Jądro Linuksa

dr inż. Krzysztof Konopko

e-mail: k.konopko@pb.edu.pl

Jądro Linuksa

Program wykładu:

- Właściwości jądra Linuksa.
- Pliki źródłowe jądra.
- Konfiguracja jądra.
- Kompilacja i kompilacja skrośna jądra.
- Moduły jądra.

Właściwości jądra Linuksa

Podstawowe funkcje jądra Linuksa:

- zarządza wszystkimi zasobami sprzętowymi: CPU, pamięć, I / O,
- dostarcza przenośnych, niezależnych od architektury sprzętowej interfejsów (API), które umożliwiają aplikacjom i bibliotekom korzystanie z zasobów sprzętowych.
- jednoczesny dostęp i wykorzystywanie zasobów sprzętowych przez różne aplikacje.

Właściwości jądra Linuksa

Podstawowe cechy jądra Linuksa:

- przenośność i wsparcie sprzętowe - działa na większości architektur,
- skalowalność - można uruchomić na superkomputerów, jak i na urządzeniach wbudowanych (wystarcza 4 MB pamięci RAM),
- bezpieczeństwo – otwartość kodu umożliwia wyszukiwanie wszystkich dziur (kod jest weryfikowana przez wielu użytkowników i specjalistów),
- stabilność i niezawodność,
- modułowość – (uruchamiamy tylko to z czego korzystamy i to nawet w czasie wykonywania),
- bardzo dobra obsługa sieci,
- łatwy do programowania – łatwy dostęp do już istniejącego kodu, wiele przydatnych zasobów w sieci.

Wersje i rozwój jądra Linuksa

Do wersji 2.6 istniały dwie gałęzie rozwoju jądra w których przyjęto następujący sposób numeracji

- pierwsza cyfra oznacza serię produkcyjną,
- druga określa serię jądra - parzysty numer serii jądra oznacza, że mamy do czynienia z jego wersją stabilną, a numer nieparzysty z wersją rozwojową,
- trzecia oznacza numer kolejnej wersji tej serii (im wyższy jest ten numer, tym nowsze jądro).

Pliki źródłowe jądra

Oficjalne wersje jądra Linux, są dostępne przez stronę:

http://www.kernel.org

Lista rozwojowych gałęzi jądra, utrzymywanych przez różnych developerów znajduje się pod adresem

http://git.kernel.org

Oprócz nich dostępne są wersje w postaci repozytoriów dostosowanych do poszczególnych urządzeń lub rozwijanych przez poszczególne firmy. Np. wersje jądra przygotowane przez producentów raspberry pi znajdują się pod adresem :

https://github.com/raspberrypi/linux

Wielkość plików źródłowych jądra

Jądro Linux 4.9.56 pobrane z https://github.com/raspberrypi/linux to:

- wielkość 629 MB, (56775 plików, 3859 katalogów)
- wielkość archiwum tar.gz 138,2 MB

ale minimalna wielkość jądra (wersja 3.17) przygotowanego pod platformę ARM (dysk twardy przez PCI, system plików ext2, obsługa konsoli i urządzeń wejściowych) to w wersji skompresowanej około 900 kB i w wersji nieskompresowanej około 2,3 MB.

Skąd tak wielka różnica?

Pliki nagłówkowe jądra

Jądro Linux 4.9.56 pobrane z https://github.com/raspberrypi/linux to:

drivers	53,60%	scripts	0,45%
arch	18,13%	mm	0,43%
fs	4,87%	crypto	0,41%
include	4,67%	security	0,31%
Documentation	4,63%	block	0,15%
sound	4,27%	samples	0,12%
net	3,61%	virt	0,06%
tools	1,90%	ipc	0,03%
kernel	0,97%	init	0,02%
firmware	0,78%	usr	0,00%
lib	0,48%	certs	0,00%

