



Systemy Operacyjne

Automatyzacja skryptowa

Dr hab. inż. Krzysztof Rzecki, prof. AGH

Sprawy organizacyjne

- Prowadzący: Krzysztof Rzecki, <http://rzecki.pl>
- Wykłady: 30h
- Laboratorium: 15h
- ECTS: 3.0
- Warunki zaliczenia:
 - Warunkiem zaliczenia wykładu jest pozytywna ocena z kolokwium teoretycznego.
 - Warunkiem zaliczenia laboratorium są pozytywne oceny z odpowiedzi oraz kolokwium praktycznego.
 - Warunkiem zaliczenia przedmiotu są trzy pozytywne, w/w oceny.
- Sposób obliczania oceny końcowej
 - Ocena wystawiana na podstawie średniej ważonej wyników uzyskanych z:
 - kolokwium teoretycznego z zakresu wykładów (30%),
 - kolokwium praktycznego z zakresu ćwiczeń laboratoryjnych (40%),
 - ocen[-al-y] z odpowiedzi ustnej uzyskanej w trakcie ćwiczeń laboratoryjnych (30%).

Edukacja / Nauka

- **Informatyka:**
 - inżynier i magister - EAliE AGH,
 - doktor - IITiS PAN,
 - habilitacja - RD ITiT AGH.
- **Zarządzanie i Marketing:**
 - magister - WZ AGH.

Certyfikaty: POWR.3.5, TOP 500 Innovators, ITIL Foundation, PRINCE2.

Badania

- metody biometryczne, gesty,
- systemy wizyjne i bronchoskopia,
- radioterapia i promieniowanie kosmiczne,
- laser-induced breakdown spectroscopy,
- wirtualizacja,
- świadomość kontekstu,
- architektury zorientowane na usługi,
- protokoły sieciowe.

Zawód / Przemysł / Biznes

- Profesor uczelni - KBiB, EAIIB, AGH.
- CEO - Live-Docs.com Sp. z o.o., Kraków.
- Kierownik projektu - KI, IMF, PK, Kraków.
- Kierownik B+R - EPL Sp. z o.o., Międzyrzecz.
- Recenzent w NCBiR.

Doświadczenie

- ComArch Healthcare S.A., Kraków,
- VSoft SA, Kraków,
- CCNS SA, Kraków,
- Telekomunikacja Polska SA, Warszawa,
- Siemens AG, Monachium,
- Nokia Siemens Networks AG, Monachium.

Programy: Fulbright Cybersecurity, ISS on DL.

Linked in:

<https://www.linkedin.com/in/krzysztof-rzecki/>

Systemy operacyjne

1. Wprowadzenie i podstawy automatyzacji skryptowej.
2. Rodzaje i architektura systemów operacyjnych.
3. Program, proces, wątek, tworzenie i terminowanie, stany, współbieżność i równoległość.
4. Komunikacja międzyprocesowa, synchronizacja, zakleszczenia.
5. Pamięć operacyjna, pamięć główna, pamięć wirtualna, przestrzeń wymiany, zarządzanie pamięcią.
6. Magazyn danych i system plików.
7. Gniazda i komunikacja sieciowa.
8. Ochrona i bezpieczeństwo systemów operacyjnych.

Co w ramach tego przedmiotu byłoby interesujące, a nie mieści się na w/w liście?

Literatura

- A. Silberschatz, P. B. Galvin, G. Gagne, Operating System Concepts Essentials, 10th ed.
- A. S. Tanenbaum, Bos H., Modern Operating Systems, Pearson, 4th ed.
- Cooper M., Advanced Bash-Scripting Guide. An in-depth exploration of the art of shell scripting, 2014, online: <https://www.tldp.org/LDP/abs/html/>
- W. Richard Stevens, Stephen A. Rago, Advanced Programming in the UNIX Environment, 3rd ed.
- W. R. Stevens, UNIX Network Programming, 2nd ed.
- R. Love, Linux System Programming: Talking Directly to the Kernel and C Library, 1st ed.
- P. Yosifovich, M. E. Russinovich, D. A. Solomon, A. Ionescu, Windows Internals, Part 1: System architecture, processes, threads, memory management, and more, 7th ed.

Przygotowanie środowiska

- VPN do sieci AGH
 - <https://pomoc-it.agh.edu.pl/vpn-zdalny-dostep-do-sieci/>
- Konto na serwerze studenckim
 - <https://pomoc-it.agh.edu.pl/konta-unix/instrukcje/zakladanie-konta-unix/>
- Własna instalacja systemu Linux (2+ cores of CPU, 4+ GB RAM, 25+ GB HDD)
 - Natywna, czyli bezpośrednio na komputerze
 - Maszyna wirtualna:
 - VirtualBox: <https://www.virtualbox.org/>
 - QEMU: <https://www.qemu.org/>
 - VMware Player: <https://www.vmware.com/products/workstation-player.html>
- **Uwaga!** Rozwiązania w postaci terminala MacOS, PowerShell, czy Windows Subsystem for Linux(WSL) są niewystarczające do realizacji ćwiczeń.

Przygotowanie środowiska do zajęć on-line

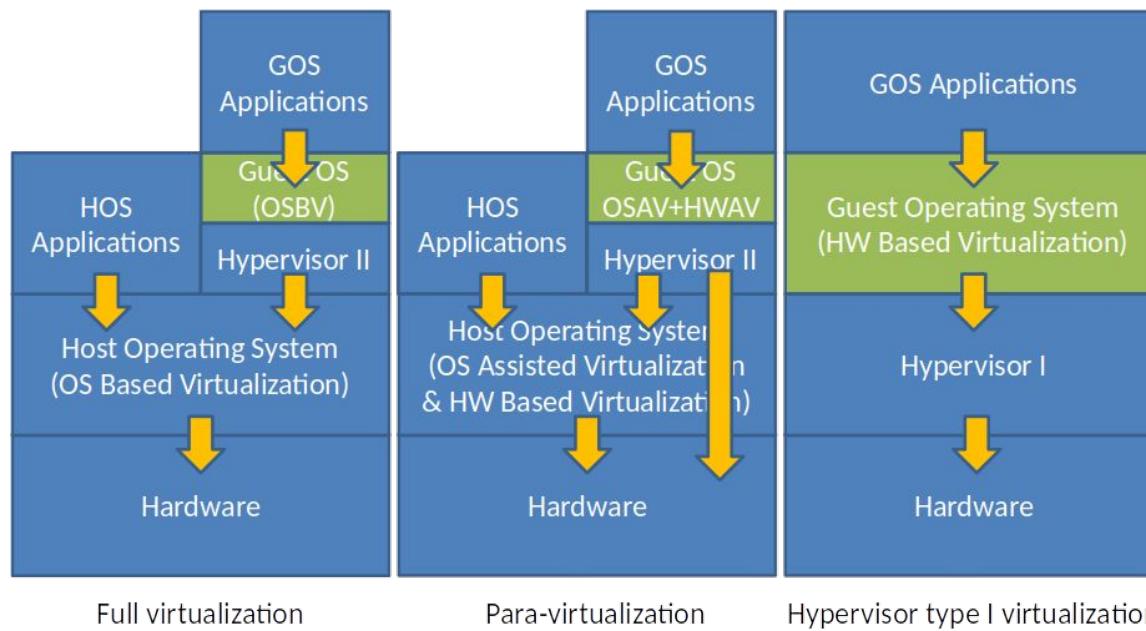
- **Sprawdź działanie swojego systemu audio** - połącz się przed zajęciami z Koleżanką/Kolegą za pomocą aplikacji, którą użyjesz do zajęć on-line i zapytaj, czy Cię słyszy, wyreguluj ustawienia mikrofonu tak, aby w czasie zajęć było Cię odpowiednio słychać, ale nie przesteruj wzmacnienia.
- **Sprawdź działanie kamery** - połącz się j.w. i zapytaj, czy Cię widać.
- **Sprawdź działanie udostępniania pulpitu i okna aplikacji** - połącz się j.w. i przetestuj działanie udostępniania całego pulpitu oraz poszczególnych okien.

Własna instalacja systemu Linux

Najpopularniejsze dystrybucje:

- Debian
- Ubuntu
- **Mint Cinnamon**
- Red Hat / CentOS
- Fedora
- openSUSE
- Mandriva
- Slackware

Virtualizacja





Implementacje wirtualizacji

Pełna wirtualizacja Hipernadzorza II typu	Para-wirtualizacja	Bare Metal Hypervisor Hipernadzorza I typu
<ul style="list-style-type: none">• Wirtualizacja sprzętowa BT (Binary Translation):<ul style="list-style-type: none">◦ VirtualBox, 32bit host,◦ VMware Workstation, 32bit,◦ VMware Server,◦ Parallels Desktop.• Wirtualizacja sprzętowa VT (Virtualization Technology): VT-x, AMD-V<ul style="list-style-type: none">◦ VMware Workstation, 64bit,◦ VirtualBox, 64bit host,◦ QEMU.	<ul style="list-style-type: none">• VirtualBox (tryb parawirtualizacji)• Xen (tryb parawirtualizacji)• Oracle VM (baza: Xen)• IBM LPAR	<ul style="list-style-type: none">• KVM (moduł jądra Linux)• Microsoft Hyper-V• VMware ESXi• Xen• Oracle VM Server (baza: Xen) <p>Również implementacje wspierane sprzętowo VT, z pierwszej kolumny:</p> <ul style="list-style-type: none">◦ VMware Workstation, 64bit,◦ VirtualBox, 64bit host.

Wirtualizacja

- Pełna wirtualizacja – binarna translacja instrukcji z systemu gościa do sprzętu poprzez system hosta:
 - Wirtualizowany system operacyjny nie wymaga żadnych zmian/modyfikacji.
 - Większe bezpieczeństwo w izolacji awarii.
- Parawirtualizacja – część instrukcji i odwołań systemu gościa jest tłumaczonych, a część jest przekazywana natywnie do sprzętu:
 - Efektywniejsze wykorzystanie współdzielonych zasobów.
 - Mniejszy narzut obliczeniowy na hosta.

Podstawowe polecenia - przypomnienie

```
pwd, cd, ls, cat, cp, mv, mkdir, rm, touch, locate,  
find, grep, df, du, head, tail, diff, tar, chmod, chown,  
id, jobs, kill, ping, wget, history, man, echo, zip, unzip,  
hostname, useradd, userdel, curl, df, diff, echo, exit, finger, free,  
grep, groups, less, passwd, ping, shutdown, ssh, reboot, sudo, top,  
uname, w, whoami
```



Hello world...

```
#!/bin/bash  
  
echo "Hello world..."
```

Advanced Bash-Scripting Guide:

<https://tldp.org/LDP/abs/html/>

Uprawnienia do plików

```
krz@zinc:~/abc$ ls -al
razem 44
drwxrwxr-x  2 krz krz  4096 paź  9 18:40 .
drwx----- 98 krz krz 28672 paź  9 18:40 ..
-rw-rw-r--  1 krz krz     0 paź  9 18:40 file.txt
krz@zinc:~/abc$ █
```

- Katalog bieżący: ~/abc oraz .
- Katalog nadrzędny: ..

Ustawianie uprawnień:

```
$ chmod uprawnienia plik
$ chmod 644 file.txt
$ chmod a+rwx,a-w directory
```

Uprawnienia, przykład:

```
drwxr-x---
```

0123456789

Pozycja 0: d (dir), l (link), b (block), c (character)

Pozycja 1, 2 i 3: uprawnienia właściciela 'u'

Pozycja 4, 5 i 6: uprawnienia grupy 'g'

Pozycja 7, 8 i 9: uprawnienia pozostałych 'o'

Pozycje 1..9: uprawnienia wszystkich 'a'

rwx - read, write, eXecute

421 - zapis binarny, np. r-x = 5, rw- = 6, r-- = 4.

Uprawnienia do plików

0	---	brak uprawnień	blokada
1	--x	wykonywanie	nieprzydatne
2	-w-	zapis	zbieranie sekretnych logów
3	-wx	zapis i wykonywanie	nieprzydatne
4	r--	odczyt	stała konfiguracja
5	r-x	odczyt i uruchamianie	pliki wykonywalne
6	rw-	odczyt i zapis	pliki edytowalne
7	rwx	odczyt, zapis i uruchamianie	skrypty i programy
	??s	bit suid	programy specjalne

Uprawnienia do katalogów

0	---	brak uprawnień	blokada
1	--x	wykonywanie	katalog bez podglądu
2	-w-	zapis	nieprzydatne
3	-wx	zapis i wykonywanie	nieprzydatne
4	r--	odczyt	nieprzydatne
5	r-x	odczyt i uruchamianie	katalogi bez możliwości zmian zaw.
6	rw-	odczyt i zapis	nieprzydatne
7	rwx	odczyt, zapis i uruchamianie	katalogi z możliwością zmian zaw.
	??t	sticky bit	katalog specjalny

Uprawnienia do uruchomienia

Podstawowe uprawnienie do wczytania i uruchomienia skryptu:

```
$ chmod 555 skrypt
```

lub:

```
$ chmod +rx skrypt
```

Sprawdź:

```
$ chmod u+s skrypt
```

Skrypty - podstawy

Przypisanie wartości do zmiennej:

```
$ a=5  
$ b=$a
```

Przypisanie wyniku działania polecenia do zmiennej:

```
$ a=`ls /var/log | wc -l` lub nowsze: a=$(ls /var/log | wc -l)
```

Wypisanie wartości zmiennej na ekran:

```
$ echo "Wartość a wynosi $a"
```

Wybrane znaki specjalne

* wildcard

```
$ ls /usr/*bin/
```

? test operator

```
$ variable = a<10?5:7
```

{a,b,c} rozwijanie

```
$ file.{bin,txt} == file.bin file.txt
```

```
{a..z} lub {1..9}
```

\$[...] obliczenie

```
$ a=5; c=$[$a+5] lub: $ let c=$a+5
```

\$ oznaczenie zmiennej

```
$ a=5; echo $a
```

Operacje na zmiennych

Podstawienie

```
$ a='Ala ma kota'  
$ b=${a/Ala/Piotr}  
$ echo $b
```

Wyzerowanie zmiennej:

```
$ a=''      lub  a=    lub a=""
```

Ćwiczenie:

```
$ a=5  
$ a+=6    vs.  $ let a+=6  
$ echo $a
```



Warunek if

```
if [ condition ]      if [ condition ]      if [ condA ] && [ condB ]  
then                  then                  then  
    command            command            command  
    ...  
fi                   elif [ condition ]    elif [ condC ] || [ codD ]  
                      command            command  
                      ...  
                      else                else  
                      command            command  
                      ...  
fi                   fi
```

Pętla for

```
for arg [list]
do
    command
...
done
```

```
for n in one two
do
    command
...
done
```

```
for n in {1..5}
do
    command
...
done
```

```
for n in "a b" "c d"
do
    Set -- $n
    cmd with $1
    cmd with $2
...
done
```

Pętla while

```
while [ condition ]  
do  
    command  
    ...  
done
```

```
while [ $a -lt $b ]  
do  
    command  
    ...  
done  
lt, le, gt, ge
```

```
cond()  
{ if ... return 0;  
else return 1; }  
  
while cond  
do  
    command  
    ...  
done
```



Pętla until

Czy jest sterowana odwrotnym warunkiem niż while ?



Podnoszenie uprawnień: su, sudo (doas)

Zalogowany jako 'root':

```
# id          -> uid=0(root) gid=0(root) grropy=0(root)
```

Zalogowany jako użytkownik:

```
$ id          -> uid=1000(krz) gid=1000(krz) grropy=1000(krz),...
```

```
$ su          zmiana id użytkownika na 0 (superużytkownik, root)
```

wymagane: hasło użytkownika root

wykonanie pojedynczego polecenia: \$ su -c <polecenie>

```
$ sudo        wykonanie polecenia z uprawnieniami root'a
```

wymagane: hasło bieżącego użytkownika + uprawnienia w /etc/sudoers

uzyskanie powłoki: \$ sudo -i

Podnoszenie uprawnień: su, sudo - c.d.

```
$ sudo id          -> uid=0(root) gid=0(root) grumpy=0(root)  
$ su - -c "id"--> uid=0(root) gid=0(root) grumpy=0(root)
```

Aby możliwe było użytkowanie polecenia `su` należy ustawić hasło dla root:

- Typowa instalacja Linux: hasło ustawione jest podczas instalacji systemu.
- Instalacja Ubuntu i pochodnych: hasło trzeba ustawić poprzez `sudo`:

```
$ sudo passwd root
```

Aby możliwe było korzystanie z polecenia `sudo` należy ustawić uprawnienia (`visudo`):

- Typowa instalacja Linux: jest konto 'root', brak ustawień sudoers.
- Instalacja Ubuntu i pochodnych: użytkownik konfigurowany w trakcie instalacji ma uprawnienia.

