawdzanie zawartości obrazu systemu

2.1 Konfiguracja obrazu systemu

Obraz systemu tworzony jest przez narzędzie **mkifs** na podstawie pliku projektowego (ang. *make image filesystem*) i plików zawartych w systemie.

```
mkifs buildfile [imagefile] [-v]
```

Gdzie:

buildfile – plik projektowy systemu
imagefile – plik z obrazem systemu
-v - rozszerzone raportowanie

Plik projektowy systemu (ang. *buildfile*) zawiera reguły według których system jest budowany. Plik projektowy systemu zawiera:

- Skrypt ładowania systemu (ang.*bootstrap script*)
- Skrypt startowy (ang. *startup script*)
- Listę plików (ang. *file list*)
- Listę plików do odlinkowania (ang. *unlink list*) – opcja

Skrypt ładowania systemu

Sposób ładowania systemu zależy od typu procesora. Skrypt ładowania systemu specyfikuje jaki program ma być użyty do ładowania systemu i jakie jego opcje mają być aktywne.

```
[virtual=x86,bios +compress] .bootstrap={
  startup-bios -s 64k
  PATH=/proc/boot LD_LIBRARY_PATH=/proc/boot:/usr/lib
  procnto
}
```

Pierwsza linia informuje że jest to skrypt startowy, część w nawiasach specyfikuje atrybuty:

virtual=x86 – praca w trybie wirtualnym procesora x86
bios - system z BIOS
+compress - użyty skompresowany obraz systemu

Program **startup-bios** wykonuje dekompresję obrazu systemu, załadowanie go do RAM i przekazanie sterowania do jądra (moduł **procnto**).

Plik **procnto** zawiera mikrojądro. Jest ono odpowiedzialne za zarządzanie pamięcią, procesami, i komunikację międzyprocesową. Wartości zmiennych **PATH** i **LD_LIBRARY_PATH** są dziedziczone przez procesy potomne i wskazują gdzie szukać programów i bibliotek. Od tego momentu system może tworzyć nowe procesy.

Skrypt startowy

Skrypt startowy wykonywany jest po uruchomieniu jądra i startuje niezbędne programy obsługi urządzeń. Przykład podany jest poniżej.

```
[+script] .script={
  seedres
  devc-con -n2 &
  reopen /dev/con2
  [+session] ksh &
  reopen /dev/con1
  [+session] ksh &
 }
```

seedres - program odczytuje z PnP BIOS dane o konfiguracji sprzętowej (portach we/wy przerwaniach) i przekazuje do bazy danych jądra

devc-con - uruchomienie programu obsługi konsoli

reopen - otwiera stdin, stdout, stderr i przypisuje je do danego urządzenia

[+session] ksh & - startuje nową kopię interpretera poleceń i czyni ją liderem sesji

Lista plików

Kolejną częścią obrazu systemu jest lista plików które są potrzebne na tym etapie pracy systemu. Fragment listy plików dany jest poniżej.

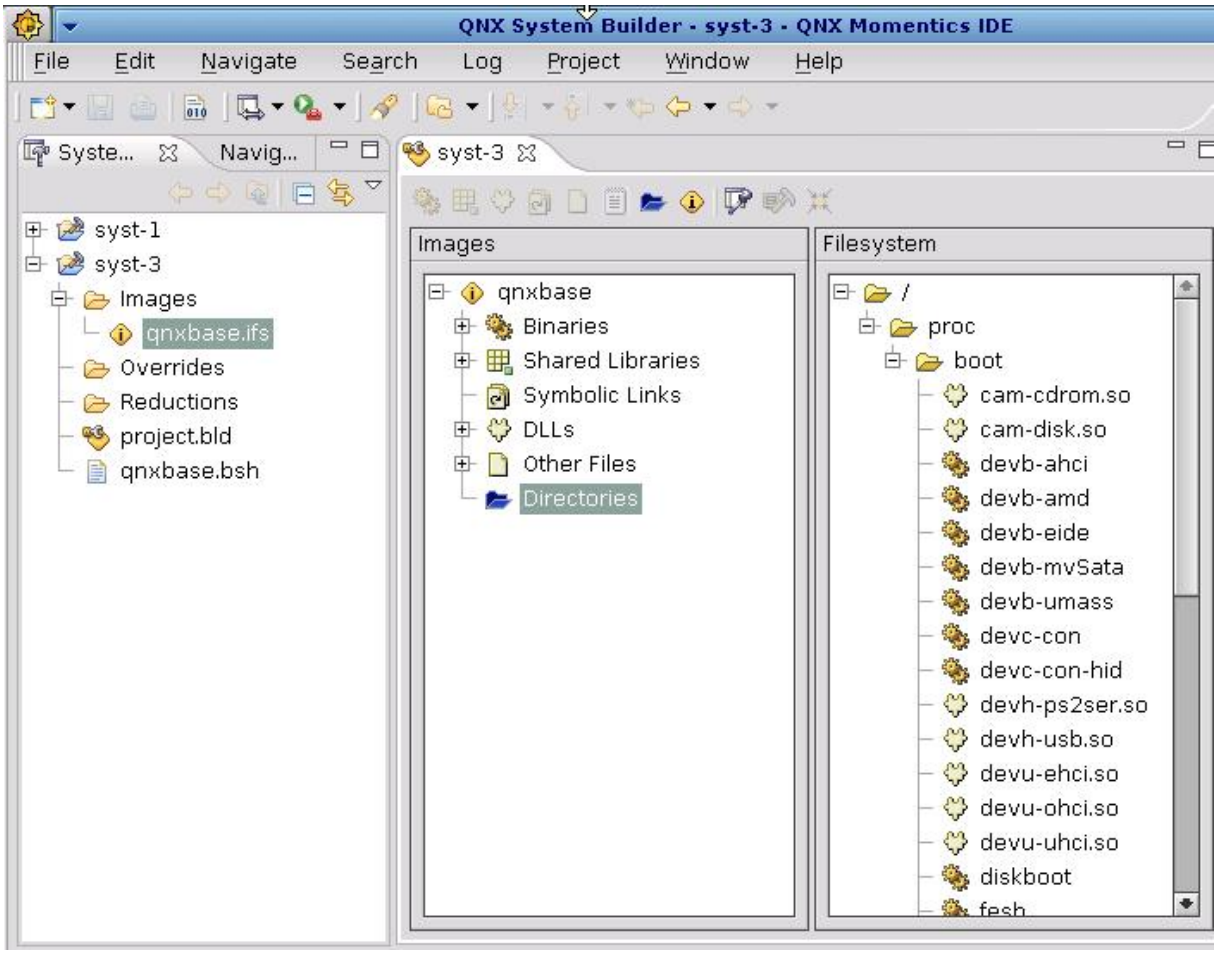
```
libc.so
# Dodanie linku symbolicznego
[type=link] /usr/lib/ldqnx.so.2=/proc/boot/libc.so
[code=uip data=copy perms=+r,+x]
seedres
kill
…
```

libc.so - Należy w obrazie systemu umieścić tę bibliotekę gdyż jest ona potrzebna w większości programów w języku C.

[type=link] /usr/lib/ldqnx.so.2=/proc/boot/libc.so – Różne aplikacje mogą się spodziewać plików w różnych miejscach. Aby ich nie kopiować tworzy się linki

[code=uip data=copy perms=+r,+x] – Wymienione dalej programy będą umieszczone w pamięci operacyjnej. Nie ma więc potrzeby kopiować ich segmentu kodu a tylko segment danych. Sekcja **perms=+r,+x** mówi że pliki mają prawo odczytu i wykonania.

Obraz systemu można również uzyskać posługując się narzędziem Momentics (perspektywa *System builder*)



Rys. 2-1 Zastosowanie narzędzia Momentics (perspektywa System builder) do generowania obrazu systemu operacyjnego

Narzędzie umożliwia:
Dodawanie / usuwanie składników do obrazu systemu jak:

- Programy binarne
- Sterowniki urządzeń
- Biblioteki

Edycję systemu plików

2.2 Przykład 1 – system wczytywany z dyskietki

Przykład kompletnego pliku projektowego dla systemu x86 podano poniżej. System umożliwia wystartowanie powłoki **ksh** i prostą eksplorację systemu za pomocą poleceń: **ps, pidin, sin.**

```
[virtual=x86,bios +compress] .bootstrap={
  startup-bios -s 64k
  PATH=/proc/boot LD_LIBRARY_PATH=/proc/boot:/usr/lib
  procnto
}
[+script] .script={
  seedres
  display_msg "Moj obraz syst QNX..."
  display_msg "Dzien dobry"
  # dwie wirtualne konsole
  # przelaczanie Ctrl+Alt+1 and Ctrl+alt+2
  devc-con -n2 &
  reopen /dev/con2
  [+session] ksh &
  reopen /dev/con1
  [+session] ksh &
}
libc.so
# Dodanie linku symbolicznego
[type=link] /usr/lib/ldqnx.so.2=/proc/boot/libc.so
[code=uip data=copy perms=+r,+x]
seedres
kill
cat
ls
ksh
devc-con
less
ps
sin
pidin
#Tutaj dodajemy edytor...
vi
```

Przykład 2-2 Prosty plik projektowy „buldfile” bld1.build

Budowanie systemu

Po edycji pliku projektowego **bld1.build** tworzymy obraz systemu pisząc:

```
$ mkifs -v bld1.build bld1.ifs
```

Gdy budowa się powiedzie powstanie plik **bld1.ifs** będący obrazem systemu.

Tworzenie dyskietki startowej

Gdy system zostanie utworzony należy sprawdzić jego działanie. Tak więc należy utworzyć taki nośnik z którego da się wczytać system. Najłatwiej będzie użyć dyskietki. W tym celu przygotowujemy dyskietkę 3.5 ” i wkładamy ją do napędu dyskietek. Następnie dokonujemy inicjacji dyskietki z przeniesieniem obrazu systemu. Używamy polecenia **dinit**:

```
$dinit –f bld1.ifs /dev/fd0
```

po przełączniku –f umieszczona jest nazwa pliku który jest obrazem systemu – w tym przypadku jest to plik **bld1.ifs**. Drugi parametr określa urządzenie do którego włożona jest dyskietka czyli **/dev/fd0**. W kolejnym kroku należy sprawdzić co jest na utworzonej dyskietce startowej. Aby to uczynić należy zamontować dyskietkę w systemie plików. Wykonujemy to pisząc polecenie:

```
$mount –t qnx4 /dev/fd0 /fs/a
```

Dyskietka umieszczona w napędzie **fd0** zostanie zamontowana w folderze **/fs** jako katalog **/fs/a**. Można zbadać jej zawartość za pomocą polecenia **ls**.