Konfiguracja jądra

- Obraz jądra to pojedynczy plik, wynikający z łączenia wszystkich plików obiektowych, które odpowiadają funkcjom włączanym podczas konfiguracji (jest to plik, który jest ładowany do pamięci przez bootloader),

- Niektóre funkcje (sterowniki urządzeń, systemy plików, itp.) mogą być jednak kompilowane jako tzw. moduły
  - są wtyczki, które mogą być ładowane/usuwane dynamicznie uzupełniając funkcje jądra
  - każdy moduł jest zapisany w oddzielnym pliku w systemie plików, a więc dostęp do modułów możliwy jest wyłącznie poprzez systemu plików (brak możliwości ładowania modułów we wczesnej fazie uruchamiania systemu).

Konfiguracja jądra

- Jądro zawiera tysiące sterowników urządzeń, różnych systemu plików, protokołów sieciowych i innych elementów.
- Dostępne są tysiące opcji, które są wykorzystywane do selektywnej kompilacji części kodu źródłowego jądra.
- Konfiguracja jądra polega na określaniu wartości szeregu parametrów, za pomocą których określane są właściwości kompilowanego jądra.
- Opcje można podzielić na:
  - sprzętowe – np. sterowniki urządzeń,
  - funkcjonalne związane z realizacją określonych zadań przez kompilowane jądro – np. funkcje sieciowe, rodzaje systemów plików, praca w czasie rzeczywistym itp..

Konfiguracja jądra

- Konfiguracja i kompilacja jądra polega na wykonaniu wielu skryptów „Makefiles”.

- Użytkownik bezpośrednio uruchamia tylko jeden skrypt „Makefile” umieszczony w głównym katalogu.

- Konfiguracja i kompilacja realizowana jest za pomocą narzędzia *make* z parametrami oznaczającymi kompilację, konfigurację, instalację itd. (wyświetlenie możliwych parametrów można zrealizować przez *make help*)

Konfiguracja jądra

- Wynikowy plik z konfiguracją *.config* znajduje się w głównym katalogu.

- Plik ten zazwyczaj nie jest konfigurowany ręcznie ale przez jeden z dostępnych interfejsów graficznych bądź tekstowych:
  - make xconfig, make gconfig (graficznie),
  - make menuconfig, make nconfig (tekstowo).

- Można korzystać z różnych interfejsów wszystkie korzystają z tego samego pliku *.config* i dysponują tym samym zestawem opcji.

Konfiguracja jądra

Istnieją różne typy opcji:

- dwuwartościowe opcje typu *prawda*, *fałsz* włączające bądź wyłączające daną funkcję z jądra,
- trójwartościowe opcje umożliwiające realizację funkcji wkompilowanej w jądro, realizowanej przez moduł, bądź rezygnacji z danej opcji,
- numeryczne związane z określeniem wartości danego parametru,
- opcje tekstowe związane z wprowadzeniem ciągu znaków.

Konfiguracja jądra

Pomiędzy poszczególnymi opcjami mogą występować zależności:

- zależność od zależności (w tym przypadku opcja A, która zależy od opcji B nie jest widoczna do momentu włączenia opcji B),
- wybór zależności (w tym przypadku gdy opcja A jest włączona, zależna opcja B jest również automatycznie włączana).

Kompilacja i kompilacja skrośna jądra

Czy warto kompilować jądro:

- w przypadku gdy instaluje się dystrybucję Linuksa, jądro zostaje przygotowane przez twórców tej dystrybucji. Zawiera ono zazwyczaj wszystko co jest potrzebne, dlatego w praktyce bardzo rzadko użytkownik musi kompilować jądro.
- w przypadku systemów wbudowanych w związku z koniecznością „wydobycia” jak największej funkcjonalności z dostępnego sprzętu jest to wręcz konieczne.

Jądro dla systemów wbudowanych najczęściej ze względu na:

- zbyt małe zasoby (procesor, pamięć, urządzenia IO),
- małą wydajność (moc obliczeniową) w porównaniu do stacji roboczej,
- brak możliwość (potrzeba) instalacji całego środowiska deweloperskiego na komputerze docelowym.

buduje się z zastosowaniem kompilacji skrośnej.