Podnoszenie uprawnień: su, sudo - c.d. 2

Połączenie poleceń:

```
$ sudo su      - mając w /etc/sudoers uprawnienia do sudo bez hasła przejdziemy do 'root'  
$ sudo -i
```

Taki sposób **nie wywołuje** wykonania .bashrc konta root:

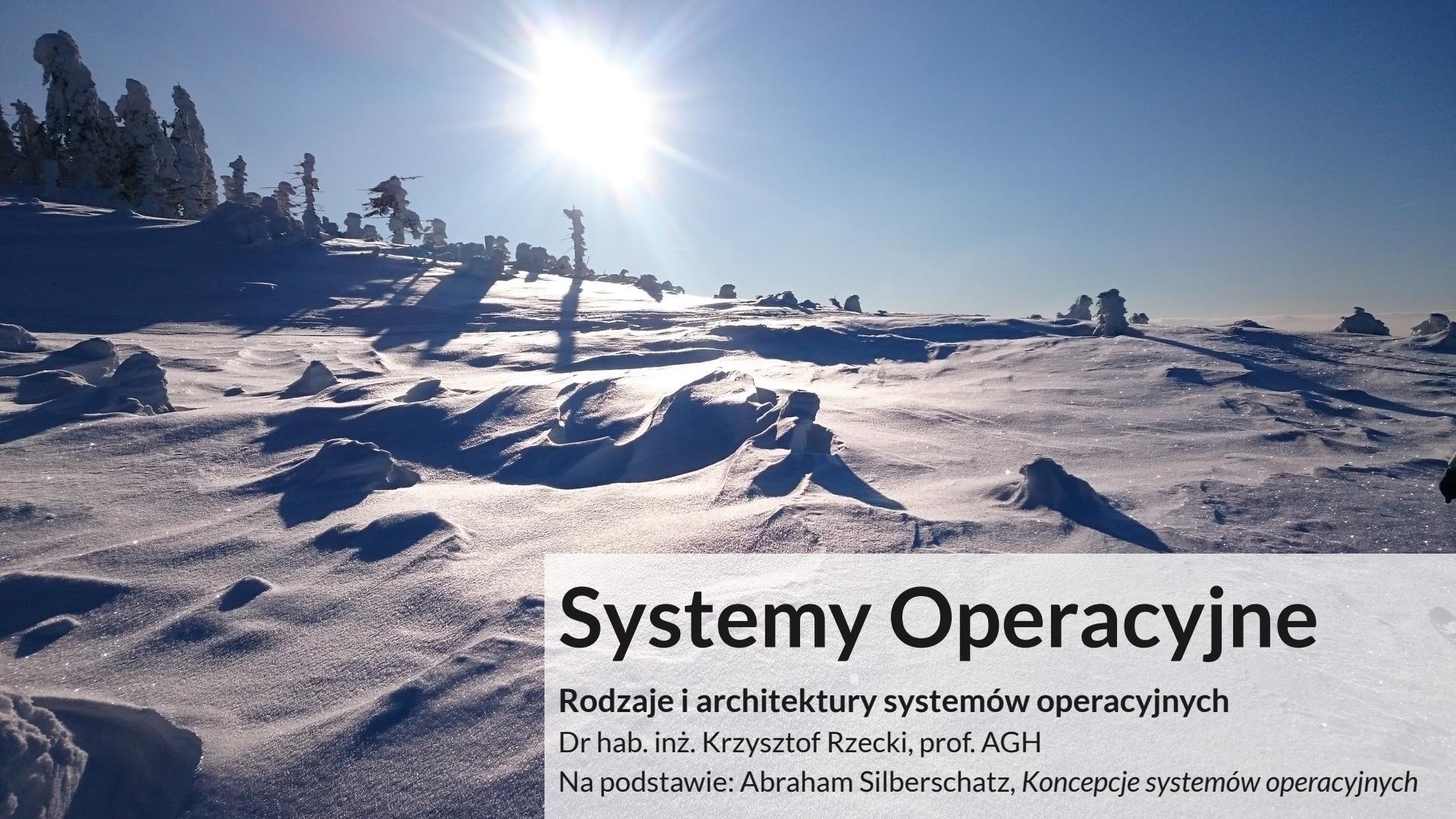
```
$ su -c <polecenie>  
$ sudo polecenie
```

Taki sposób **wywołuje** wykonanie .bashrc konta root (odpowiednik: \$ source .bashrc):

```
$ su - -c <polecenie>  
$ sudo su - -c <polecenie> !!!
```



Czerwone Wierchy



Systemy Operacyjne

Rodzaje i architektury systemów operacyjnych

Dr hab. inż. Krzysztof Rzecki, prof. AGH

Na podstawie: Abraham Silberschatz, *Koncepcje systemów operacyjnych*

System operacyjny

System operacyjny - zarządza sprzętem komputerowym, jest pośrednikiem między użytkownikiem, a sprzętem komputerowym

Sprzęt komputerowy - CPU, RAM, I/O devices

Zadania systemu operacyjnego:

- dostarczenie środowiska do uruchamiania i zarządzania (ang. *control program*) programami użytkownika (*wygoda*),
- dystrybucja zasobów (ang. *resource allocator*) do efektywnej eksploatacji sprzętu komputerowego (*wydajność*).

Użytkownicy	
Programy użytkowe	
System operacyjny	Powłoka
	Jądro i moduły
BIOS - <i>Basic Input Output System</i>	
Sprzęt komputerowy	

System operacyjny jako program sterujący

Program sterujący (ang. *control program*):

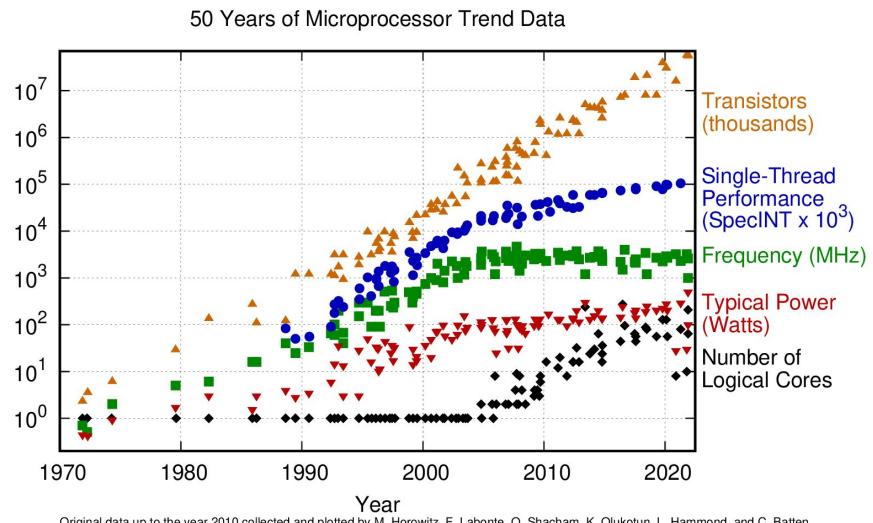
- Nadzorowanie działania programów użytkowych.
- Przechwytywanie i przeciwdziałanie błędom.
- Udostępnianie systemu komputerowego użytkownikom.
- Kontrola dostępu użytkowników i programów do zasobów.
- Obsługa i kontrola pracy urządzeń wejścia-wyjścia.

Dystrybucja zasobów

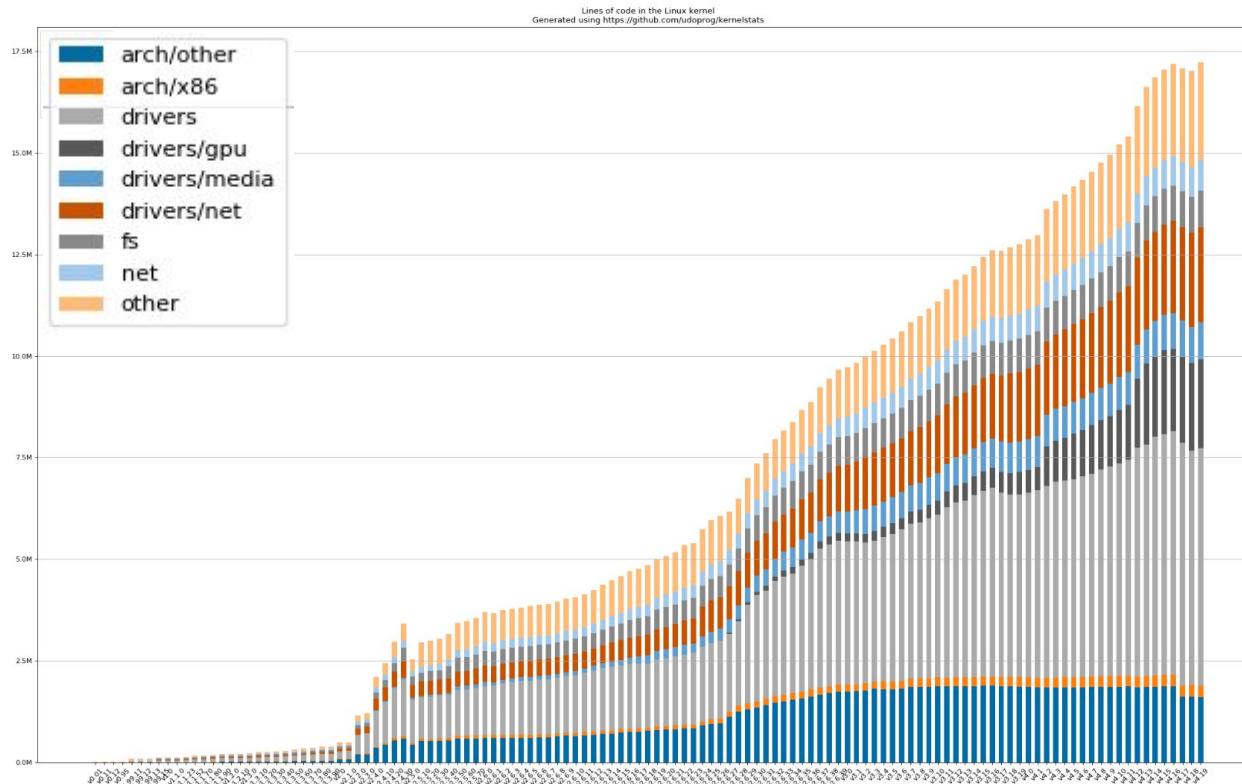
Dystrybucja zasobów obejmuje:

- Planowanie i przydział czasu procesora.
- Kontrola i przydział pamięci operacyjnej.
- Zarządzanie pozostałymi zasobami, jak oprogramowanie czy dostęp do sieci internet.
- Dostarczenie mechanizmów do synchronizacji zadań i komunikacji między zadaniami.

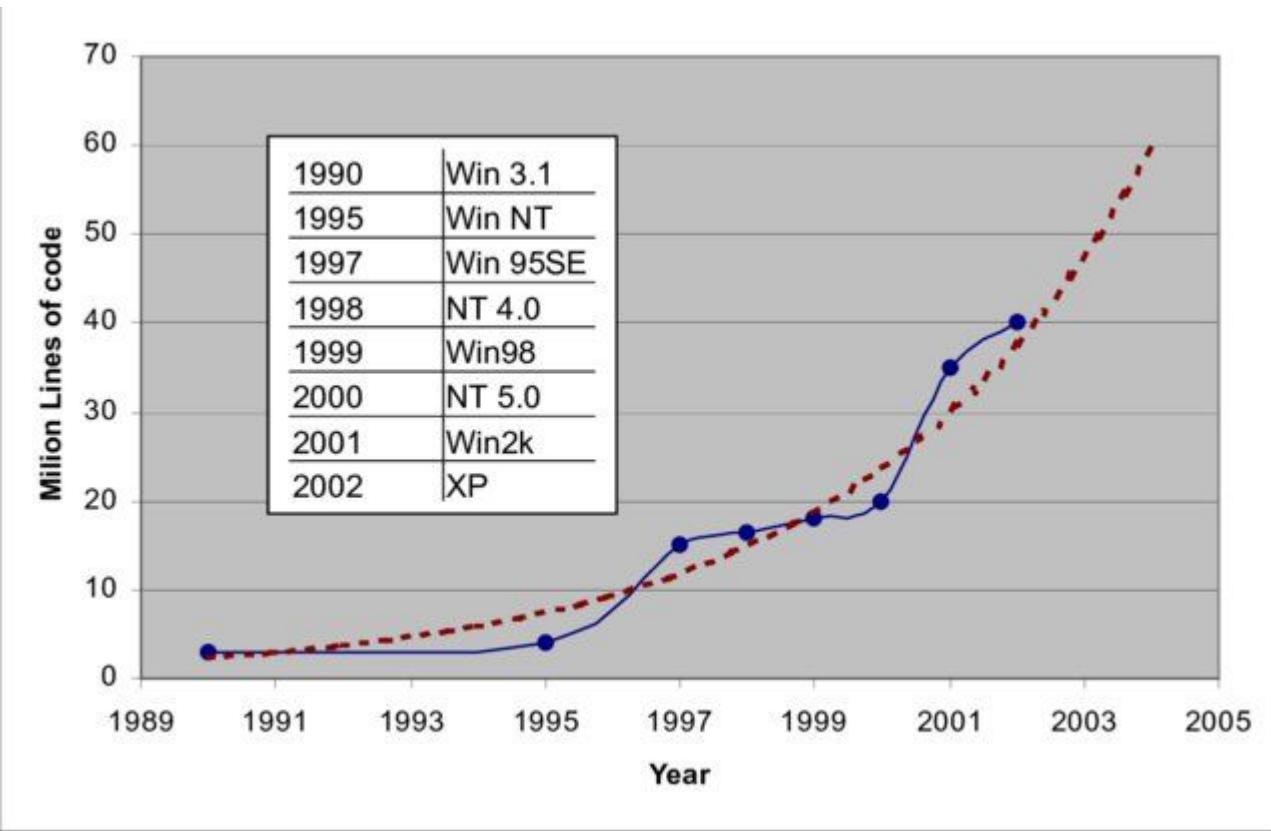
Rozwój sprzętu - prawo Moore'a



Źródło: <https://github.com/karlrupp/microprocessor-trend-data>



Źródło: <https://github.com/udoprog/kernelstats>



Źródło: <https://github.com/udoprog/kernelstats>

Podział systemów komputerowych

- Ze względu na wielkość:
 - Mainframe
 - Minicomputer
- Ze względu na zasoby:
 - Server
 - Workstation

Czy system operacyjny to program ?