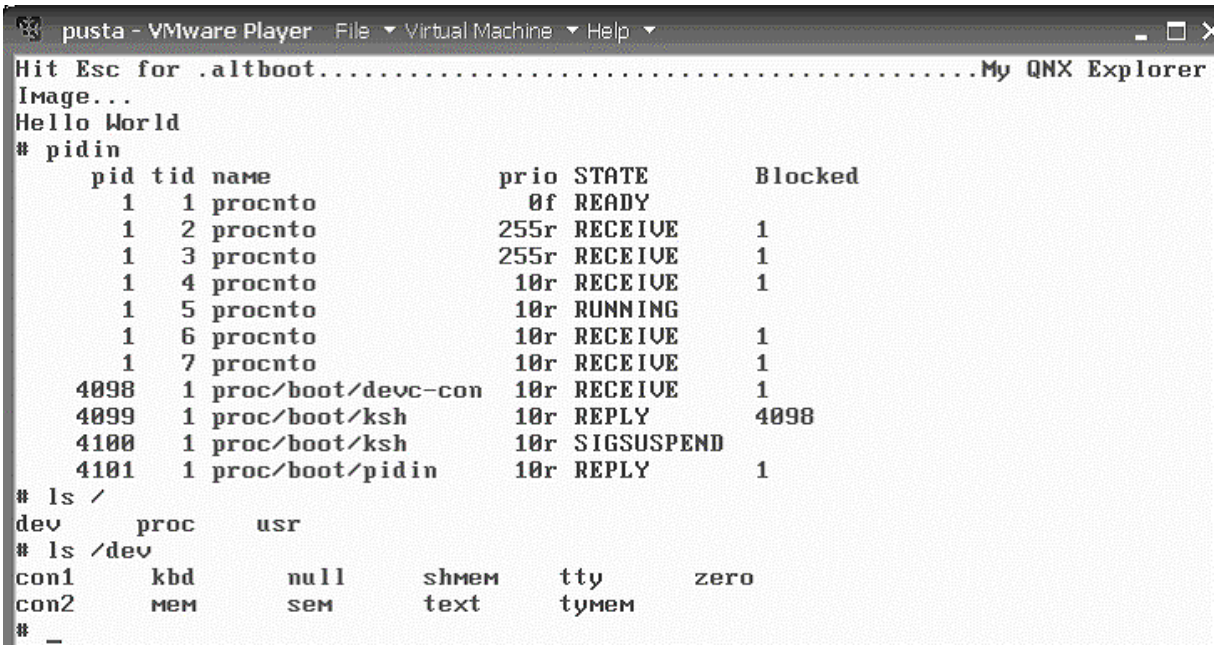
```
$ls –l /fs/a
total 1423
drwxrwxr-x  3 root  root   2048 Oct 12 12:39 .
drwxrwxr-x  3 root  root   2048 Oct 12 12:39 ..
-rw-------  1 root  root      0 Oct 12 12:39 .altboot
-r--r--r--  1 root  root    360 Oct 12 12:39 .bitmap
-rw-------  1 root  root 723432 Oct 12 12:39 .boot
-r--r--r--  1 root  root    512 Oct 12 12:39 .inodes
```

Wynik 2-1 Zawartość dyskietki startowej

Uprzednie polecenie **dinit –f bld1.ifs /dev/fd0** skopiowało plik obrazu systemu do pliku **.boot**.

Uruchamianie systemu

Działanie systemu możemy przetestować posługując się maszyną wirtualną. Tutaj użyto darmowej maszyny VMWare Player. Wyniki wczytywanie systemu pokazane są poniżej.



Przykład 2-3 Działanie systemu zbudowanego w oparciu o plik bld1.build

2.3 Przykład 2 – system wczytywany z dysku Flash USB

Na podstawie wzorca **bios.build** budowany będzie minimalny obraz systemu. Do programów wykonywalnych dołączony będzie przykładowy program **pcm_dido** który zmienia stan wyjść cyfrowych na karcie PCM3718. Program ten wystartuje automatycznie. Na jego miejsce można użyć dowolnego innego programu aplikacyjnego.

Aby otrzymać właściwy plik projektowy należy:

1. Skopiować plik **/boot/build/bios.build** do pliku **bios2.build** w tym samym folderze
2. Skopiować plik **pcm_dido** do katalogu **/usr/qnx632/target/qnx6/x86/bin**
3. Poddać edycji plik **bios2.build** otrzymując plik jak poniżej

```
# ONX Neutrino dla systemów x86 z BIOS
# Użyty minimalny system plików
# Startowany proces pcm_dido który zmienia stany
# wyjść cyfrowych karty PCM3718

[virtual=x86,bios +compress] boot = {
    startup-bios
    PATH=/proc/boot procnto -vv
}

[+script] startup-script = {
# Programy wymagają linkera czasu wykonania
ldqnx.so.2
    procmgr_symlink ../../proc/boot/libc.so.2
/usr/lib/ldqnx.so.2

# Start 4 konsoli
    devc-con -n4 &
    reopen /dev/con1
    display_msg Witamy w QNX Neutrino na PC z BIOS
    slogger &

# Start serwera pci
    seedres
    pci-bios &
    waitfor /dev/pci

# Start innych serwerow
```

```
    pipe &
    mqueue &
    devc-ser8250 -e &
    devc-pty &

# Start serwera debugowania na zlaczu szeregowym
    waitfor /dev/ser1
    [+session] pdebug /dev/ser1 &

# Te zmienne otoczenia będą odziedziczone przez
# wszystkie następne programy
    SYSNAME=nto
    TERM=qansi

# Start interpreterów poleceń shell na innych
# konsolach
    reopen /dev/con2
    [+session] sh &
    reopen /dev/con3
# Start naszej aplikacji do sterowanie diodami
    pcm_dido &

# Start glownego shella
    reopen /dev/con1
    [+session] sh
#   [+session] login -p
}

# Ustalenie folderu /temp w pamieci RAM
[type=link] /tmp=/dev/shmem

# Dolaczenie biblioteki dzielonej “c” zawierającej
# też runtime linker
libc.so

# Dołączenie biblioteki zmiennego przecinka
# fpemu.so.2

# Pliki powyżej mogą być dzielone przez wiele
procesów
[data=c]
# Programy wykonywalne muszą być poniżej tej linii
# Driver konsoli
devc-con
```

```
# Serwer pci
pci-bios
seedres

# Use the "public domain korn shell" as the default
shell "sh"
# Używamy ksh jako sh
sh=ksh
# Lub też używamy mniejszego shella fesh jako ”sh"
#sh=fesh
# Inne programy wykonywalne
pdebug
pipe
mqueue
devc-ser8250
devc-pty
ls
cat
pidin
uname
slogger
sloginfo
pcm_dido
```

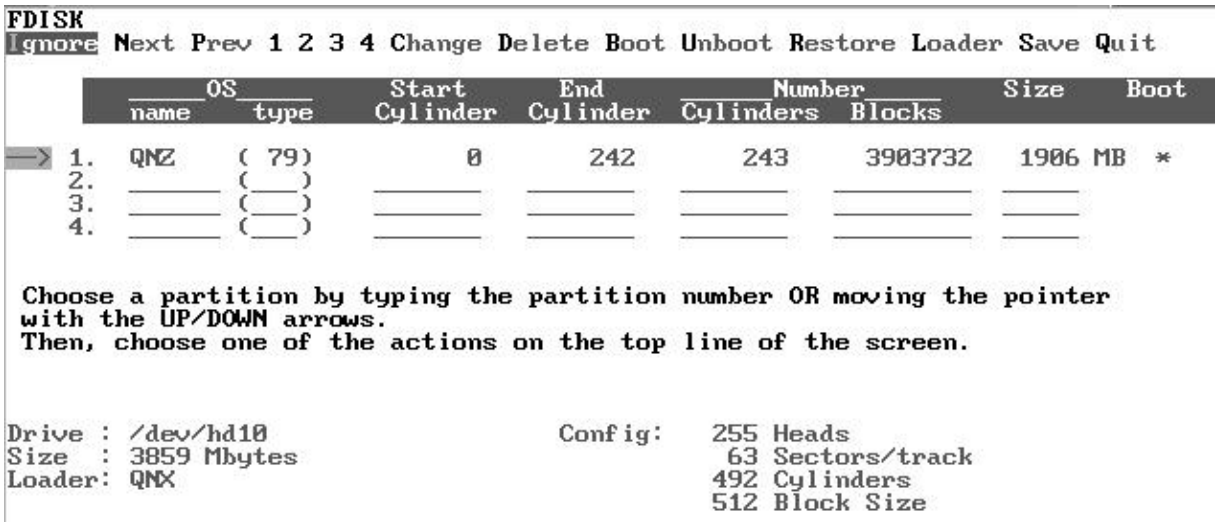
Przykład 2-4 Plik projektowy systemu dla komputerów PC z BIOS bios.build

Aby zbudować system na dysku USB lub flash należy:

1. Przejść do folderu **/boot/build**
   #cd /boot/build
2. Zbudować obraz systemu na podstawie pliku projektowego **bios2.build**
   # mkifs bios2.build bios2.ifs
   Powstanie obraz systemu **bios2.ifs**
3. Włożyć dysk USB lub flash w odpowiednim złączu. Powinien się on pojawić w katalogu **/dev** jako np. **/dev/hd10**
   #ls /dev
   hd10

   Uruchomić program **fdisk**
   #fdisk /dev/hd10

   Utworzyć bootowalną partycję typu 77, 78 lub 79



Przykład 2-5 Tworzenie partycji na dysku USB

4. Zainicjować na nowej partycji **/dev/hd10t79** system plików i przepisać plik **/boot/build/bios.ifs** jako obraz systemu. Będzie on na nowym dysku widoczny jako /.boot

```
#dinit -hq -f /boot/build/bios.ifs /dev/hd10t79
```

4. Zainstalować na dysku programy ładujące pierwszego i drugiego poziomu

```
#dloader /dev/hd10 pc1
#dloader /dev/hd10t77 pc2
```

Można sprawdzić czy pliki przepisały się prawidłowo i ewentualnie dopisać nowe pliki i programy

```
#ls  /fs/hd10-qnx4-1
#
# ls /fs/hd10-qnx4-1
.                .bitmap          .longfilenames
..               .boot            bin
.altboot         .inodes          etc
```

5. Umieścić dysk w komputerze docelowym i uruchomić go ponownie

W powyższym przykładzie użyty został program **dloader**.

dloader device loader

Program zapisuje na danym dysku określonym przez parametr **device** program ładujący **loader** poziomu 1 lub 2. Używa się 2 standardowych nazw programów ładujących

pc1 – standardowy program ładujący pierwszego poziomu dla PC (ładuje partycję)

pc2 – standardowy program ładujący drugiego poziomu dla QNX

Używa się także programów ładujących **pc1-flop** i **pc2-flop** dla dyskietek i dysków stałych pojemności do 8 GB.