Kompilacja i kompilacja skrośna jądra

Architektura docelowego procesora i jego parametry definiowane są przez dwie zmienne:

- `ARCH =` oznacza architekturę procesora, dla której budujemy jądro (pisana małymi literkami oznacza odpowiedni podkatalog w katalogu arch/ należy ją podać zarówno przy konfiguracji, kompilacji jak i instalacji np.: *`ARCH=arm`*),
- `CROSS_COMPILE =` oznacza przedrostek kompilatora i pozostałych narzędzi używanych przy konfiguracji, kompilacji jak i instalacji np.:

```
CROSS_COMPILE=arm-unknown-linux-gnueabihf-gcc-
```

Kompilacja i kompilacja skrośna jądra

W niektórych architektach (np. ARM) wymagana jest specyficzna konfiguracja uwzględniająca wszystkie funkcje danego urządzenia. Twórcy danego urządzenia mogą dostarczyć gotowe pliki konfiguracyjne, które można „załadować” podczas konfiguracji komendą

```
make ARCH=arm xx_defconfig
```

Listę wszystkich plików konfiguracyjnych można wyświetlić po wykonaniu

```
make ARCH=arm help
```

Kompilacja i kompilacja skrośna jądra

- Wiele systemów wbudowanych posiada specyficzne peryferia.
- W zależności od architektury, sterowniki dla takiego sprzętu są albo bezpośrednio kompilowane w jądro, lub są obsługiwane za pomocą specjalnego języka opisu sprzętu tworząc tzw. *Device Tree*
- W Linuksie *Device Tree*, (ARM), znajduje się w arch/arm/boot/dts. Są tam:
  - .dts – pliki dla konkretnych urządzeń,
  - .dtsi – „biblioteki” dla konkretnych SoC-ów.
- Kompilator (scripts/dtc) tworzy z nich wersję binarną (Blob) – .dtb. Ten właśnie plik musi być załadowany przez Bootloader.

Kompilacja skrośna jądra dla Raspberry Pi 2

- Instalacja i konfiguracja toolchaina (np. dostępnego w Ubuntu)

```
sudo apt-get install gcc-arm-linux-gnueabihf
```

- Pobranie plików źródłowych jądra:

```
git clone https://github.com/raspberrypi/linux
```

- Skopiowanie pliku z konfiguracją jądra dla raspberry pi 2 (skorzystamy z toolchaina dostarczonego przez Ubuntu)

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- \
   bcm2709_defconfig
```

- kompilacja jądra i systemu Device Tree

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage \
   dtbs
```

Moduły

Modułami można operować za pomocą następujących programów:

- `insmod` - polecenie to pozwala na załadowanie modułu do pamięci,
- `rmmod` - usuwa moduł,
- `lsmod` - wyświetla listę załadowanych modułów,
- `depmod` - tworzy bazę danych z informacjami o dostępnych w systemie modułach które mogą zostać umieszczone w jądrze,
- `modinfo` - pobiera informacje o module: opis, parametry i zależności,
- `modprobe` - próbuje załadować wszystkie zależne, a następnie wskazany moduł.

Moduły

Gdy nowy moduł jest ładowany informacje o nim można podejrzeć w logach jądra za pomocą komendy

`dmesg` (diagnostic message)

Aktualne wartości parametrów załadowanych modułów można zobaczyć w:

`/sys/module/<name>/parameters`

Kompilacja skrośna modułów dla Raspberry Pi 2

- kompilacja modułów

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- modules
```

- instalacja modułów

```
sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
   INSTALL_MOD_PATH=path modules_install
```

`INSTALL_MOD_PATH` to ścieżka, do której zostaną zainstalowane moduły

Dziękuję za uwagę

Zapraszam
za tydzień :)