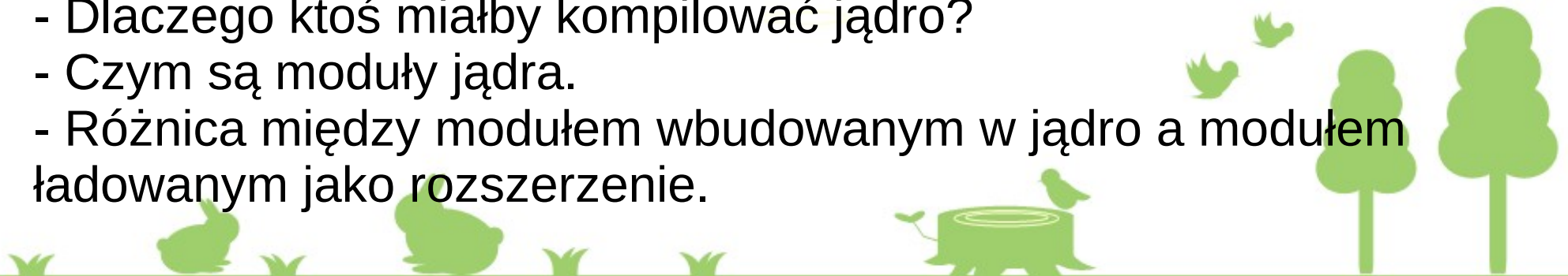


Jądro systemu operacyjnego

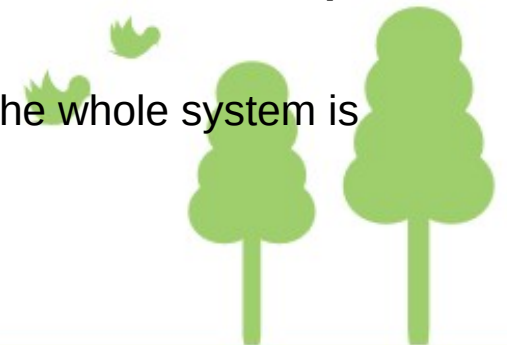
Zostaną poruszone następujące tematy:

- Czym jest Linux?
- Czym jest jądro systemu i jaka jest jego rola względem systemu operacyjnego.
- Krótka historia na temat jądra.
- Różnice i podobieństwa między jądrem linuxa w androidzie i w linuxie desktopowym.
- Jądro w systemach RTOS i systemach wbudowanych.
- Podsystemy jądra.
- Jakiego rodzaju jądra systemu są dostępne.
- Jakie są opcje konfiguracji jądra.
- Dlaczego ktoś miałby kompilować jądro?
- Czym są moduły jądra.
- Różnica między modułem wbudowanym w jądro a modułem ładowanym jako rozszerzenie.



Czym jest Linux? Let me interject for a moment...

- What you guys are referring to as Linux, is in fact, GNU/Linux, or as I've recently taken to calling it, GNU plus Linux.
- Linux is not an operating system unto itself, but rather another free component of a fully functioning GNU system made useful by the GNU corelibs, shell utilities and vital system components comprising a full OS as defined by POSIX. Many computer users run a modified version of the GNU system every day, without realizing it.
- Through a peculiar turn of events, the version of GNU which is widely used today is often called "Linux", and many of its users are not aware that it is basically the GNU system, developed by the GNU Project.
- **There really is a Linux, and these people are using it, but it is just a part of the system they use. Linux is the kernel: the program in the system that allocates the machine's resources to the other programs that you run. The kernel is an essential part of an operating system, but useless by itself; it can only function in the context of a complete operating system.**
- Linux is normally used in combination with the GNU operating system: the whole system is basically GNU with Linux added, or GNU/Linux.



Czym jest jądro systemu?

Jak dowiedzieliśmy się z copypasty, Linux to tak naprawdę jądro systemu i jego rolą jest zarządzanie zasobami systemu – przydzielając je innym programom działającym w *user space*. Nie jest to jego jedyna rola.



User space i kernel space

System operacyjny zazwyczaj dzieli pamięć wirtualną na dwie części. W części user space działają wszystkie programy z którymi użytkownik prowadzi interakcje, przykładowo Firefox, notepad, terminal, kalkulator. Programy w user space nie mają bezpośredniego dostępu do hardware. Zamiast tego korzystają z interfejsu udostępnianego przez kernel.



User space i kernel space cz.2

Podział między user space i kernel space służy bezpieczeństwu systemu. Jeśli program w user space przestaje działać prawidłowo, nie ma to wpływu na inne programy lub na system operacyjny (przykładowo gra może crashować).