2.4 Przesyłanie obrazu systemu do komputera docelowego

Po wytworzeniu obrazu systemu należy umieścić go na komputerze docelowym. Wyróżnić tu można dwa zasadniczo różne przypadki:

A) W komputerze docelowym istnieje BIOS, ROM monitor czy inny rodzaj programu ładującego
B) W komputerze docelowym brak jakiegokolwiek programu ładującego

W przypadku A) posłużyć się można następującymi metodami:

Załadować obraz systemu przy użyciu protokołu TFTP	Sieć ETHERNET
Załadować obraz systemu przy użyciu transmisji szeregowej	RS232
Zapisać obraz na dysku Compact Flash	Kontroler IDE
Zapisać obraz na dysku USB	Kontroler USB

Tab. 2-1 Ładowanie obrazu systemu w systemach z programem ładującym

W przypadku B) posłużyć się możliwe są następujące rozwiązania

Utworzyć system plików typu Flash i w nim umieścić obraz program ładujący IPL systemu	Zewnętrzny programator pamięci Flash
Jak wyżej	Programator JTAG

Tab. 2-2 Przeniesienie obrazu systemu w systemach bez programu ładującym

3. Dalsza konfiguracja systemu i pliki inicjalizacyjne

3.1 Konfiguracja systemu

Konfiguracja systemu polega na:
1. Ustanowieniu konsoli (do monitorowania pracy systemu)
2. Uruchomieniu programów sterujących urządzeniami (ang. *driverów*)
3. Uruchomieniu procesów aplikacyjnych

Ustanowienie konsoli

Konsola przydatna jest do obserwacji czy system i aplikacje zachowują się prawidłowo. Konsola może być uruchomiona na:
- Monitor + klawiatura
- Port szeregowy

W poprzednim pliku konfiguracyjnym systemu zawarte były polecenia:

```
devc-con -n2 &
reopen /dev/con2
[+session] ksh &
```

Pierwszy wiersz uruchamia sterownik konsoli, drugi powoduje że standartowe we/wy przekierowane będzie na urządzenie **`/dev/con2`** a trzeci wiersz uruchamia interpreter poleceń **`ksh`**. W podanym niżej przykładzie uruchamiamy konsolę na porcie szeregowym **`/dev/ser1`**

```
devc-ser8250 -e -b115200 &
reopen /dev/ser1
```

Użyte wyżej polecenie **`reopen`** jest wewnętrznym poleceniem dla narzędzia **`mkifs`**.

Uruchomieniu programów sterujących urządzeniami

Kolejnym krokiem będzie uruchomienie programów sterujących urządzeniami a w szczególności urządzeniami pamięciowymi na których rezydują inne programy. QNX wspiera następujące urządzenia:

dyski	**`devb-*`**
Pamięci flash	**`devf-*`**
Sieć	**`defn-*,defnp-*`**
Urządzenia wejściowe	**`devi-*`**
Urządzenia USB	**`devu-*`**
Systemy plików	**`fs-*`**

Być może drivery urządzeń będą potrzebowały obiektów dzielonych *.so. Powinny być one zawarte w obrazie systemu. Gdy driver urządzenia zostanie uruchomiony ładuje on automatycznie odpowiednią bibliotekę dzieloną. Drivery zaczynające się od **`fs-*`** obsługują poszczególne systemy plików. QNX zawiera następujące biblioteki związane z obsługą systemów plików:

MS-DOS	**`fs-dos.so`**
Linux	**`fs-ext2.so`**
Macintosh HFS I HFS Plus	**`fs-mac.so`**
Windows NT	**`fs-nt.so`**
QNX 4	**`fs-qnx4.so`**
Power-Safe	**`fs-qnx6.so`**
ISO-9660 CD-ROM, Universal Disk Format (UDF)	**`fs-udf.so`**

Tab. 3-1 Systemy plików i obsługujące je biblioteki

Pamięci flash

Aby uzyskać dostęp do sytemu plików flash należy uruchomić odpowiedni dla danego typu pamięci driver (zaczynają się od **`devf-*`).** Uniwersalny driver pamięci flash nazywa się **`devf-generic`**.

Drivery sieci

Driver sieci może być uruchamiany / zatrzymywany dynamicznie przez polecnie **`mount`**.

```
mount -T io-pkt devn-ne2000.so
```

Sieciowe systemy plików

QNX wspiera dwa sieciowe systemy plików:

- NFS – Umożliwia dostęp do sieciowego systemu plików typowego dla systemów UNIX (**`fs-nfs2, fs-nfs3`**)
- CIFS - Umożliwia dostęp do sieciowego systemu plików systemu Windows (fs-cifs),

Uruchamianie aplikacji

Aplikacje które mają być uruchomione powinny być zawarte w skrypcie inicjalizacyjnym po tym jak wymagane drivery zostaną uruchomione. Gdy aplikacja potrzebuje jakiegoś urządzenia może czekać na jego uruchomienie za pomocą polecenia **`waitfor`**.

3.2 Skrypt sysinit

1. Skrypt **sysinit** startuje procesy systemowe:
2. Program **slogger** gdy nie jest jeszcze uruchomiony.
3. Administrator kolejek **mqueue**.
4. Ustawia strefę czasową
5. Uruchamia program **/etc/rc.d/rc.rtc** ustawiający zegar RTC
6. Ustawia zmienną środowiska $HOSTNAME
7. Uruchamia skrypt **/etc/rc.d/rc.devices** znajdujący urządzenia i uruchamia proces **io-net** zarządzania siecią.
8. Gdy plik **/etc/system/config/useqnet** istnieje uruchamia administrator sieci Qnet **npm-qnet.so**
9. Uruchamia skrypt **/etc/rc.d/rc.sysinit**.
10. Gdy się to uda uruchamia shell **sh** lub gdy się nie uda **fesh**.

/etc/rc.d/rc.sysinit	Startuje procesy systemowe
/etc/rc.d/rc.rtc	Ustawia czas
/etc/rc.d/rc.devices	Uruchamia urządzenia
/etc/rc.d/rc.local	Uruchamia ustawienia lokalne (tutaj powinny być umieszczone procesy użytkownika)

Tab. 3-2 Skrypty startowe systemu QNX6 Neutrino

3.3 Skrypt rc.sysinit

Skrypt **/etc/rc.d/rc.sysinit** wykonuje następujące funkcje:

1. Startuje generator liczb losowych **random.**
2. Gdy istnieje katalog **/var/dumps** startuje proces **dumper.**
3. Gdy istnieje plik **/etc/host_cfg/$HOSTNAME/rc.d/rc.local** i jest to plik wykonywalny to go startuje. Gdy nie ma takiego pliku startuje **/etc/rc.d/rc.local** gdy plik ten istnieje i jest wykonywalny. W plikach tych użytkownik może uruchamiać własne programy
4. Uruchamia proces **tinit**. Domyślnie starowany jest Photon ale gdy utworzymy plik **/etc/system/config/nophoton** wystartuje powłoka w trybie tekstowym.

Przykład pliku **/etc/rc.d/rc.local**

```
inetd &
gns –c
```

3.4 Skrypt rc.devices

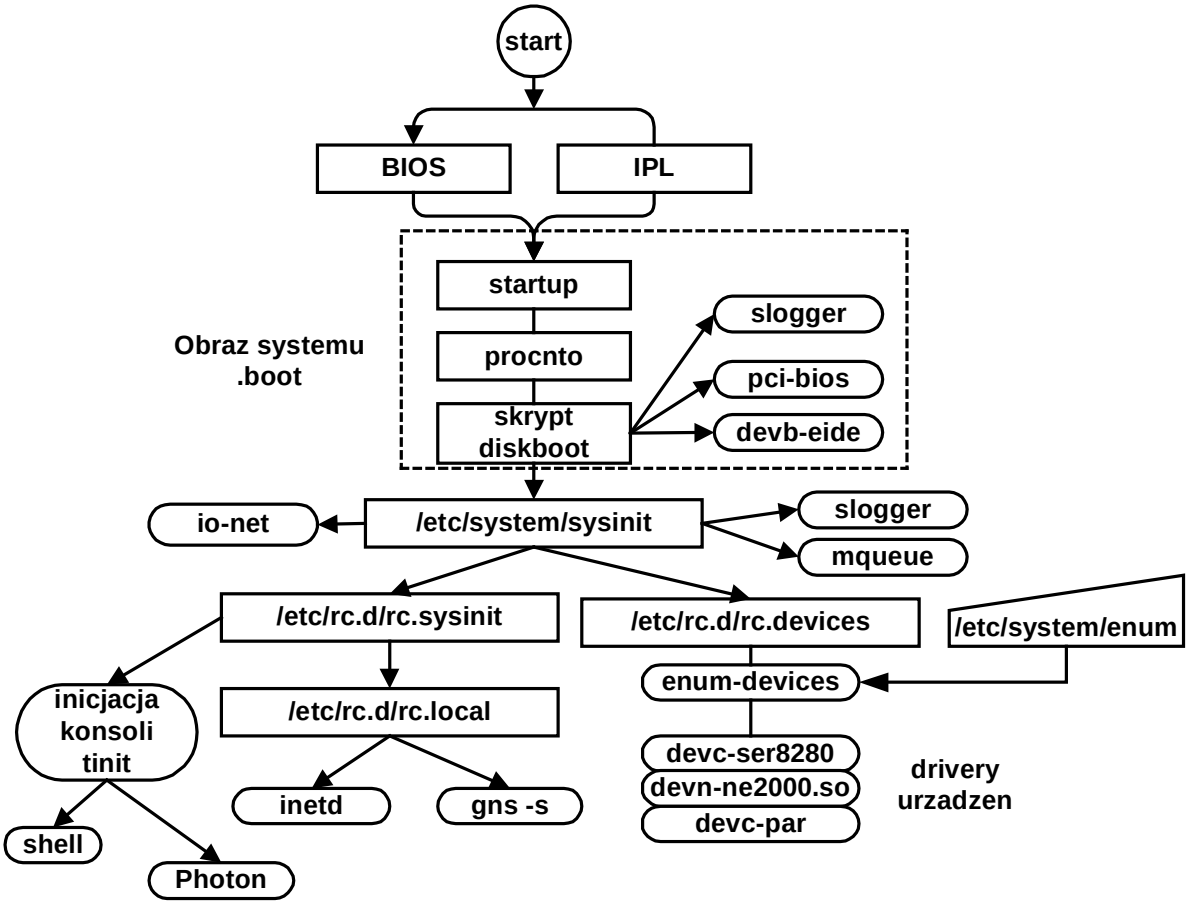
Skrypt **/etc/rc.d/rc.devices** startuje program **enum-devices** aby wykryć zainstalowany sprzęt i uruchomić potrzebne drivery. Po wykryciu danego urządzenia wykonywane są akcje wyspecyfikowane w plikach katalogu **/etc/system/enum**
Katalog ten zawiera podkatalog **devices** w którym są pliki **net, block, audio, bridge, char, graphic, input, media, printer, serial**. W plikach są informacje dla którego urządzenia jaki driver uruchomić.
Przykładowo plik **/etc/system/enum/devices/net** wskazuje jaki driver karty uruchomić i startuje program **netmanager**. Program ten ustala konfigurację sieci pobieraną z pliku **/etc/net.cfg**

```
# nto network config file v1.2
version v1.2

[global]
hostname compaq1
domain telsat.wroc.pl
nameserver 194.54.16.34
nameserver 192.168.0.1
nameserver 192.168.1.1
route 192.168.0.1 0.0.0.0 0.0.0.0

[en0]
type ethernet
mode dhcp
```