- System operacyjny to jedyny program działający cały czas na komputerze
 - (uwaga: systemy wbudowane)
- Program ten zwykle nazywamy: **jądro** (ang. *kernel*)
- Pozostałe typy programów:
 - Programy systemowe (część z nich to polecenia systemowe)
 - Programy aplikacyjne (aplikacje użytkowe)

Jednostka danych

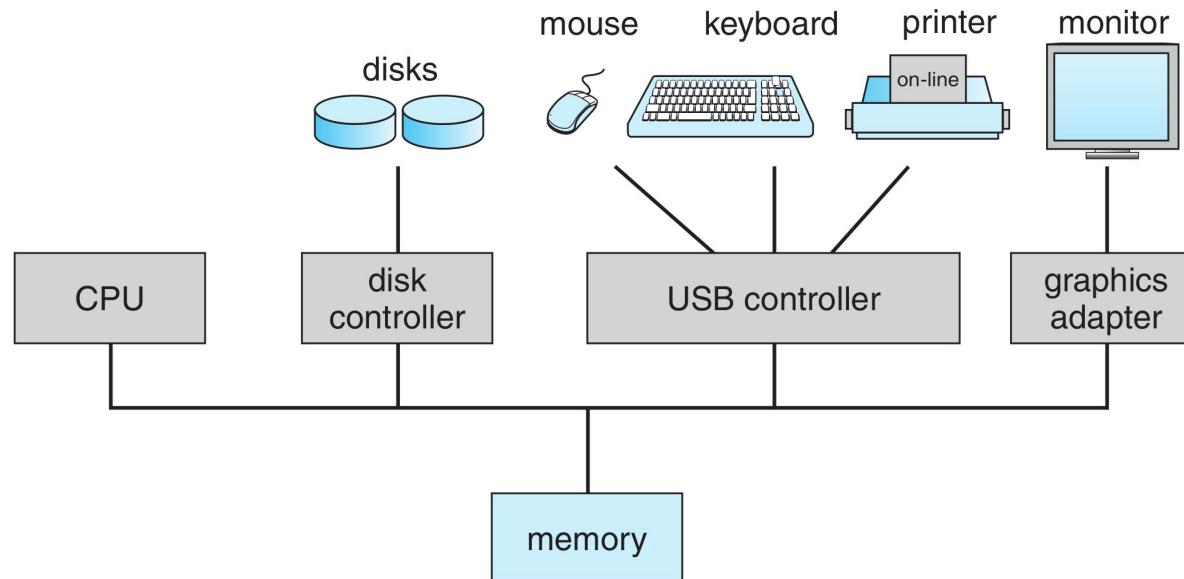
- **bit** - podstawowa jednostka informacji, oznaczenie: b
- bit może przyjmować wartość 0 lub 1
- kombinacja bitów reprezentuje cyfry, litery, obrazy, wideo, dźwięki, dokumenty, programy, itp.
- **bajt** to 8 bitów, oznaczenie B
- **word** to natywna dla danej architektury komputera jednostka informacji
- np. w architekturze 64-bitowej słowo to 8 bajtów
- Mnożniki:
 - $1 \text{ KB} = 1024 \text{ B} = 2^{10} \text{ B}$
 - $1 \text{ MB} = 1024 \text{ KB} = 1024^2 \text{ B} = 2^{20} \text{ B}$ (małe k = 10^3 , wielkie K = 2^{10})
 - $1 \text{ GB} = 2^{30} \text{ B}$, itd.
 - $1 \text{ Kib} = 1024 \text{ b}$ i analogicznie Mib = 2^{20} b , Gib = 2^{30} b , itd.

Młody informatyk myśli, że 1 kilobajt to 1000 bajtów, stary informatyk jest przekonany, że 1 kilometr to 1024 metry...

Elementy systemu komputerowego

- Sprzęt: procesor CPU, pamięć RAM, urządzenia we-wy
- Systemy operacyjne: Linux/Unix, Windows, MacOS, itp.
- Programy użytkowe: aplikacje, systemy baz danych, gry komputerowe, oprogramowanie biurowe, środowiska programistyczne, itp.
- Użytkownicy: ludzie, programy, maszyny

Organizacja systemu komputerowego



Działanie systemu komputerowego

- Uruchamianie systemu komputerowego:
 - BIOS - Basic Input-Output System
 - Bootstrap program przechowywany w ROM - read-only memory lub EEPROM - erasable programmable read-only memory zwany zwykle firmware
 - Następuje znalezienie i załadowanie systemu operacyjnego oraz pierwszego procesu "init"
 - Oczekивание на здравия
- Wystąpienie zdarzenia sygnalizowane jest przez przerwanie (ang. *interrupt*):
 - Sprzętowe przerwanie może być wyzwolone przez przesłanie sygnału przez szynę systemową do procesora
 - Programowe przerwanie może być wyzwolone specjalną operacją, tzw. *system call* lub *monitor call*
- W momencie wystąpienia przerwania, procesor przerywa aktualnie wykonywaną operację, wykonuje procedurę przewidzianą dla danego zdarzenia i wraca do przerwanej operacji.

Struktura pamięci

- RAM - *random access memory* - pamięć, do której ładowane są programy do wykonania oraz dane dla tych programów
- Pamięci RAM to zwykle technologia półprzewodnikowa DRAM - *dynamic random-access memory*
- ROM - *read-only memory* - pamięć, w której przechowywane są niezmienialne programy, np. wgrane gry do konsol video
- EEPROM - *erasable programmable read-only memory* - pamięć, która nie może być zbyt często nadpisywana, np. system operacyjny urządzeń mobilnych
- Rejestr - szybka, niewielka pamięć przy/w procesorze na czas wykonywania pojedynczej instrukcji
- Pamięć stała: dysk twardy HDD - *hard disk drive*, dysk optyczny, pen drive, itp.
- Pamięć NVRAM - *nonvolatile RAM* - odpowiedni pamięci RAM podtrzymywany baterijnie

Proste systemy wsadowe

- Pierwsze komputery:
 - Wejście: czytniki kart i przewijaki taśm
 - Wyjście: drukarki wierszowe, przewijaki taśm, perforatory kart
- Zadanie na karcie perforowanej: program, dane, karty sterujące
- Czas obliczeń: minuty+ (czasem dni)
- System operacyjny umieszczony na stałe w pamięci operacyjnej
- Grupowanie zadań o podobnych wymaganiach: wsad (ang. *batch*)
- Komputer obsługiwał operator, który pobierał i sortował programy
- Istotna różnica w szybkości działania procesora w porównaniu z we/wy
- Następstwem było wprowadzenie technologii dyskowej



Wieloprogramowe systemy wsadowe

- Zastosowanie pamięci o dostępie swobodnym (dysków)
- Wczytywanie kart na dysk i zapamiętanie położenia danych => spooling
- Pula zadań (ang. *job pool*) - wczytanie pewnej liczby zadań na dysk
- Możliwość dobierania zadań z dysku tak, aby zwiększyć efektywność jednostki centralnej
- Planowanie zadań (ang. *scheduling*) - planowanie zadań i planowanie przydziału procesora

Systemy z podziałem czasu (1960r.)

- Problemy systemów wieloprogramowych:
 - wielowariantowość ścieżek wykonywania
 - brak możliwości modyfikowania programu
 - długie czasy od rozpoczęcia tworzenia programu do wyniku jego działania
- Podział czasu, wielozadaniowość, ang. *multitasking*
- Interakcyjność, ang. *hands-on* - wymiana danych z programem w ciągu jego trwania
- System plików (ang. *file, filesystem*):
 - zestaw powiązanych informacji
 - format, typ
 - organizacja w katalogi
- Bezpośredni dostęp użytkownika do komputera (bez operatora)
- Pamięć wirtualna - wspomaganie pamięci operacyjnej pamięcią dyskową
- Problem zakleszczenia - wzajemne oczekiwanie programów na zasoby

Systemy dla komputerów osobistych ('70)

- Zmniejszenie cen sprzętu
- Rozwój linii komputerów PC (ang. *personal computer*)
- Początkowo systemy operacyjne dla pierwszych komputerów osobistych: nie wielostanowiskowe, nie wielozadaniowe, lecz z czasem je rozwinięto
- Microsoft: MS-DOS, później Microsoft Windows
- IBM: MS-DOS (od Microsoft), później OS/2
- Apple: Macintosh, później iOS
- Bell Laboratories: UNIX dla PDP-11 (wiele koncepcji z systemu MULTICS dla komputera GE645)
- Na bazie rozwiązań UNIX w latach '80 => Windows NT, IBM OS/2, Macintosh Operating System

Systemy wieloprocesorowe, równoległe

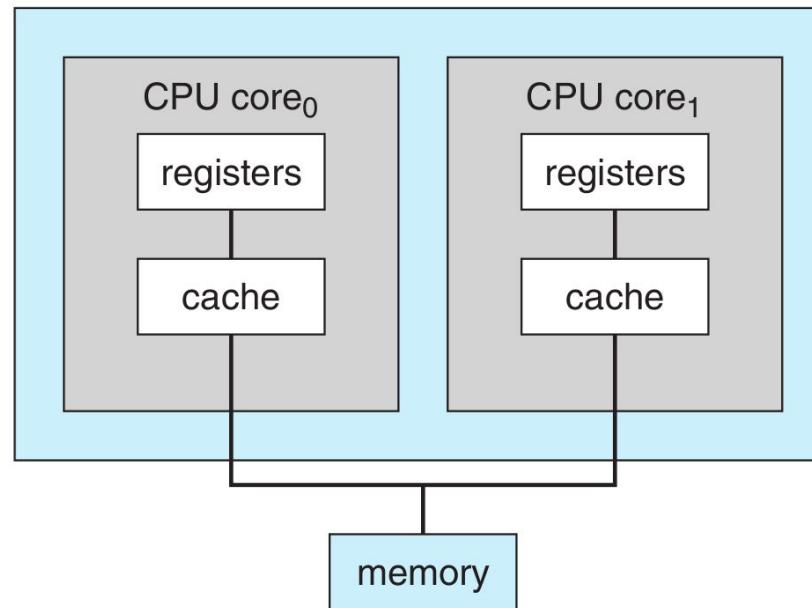
- Zwiększała przepustowość
- Współczynnik przyspieszenia, który nie zawsze powiela wydajność
- Współużytkowanie urządzeń zewnętrznych
- Zwiększenie niezawodności (redundancja/nadmiarowość węzłów obliczeniowych)
- Wieloprzetwarzanie symetryczne - w każdym procesorze działa identyczna kopia
 - Działa N procesów na N egzemplarzach jednostki centralnej
 - Może się zdarzyć niezbalansowanie obciążenia procesorów
 - Wersja Encore systemu UNIX dla komputera Multimax
 - SunOS w wersji 5 (Solaris 2) dla komputerów Sun
- Wieloprzetwarzanie asymetryczne - każdy procesor ma inne zadanie (+procesor główny)
 - Np. słabsze procesory obsługują komunikację (ang. front-end)
 - IBM i komputer IBM Series/1 jako procesor czołowy
 - SunOS w wersji 4 dla komputerów Sun

UMA/NUMA

- UMA - *uniform memory access* - pamięć o jednorodnym czasie dostępu
- NUMA - *non-uniform memory access* - pamięć o niejednorodnym czasie dostępu

System operacyjny musi tak zarządzać dostępem do pamięci, aby minimalizować efekt NUMA.

Procesor wielordzeniowy



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Systemy rozproszone, systemy klastrowe

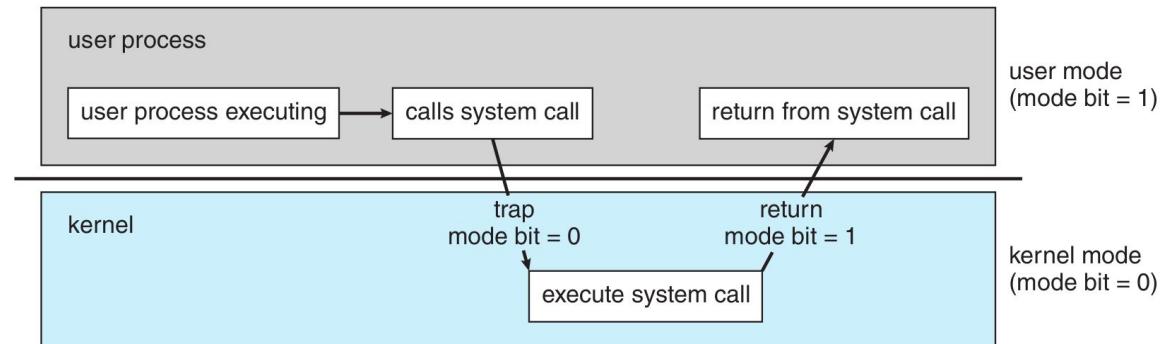
- Systemy luźno powiązane (ang. *loosely coupled*), rozproszone (ang. *distributed systems*)
- Rozdzielenie geograficzne
- Połączenie przez linie telekomunikacyjne (internet, linie telefoniczne, itp.)
- Zróżnicowanie architektur poszczególnych węzłów (ang. *node*)
- Zróżnicowanie mocy obliczeniowych poszczególnych węzłów
- Cechy:
 - Podział zasobów
 - Przyspieszenie obliczeń
 - Niezawodność
 - Komunikacja

Systemy czasu rzeczywistego (ang. *real-time*)

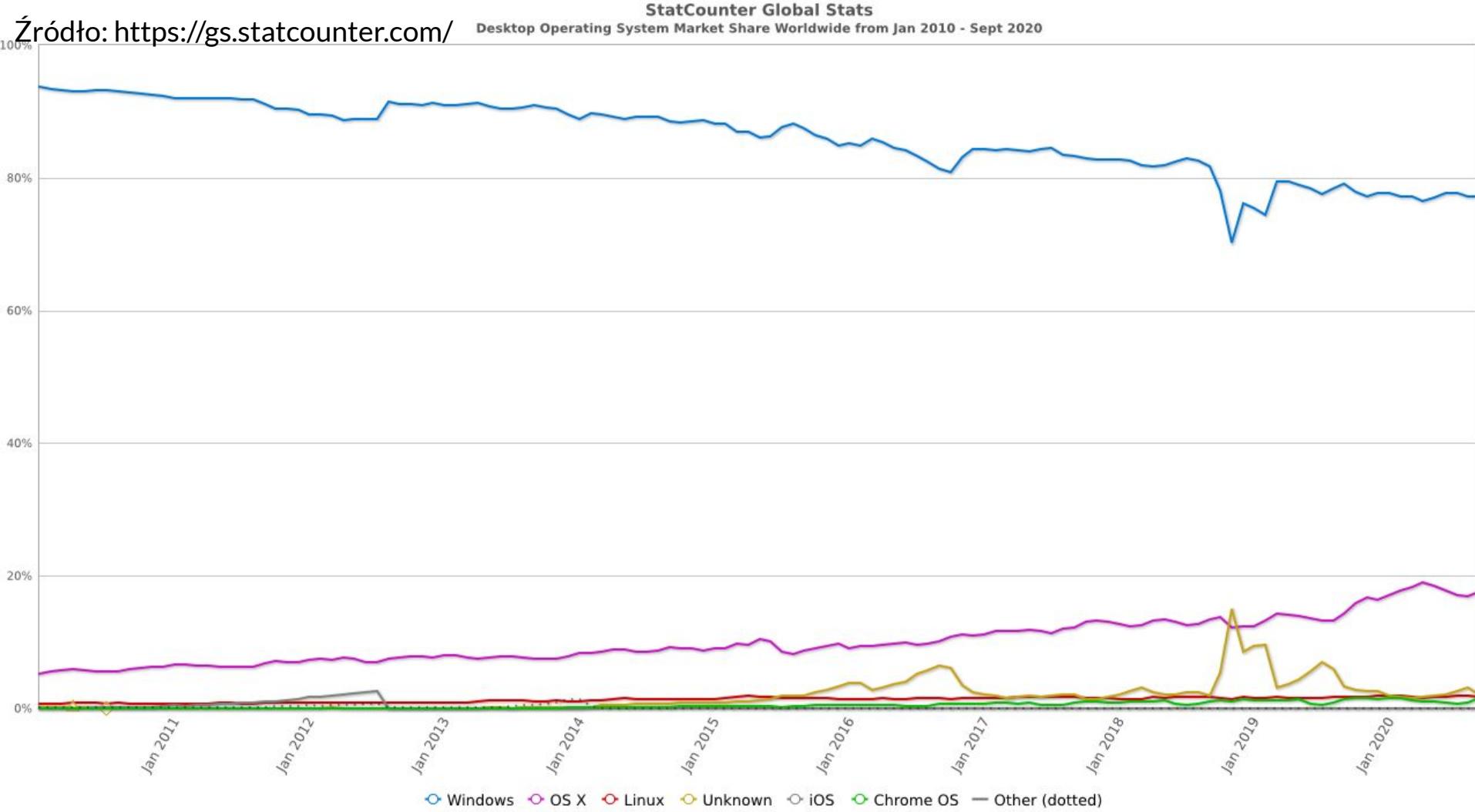
- Zastosowanie: surowe wymagania na czas wykonania operacji lub przepływu danych
- Przykłady:
 - Jednokierunkowe sterowanie maszyną według zadanego programu
 - Odczytywanie wartości czujników
 - Analiza odczytanych wartości czujników i adekwatna reakcja robota
 - Analiza otoczenia, przetwarzanie danych (sygnałów, obrazów) i podejmowanie decyzji
- Odmiany systemów czasu rzeczywistego:
 - Rygorystyczny (ang. *Hard real-time system*) - terminowe wypełnienie krytycznych zadań
 - Łagodny (ang. *Soft real-time system*) - krytyczne zadanie do obsługi otrzymuje pierwszeństwo