Kernel przydziela każdemu procesowi z user space część pamięci poprzez *virtual addressing*. Zakresy adresów używane przez procesy mogą być odrębne, przez co crash jednej aplikacji nie wpłynie na inne.

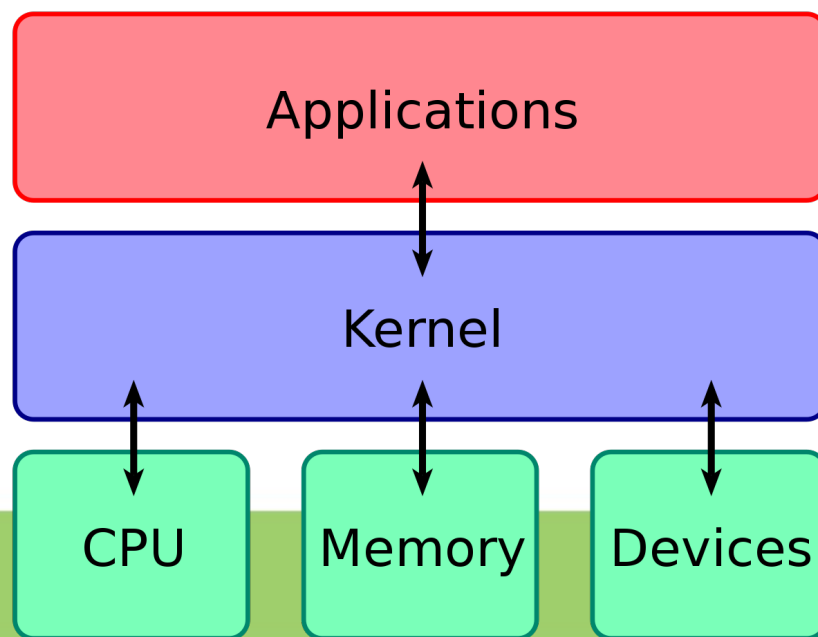


Interfejs udostępniany przez kernel

Kilka programów może chcieć jednocześnie korzystać z drukarki.

Kernel ma za zadanie kolejkować te zadania do hardware w ten sposób, aby drukarka nie drukowała raz strony z jednego dokumentu a raz z drugiego.

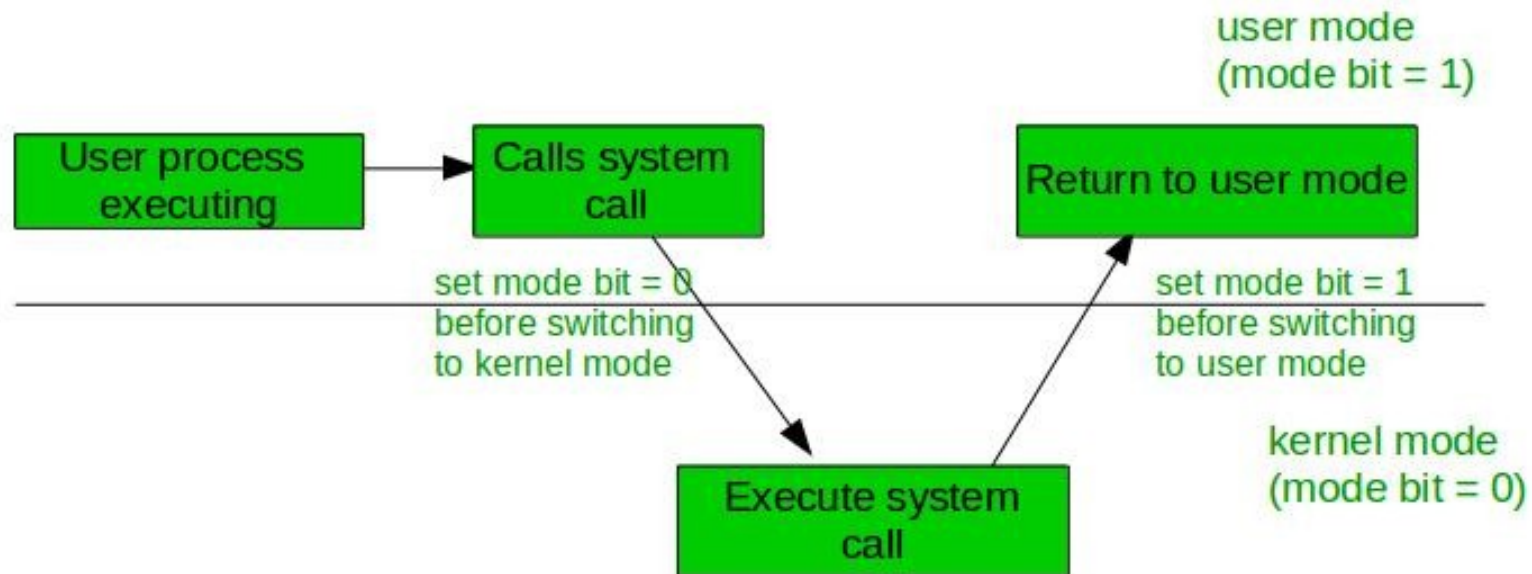
Zamiast tego ma wydrukować pełen dokument i dopiero wtedy drukować w pełni kolejny, itd.