Przykład 3-1 Plik konfiguracji sieci /etc/net.cfg

3.5 Plik startowy Photona

Gdy startuje Photon to wykonuje skrypt **$HOME/.ph/phapps** (o ile istnieje i ma atrybut wykonywalności).
Np. plik **/root/.ph/phapps** zawiera:
/usr/photon/bin/devi-penmount dmc9000 –i fd –d/dev/ser3 abs &

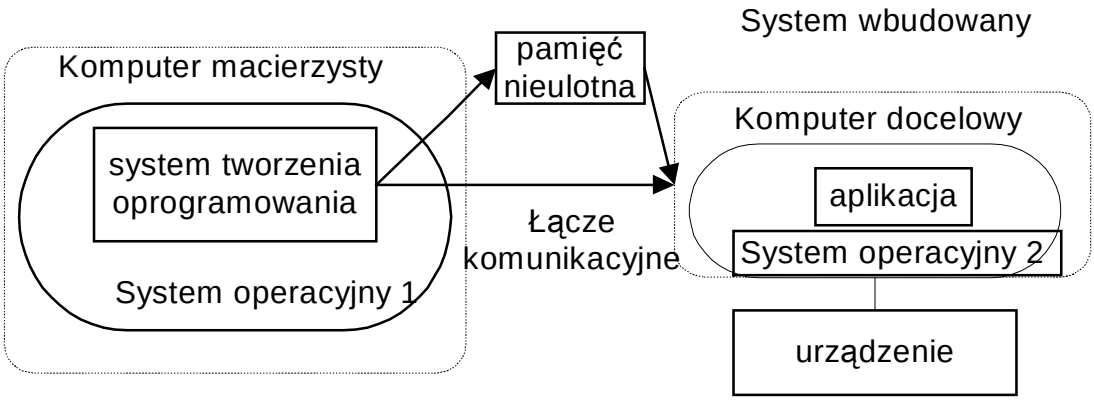


Rys. 3-1 Start systemu QNX6 Neutrino

Instalacja i konfiguracja systemu operacyjnego

1. QNX6 w systemach wbudowanych

W systemach wbudowanych oprogramowanie tworzone jest na tak zwanym komputerze macierzystym (*ang. host*) a wykonywane na komputerze docelowym (*ang. target*).



Rys. 1-1 System skrośnego rozwoju oprogramowania

Aplikacje mogą być tworzone w środowiskach systemów:

- QNX6 Neutrino, Windows (NT, 2000, XP),
- Linux,
- Solaris SPARC.

Narzędziem do tworzenia takich aplikacji jest Momentics Development Suite. Środowisko to zawiera:

1. Zintegrowane środowisko rozwoju oprogramowania IDE (*ang. Integrated Development Environment*).
2. Biblioteki ANSI C, POSIX, DINKUM C++, GNU C++.
3. Dokumentację.
4. Narzędzia BSP (*ang. Board Suport Packages*) służące do dostosowania systemu QNX6 Neutrino do różnorodnych środowisk sprzętowych.
5. Narzędzia DDK (*ang. Driver Development Kits)* wspomagające tworzenie sterowników urządzeń zewnętrznych (grafiki, audio, sieciowe, USB).
6. Narzędzia TDK (*ang. Technology Development Kits)* które ułatwiają tworzenie wbudowanych systemów plików, systemów wieloprocesorowych SMP, obsługi multimediów i grafiki 3D.

System docelowy na komputerach z procesorami:

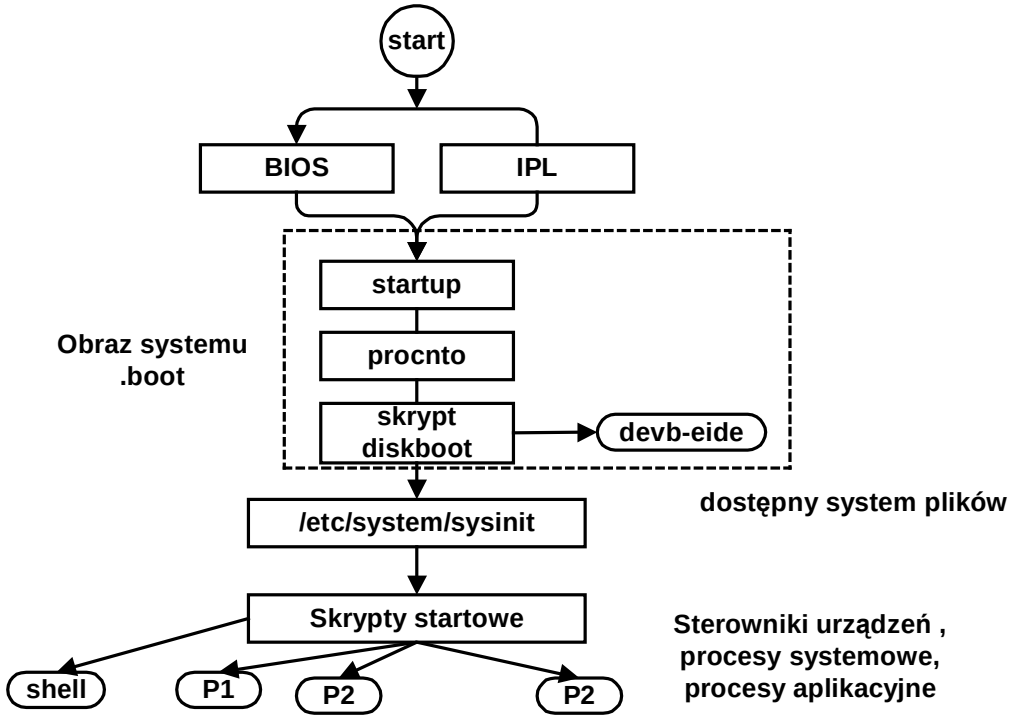
- ARM,
- MIPS, PowerPC,
- SH-4,
- StrongARM (w tym Xscale)
- x86.

2. Pojęcia podstawowe, wczytywanie systemu

2.1 Program ładowania wstępnego IPL i BIOS

Gdy procesor startuje brak jeszcze systemu operacyjnego – musi on być załadowany, skonfigurowany i wystartowany. Role te pełni IPL (ang. *Initial Program Loader*). Funkcje IPL to:

- Start od obszaru kodu wskazywanego przez „reset vector”
- Konfiguracja kontrolera pamięci, kontrolera PCI i innych kontrolerów
- Konfiguracja zegara
- Kopiuje obraz systemu do pamięci operacyjnej
- Wykonuje skok do początku obrazu systemu



Rys. 2-1 Start systemu QNX6 Neutrino

Wyróżnia się dwa rodzaje sposobów pracy IPL:

- Zimny start – pierwszy program jaki się w komputerze wykonuje, składniki sprzętu nie są zainicjowane
- Ciepły start – program uruchomiony przez BIOS lub monitor zawarty w pamięci ROM, pewne składniki sprzętu są już zainicjowane.