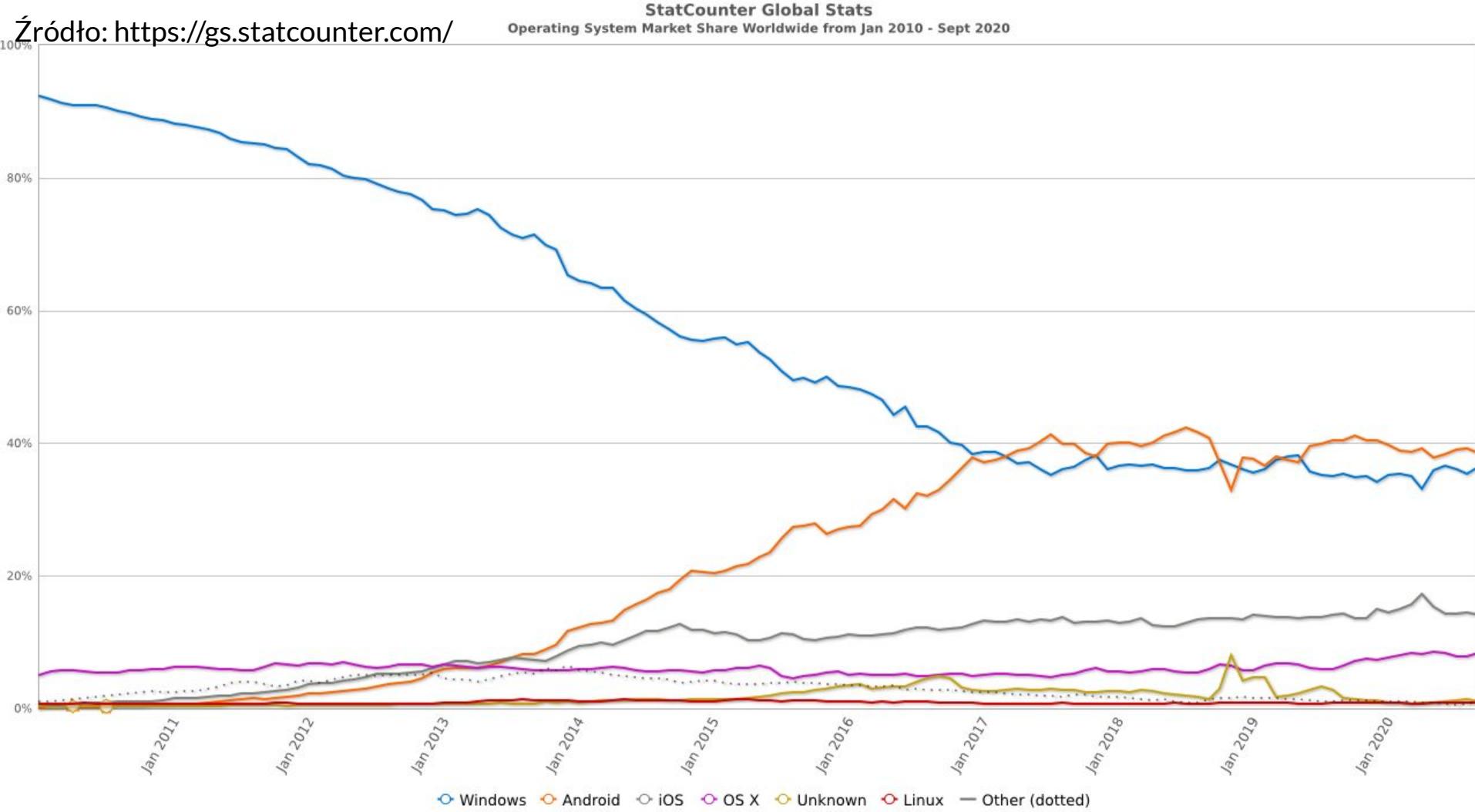
Tryby pracy systemu operacyjnego

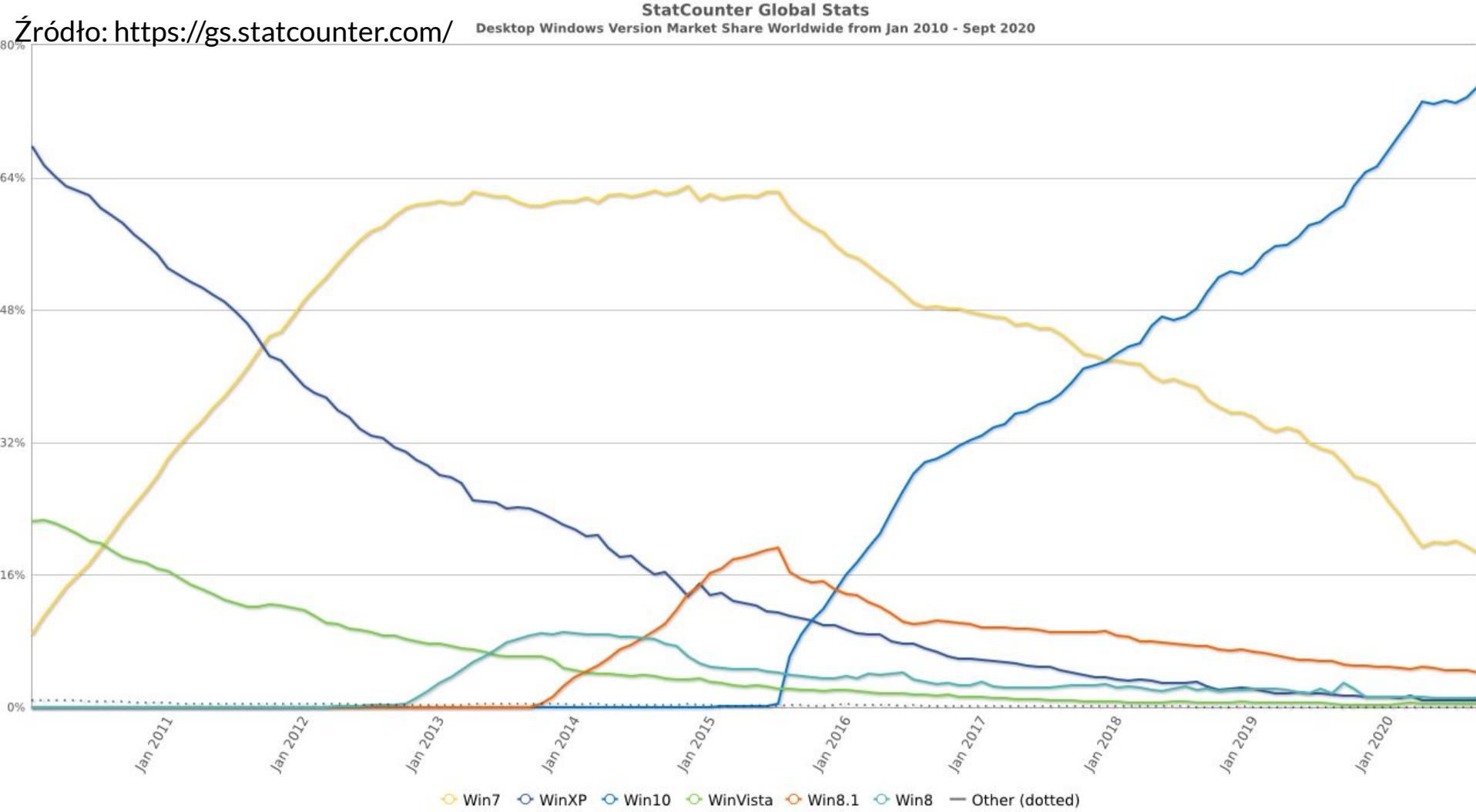
- Tryb użytkownika (ang. *user mode*)
- Tryb jądra (ang. *kernel mode*)
- Zaimplementowany w sprzęcie *mode bit*: 0 - kernel mode, 1 - user mode

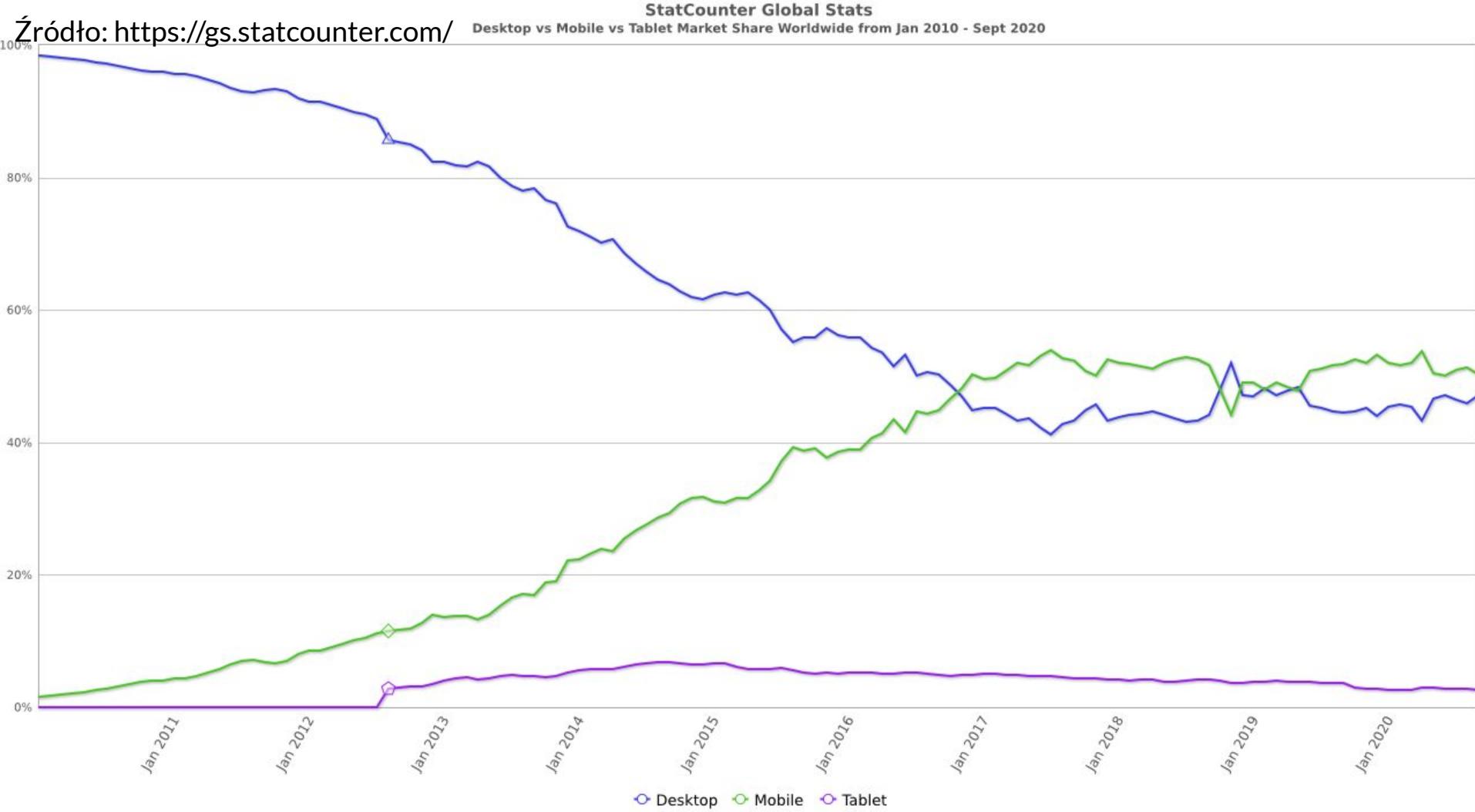


Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*





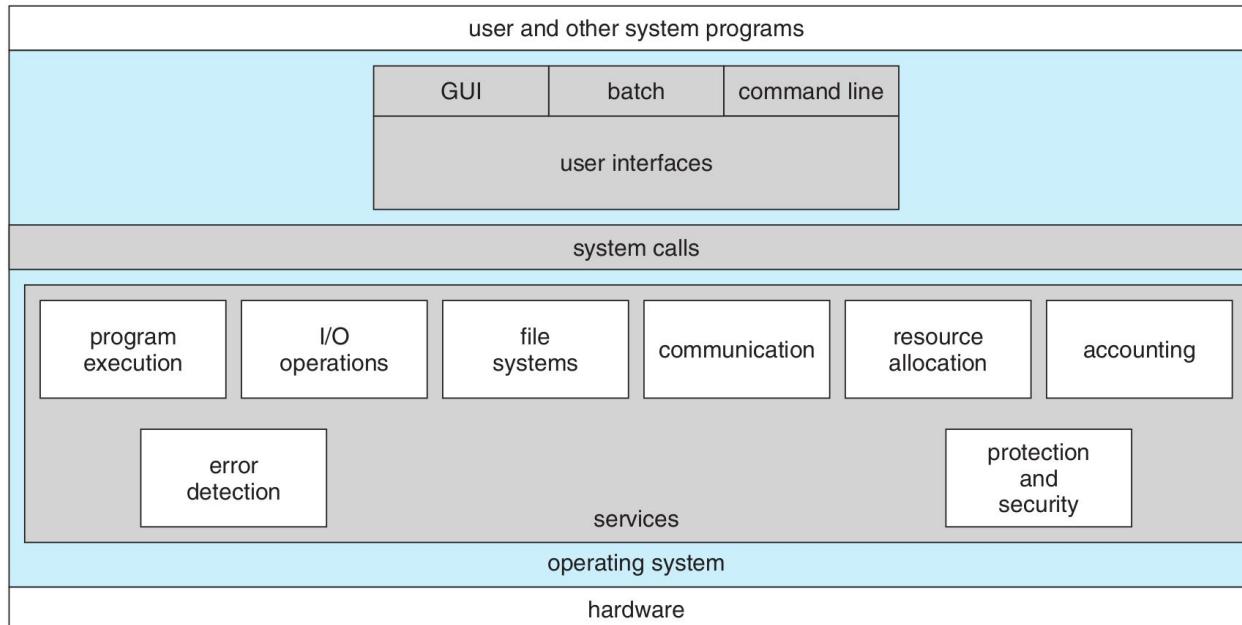




Pytania...

- Jakie zagrożenia niesie za sobą wielodostęp ?
- Co jest główną zaletą wieloprogramowości ?
- Czy współcześnie może mieć zastosowanie system bez dysku twardego ?
- Jakie uzasadnienie istnienia mają systemy rozproszone ?
- Jaka jest największa trudność w tworzeniu systemu czasu rzeczywistego ?
- Jakie znaczenie ma tryb jądra i tryb użytkownika dla bezpieczeństwa systemu ?

Struktura systemu operacyjnego



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Interfejs użytkownika

- Graficzny interfejs użytkownika (GUI - graphical user interface) to najpopularniejszy wśród użytkowników biurowych interfejs człowiek-komputer.
- Interpreter poleceń (CLI - command-line interface) jest interfejsem użytkownika (UI) przyjmującym komendy tekstowe oraz funkcje.
- Interfejs *batch* to zbiór komend i dyrektyw kontrolujących te komendy zapisanych w pliku, który po uruchomieniu kolejno uruchamia komendy z tego pliku.
- Interpreter poleceń == *shell*: sh (Bourn shell), bash (Bourne-Again shell), csh (C shell), zsh (Z shell), ksh (Korn shell), itp.
- Interpretacja poleceń: polecenie systemowe (np. cd) lub program (np. rm).