Tryby CPU: User mode & kernel mode

Są dwa bazowe tryby CPU – user mode i kernel mode. (ustawiane poprzez zmianę bitu na 1 lub 0). Programy działające w kernel space mają pełen dostęp do systemu oraz hardware.

W momencie gdy program chce użyć hardware wykonuje on syscall do kernela. Wtedy CPU zmienia tryb na kernel mode, wykonuje zadanie określone przez syscall, następnie zmienia tryb z powrotem na user mode.



Role kernela podsumowanie

- Zarządza przydzielaniem pamięci RAM oraz czasu CPU procesom.
- Zarządza dostępem do urządzeń I/O jak drukarka, myszka, karta sieciowa.
- Pozwala procesom na komunikację między sobą
- Zarządza pamięcią poprzez virtual addressing.
- Udostępnia interfejs dostępu do hardware dla programów z user space.



Krótką historia na temat jądra Linux

- Zainspirowany projektem GNU Richarda Stallmana
- Wersja 0.01 w roku 1991.
- Otwarto-źródłowy (GPL) – każdy może analizować kod, forkować oraz założyć pull request z proponowanymi zmianami.
- Projekt rozpoczęty przez Linusa Torvaldsa jako hobby. Ponad 30 lat później +90% serwerów działa na systemach z jądrem Linux.
- Jądro linux jest również używane w komputerach osobistych, smartfonach z systemem Android, routerach, urządzeniach Internet of Things, Raspberry Pi oraz superkomputerach.

W 2010 Google przy tworzeniu systemu Android zrobiło fork kernela Linux.



Jądro Linuxa w Androidzie

Ktoś mógłby zapytać: “Jeżeli w Androidzie jest Linux oraz w Ubuntu jest Linux, to dlaczego nie można uruchamiać aplikacji z Androida na Gnu/Linuxie i na odwrót?”

Android korzysta jedynie z jądra Linux, nie zawiera GNU Core Utilities tak jak to jest w przypadku większości dystrybucji linuxa. Jako że programy w większości przypadków korzystają z dynamicznie łączonych bibliotek (w przeciwieństwie do statycznie łączonych), to jeśli tych bibliotek na danym systemie nie ma to program się nie uruchomi.



Program działający na Androidzie oraz na Gnu/Linuxie

Możliwe jest utworzenie programu działającego na obu systemach operacyjnych. Przykładowo jeśli skompilujemy program ze statycznie łącznie bibliotekami na raspberry pi 1 (korzystający z procesora ARM), a następnie przeniesiemy plik wykonywalny na Androida również z procesorem ARM program ten zadziała na obu systemach tak samo.



Linkowanie statyczne i dynamiczne

Statyczne linkowanie oznacza że wszystkie biblioteki potrzebne do uruchomienia programu zostają spakowane razem z programem.

Dynamiczne linkowanie zakłada, że biblioteka z której ma korzystać program jest dostępna na danym systemie.



Bionic vs Gnu C library

- W systemie Android, używaną przez kernel biblioteką jest Bionic w kontraście do GNU/Linux które używa GNU C library.
- Bionic jest zaprojektowana dla urządzeń mających mniej pamięci oraz mocy obliczeniowej niż komputery PC, laptopy i serwery.



Różnice pomiędzy kernelem Androida i Gnu/Linux

- W androidzie komunikacja pomiędzy procesami realizowana jest poprzez mechanizm Binder.
- Wakelock – mechanizm umożliwiający aplikacjom blokadę możliwości uśpienia systemu w przypadku gdy użytkownik nie ma bezpośredniej interakcji z urządzeniem (przykładowe zastosowanie – oglądanie filmu).
- Lista różnic dostępna jest na stronie https://elinux.org/Android_Kernel_Features



Inter-process communication

IPC jest to mechanizm umożliwiający procesom dzielenie się informacjami. Jest to na przykład:

- Plik nadpisywany przez proces “serwer” i czytany przez procesy “klientów”.
- Pipe – dane z jednego procesu przekazywane są do innego za pomocą operatora |.

```
(base) fenix@fenix:~/Downloads$ echo hello | cat  
hello  
(base) fenix@fenix:~/Downloads$ _
```

- Socket – dane przesyłane poprzez interfejs sieciowy do innego procesu na tym samym lub innym komputerze w sieci.