Po zakończeniu pracy IPL wymagany jest następujący stan systemu:
- Kontroler pamięci jest skonfigurowany i umożliwia do niej dostep
- Niezbędne minimum konfiguracji sprzętu zostało wykonane
- Obraz systemu operacyjnego umieszczony jest w liniowo adresowalnej pamięci
- Początkowa część obrazu systemu operacyjnego umieszczona jest w pamięci RAM

2.2 BIOS i ładowanie systemu operacyjnego

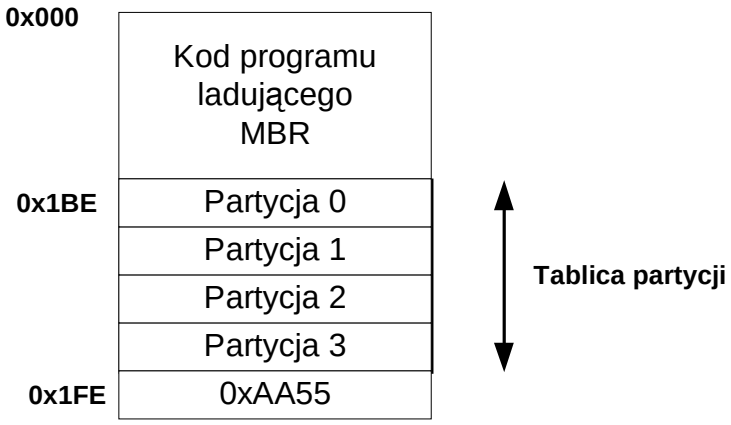
BIOS (ang. *Basic Input Output System*) jest programem który startuje jako pierwszy po włączeniu komputera. Zapewnia dostęp do podstawowych komponentów komputera. Jego funkcje to:

1. Dostarczenie interfejsu do konfiguracji systemu (BIOS Setup)
2. Wczytywanie systemu operacyjnego
3. Diagnostyka systemu

Po włączeniu zasilania BIOS wykonuje następujące czynności:
1. Sprawdzenie integralności kodu programu BIOS
2. Ustalenie rozmiaru i sprawdzenie pamięci głównej
3. Znalezienie, zainicjalizowanie i skatalogowanie wszystkich magistrali
4. Dostarczenie interfejsu do konfiguracji systemu (BIOS Setup)
5. Zidentyfikowanie urządzeń zdolnych do wczytywania systemu (dyski IDE, SATA, Floppy, CDROM, FLASH).
6. Załadowane z wybranego urządzenia (użytkownik może określić z którego urządzenia dysku należy załadować system operacyjny gdy urządzeń jest więcej) pierwszych 512 bajtów do pamięci operacyjnej pod adres 0000:7C00. Bajty ładowane są z obszaru znajdującego się na samym początku dysku (sektor 0, cylinder 0, powierzchnia 0). W obszarze ty znajduje się pierwotny program ładujący MBR (ang. *Master Boot Record*) i tablica partycji dysku.
7. Następuje przekazanie sterowania do programu ładującego zawartego w MBR (rozkazu znajdującego się pod adresem

0000:7C00). Program ten stwierdza która partycja jest aktywna i ładuje do pamięci operacyjnej zawarty na jej początku program ładujący (ang. *Boot Record*). Program ten jest specyficzny dla systemu operacyjnego i wie jak go załadować. Dalsze ładowanie systemu przeprowadza program ładujący.

Znane są programy ładujące które mogą wybrać inną partycję dysku z której ma być załadowany system operacyjny. Przykładem jest LILO lub GRUB, U-boot.



Rys. 2-2 Struktura sektora ładującego MBR

1	Flaga aktywności
2	Początek partycji
3	Typ partycji
4	Koniec partycji
5	Sektor początkowy partycji
6	Liczba sektorów partycji

Rys. 2-3 **Zawartość tablicy partycji**

Jedna partycja oznaczona jest jako aktywna. Zawiera ona program ładujący.

2.3 Program ładujący i obraz systemu

Obraz systemu jest plikiem o nazwie **`/.boot`** a zapasowym plikiem **`/.altboot`**. Zawiera on:

- Program startowy (ang. *startup program)*
- Mikrojądro i administrator procesów
- Programy i skrypty startowe

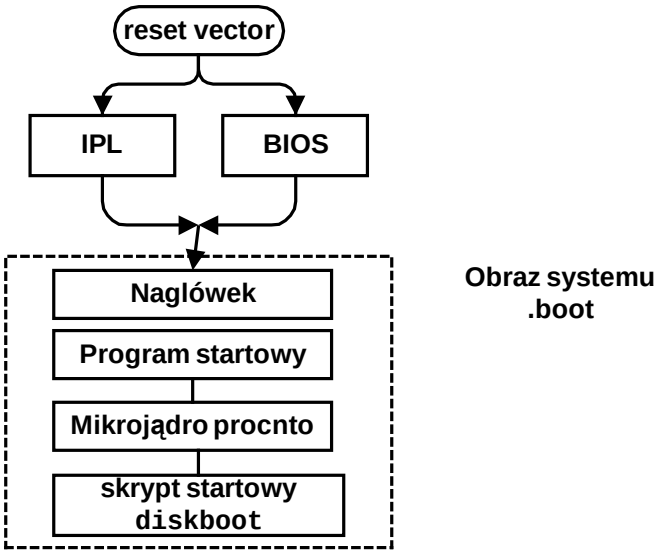
Funkcje wykonywane przez program startowy to:

- Inicjalizacja sprzętu
- Wypełnia „stronę systemową ” zawierającą informacje o sprzęcie
- Kopiowanie do pamięci operacyjnej dalszą część obrazu systemu (mikrojądro (ang. *microkernel*) i administrator procesów **`procnto`**) i jego dekompresja (gdy konieczna)
- Startuje procesy zawarte w **`boot scripts`** które dokonują dalszego ustanawiania środowiska sprzętowego i programowego poprzez wykonanie skryptów i uruchomienie programów w C

Inicjalizacja sprzętu

Ustanowienie wywołań systemowych do mikrojądra

- Interfejs do debugowania
- Interfejs do zegara i czasomierza
- Interfejs kontrolera przerwań
- Interfejs kontrolera pamięci podręcznej
- Interfejs zarządzania zasilaniem



Rys. 2-4 Pierwsza faza startu systemu

Skrypt startowy **diskboot**

Typowy skrypt **diskboot** wykonuje następujące funkcje:

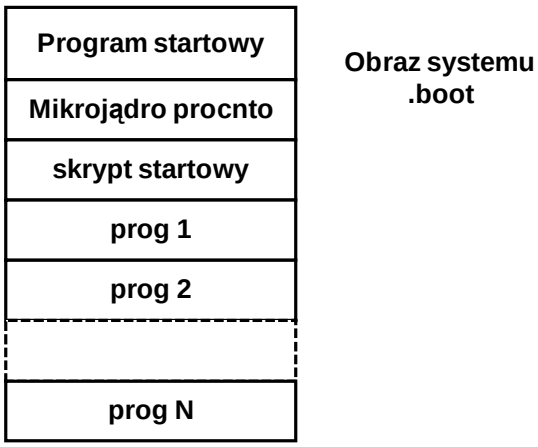
1. Startuje proces logujący **slogger** zapisujący zdarzenia przy starcie systemu. Zawartość tego pliku może być odczytana poleceniem **sloginfo**.
2. Uruchamia program **seedres** który odczytuje ustawienia PnP BIOS i przekazuje je do **procnto**.
3. Uruchamia program **pci-bios**
4. Startuje driver dysku **dev-eide** (lub inny driver dysku)
5. Montuje systemy plików : CD-ROM, DOS, Ext2, QNX4. Systemy plików montowane są w katalogu **/fs**.
6. Opcjonalnie uruchamia shell **fesh** dla systemów wbudowanych.
7. Staruje driver konsoli **devc-con-hid**
8. Uruchamia skrypt **/etc/system/sysinit**

3. Obraz systemu i jego tworzenie

Obraz systemu .**boot** jest plikiem zawierającym programy wykonywalne i dane. Może być rozpatrywany jako mały system plików zawierający foldery i pliki. W szczególności obraz systemu zawiera:

- Program ładujący
- Mikrojądro
- Administrator procesów **procnto**

Inne programy i skrypty startowe



Obraz systemu może być ładowalny (ang. *bootable*) i nie ładowalny (ang. *non bootable*). Obraz ładowalny zawierać musi program ładujący. Obraz systemu tworzony jest za pomocą narzędzia **mkifs**.

Zawartość pliku **boot** może być sprawdzona za pomocą polecenia:

ls /proc/boot

```
#ls /proc/boot
.script     ping     cat    data1        pidin
ksh         ls       ftp    procnto      devc-
ser8250-ixp2400
```

Przykład 3-1 Sprawdzanie zawartości obrazu systemu

3.1 Konfiguracja obrazu systemu

Obraz systemu tworzony jest przez narzędzie **mkifs** na podstawie pliku projektowego (ang. *make image filesystem*) i plików zawartych w systemie.

```
mkifs buildfile [imagefile] [-v]
```

Gdzie:

buildfile – plik projektowy systemu
imagefile – plik z obrazem systemu
-v - rozszerzone raportowanie

Plik projektowy systemu (ang. *buildfile*) zawiera reguły według których system jest budowany. Plik projektowy systemu zawiera:

- Skrypt ładowania systemu (ang.*bootstrap script*)
- Skrypt startowy (ang. *startup script*)
- Listę plików (ang. *file list*)
- Listę plików do odlinkowania (ang. *unlink list*) – opcja

Skrypt ładowania systemu

Sposób ładowania systemu zależy od typu procesora. Skrypt ładowania systemu specyfikuje jaki program ma być użyty do ładowania systemu i jakie jego opcje mają być aktywne.

```
[virtual=x86,bios +compress] .bootstrap={
  startup-bios -s 64k
  PATH=/proc/boot LD_LIBRARY_PATH=/proc/boot:/usr/lib
  procnto
}
```

Pierwsza linia informuje że jest to skrypt startowy, część w nawiasach specyfikuje atrybuty:

virtual=x86 – praca w trybie wirtualnym procesora x86
bios - system z BIOS
+compress - użyty skompresowany obraz systemu

Program **startup-bios** wykonuje dekompresję obrazu systemu, załadowanie go do RAM i przekazanie sterowania do jądra (moduł **procnto**).

Plik **procnto** zawiera mikrojądro. Jest ono odpowiedzialne za zarządzanie pamięcią, procesami, i komunikację międzyprocesową. Wartości zmiennych **PATH** i **LD_LIBRARY_PATH** są dziedziczone przez procesy potomne i wskazują gdzie szukać programów i bibliotek. Od tego momentu system może tworzyć nowe procesy.

Skrypt startowy

Skrypt startowy wykonywany jest po uruchomieniu jądra i startuje niezbędne programy obsługi urządzeń. Przykład podany jest poniżej.

```
[+script] .script={
  seedres
  devc-con -n2 &
  reopen /dev/con2
  [+session] ksh &
  reopen /dev/con1
  [+session] ksh &
 }
```

seedres - program odczytuje z PnP BIOS dane o konfiguracji sprzętowej (portach we/wy przerwaniach) i przekazuje do bazy danych jądra

devc-con - uruchomienie programu obsługi konsoli

reopen - otwiera stdin, stdout, stderr i przypisuje je do danego urządzenia

[+session] ksh & - startuje nową kopię interpretera poleceń i czyni ją liderem sesji

Lista plików

Kolejną częścią obrazu systemu jest lista plików które są potrzebne na tym etapie pracy systemu. Fragment listy plików dany jest poniżej.