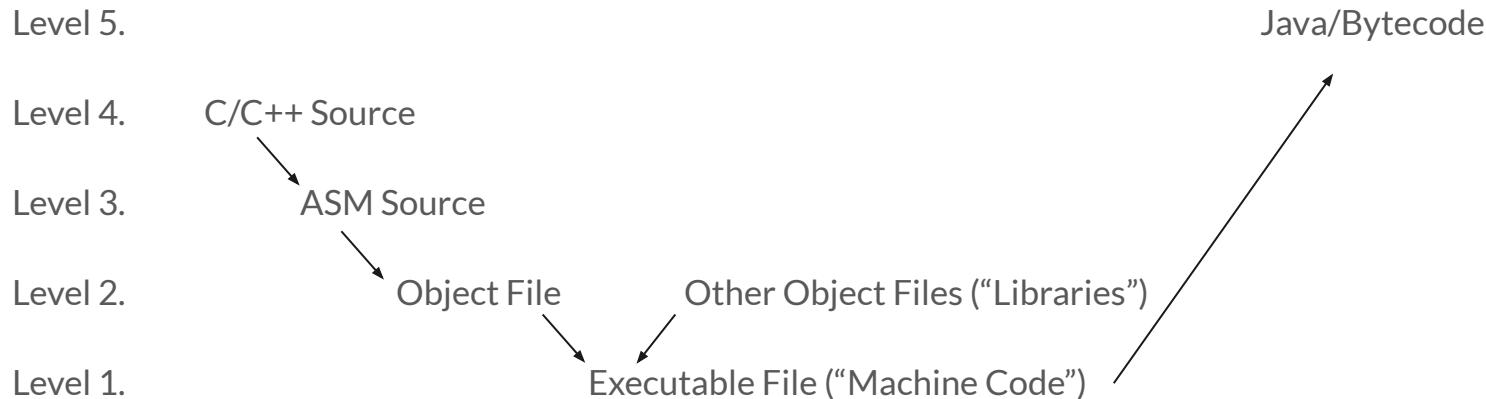
Podstawowe polecenia bash

- Dokumentacja na temat poleceń dostępna jest (wychodzimy klawiszem 'q'):

```
$ man <nazwa polecenia lub programu>
```

- **Polecenia:** pwd, cd, ls, cat, cp, mv, mkdir, rm, touch, locate, find, grep, df, du, head, tail, diff, tar, chmod, chown, jobs, kill, ping, wget, history, man, echo, zip, unzip, hostname, useradd, userdel, curl, df, diff, echo, exit, finger, free, grep, groups, less, passwd, ping, shutdown, ssh, reboot, sudo, top, uname, w, whoami
- **Instrukcje:** instrukcjami skryptowymi: if, for, while, until, etc.
- **Edytory tekstu:** vim, pico, etc.

Poziomy języków programowania





Skrzyczne



Systemy Operacyjne

Procesy

Dr hab. inż. Krzysztof Rzecki, prof. AGH

Na podstawie: Abraham Silberschatz, *Koncepcje systemów operacyjnych*



Od programu do procesu

Tworzenie programu:

Kod źródłowy -> Kompilacja -> Linkowanie -> Kod binarny

Uruchomienie programu:

Kod binarny, czyli program -> Wczytanie do pamięci operacyjnej -> Nadanie PCB -> Proces



Proces

- Pierwsze systemy operacyjne: jeden program, który ma dostęp do wszystkich zasobów
- Obecnie: zarządzanie **uruchomionymi programami -> procesami**
- Proces to inaczej:
 - wczytany do pamięci i uruchomiony kod
 - jednostka pracy w systemie z dzieleniem czasu
- Zarządzanie to obejmuje: kontrola i separacja
- Efekt: system zawiera kolekcje procesów:
 - procesy systemu operacyjnego
 - procesy użytkownika
- Potencjalnie wszystkie w/w procesy uruchamiane są jednocześnie
- Procesor(y) przełączają się między procesami, co zwiększa efektywność systemu komputerowego

Powoływanie i terminowanie procesu (*)

Powoływanie (tworzenie ?):

- Nowe zadanie wsadowe
- Interaktywne logowanie
- Utworzenie usługi przez SO
- Podział (*spawn*) istniejącego procesu

Terminowanie:

- Prawidłowe zakończenie
- Brak pamięci
- Naruszenie ochrony
- Interwencja operatora lub SO

Proces informuje o zakończeniu:

- Instrukcja HALT
- Akcja użytkownika (np. log off)
- Awaria lub błąd
- Terminowanie procesu rodzica

Powoływanie procesu - interfejs

Program w ścieżce:

\$ xclock

Program użytkownika:

\$./program

Program w tle:

\$ xclock &

Terminowanie programu:

ctrl + c

Podgląd zadań:

\$ jobs

Przeniesienie w tło:

\$ bg %1

Wstrzymanie pracy programu:

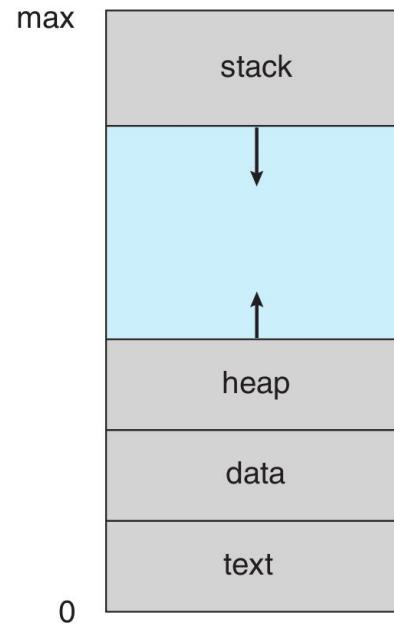
ctrl + z

Przeniesienie na pierwszy plan:

\$ fg %1

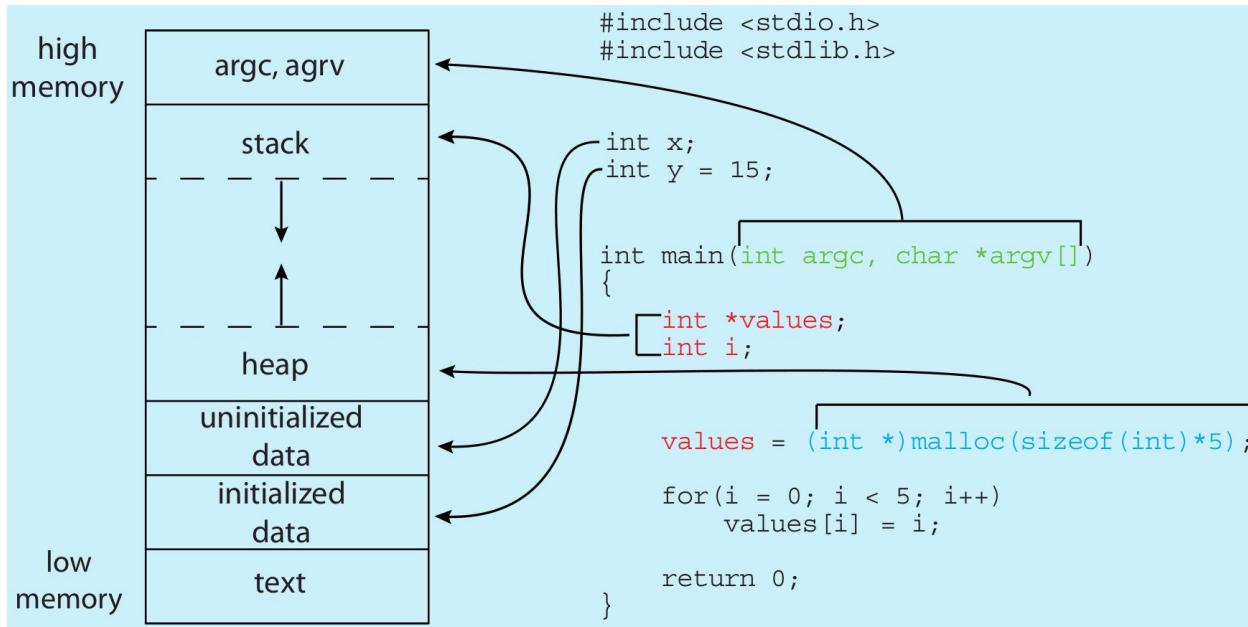
Koncepcja procesu

- System wsadowy: *jobs*, system współdzielony: *user programs* lub *tasks* -> obecnie: *process*
- Proces:
 - Kod programu, *text section*
 - Licznik programu, *program counter* == Licznik rozkazów == Wskaźnik wykonywanej instrukcji
 - Stos procesu, *process stack* (parametry procedur, adresy powrotne, zmienne tymczasowe)
 - Sekcja danych, *data section*, czyli zmienne globalne
 - Sterta, *heap*, czyli pamięć przydzielana dynamicznie
- Program jest pasywny - plik wykonywalny zawierający listę instrukcji
- Proces jest aktywny - licznik rozkazów wskazuje następną instrukcję

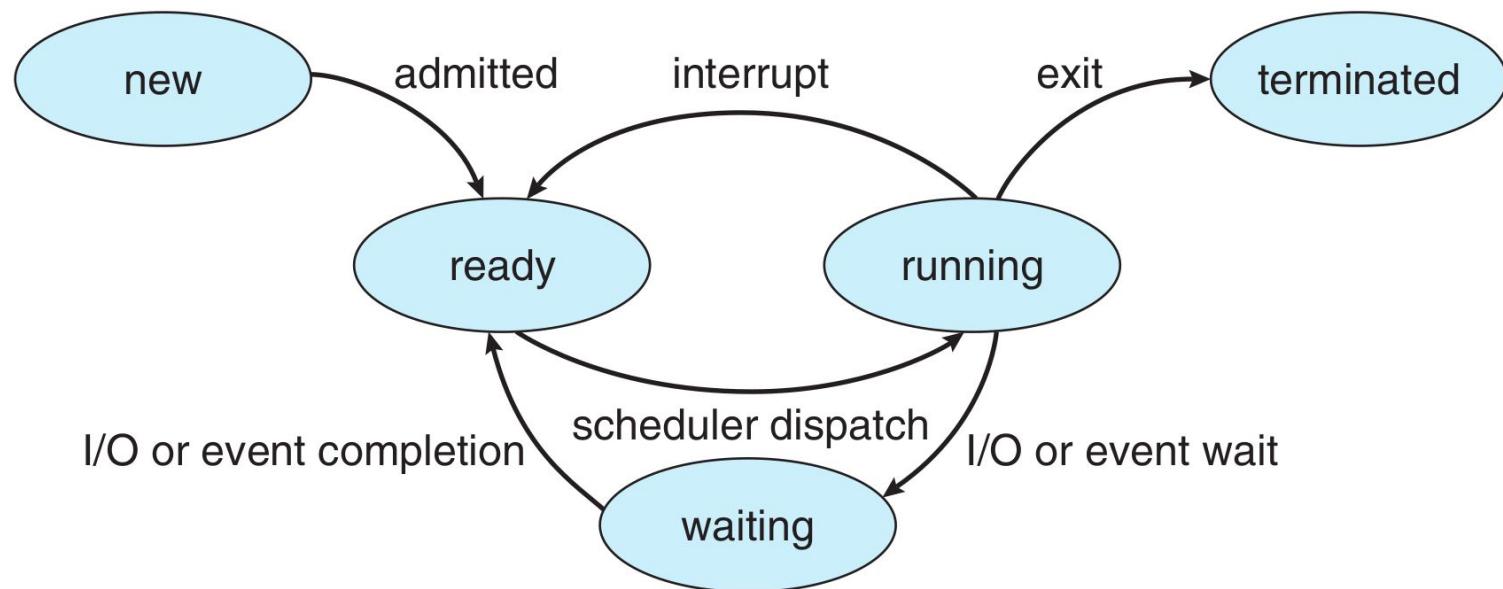


Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Program w C

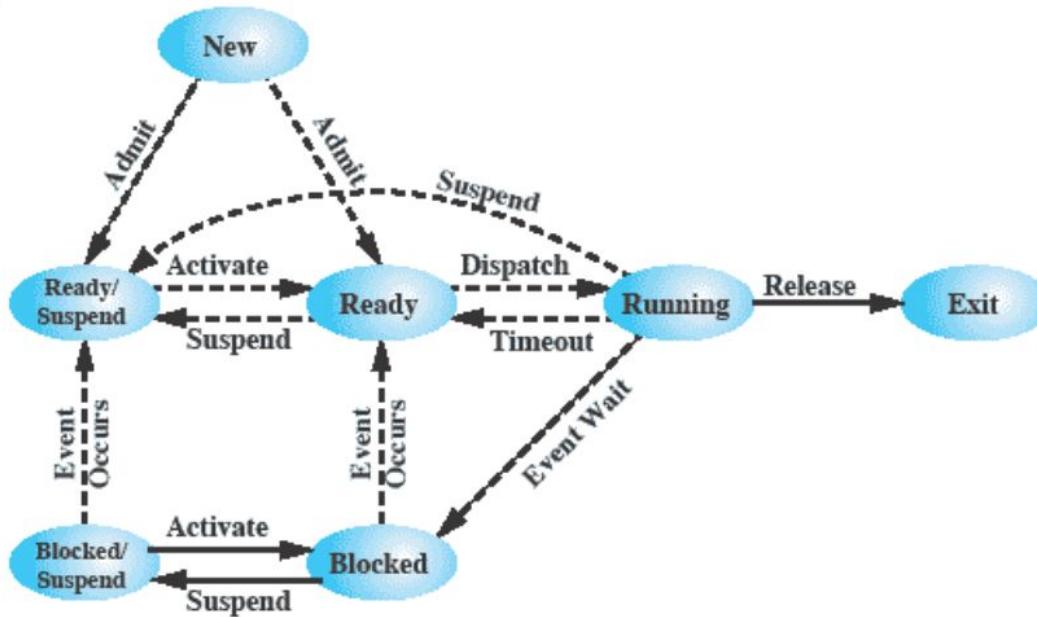


Stany procesu (model pięciostanowy)



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Dwa stany zawieszenia (*)



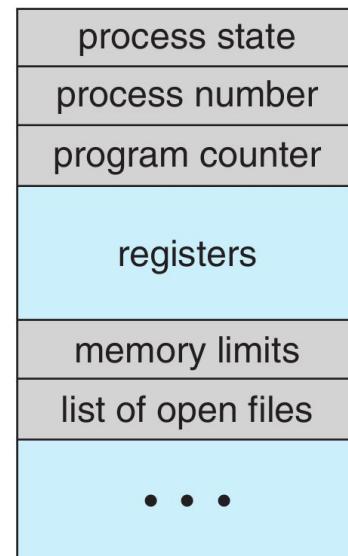
Powody zawieszania procesu (*)

- **Swapping** - system operacyjny musi zwolnić pewną ilość pamięci (głównej), aby uruchomić proces, który jest gotowy do wykonania.
- **Inny problem SO** - błędy.
- **Interaktywne żądanie użytkownika**
- **Timing** - periodyczne wywołanie procesu (np. monitorowanie) i może zostać on zawieszony do następnego wywołania.
- **Żądanie procesu macierzystego** - SIGSTOP / SIGTSTP / SIGCONT

Blok kontrolny procesu

PCB - ang. *process control block*

Jest to obszar pamięci zawierający różne informacje skojarzone z procesem, którego dotyczy.



process state
process number
program counter
registers
memory limits
list of open files
• • •

Blok kontrolny procesu

- Stan procesu (kolejny slajd)
- Numer procesu (kolejny slajd)
- Licznik rozkazów (kolejny slajd)
- Rejestry procesora - CPU registers - m.in. akumulatory, rejestrze indeksowe, wskaźniki stosu, rejestrzy ogólnego przeznaczenia, rejestrzy warunków, etc.
- Informacje o planowaniu przydziału procesora - CPU-scheduling information - m.in. priorytet procesu, wskaźnik do kolejek, etc.
- Informacje o zarządzaniu pamięcią - Memory-management information - m.in. zawartość rejestrów granicznych, tablice stron lub tablice segmentów, etc.
- Informacje do rozliczeń - accounting information - ilość zużytego czasu procesora i czasu rzeczywistego, ograniczenia czasowe, numery kont, numer zadania lub procesu, itp.
- Informacja o stanie wejścia-wyjścia - I/O status information - m.in. lista urządzeń we/wy przydzielonych do procesu, lista otwartych plików, itd.

Blok kontrolny procesu

Linux kernel:

```
$ /usr/src/  
$ linux-headers-[kernel v.]/  
$ include/linux/sched.h
```

Zdefiniowane:

task_struct

Kilkadziesiąt pól !