Czym jest system RTOS.

Real-Time Operating System to system, od którego wymaga się deterministycznego czasu oraz kolejności realizacji zadań.

RTOS mają zastosowania w przypadkach gdy czas reakcji jest kluczowy przykładowo: samochody, system kontroli lotów, roboty.

RTOS powinien działać bez przerwy bez konieczności restartu.

Jest możliwość aby zadanie o wysokim priorytecie miało wyłączny dostęp do CPU (inne procesy nie mogą wtedy być wykonywane).


Przykłady systemów RTOS: FreeRTOS, Micrium uC/OS, ThreadX.



Jądro w systemach embedded

W przeciwieństwie do komputerów osobistych, projekty zawierające systemy wbudowane przeznaczone są do jednego z góry określonego celu. W związku z tym wiele funkcjonalności kernela desktopowego może nie być wykorzystywane (na przykład sterowniki bluetooth lub sterowniki karty graficznej).

Jądro w systemach wbudowanych jest konfigurowane tak aby zawierało tylko części istotne z punktu widzenia danego projektu w celu zmniejszenia wymagań sprzętowych oraz zwiększenia energooszczędności.



Podsystemy jądra

Dyspozytor procesów Wirtualny system plików
 Manager pamięci Interfejs sieciowy
 Interfejs komunikacji między-
 procesowej(IPC)

Various layers within Linux, also showing separation between the **userland** and **kernel space**

User mode	User applications	bash, LibreOffice, GIMP, Blender, 0 A.D., Mozilla Firefox, ...				
	System components	init daemon: OpenRC, runit, systemd...	System daemons: polkitd, smbd, sshd, udevd...	Window manager: X11, Wayland, SurfaceFlinger (Android)	Graphics: Mesa, AMD Catalyst, ...	Other libraries: GTK, Qt, EFL, SDL, SFML, FLTK, GNUstep, ...
	C standard library	fopen, execv, malloc, memcpy, localtime, pthread_create ... (up to 2000 subroutines) glibc aims to be fast, musl aims to be lightweight, uClibc targets embedded systems, bionic was written for Android, etc. All aim to be POSIX/SUS-compatible.				
Kernel mode	Linux kernel	stat, splice, dup, read, open, ioctl, write, mmap, close, exit, etc. (about 380 system calls) The Linux kernel System Call Interface (SCI), aims to be POSIX/SUS-compatible ^[104]				
		Process scheduling subsystem	IPC subsystem	Memory management subsystem	Virtual files subsystem	Network subsystem
		Other components: ALSA, DRI, evdev, klibc, LVM, device mapper, Linux Network Scheduler, Netfilter Linux Security Modules: SELinux, TOMOYO, AppArmor, Smack				
Hardware (CPU, main memory, data storage devices, etc.)						


Dyspozytor procesów

Dyspozycja procesami (scheduling) to przydzielanie czasu procesora zadaniom na podstawie ich priorytetu.

Gdy uruchamiamy program (na przykład “cat”) kernel tworzy proces i nadaje mu Process ID (PID). Znając PID procesu możemy wysłać do kernela sygnał aby proces został zakończony (“kill 79981”). Sygnał Terminate może również być wysłany poprzez skrót klawiszowy Ctrl+C. Sygnał uśpienia procesu wysyłany jest poprzez Ctrl+Z.

```
(base) fenix@fenix:~/Downloads$ cat
^Z
[2]+  Stopped                  cat
(base) fenix@fenix:~/Downloads$ ps aux
```


USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
fenix	79981	0.0	0.0	5632	580	pts/3	T	10:20	0:00	cat



Dyspozytor procesów cz.2

Celem dyspozytora procesów jest maksymalizacja wykorzystania procesora (minimalizacja czas, w którym procesor czeka beczynnie) oraz zapewnienie procesom sprawiedliwego przydziału czasu procesora.

Procesor (załóżmy że jednowątkowy) jest w stanie wykonywać tylko jedno zadanie na raz. Aby zapewnić wrażenie że procesy wykonują się równocześnie, dany proces zajmuje CPU na krótką chwilę. Następnie inny proces otrzymuje czas CPU. Przełączenie z jednego procesu na inny kosztuje cykle procesora, przez które mógłby on wykonywać instrukcje.



Manager pamięci (RAM)

Zajmuje się alokacją pamięci dla programów, mapowaniem plików do przestrzeni adresowej procesów. Przykładowo aplikacje nie mają dostępu do pamięci systemu operacyjnego. Z perspektywy aplikacji wygląda to jakby system zajmował się tylko nią (mimo że uruchomione jest wiele aplikacji na raz). Manager pamięci zajmuje się przenoszeniem danych zapisanych na dysku do pamięci RAM i z powrotem.