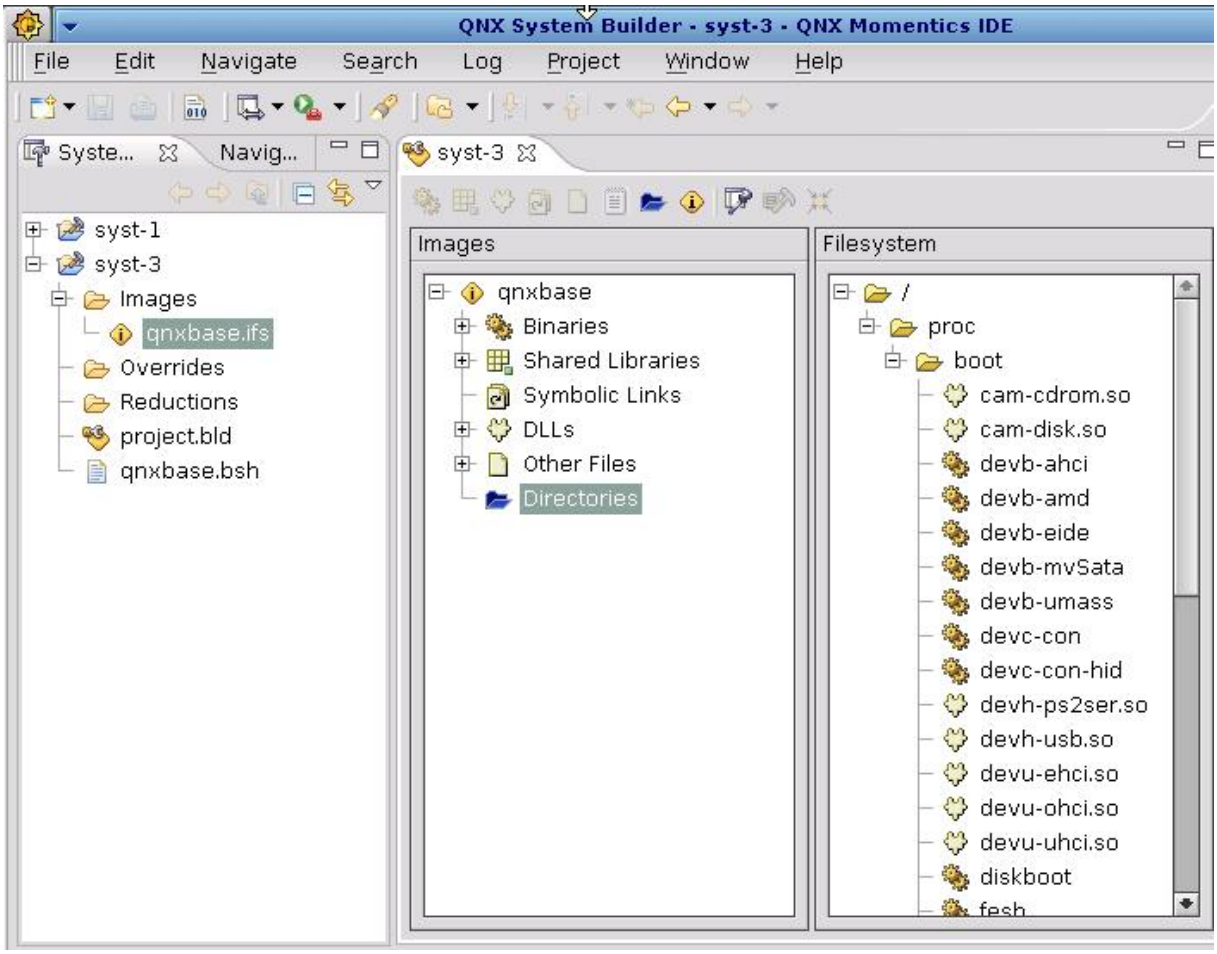
```
libc.so
# Dodanie linku symbolicznego
[type=link] /usr/lib/ldqnx.so.2=/proc/boot/libc.so
[code=uip data=copy perms=+r,+x]
seedres
kill
…
```

libc.so - Należy w obrazie systemu umieścić tę bibliotekę gdyż jest ona potrzebna w większości programów w języku C.

`[type=link] /usr/lib/ldqnx.so.2=/proc/boot/libc.so –` Różne aplikacje mogą się spodziewać plików w różnych miejscach. Aby ich nie kopiować tworzy się linki

`[code=uip data=copy perms=+r,+x]` – Wymienione dalej programy będą umieszczone w pamięci operacyjnej. Nie ma więc potrzeby kopiować ich segmentu kodu a tylko segment danych. Sekcja **`perms=+r,+x`** mówi że pliki mają prawo odczytu i wykonania.

Obraz systemu można również uzyskać posługując się narzędziem Momentics (perspektywa *System builder*)



Rys. 3-1 Zastosowanie narzędzia Momentics (perspektywa System builder) do generowania obrazu systemu operacyjnego

Narzędzie umożliwia:
Dodawanie / usuwanie składników do obrazu systemu jak:

- Programy binarne
- Sterowniki urządzeń
- Biblioteki

Edycję systemu plików

3.2 Przykład 1 – system wczytywany z dyskietki

Przykład kompletnego pliku projektowego dla systemu x86 podano poniżej. System umożliwia wystartowanie powłoki **ksh** i prostą eksplorację systemu za pomocą poleceń: **ps, pidin, sin.**

```
[virtual=x86,bios +compress] .bootstrap={
  startup-bios -s 64k
  PATH=/proc/boot LD_LIBRARY_PATH=/proc/boot:/usr/lib
  procnto
}
[+script] .script={
  seedres
  display_msg "Moj obraz syst QNX..."
  display_msg "Dzien dobry"
  # dwie wirtualne konsole
  # przelaczanie Ctrl+Alt+1 and Ctrl+alt+2
  devc-con -n2 &
  reopen /dev/con2
  [+session] ksh &
  reopen /dev/con1
  [+session] ksh &
}
libc.so
# Dodanie linku symbolicznego
[type=link] /usr/lib/ldqnx.so.2=/proc/boot/libc.so
[code=uip data=copy perms=+r,+x]
seedres
kill
cat
ls
ksh
devc-con
less
ps
sin
pidin
#Tutaj dodajemy edytor...
vi
```

Przykład 3-2 Prosty plik projektowy „buldfile” bld1.build

Budowanie systemu

Po edycji pliku projektowego **bld1.build** tworzymy obraz systemu pisząc:

```
$ mkifs -v bld1.build bld1.ifs
```

Gdy budowa się powiedzie powstanie plik **bld1.ifs** będący obrazem systemu.

Tworzenie dyskietki startowej

Gdy system zostanie utworzony należy sprawdzić jego działanie. Tak więc należy utworzyć taki nośnik z którego da się wczytać system. Najłatwiej będzie użyć dyskietki. W tym celu przygotowujemy dyskietkę 3.5 ” i wkładamy ją do napędu dyskietek. Następnie dokonujemy inicjacji dyskietki z przeniesieniem obrazu systemu. Używamy polecenia **dinit**:

```
$dinit –f bld1.ifs /dev/fd0
```

po przełączniku –f umieszczona jest nazwa pliku który jest obrazem systemu – w tym przypadku jest to plik **bld1.ifs**. Drugi parametr określa urządzenie do którego włożona jest dyskietka czyli **/dev/fd0**. W kolejnym kroku należy sprawdzić co jest na utworzonej dyskietce startowej. Aby to uczynić należy zamontować dyskietkę w systemie plików. Wykonujemy to pisząc polecenie:

```
$mount –t qnx4 /dev/fd0 /fs/a
```

Dyskietka umieszczona w napędzie **fd0** zostanie zamontowana w folderze **/fs** jako katalog **/fs/a**. Można zbadać jej zawartość za pomocą polecenia **ls**.

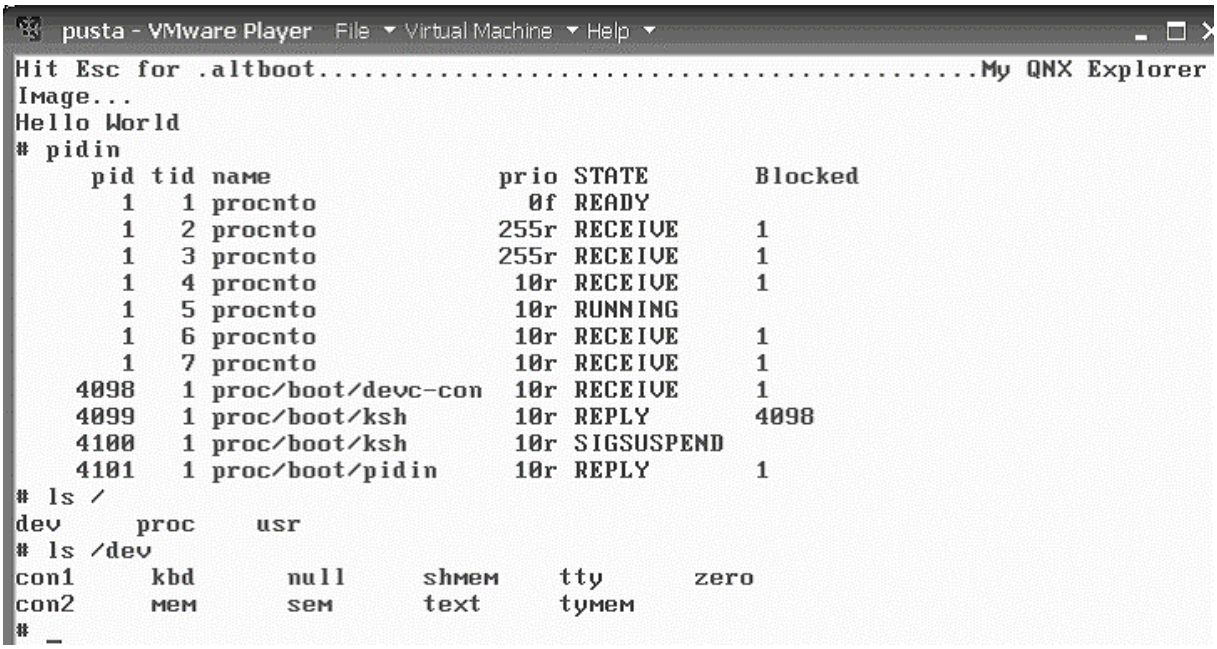
```
$ls –l /fs/a
total 1423
drwxrwxr-x  3 root  root   2048 Oct 12 12:39 .
drwxrwxr-x  3 root  root   2048 Oct 12 12:39 ..
-rw-------  1 root  root      0 Oct 12 12:39 .altboot
-r--r--r--  1 root  root    360 Oct 12 12:39 .bitmap
-rw-------  1 root  root 723432 Oct 12 12:39 .boot
-r--r--r--  1 root  root    512 Oct 12 12:39 .inodes
```

Wynik 3-1 Zawartość dyskietki startowej

Uprzednie polecenie **dinit –f bld1.ifs /dev/fd0** skopiowało plik obrazu systemu do pliku **.boot**.