User mode:

```
$ /proc/[PID procesu]  
$ ps
```

PCB - stan procesu

Process state - stan procesu

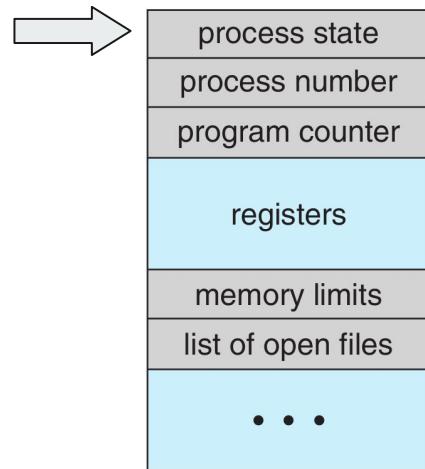
```
$ ps -p [PID procesu] -o pid,status,comm
```

Lub:

```
$ cat /proc/[PID procesu]/status | grep Stat
```

Możliwe stany:

- R=running,
- S=sleeping in an interruptible wait,
- D=waiting in uninterruptible disk sleep,
- Z=zombie,
- T=traced or stopped (on a signal),
- W=paging



PCB - numer procesu

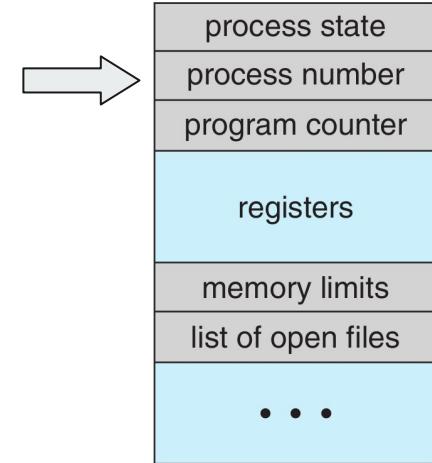
PID - Process identifier - identyfikator procesu

```
$ pidof <nazwa programu>
```

lub:

```
$ ps aux | grep <nazwa programu>
```

W wyniku otrzymamy liczbę, która jest numerem PID procesu, który jest liczbą typu integer niepowtarzalną w przestrzeni systemu operacyjnego.



PCB - numer procesu

Odczyt własnego PID (programistycznie):

```
$ pid_t getpid(void); // unistd.h
```

Odczyt PID rodzica:

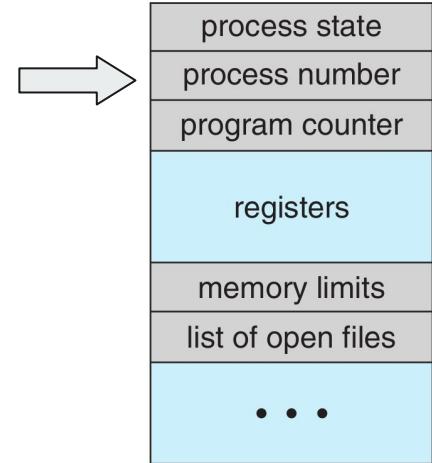
```
$ pid_t getppid(void); // unistd.h
```

Utworzenie procesu i pobranie jego PID:

```
$ pid_t fork(void); // unistd.h
```

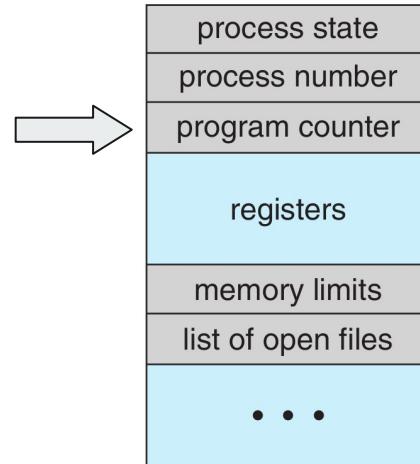
Funkcje, w których argumentem jest PID:

kill, wait, nice

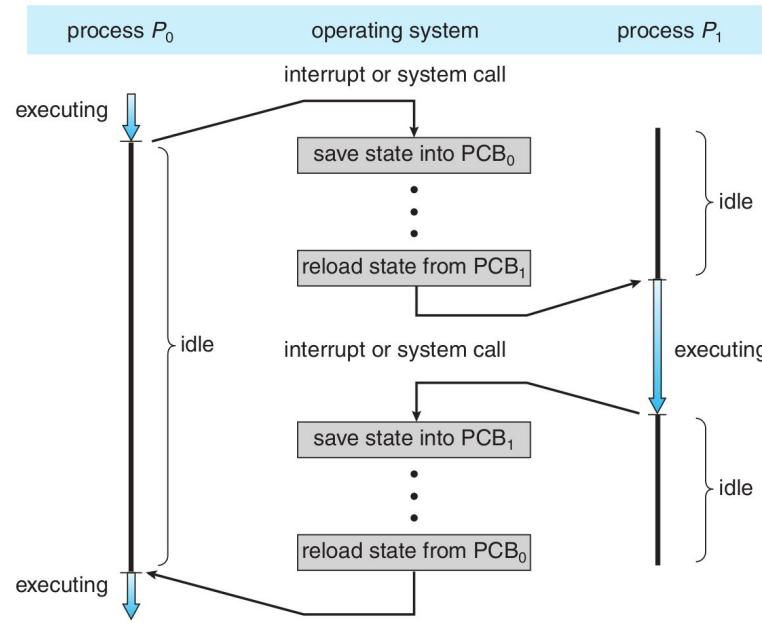


PCB - licznik rozkazów

Licznik rozkazów - program counter - adres kolejnej instrukcji do wykonania

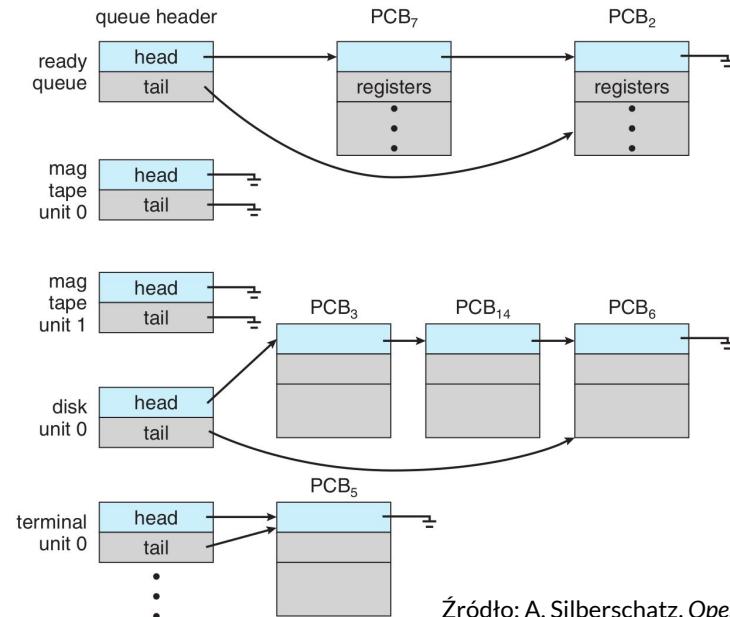


Blok kontrolny procesu - przełączanie CPU



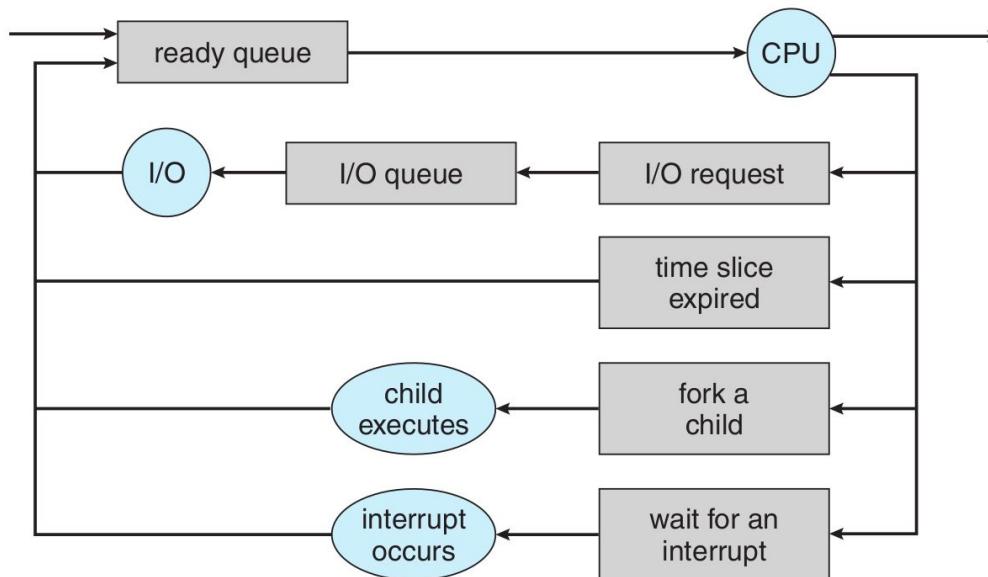
Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Kolejkowanie urządzeń we/wy



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Diagram kolejkowania procesu



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Operacje na procesach - tworzenie procesu (1)

- Tworzenie procesu:
 - Dany proces (rodzic - *parent*) może utworzyć wiele nowych procesów (dziecko - *child*)
 - Każdy z nowo powstałych procesów może tworzyć kolejne, powstaje drzewo - *tree* procesów
 - Identyfikacja procesu w większości systemów odbywa się przez identyfikator PID

```
$ pstree -p | head -n 10
```

```
systemd(1)-+-ModemManager(824)-+-{ModemManager}(854)
           |           `-{ModemManager}(857)
           | -NetworkManager(711)-+-{NetworkManager}(819)
           |           `-{NetworkManager}(821)
           | -accounts-daemon(10363)-+-{accounts-daemon}(10366)
           |           `-{accounts-daemon}(10372)
           | -acpid(10526)
           | -agetty(876)
```

```
$ ps -ax | head -n 10
```

PID	TTY	STAT	TIME	COMMAND
1	?	Ss	0:01	/sbin/init splash
2	?	S	0:00	[kthreadd]
3	?	I<	0:00	[rcu_gp]
4	?	I<	0:00	[rcu_par_gp]
6	?	I<	0:00	[kworker/0:0H-kblockd]
9	?	I<	0:00	[mm_percpu_wq]
10	?	S	0:00	[ksoftirqd/0]
11	?	I	0:01	[rcu_sched]
12	?	S	0:00	[migration/0]

Pytanie: skąd różnica w nazwie: systemd vs. init ?

Poszukiwanie PID procesu macierzystego

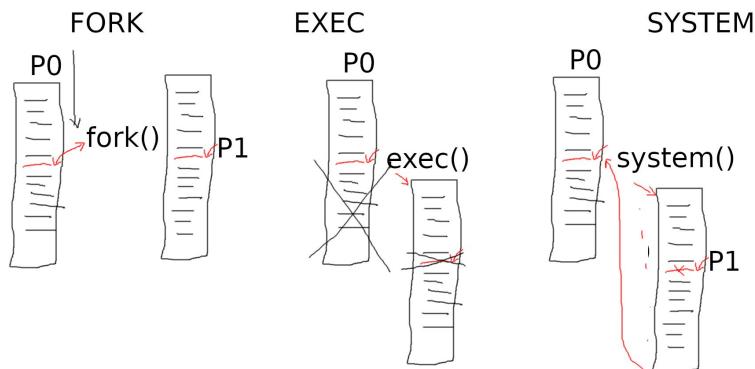
```
$ pidof <nazwa programu>  
  
$ ps -p `pidof <nazwa programu>` -o ppid,pid,status,comm
```

lub:

```
$ cat /proc/`pidof <nazwa programu>`/status | grep PPid
```

Operacje na procesach - tworzenie procesu (2)

- Kiedy dany proces utworzy proces potomny, mogą zaistnieć dwa przypadki:
 - Proces rodzica wykonuje się nadal jednocześnie z procesem potomnym (fork)
 - Proces rodzica czeka, aż któryś lub wszystkie procesy potomne zakończą działanie (fork lub system)
- Są także dwie możliwości adresowania pamięci dla nowego procesu:
 - Proces potomny jest duplikatem procesu rodzica, tzn. ma ten sam kod programu jak rodzic (fork)
 - Proces potomny to na nowo załadowany program (exec i system)



Operacje na procesach - tworzenie procesu (3)

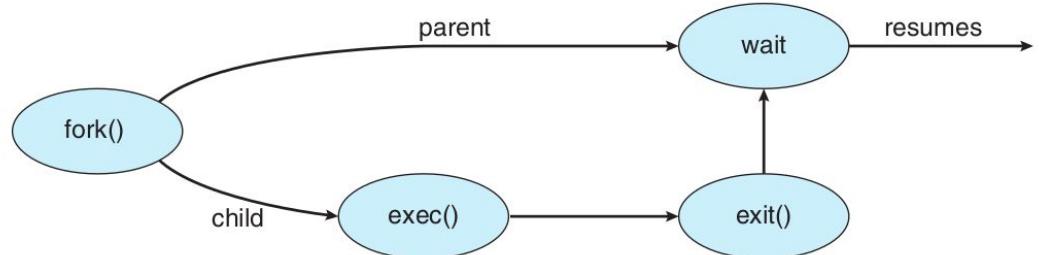
```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

/* fork a child process */
pid = fork();

if (pid < 0) { /* error occurred */
    fprintf(stderr, "Fork Failed");
    return 1;
}
else if (pid == 0) { /* child process */
    execlp("/bin/ls","ls",NULL);
}
else { /* parent process */
    /* parent will wait for the child to complete */
    wait(NULL);
    printf("Child Complete");
}

return 0;
}
```



Uruchamianie skryptów, a proces potomny

Tryb bez źródła:

```
$ ./skrypt.sh
```

Tryb ze źródłem:

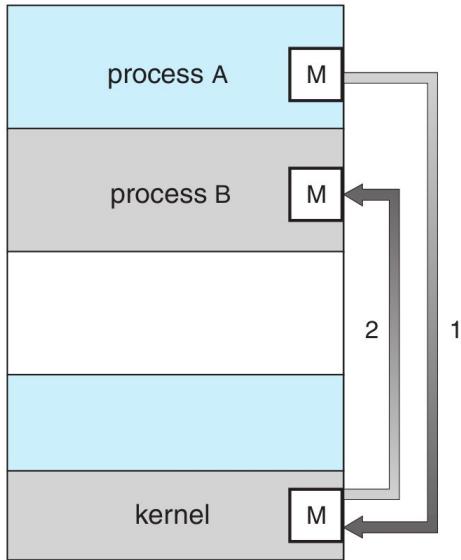
```
$ ././skrypt.sh
```

```
$ source ./skrypt.sh
```

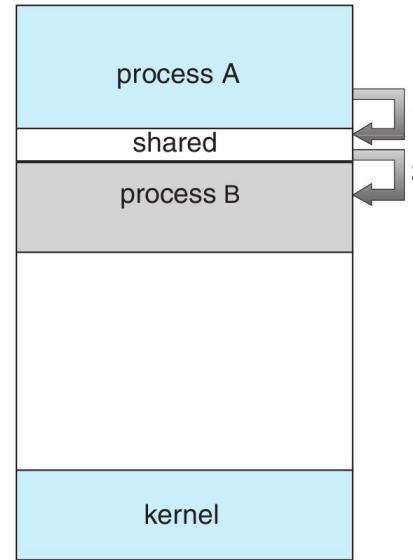
Komunikacja międzyprocesowa

- Współistnienie procesów:
 - Proces niezależny - nie wpływają na niego, ani nie wpływa na inne procesy
 - Proces kooperujący - może wpływać na inne procesy lub inne procesy mogą wpływać na niego
- Znaczenie komunikacji międzyprocesowej:
 - Współdzielenie informacji (dane, wymiana komunikatów)
 - Przyspieszenie obliczeń (czy wszystkie można zrównoleglić?)
 - Modularność systemu
 - Wygoda
- IPC - interprocess communication = komunikacja międzyprocesowa

Modele komunikacji międzyprocesowej (1)



Message passing



Shared memory

Modele komunikacji międzyprocesowej (2)

- Message passing:
 - Użyteczny do wymiany niewielkich ilości danych (brak konieczności zapobiegania konfliktom)
 - Łatwiejszy w implementacji
- Shared memory:
 - Najszybsza możliwa komunikacja
 - Jedyna wymagana interwencja z kernela to utworzenie tej pamięci

Producent i konsument

- Model producent - konsument
 - Serwer www - przeglądarka html
 - Kompilator tex - pdf viewer
 - Etc.