Manager pamięci (RAM)

- Aplikacja w chwili uruchomienia jest przez manager pamięci ładowana do RAMu. Wtedy też są przydzielane adresy w pamięci.

```
(base) fenix@fenix:~/Desktop/kernel_video/c$ cat main.c
#include <stdio.h>

int main(void){

    int x = 123;

    printf("x value = %d\n", x);
    printf("x address = %p\n",&x);

    return 0;
}
(base) fenix@fenix:~/Desktop/kernel_video/c$ gcc main.c
(base) fenix@fenix:~/Desktop/kernel_video/c$ ./a.out
x value = 123
x address = 0x7ffcd5a97f94
(base) fenix@fenix:~/Desktop/kernel_video/c$ ./a.out
x value = 123
x address = 0x7ffe1f7d32d4
(base) fenix@fenix:~/Desktop/kernel_video/c$
```



Pamięć logiczna a pamięć fizyczna

Dla każdego programu przydzielona jest pamięć logiczna, która z kolei jest mapowana na pamięć fizyczną. Adres zwracany przez `printf("x address = %p\n",&x);` jest adresem logicznym (adres fizyczny może być inny ale może też być taki sam). Mapowanie pomiędzy pamięcią logiczną i fizyczną nazywa się Virtual Memory. Pamięć zarezerwowana dla danego procesu jest zazwyczaj wyłączna i inne programy nie mają do niej dostępu. Dzięki pamięci logicznej system operacyjny może ograniczyć ilość pamięci dostępnej dla danego procesu (aby jeden proces nie zabrał całej pamięci).

Wirtualny system plików (VFS)

Według kernela Linuxa jeśli na czymś możemy wykonać operację `open()`, `close()`, `read()` i `write()` to jest to plik.

VFS jest to warstwa abstrakcji umożliwiaująca aplikacjom dostęp do systemu plików.

Dzięki VFS możliwe jest równoczesne korzystanie z różnych systemów plików (ntfs, ext4, fat32, nfs).

Filozofia pochodząca z Unixa “wszystko jest plikiem”.

<https://medium.com/@boutnaru/linux-memory-management-part-1-introduction-896f376d3713>

<https://www.kernel.org/doc/html/latest/filesystems/vfs.html>

<https://www.youtube.com/watch?v=KffhM5N9wVg>



Wirtualny system plików cz.2

Filozofię “wszystko jest plikiem” możemy zweryfikować próbując zapisać do konsoli dane, traktując ją jak plik. Komendą `ps` sprawdzamy na jaki plik jest mapowana obecna konsola, następnie przekierowujemy wiadomość do tego pliku poprzez `echo` i operator `>`.

```
(base) fenix@fenix:~$ ps
  PID TTY          TIME CMD
 89917 pts/5        00:00:00 bash
 90060 pts/5        00:00:00 ps
(base) fenix@fenix:~$ echo "hello" > /dev/pts/5
hello
(base) fenix@fenix:~$
```

Różne typy kerneli

Kernel monolityczny (Linux) i Mikrokernel

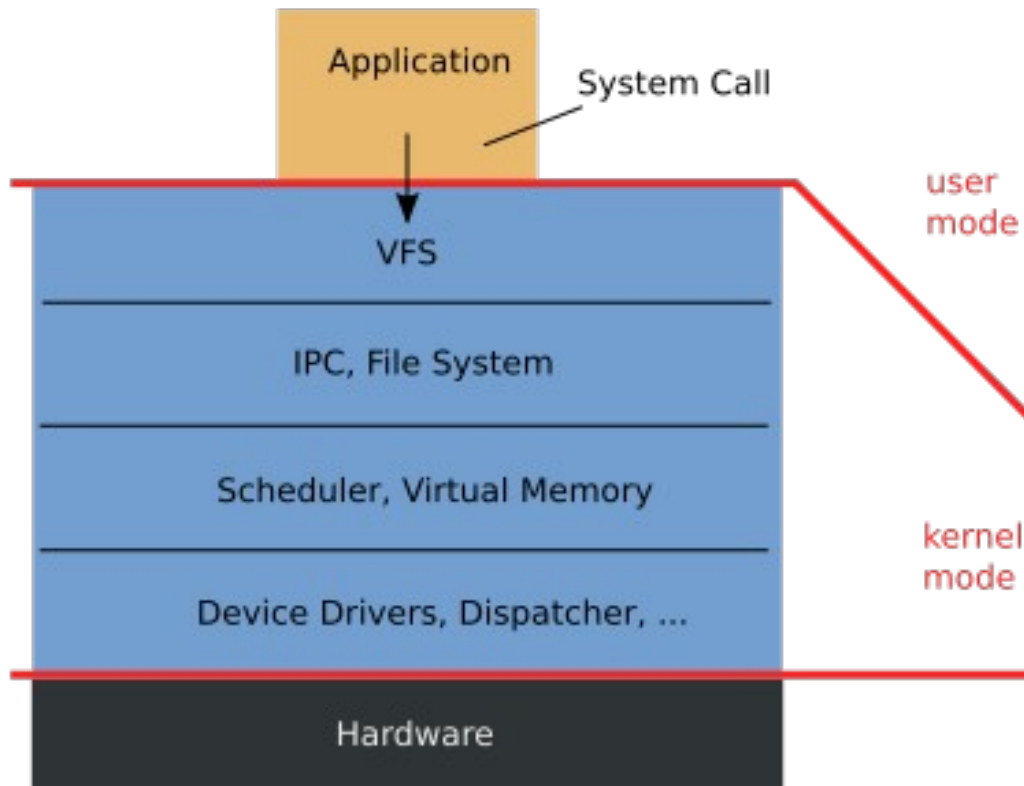
Cechy mikrokernelsa:

- Zajmuje mniej pamięci ze względu na to że część funkcjonalności jest implementowana w przestrzeni użytkownika (user space).
- Działa wolniej ponieważ komunikacja (IPC) oraz przełączanie trybu procesora (user mode i kernel mode) pomiędzy częściami z kernel space i user space kosztuje czas.

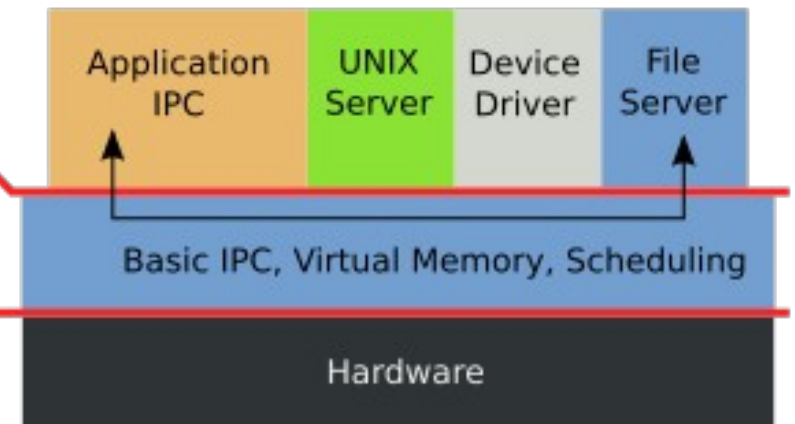


Monolit vs mikrokernel

Monolithic Kernel
based Operating System



Microkernel
based Operating System



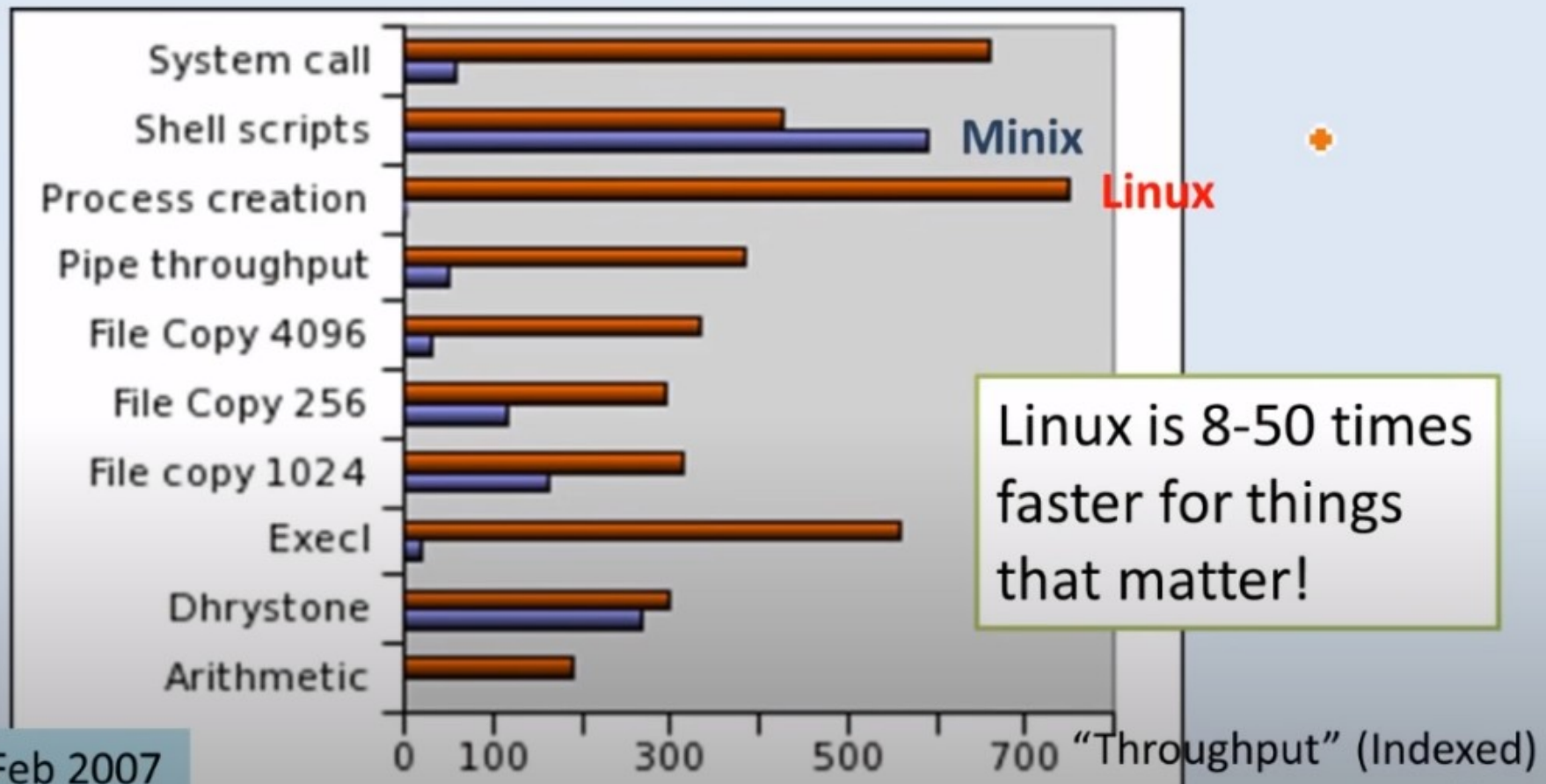
Monolit vs mikrokernel

Mikrokernel próbuje zmniejszyć ilość kodu działającą w trybie kernel mode. Dzięki temu może zapewnić większe bezpieczeństwo (mniejsza ilość kodu ma pełny dostęp do systemu).

Mniejszy kernel łatwiej jest debugować i jeśli jedna z części systemu działająca w user space ulegnie awarii nie zatrzyma to działania systemu.



Really Comparing Linux and Minix



Minix jest jednym z mikrokerneli.

Opcje konfiguracji kernela

Linux jest otwartoźródłowy, zatem każdy może pobrać źródło kernela i samemu go skompilować.

Jednak dlaczego ktoś miałby to robić jeśli są dostępne wersje iso całego systemu operacyjnego np. Debian?

Jeśli samemu skompilujemy kernel mamy możliwość go skonfigurować tak aby pasował do naszych potrzeb. Przykładowo jeśli komputer ma mało pamięci RAM, możemy wykluczyć części kernela, których nie potrzebujemy aby zaoszczędzić pamięć.



Opcje konfiguracji kernela - instrukcja

1. Pobierz źródło z strony <https://www.kernel.org>
2. Rozpakuj archiwum
3. Komenda “make menuconfig” pokaże ekran konfiguracji kernela.