Uruchamianie systemu

Działanie systemu możemy przetestować posługując się maszyną wirtualną. Tutaj użyto darmowej maszyny VMWare Player. Wyniki wczytywanie systemu pokazane są poniżej.



Przykład 3-3 Działanie systemu zbudowanego w oparciu o plik bld1.build

3.3 Przykład 2 – system wczytywany z dysku Flash USB

Na podstawie wzorca **bios.build** budowany będzie minimalny obraz systemu. Do programów wykonywalnych dołączony będzie przykładowy program **pcm_dido** który zmienia stan wyjść cyfrowych na karcie PCM3718. Program ten wystartuje automatycznie. Na jego miejsce można użyć dowolnego innego programu aplikacyjnego.

Aby otrzymać właściwy plik projektowy należy:

1. Skopiować plik **/boot/build/bios.build** do pliku **bios2.build** w tym samym folderze
2. Skopiować plik **pcm_dido** do katalogu **/usr/qnx632/target/qnx6/x86/bin**
3. Poddać edycji plik **bios2.build** otrzymując plik jak poniżej

```
# ONX Neutrino dla systemów x86 z BIOS
# Użyty minimalny system plików
# Startowany proces pcm_dido który zmienia stany
# wyjść cyfrowych karty PCM3718

[virtual=x86,bios +compress] boot = {
    startup-bios
    PATH=/proc/boot procnto -vv
}

[+script] startup-script = {
# Programy wymagają linkera czasu wykonania
ldqnx.so.2
    procmgr_symlink ../../proc/boot/libc.so.2
/usr/lib/ldqnx.so.2

# Start 4 konsoli
    devc-con -n4 &
    reopen /dev/con1
    display_msg Witamy w QNX Neutrino na PC z BIOS
    slogger &

# Start serwera pci
    seedres
    pci-bios &
    waitfor /dev/pci

# Start innych serwerow
```

```
    pipe &
    mqueue &
    devc-ser8250 -e &
    devc-pty &

# Start serwera debugowania na zlaczu szeregowym
    waitfor /dev/ser1
    [+session] pdebug /dev/ser1 &

# Te zmienne otoczenia będą odziedziczone przez
# wszystkie następne programy
    SYSNAME=nto
    TERM=qansi

# Start interpreterów poleceń shell na innych
# konsolach
    reopen /dev/con2
    [+session] sh &
    reopen /dev/con3
# Start naszej aplikacji do sterowanie diodami
    pcm_dido &

# Start glownego shella
    reopen /dev/con1
    [+session] sh
#   [+session] login -p
}

# Ustalenie folderu /temp w pamieci RAM
[type=link] /tmp=/dev/shmem

# Dolaczenie biblioteki dzielonej “c” zawierającej
# też runtime linker
libc.so

# Dołączenie biblioteki zmiennego przecinka
# fpemu.so.2

# Pliki powyżej mogą być dzielone przez wiele
procesów
[data=c]
# Programy wykonywalne muszą być poniżej tej linii
# Driver konsoli
devc-con
```

```
# Serwer pci
pci-bios
seedres

# Use the "public domain korn shell" as the default
shell "sh"
# Używamy ksh jako sh
sh=ksh
# Lub też używamy mniejszego shella fesh jako ”sh"
#sh=fesh
# Inne programy wykonywalne
pdebug
pipe
mqueue
devc-ser8250
devc-pty
ls
cat
pidin
uname
slogger
sloginfo
pcm_dido
```

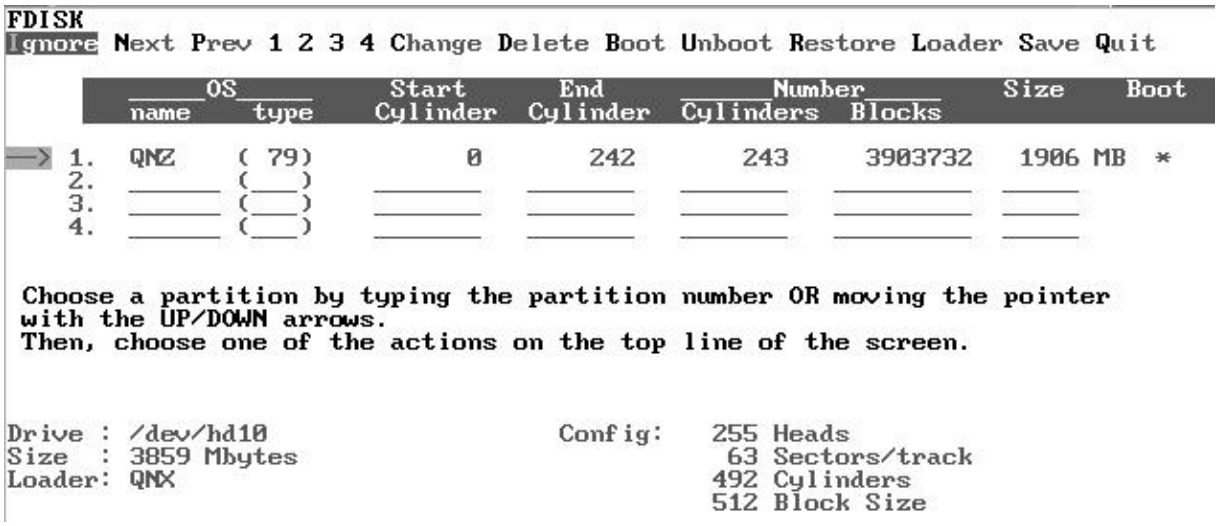
Przykład 3-4 Plik projektowy systemu dla komputerów PC z BIOS bios.build

Aby zbudować system na dysku USB lub flash należy:

1. Przejść do folderu **/boot/build**
   #cd /boot/build
2. Zbudować obraz systemu na podstawie pliku projektowego **bios2.build**
   # mkifs bios2.build bios2.ifs
   Powstanie obraz systemu **bios2.ifs**
3. Włożyć dysk USB lub flash w odpowiednim złączu. Powinien się on pojawić w katalogu **/dev** jako np. **/dev/hd10**
   #ls /dev
   hd10

   Uruchomić program **fdisk**
   #fdisk /dev/hd10

   Utworzyć bootowalną partycję typu 77, 78 lub 79



Przykład 3-5 Tworzenie partycji na dysku USB

4. Zainicjować na nowej partycji **/dev/hd10t79** system plików i przepisać plik **/boot/build/bios.ifs** jako obraz systemu. Będzie on na nowym dysku widoczny jako /.boot

   #dinit -hq -f /boot/build/bios.ifs /dev/hd10t79

4. Zainstalować na dysku programy ładujące pierwszego i drugiego poziomu

   #dloader /dev/hd10 pc1
   #dloader /dev/hd10t77 pc2

   Można sprawdzić czy pliki przepisały się prawidłowo i ewentualnie dopisać nowe pliki i programy

   #ls /fs/hd10-qnx4-1

```
#
# ls /fs/hd10-qnx4-1
.                .bitmap          .longfilenames
..               .boot            bin
.altboot         .inodes          etc
```

5. Umieścić dysk w komputerze docelowym i uruchomić go ponownie

W powyższym przykładzie użyty został program **dloader**.
dloader device loader

Program zapisuje na danym dysku określonym przez parametr **device** program ładujący **loader** poziomu 1 lub 2. Używa się 2 standardowych nazw programów ładujących

pc1 – standardowy program ładujący pierwszego poziomu dla PC (ładuje partycję)

pc2 – standardowy program ładujący drugiego poziomu dla QNX

Używa się także programów ładujących **pc1-flop** i **pc2-flop** dla dyskietek i dysków stałych pojemności do 8 GB.

3.4 Przesyłanie obrazu systemu do komputera docelowego

Po wytworzeniu obrazu systemu należy umieścić go na komputerze docelowym. Wyróżnić tu można dwa zasadniczo różne przypadki:

A) W komputerze docelowym istnieje BIOS, ROM monitor czy inny rodzaj programu ładującego
B) W komputerze docelowym brak jakiegokolwiek programu ładującego

W przypadku A) posłużyć się można następującymi metodami:

Załadować obraz systemu przy użyciu protokołu TFTP	Sieć ETHERNET
Załadować obraz systemu przy użyciu transmisji szeregowej	RS232
Zapisać obraz na dysku Compact Flash	Kontroler IDE
Zapisać obraz na dysku USB	Kontroler USB

Tab. 3-1 Ładowanie obrazu systemu w systemach z programem ładującym

W przypadku B) posłużyć się możliwe są następujące rozwiązania

Utworzyć system plików typu Flash i w nim umieścić obraz program ładujący IPL systemu	Zewnętrzny programator pamięci Flash
Jak wyżej	Programator JTAG

Tab. 3-2 Przeniesienie obrazu systemu w systemach bez programu ładującym

4. Dalsza konfiguracja systemu i pliki inicjalizacyjne

4.1 Konfiguracja systemu

Konfiguracja systemu polega na:
1. Ustanowieniu konsoli (do monitorowania pracy systemu)
2. Uruchomieniu programów sterujących urządzeniami (ang. *driverów*)
3. Uruchomieniu procesów aplikacyjnych

Ustanowienie konsoli

Konsola przydatna jest do obserwacji czy system i aplikacje zachowują się prawidłowo. Konsola może być uruchomiona na:
- Monitor + klawiatura
- Port szeregowy

W poprzednim pliku konfiguracyjnym systemu zawarte były polecenia:

```
devc-con -n2 &
reopen /dev/con2
[+session] ksh &
```

Pierwszy wiersz uruchamia sterownik konsoli, drugi powoduje że standartowe we/wy przekierowane będzie na urządzenie **`/dev/con2`** a trzeci wiersz uruchamia interpreter poleceń **`ksh`**. W podanym niżej przykładzie uruchamiamy konsolę na porcie szeregowym **`/dev/ser1`**

```
devc-ser8250 -e -b115200 &
reopen /dev/ser1
```

Użyte wyżej polecenie **`reopen`** jest wewnętrznym poleceniem dla narzędzia **`mkifs`**.