Shared memory - buforowanie (1)

- Bufory wymiany danych:
 - Bufor nieograniczony (unbounded buffer) - brak limitu wielkości:
 - Producent nigdy nie czeka, konsument czeka gdy bufor jest pusty
 - Bufor ograniczony (bounded buffer) - określona wielkość bufora:
 - Producent czeka, gdy bufor jest pełny, konsument czeka gdy bufor jest pusty

Shared memory - buforowanie (2)

```
#define BUFFER_SIZE 10

typedef struct {
    . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- kolejka FIFO
- in - następna wolna pozycja w buforze
- out - pierwsza wolna pozycja w buforze
- in == out - bufor jest pusty
- ((in+1) % BUFFER_SIZE) == out - bufor jest pełny





Shared memory - buforowanie (3)

Producent

```
item nextProduced;

while (true) {
    /* produce an item in nextProduced */
    while (((in + 1) % BUFFER_SIZE) == out)
        ; /* do nothing */
    buffer[in] = nextProduced;
    in = (in + 1) % BUFFER_SIZE;
}
```

Konsument

```
item nextConsumed;

while (true) {
    while (in == out)
        ; // do nothing

    nextConsumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
    /* consume the item in nextConsumed */
}
```

Shared memory

- W oparciu o przykład napisz program typu chat, w którym dwa procesy wymieniają między sobą komunikaty (liczby lub tekst).

```
#include <stdio.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main()
{
/* the identifier for the shared memory segment */
int segment_id;
/* a pointer to the shared memory segment */
char *shared_memory;
/* the size (in bytes) of the shared memory segment */
const int size = 4096;

/* allocate a shared memory segment */
segment_id = shmget(IPC_PRIVATE, size, S_IRUSR | S_IWUSR);

/* attach the shared memory segment */
shared_memory = (char *) shmat(segment_id, NULL, 0);

/* write a message to the shared memory segment */
sprintf(shared_memory, "Hi there!");

/* now print out the string from shared memory */
printf("%s\n", shared_memory);

/* now detach the shared memory segment */
shmdt(shared_memory);

/* now remove the shared memory segment */
shmctl(segment_id, IPC_RMID, NULL);

return 0;
}
```

Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Synchronizacja

- Message passing może być blokujący lub nieblokujący:
 - Blokujące wysyłanie
 - Nieblokujące wysyłanie
 - Blokujący odbiór
 - Nieblokujący odbiór

Buforowanie

- Zerowa pojemność (zero capacity) - kolejka ma długość zero
- Ograniczona długość (bounded capacity) - kolejka ma ustaloną długość
- Nieograniczona długość (unbounded capacity) - kolejka ma nieograniczoną długość



Turbacz

The background image shows a close-up of a large, brownish-orange rock formation with intricate textures and crevices. A rope hangs vertically from the top right corner of the rock. Below the rock, the water is a vibrant turquoise color, appearing clear and shallow near the shore. The overall scene is natural and serene.

Systemy Operacyjne

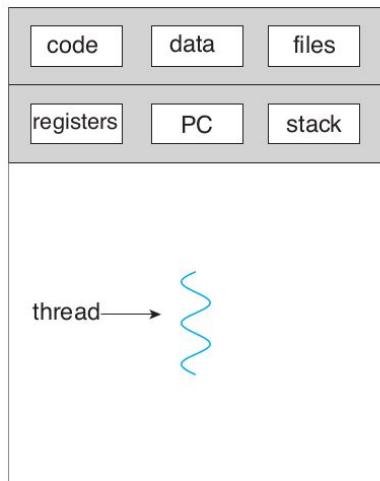
Wątki

Dr hab. inż. Krzysztof Rzecki, prof. AGH

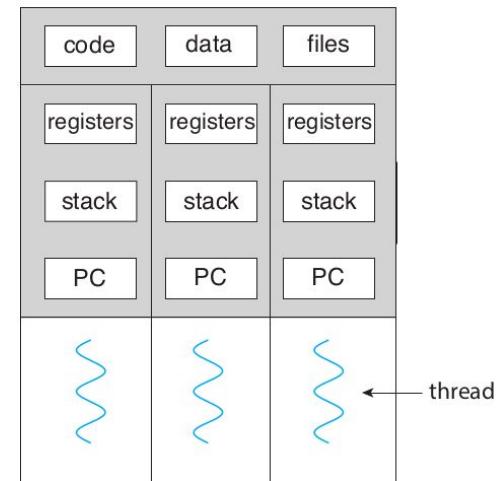
Na podstawie: Abraham Silberschatz, *Koncepcje systemów operacyjnych*

Wątek

- Wątek to podstawowa jednostka wykorzystania procesora.
- Wątek zawiera: identyfikator (numer), licznik rozkazów, rejestrory oraz stos.
- Wątek współdzieli z innymi wątkami należącymi do tego samego procesu: sekcję kodu, sekcję danych oraz inne zasoby systemu operacyjnego (np. otwarte pliki, sygnały).



single-threaded process



multithreaded process

Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*



Proces vs. Wątek

Proces

- Proces osadzony jest w dwóch charakterystykach:
 - Właściciela zasobów, w tym przestrzeni adresowej zawierającej obraz procesu.
 - Planowania i wykonywania, która przeplata się razem z innymi procesami.
- Te dwie wyżej wymienione charakterystyki są niezależnie traktowane w systemie operacyjnym.

Wątek

- Jednostką wykonywanego zadania jest *wątek*, inaczej lekki proces (ang. *lightweight process*).
- Właściciel określony jest przez proces (zadanie), do którego wątek należy.

Proces

- Proces to wykonywanie programu.
- *Heavy weight process.*
- Czaso- i zasobochłonne: tworzenie, terminacja, przełączanie kontekstu.
- Komunikacja: pamięć dzielona lub wymiana komunikatów jako mechanizmy specjalne.
- Procesy są izolowane.
- Przełączanie procesów odbywa się przez funkcje systemowe.
- Dla jądra dwa procesy to dwa procesy.
- Zablokowanie jednego procesu nie wpływa na fakt zablokowania innego procesu.

vs. Wątek

- Wątek to część danego procesu.
- *Lightweight process.*
- Szybsze i zużywające mniej zasobów na tworzenie, terminację i przełączanie kontekstu.
- Komunikacja: bezpośrednio współdzielone wszystkie zasoby danego procesu.
- Wątki współdzielą.
- Przełączanie wątków odbywa się bez wywoływania przerwań do jądra.
- Dla jądra dwa wątki to jeden proces.
- Zablokowanie procesu, to zablokowanie jego wszystkich wątków.

Proces

- Zablokowanie się procesu macierzystego uniemożliwia tworzenie procesów potomnych.
- Proces ma własny PCB, stos oraz przestrzeń adresową.
- Zmiany w procesie macierzystym nie mają wpływu na procesy potomne.

vs. Wątek

- Zablokowanie pierwszego wątku nie wpływa na działanie pozostałych wątków procesu.
- Ma rodzica PCB, własny TCB, stos oraz współdzieloną przestrzeń adresową.
- Zmiany w procesie wpływają na zmiany w wątkach tego procesu.

Obserwowanie wątków w systemie Linux

```
$ ps -T -o pid,tid,comm -p `pidof insync`
```

PID	TID	COMMAND
-----	-----	---------

4224	4224	insync
4224	4260	QXcbEventQueue
4224	4280	QDBusConnection
4224	4281	gmain
4224	4282	gdbus
4224	4284	insync
4224	4312	insync


Nazwa wątku, nazwa procesu

```
$ pstree -p `pidof insync`
```

```
insync(4224)---QtWebEngineProc(4336)---QtWebEngineProc(4338)---QtWebEngineProc(4357)---{QtWebEngineProc}(4358)
```