```
.config - Linux/x86 6.0.0 Kernel Configuration

Linux/x86 6.0.0 Kernel Configuration

Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenu ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ] excluded
<M> module < > module capable

General setup --->
[*] 64-bit kernel
Processor type and features --->
[*] Mitigations for speculative execution vulnerabilities --->
Power management and ACPI options --->
Bus options (PCI etc.) --->
Binary Emulations --->
[*] Virtualization ----
General architecture-dependent options --->
[*] Enable loadable module support ---->
[*] Enable the block layer ---->
Executable file formats --->
Memory Management options --->
[*] Networking support --->
Device Drivers --->
File systems --->
Security options --->
-* Cryptographic API --->
Library routines --->
Kernel hacking --->

<Select> < Exit > < Help > < Save > < Load >
```

Istotne opcje przy konfiguracji kernela

Kernel compression mode: wybierz algorytm odpowiedni do zastosowań. Możesz wybrać niską kompresję dzięki temu uzyskasz większą wydajność kosztem większego użycia pamięci. Szybkość dekompresji jest istotna przy uruchamianiu systemu operacyjnego. Szybkość kompresji jest istotna przy kompilacji. Jest to opcja istotna z punktu widzenia systemu wbudowanego

Timer tick handling: opcja istotna z punktu widzenia energooszczędności. Jeśli kernel ma działać na laptopie może być istotne aby ustawić opcję pozwalającą na uśpienie kernela w czasie gdy nic się nie dzieje.