Uruchomieniu programów sterujących urządzeniami

Kolejnym krokiem będzie uruchomienie programów sterujących urządzeniami a w szczególności urządzeniami pamięciowymi na których rezydują inne programy. QNX wspiera następujące urządzenia:

dyski	**`devb-*`**
Pamięci flash	**`devf-*`**
Sieć	**`defn-*,defnp-*`**
Urządzenia wejściowe	**`devi-*`**
Urządzenia USB	**`devu-*`**
Systemy plików	**`fs-*`**

Być może drivery urządzeń będą potrzebowały obiektów dzielonych *.so. Powinny być one zawarte w obrazie systemu. Gdy driver urządzenia zostanie uruchomiony ładuje on automatycznie odpowiednią bibliotekę dzieloną. Drivery zaczynające się od **`fs-*`** obsługują poszczególne systemy plików. QNX zawiera następujące biblioteki związane z obsługą systemów plików:

MS-DOS	**`fs-dos.so`**
Linux	**`fs-ext2.so`**
Macintosh HFS I HFS Plus	**`fs-mac.so`**
Windows NT	**`fs-nt.so`**
QNX 4	**`fs-qnx4.so`**
Power-Safe	**`fs-qnx6.so`**
ISO-9660 CD-ROM, Universal Disk Format (UDF)	**`fs-udf.so`**

Tab. 4-1 Systemy plików i obsługujące je biblioteki

Pamięci flash

Aby uzyskać dostęp do sytemu plików flash należy uruchomić odpowiedni dla danego typu pamięci driver (zaczynają się od **`devf-*`).** Uniwersalny driver pamięci flash nazywa się **`devf-generic`**.

Drivery sieci

Driver sieci może być uruchamiany / zatrzymywany dynamicznie przez polecnie **`mount`**.

```
mount -T io-pkt devn-ne2000.so
```

Sieciowe systemy plików

QNX wspiera dwa sieciowe systemy plików:

- NFS – Umożliwia dostęp do sieciowego systemu plików typowego dla systemów UNIX (**`fs-nfs2, fs-nfs3`**)
- CIFS - Umożliwia dostęp do sieciowego systemu plików systemu Windows (fs-cifs),

Uruchamianie aplikacji

Aplikacje które mają być uruchomione powinny być zawarte w skrypcie inicjalizacyjnym po tym jak wymagane drivery zostaną uruchomione. Gdy aplikacja potrzebuje jakiegoś urządzenia może czekać na jego uruchomienie za pomocą polecenia **`waitfor`**.

4.2 Skrypt sysinit

1. Skrypt **sysinit** startuje procesy systemowe:
2. Program **slogger** gdy nie jest jeszcze uruchomiony.
3. Administrator kolejek **mqueue**.
4. Ustawia strefę czasową
5. Uruchamia program **/etc/rc.d/rc.rtc** ustawiający zegar RTC
6. Ustawia zmienną środowiska $HOSTNAME
7. Uruchamia skrypt **/etc/rc.d/rc.devices** znajdujący urządzenia i uruchamia proces **io-net** zarządzania siecią.
8. Gdy plik **/etc/system/config/useqnet** istnieje uruchamia administrator sieci Qnet **npm-qnet.so**
9. Uruchamia skrypt **/etc/rc.d/rc.sysinit**.
10. Gdy się to uda uruchamia shell **sh** lub gdy się nie uda **fesh**.

/etc/rc.d/rc.sysinit	Startuje procesy systemowe
/etc/rc.d/rc.rtc	Ustawia czas
/etc/rc.d/rc.devices	Uruchamia urządzenia
/etc/rc.d/rc.local	Uruchamia ustawienia lokalne (tutaj powinny być umieszczone procesy użytkownika)

Tab. 4-2 Skrypty startowe systemu QNX6 Neutrino

4.3 Skrypt rc.sysinit

Skrypt **/etc/rc.d/rc.sysinit** wykonuje następujące funkcje:

1. Startuje generator liczb losowych **random.**
2. Gdy istnieje katalog **/var/dumps** startuje proces **dumper.**
3. Gdy istnieje plik **/etc/host_cfg/$HOSTNAME/rc.d/rc.local** i jest to plik wykonywalny to go startuje. Gdy nie ma takiego pliku startuje **/etc/rc.d/rc.local** gdy plik ten istnieje i jest wykonywalny. W plikach tych użytkownik może uruchamiać własne programy
4. Uruchamia proces **tinit**. Domyślnie starowany jest Photon ale gdy utworzymy plik **/etc/system/config/nophoton** wystartuje powłoka w trybie tekstowym.

Przykład pliku **/etc/rc.d/rc.local**

```
inetd &
gns –c
```

4.4 Skrypt rc.devices

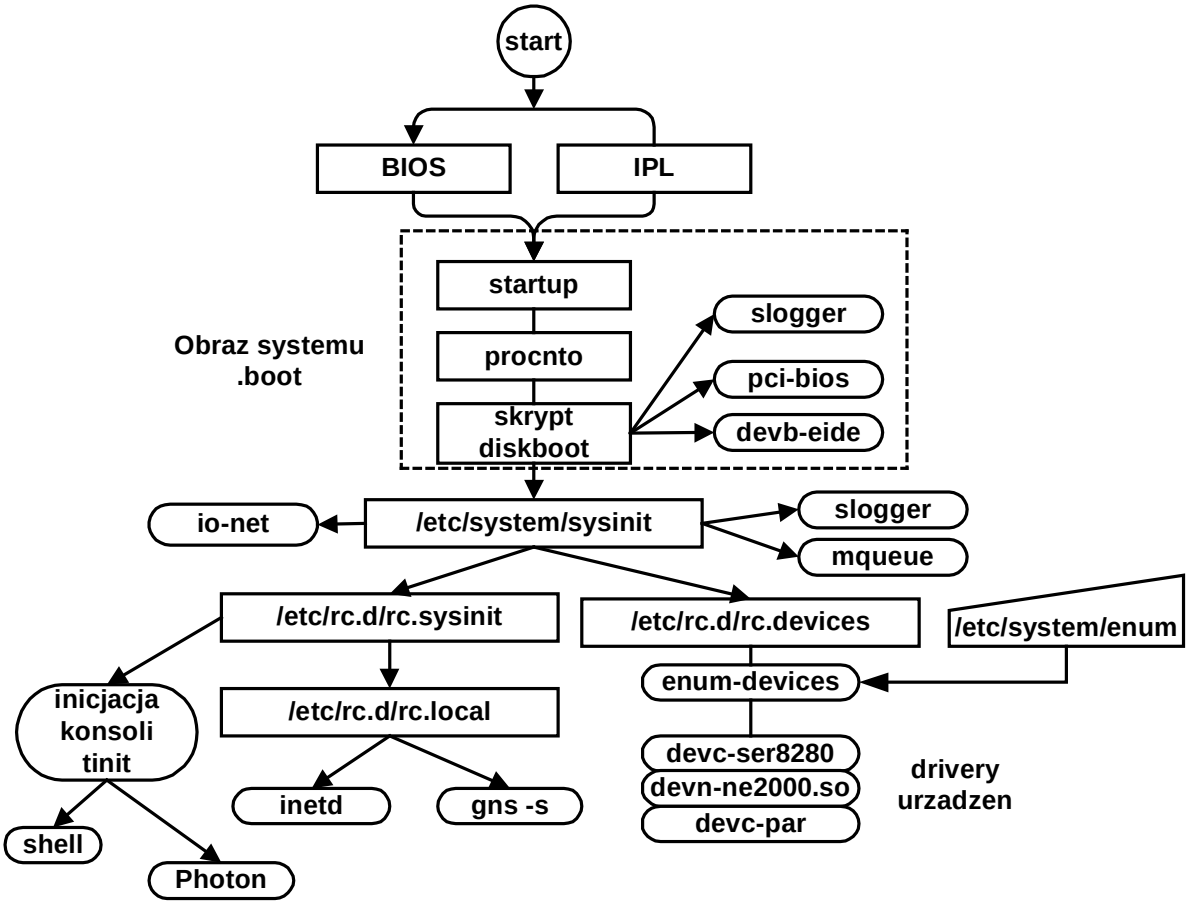
Skrypt **/etc/rc.d/rc.devices** startuje program **enum-devices** aby wykryć zainstalowany sprzęt i uruchomić potrzebne drivery. Po wykryciu danego urządzenia wykonywane są akcje wyspecyfikowane w plikach katalogu **/etc/system/enum**
Katalog ten zawiera podkatalog **devices** w którym są pliki **net, block, audio, bridge, char, graphic, input, media, printer, serial**. W plikach są informacje dla którego urządzenia jaki driver uruchomić.
Przykładowo plik **/etc/system/enum/devices/net** wskazuje jaki driver karty uruchomić i startuje program **netmanager**. Program ten ustala konfigurację sieci pobieraną z pliku **/etc/net.cfg**

```
# nto network config file v1.2
version v1.2

[global]
hostname compaq1
domain telsat.wroc.pl
nameserver 194.54.16.34
nameserver 192.168.0.1
nameserver 192.168.1.1
route 192.168.0.1 0.0.0.0 0.0.0.0

[en0]
type ethernet
mode dhcp
```

Przykład 4-1 Plik konfiguracji sieci /etc/net.cfg

4.5 Plik startowy Photona

Gdy startuje Photon to wykonuje skrypt **$HOME/.ph/phapps** (o ile istnieje i ma atrybut wykonywalności).
Np. plik **/root/.ph/phapps** zawiera:
/usr/photon/bin/devi-penmount dmc9000 –i fd –d/dev/ser3 abs &



Rys. 4-1 Start systemu QNX6 Neutrino