Processor family: jeśli masz AMD wybierz rodzinę amd jeśli masz intel, wybierz intel.



Istotne opcje przy konfiguracji kernela

Networking support: sekcja gdzie możemy wyłączyć wsparcie dla bluetooth lub innych protokołów sieciowych.

Device Drivers: sekcja gdzie możemy włączyć lub wyłączyć sterowniki urządzeń (przykładowo serwer nie potrzebuje sterownika karty dźwiękowej).



Moduły jądra systemu

Moduły mogą być dodane do jądra podczas konfiguracji (przed kompilacją) lub dynamicznie ładowane w czasie działania systemu operacyjnego (np. podczas startu systemu).

Moduł może być napisany na przykład w języku C i od niedawna w języku Rust.

Moduł może zostać dodany dynamicznie poprzez komendę “`sudo insmod mymodule.ko`”.

I usunięty poprzez “`sudo rmmod mymodule`”.

Moduł to rozszerzenie do kernela.



Przykładowy prosty moduł o nazwie hello.c

```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
```

```
static int __init start(void)
{
    printk(KERN_INFO "Loading module...");
    printk(KERN_INFO "Hello world...");
    return 0;
}
```

```
static void __exit end(void){
    printk(KERN_INFO "Bye world");
}
```

```
module_init(start);
module_exit(end);
```



Makefile dla pliku hello.c

obj-m += hello.o

all:

make -C /lib/modules/\$(shell uname -r)/build M=\$(PWD) modules

clean:

make -C /lib/modules/\$(shell uname -r)/build M=\$(PWD) clean



Dodanie modułu i obserwacja logu

```
leaf@leaf:~/Desktop/linux_kernel/hello_world$ ls
hello.c  Makefile
leaf@leaf:~/Desktop/linux_kernel/hello_world$ make
make -C /lib/modules/5.4.0-132-generic/build M=/home/leaf/Desktop/linux_kernel/hello_world modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-132-generic'
CC [M] /home/leaf/Desktop/linux_kernel/hello_world/hello.o
Building modules, stage 2.
MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /home/leaf/Desktop/linux_kernel/hello_world/hello.o
see include/linux/module.h for more information
CC [M] /home/leaf/Desktop/linux_kernel/hello_world/hello.mod.o
LD [M] /home/leaf/Desktop/linux_kernel/hello_world/hello.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-132-generic'
leaf@leaf:~/Desktop/linux_kernel/hello_world$ ls
hello.c  hello.ko  hello.mod  hello.mod.c  hello.mod.o  hello.o  Makefile  modules.order  Module.symvers
leaf@leaf:~/Desktop/linux_kernel/hello_world$ sudo insmod hello.ko
leaf@leaf:~/Desktop/linux_kernel/hello_world$ tail -f /var/log/kern.log
Nov 20 19:55:55 leaf kernel: [435342.635629] Loading module...
Nov 20 19:56:11 leaf kernel: [435342.635630] Hello world...
leaf@leaf:~$ sudo rmmod hello
Nov 20 19:56:52 leaf kernel: [435395.531344] Bye world
```



Moduł to na przykład sterownik

Możemy zaimplementować sterownik pseudo-urządzenia, który odpowiada to co zostało w nim zapisane. Lub sterownik pseudo-urządzenia które zwraca losową liczbę od 0 do 100. Może to być też sterownik prawdziwego urządzenia.



Zadania w trakcie zajęć

1. Napisz moduł, który w chwili załadowania w usunięcia wpisuje do logu kernela customową wiadomość.
2. Napisz moduł, który rejestruje sterownik urządzenia znakowego (character device). Jako dowód można pokazać wynik `"cat /proc/devices"`
3. Napisz moduł, który rejestruje sterownik urządzenia oraz mapuje to urządzenie na plik w `/dev`.
Co pozwala na odczyt customowej wiadomości z pliku za pomocą `"cat /dev/mydevice"`.
4. To samo co 3, ale urządzenie powinno zwracać losową liczbę od 0 do 10.
5. To samo co 3, ale urządzenie powinno zwrócić wiadomość tylko raz zamiast nieskończonego ciągu wiadomości.

• Źródła

#1 By Bobbo - Own work, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=4392180>

- #2 <https://www.geeksforgeeks.org/dual-mode-operations-os/>
- #3 0:17 <https://www.youtube.com/watch?v=ShcR4Zfc6Dw>
- #4 https://en.wikipedia.org/wiki/Linux_distribution