```
|
```

```
|
```

```
...
```

```
|-{insync}(4260)
```

```
|-{insync}(4280)
```

```
|-{insync}(4281)
```



Procesy potomne

PID vs. TID

- PID = *process identifier*
- TID = *thread identifier*
- Jeśli proces ma tylko jeden wątek, to PID == TID
- Jeśli proces ma wiele wątków, to pierwszy z nich ma unikalny TID w zakresie tego procesu
- Jądro systemu nie rozróżnia szczególnie *wątku od procesu*
- Dla jądra wątki to procesy, które współdzielą pewne zasoby
- Kiedy tworzymy nowy proces za pomocą `fork()`, to otrzymuje on nowy PID i TID (PID==TID)
- Kiedy tworzymy nowy wątek to otrzymuje on PID taki jak procesu oraz nowy TID.
- Alias: LWP = *Light-Weight Process*

Obserwowanie wątków kernela w Linux

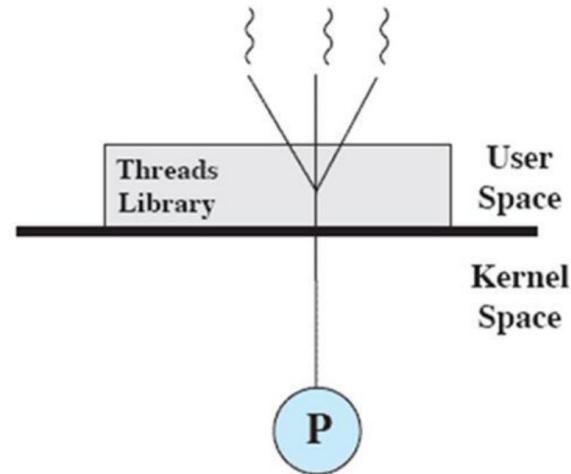
```
$ pstree -p 2
kthreadd(2)---UVM Tools Event(977)
    |-UVM deferred re(976)
    |-UVM global queu(975)
    |-acpi_thermal_pm(134)
    |-ata_sff(117)
    |-blkcg_punt_bio(115)
    |-charger_manager(164)
    |-cpuhp/0(14)
```

Dlaczego mają odmienny PID ?

Dlatego, że to jest TID.

Wątki na poziomie użytkownika

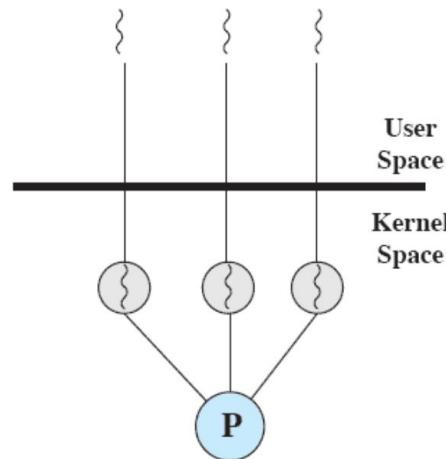
- Zarządzanie wątkami odbywa się na poziomie aplikacji.
- Jądro nie ma wiedzy na temat wątków.



Źródło: TODO

Wątki na poziomie jądra

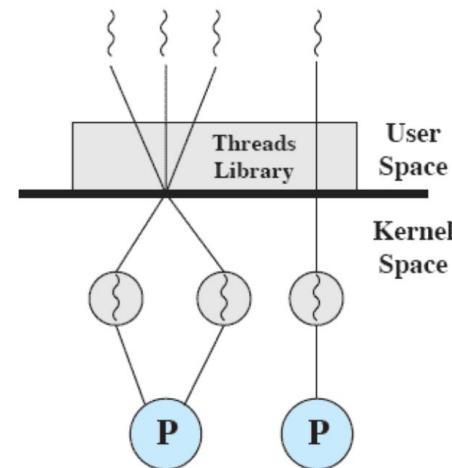
- Jądro zarządza kontekstem dla procesu oraz wątków.
Nie ma zarządzania wątkami na poziomie aplikacji.
- Zalety:
 - Kernel może jednocześnie planować realizację wielu wątków z jednego procesu na wielu procesorach/rdzeniach.
 - Jeśli jeden wątek w procesie jest zablokowany, jądro może planować inny wątek tego samego procesu.
 - Funkcjonalność jądra może być wielowątkowa.
- Wada: przekazanie kontroli między wątkami w obrębie procesu wymaga kernel-mode.



Źródło: TODO

Rozwiązanie łączone

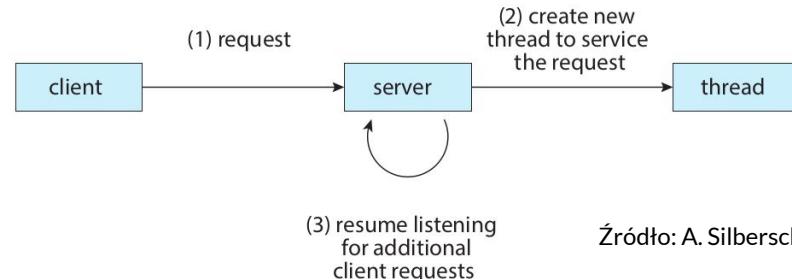
- Tworzenie wątku odbywa się w przestrzeni użytkownika.
- Planowanie (*scheduling*) oraz synchronizacja wątków odbywa się w jądrze



Źródło: TODO

Wątki - zastosowania

- Program serwera obsługujący żądania (ang. *requests*) programów klienckich:
 - Serwer stron WWW i przeglądarka internetowa.
 - Serwer poczty elektronicznej (ang. *mail transfer agent*) i program pocztowy.
 - Sprawdzanie pisowni w edytorze tekstu.
- Prowadzenie obliczeń macierzowych:
 - Wykonywanie operacji na tych samych danych.

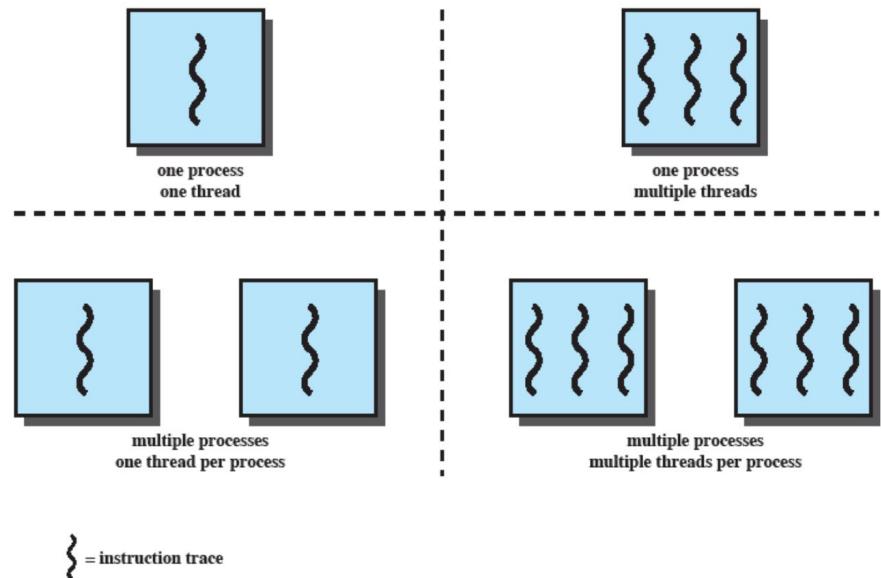


Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Uwaga! Tworzenie wątku jest mniej obciążające niż tworzenie nowego procesu.

Wielowątkowość

- Zdolność systemu operacyjnego do wspierania wielu ścieżek wykonywania w obrębie jednego procesu.
- MS-DOS - single user process with single thread.
- Some UNIX - multiple user processes with single thread per process.
- Java run-time env. - single process with multiple threads.
- Windows, Solaris, modern UNIX, Linux, etc. - multiple processes with multiple threads per process.

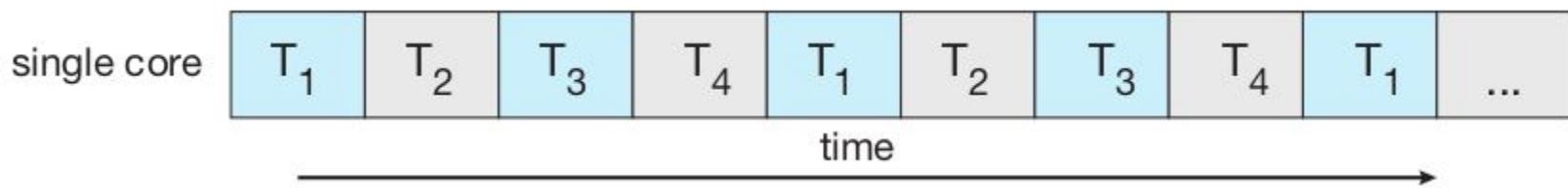


Źródło: TODO

Zalety oprogramowania wielowątkowego

- **Responsywność** - jeśli część aplikacji jest zablokowana, inna jej część może wykonywać operacje, a cała aplikacja sprawia wrażenie ciągłego działania. Zastosowanie: w aplikacji, kiedy jedna wywołana operacja wykonywana jest w tle, interfejs użytkownika pozostaje responsywny.
- **Współdzielenie zasobów** - w przypadku procesów współdzielenie zasobów odbywa się tylko poprzez pamięć współdzieloną, albo przesyłanie komunikatów. Wątki współdzierają zasoby w prost. Współdzielenie kodu i danych umożliwia wątkom działać w tej samej przestrzeni adresowej.
- **Ekonomia** - alokowanie pamięci i zasobów przy tworzeniu procesu jest bardziej kosztowne, niż w przypadku wątków. Przełączanie przełączanie kontekstu jest także szybsze w przypadku wątków.
- **Skalowalność** - aplikacje wielowątkowe mogą działać w architekturze wielordzeniowej. Aplikacja jednowątkowa może być wykonana tylko na jednym rdzeniu procesora.

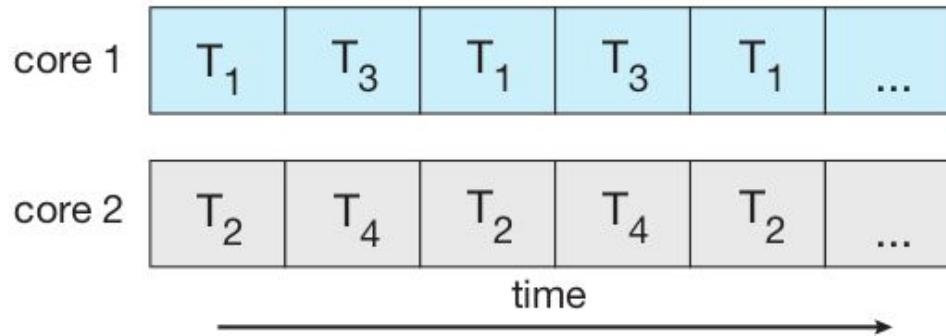
Współbieżność i równoległość



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Współbieżność (ang. *concurrency*) - umożliwia więcej niż jednemu zadaniu być wykonywanym. Do realizacji współbieżności nie jest wymagany system wielordzeniowy.

Współbieżność i równoległość



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

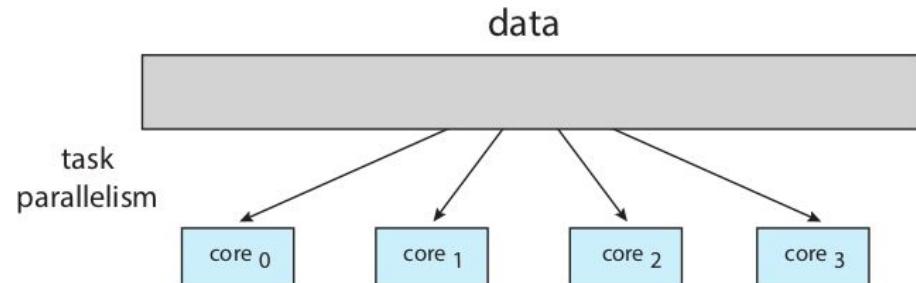
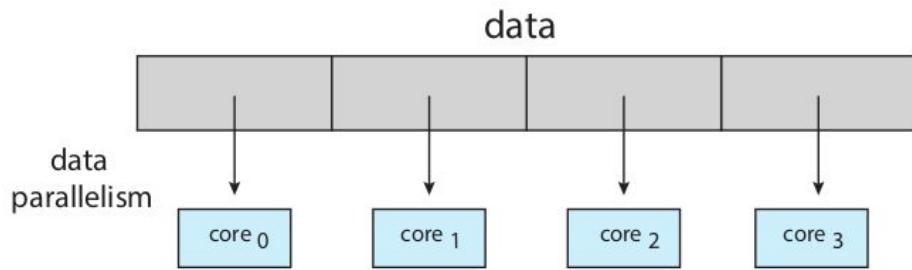
Równoległość (ang. *parallelism*) - umożliwia więcej niż jednemu zadaniu być wykonywanym **JEDNOCZEŚNIE**. Do realizacji równoległości jest wymagany system wielordzeniowy.

Pytanie: czy może zaistnieć współbieżność bez równoległości ?

Wyzwania programowe

- **Identyfikacja zadań**, w szczególności na zadania niezależne między sobą.
- **Balansowanie** zadaniami celem zrównoważenia obciążenia.
- **Dzielenie danych** między wydzielone zadania.
- **Zależność danych** występująca w szczególności przy następstwie obliczeń.
- **Testowanie i debugowanie** są zdecydowanie trudniejsze niż w programach jednowątkowych.

Typy równoległości



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Dane dzielone są między procesami/rdzeniami wykonującymi tego samego typu operacje.

Przykład: sumowanie zakresów komórek.

Zadania dzielone są między procesami/rdzeniami, a każdy wątek realizuje unikalną operację.

Przykład: jednoczesne wyznaczanie min i max.

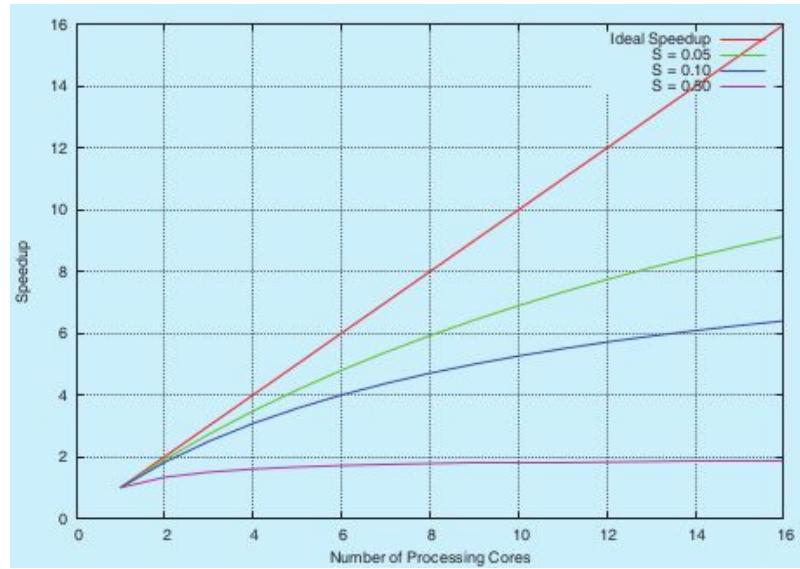
Prawo Amdahl'a

Wyraża potencjalny wzrost wydajności obliczeń przez dodanie kolejnych rdzeni obliczeniowych do obsługi aplikacji, która ma dwa komponenty: podlegający i niepodlegający zrównolegleniu.

S - procentowy udział niepodlegającego zrównolegleniu kodu.

N - liczba rdzeni przypisanych do zadania.

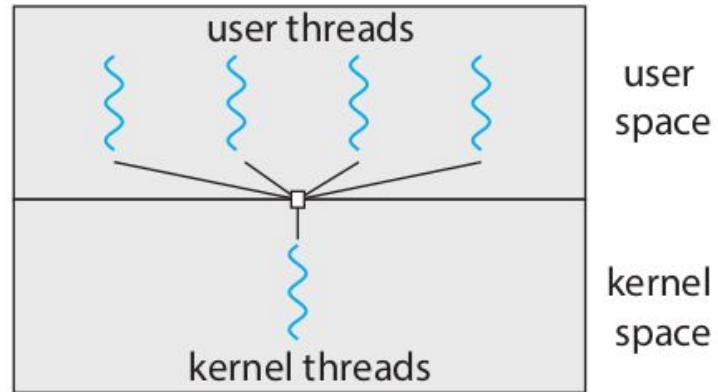
$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Model wielowątkowy - Many-to-One

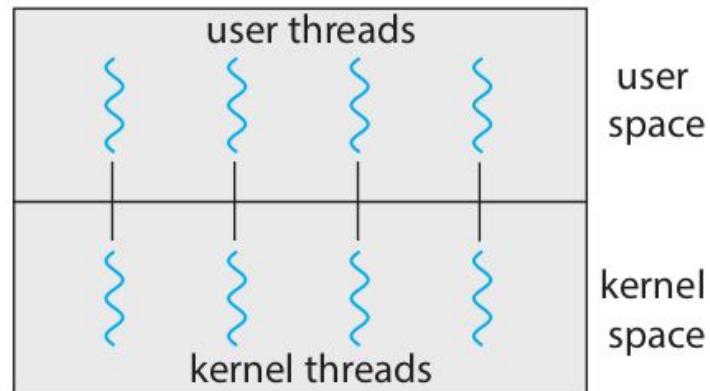
- Zarządzanie wątkami wykonywane jest w przestrzeni użytkownika (przez bibliotekę).
- Zaleta: wydajność.
- Wada 1: zablokowanie całego procesu, jeśli któryś wątek wykona blokujące wywołanie systemowe.
- Wada 2: wątki nie zostaną uruchomione równolegle w systemie wielordzeniowym.
- To tzw. zielone wątki (ang. *green threads*) - można je uruchomić w środowisku nie wspierającym wielowątkowości.
- Przykłady: Solaris, wczesne versje Java.



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Model wielowątkowy - One-to-One

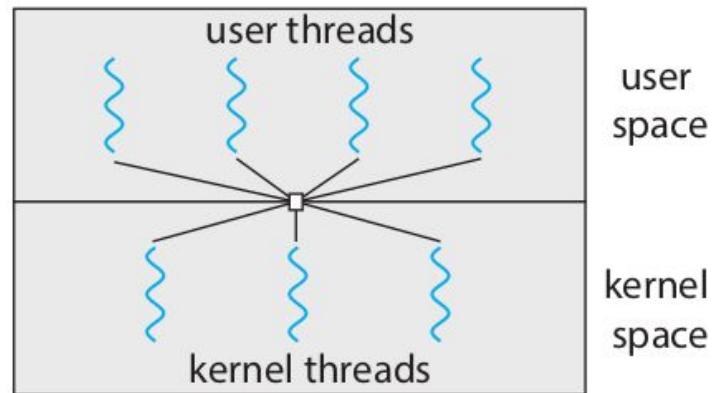
- Model wprowadza niezależną współbieżność, tzn. dany wątek może być realizowany także wtedy, gdy inny wywoła blokującą funkcję systemową.
- Model wprowadza także równoległość.
- Wada: każdy wątek użytkownika tworzy wątek w jądrze, a duża ich liczba może obniżać wydajność systemu.
- Przykłady: Linux, Windows.



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Model wielowątkowy - Many-to-Many

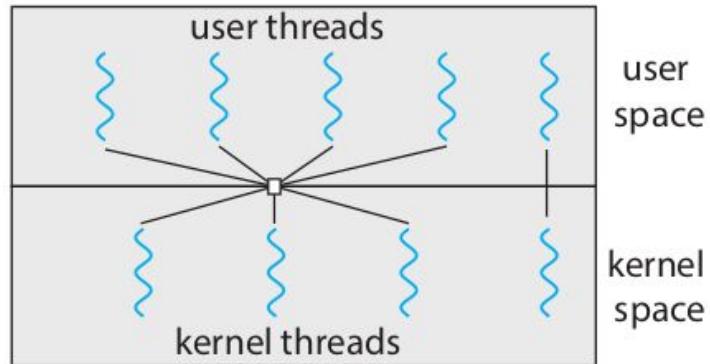
- Model multiplexuje wiele wątków w przestrzeni użytkownika z równą lub mniejszą liczbą wątków w przestrzeni jądra.
- Liczba wątków w przestrzeni jądra może być specyficzna względem aplikacji lub sprzętu.
- Model ten jest pozbawiony wad modeli Many-to-One i One-to-One.
- Trudny w implementacji.



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Model wielowątkowy - Two-level

Jak na rysunku obok.



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Biblioteki programistyczne dla wątków

- Biblioteki dostarczają API (ang. *Application Programming Interface*) do tworzenia i zarządzania wątkami.
- Podejście 1: Wątki w przestrzeni użytkownika, bez wsparcia ze strony jądra. To oznacza, że wynikiem wywołania funkcji bibliotecznej jest wywołanie funkcji lokalnej, nie systemowej.
- Podejście 2: Wątki w przestrzeni jądra ze wsparciem systemu operacyjnego. To oznacza, że kod i dane biblioteki istnieją w przestrzeni jądra, a wywołanie funkcji jest wywołaniem systemowym.

Biblioteki programistyczne dla wątków (2)

- POSIX Pthreads - przestrzeń użytkownika lub przestrzeń jądra.
 - Windows thread library - przestrzeń jądra.
 - Java thread API - bezpośrednio w programach Java, tak jak dana implementacja JVM zależna od hostującego systemu operacyjnego.
-
- POSIX i Windows - dane zadeklarowane globalnie są współdzielone między wątkami procesu.
 - Java - nie ma danych globalnych, dostęp do współdzielonych danych musi zostać nadany.

Strategie tworzenia wątków

Strategia asynchroniczna - wątek tworzy wątek potomny i następnie kontynuuje swoje działanie. Oba wątki działają współbieżnie i niezależnie.

Zastosowanie:

- Serwery wielowątkowe.
- Responsywny interfejs użytkownika.

Strategia synchroniczna - wątek tworzy wątki potomne i przechodzi w stan oczekiwania na zakończenie wykonywania ich zadań. O ile wątki potomne działają współbieżnie, to wątek macierzysty po prostu czeka.

Zastosowanie:

- Obliczenia z przesłaniem zadań cząstkowych i oczekiwaniem na wyniki.

Pula wątków

Powody:

- Tworzenie wątków zajmuje pewien czas (mniejszy niż procesów potomnych), a może mogą być wykorzystane ponownie.
- Brak kontroli liczby powstających wątków może doprowadzić do przeciążenia zasobów (procesor, pamięć) systemu.

Rozwiązanie: **pula wątków** (ang. *thread pool*)

- Utworzenie zadanej liczby wątków.
- Umieszczanie wątków w puli wątków.
- Wątki oczekują na przydzielenie zadania.
- Serwer otrzymuje żądanie.
- Serwer przekazuje żądanie do puli wątków.
- Jeśli w puli jest wolny wątek, przejmuje on żądanie i zajmuje się jego obsługą.
- Jeśli brak jest wolnych wątków w puli, zadanie jest kolejkowane.
- Po zakończeniu obsługi danego żądania wątek wraca do puli i oczekuje na nowe.
- Pula wątków najlepiej działa, gdy zadania obsługiwane są asynchronicznie.

Rozmiar puli wątków

Rozmiar puli wątków może być zależny od:

- Liczby rdzeni procesora.
- Ilości fizycznej pamięci RAM.
- Może być też dynamicznie zmieniany w zależności od aktualnie działających wątków (obserwując ich obciążenie).

Wywołania systemowe: `fork()` oraz `exec()`

Problem: czy po wywołaniu przez wątek funkcji systemowej `fork()` proces potomny duplikuje wszystkie wątki, czy nowy proces jest jedno-wątkowy ?

Odpowiedź: sprawdzić i odpowiedź przedstawić na forum UPEL.

Działanie: wywołanie `exec()` spowoduje zastąpienie całego procesu i wszystkich wątków.

Przypadek I: jeśli `exec()` wywołany jest zaraz po `fork()`, wówczas duplikowanie wszystkich wątków nie jest potrzebne, bo i tak proces zostanie zastąpiony w funkcji `exec()`.

Przypadek II: jeśli `exec()` wywołany jest później, dany `fork()` powinien zduplikować wszystkie wątki.

Obsługa sygnałów

Procedura obsługi sygnałów:

- Sygnał jest generowany przez zdarzenie.
- Sygnał jest dostarczany do procesu.
- Proces musi obsłużyć sygnał.

Sygnał synchroniczny: dzielenie przez 0, nielegalny dostęp do pamięci.

Sygnał asynchroniczny: wciśnięcie np. [ctrl+c].

Zagadka: który wątek otrzyma sygnał ?

Obsługa sygnału:

- **domyślna obsługa sygnału** - jeśli brak zdefiniowanej obsługi sygnału, zajmuje się nią jądro systemu operacyjnego (sygnał może być zignorowany lub zakończyć działanie programu),
- **zdefiniowana przez użytkownika obsługa sygnału.**

Sygnal a program wielowątkowy

Gdzie dostarczyć sygnał ?

- Do wątku, do którego sygnał pasuje.
- Do każdego wątku w procesie.
- Do wybranych wątków w procesie.
- Wybrać jeden wątek do przechwytywania wszystkich sygnałów danego procesu.

Wysyłanie sygnału do procesu:

```
kill(pid_t pid, int signal)
```

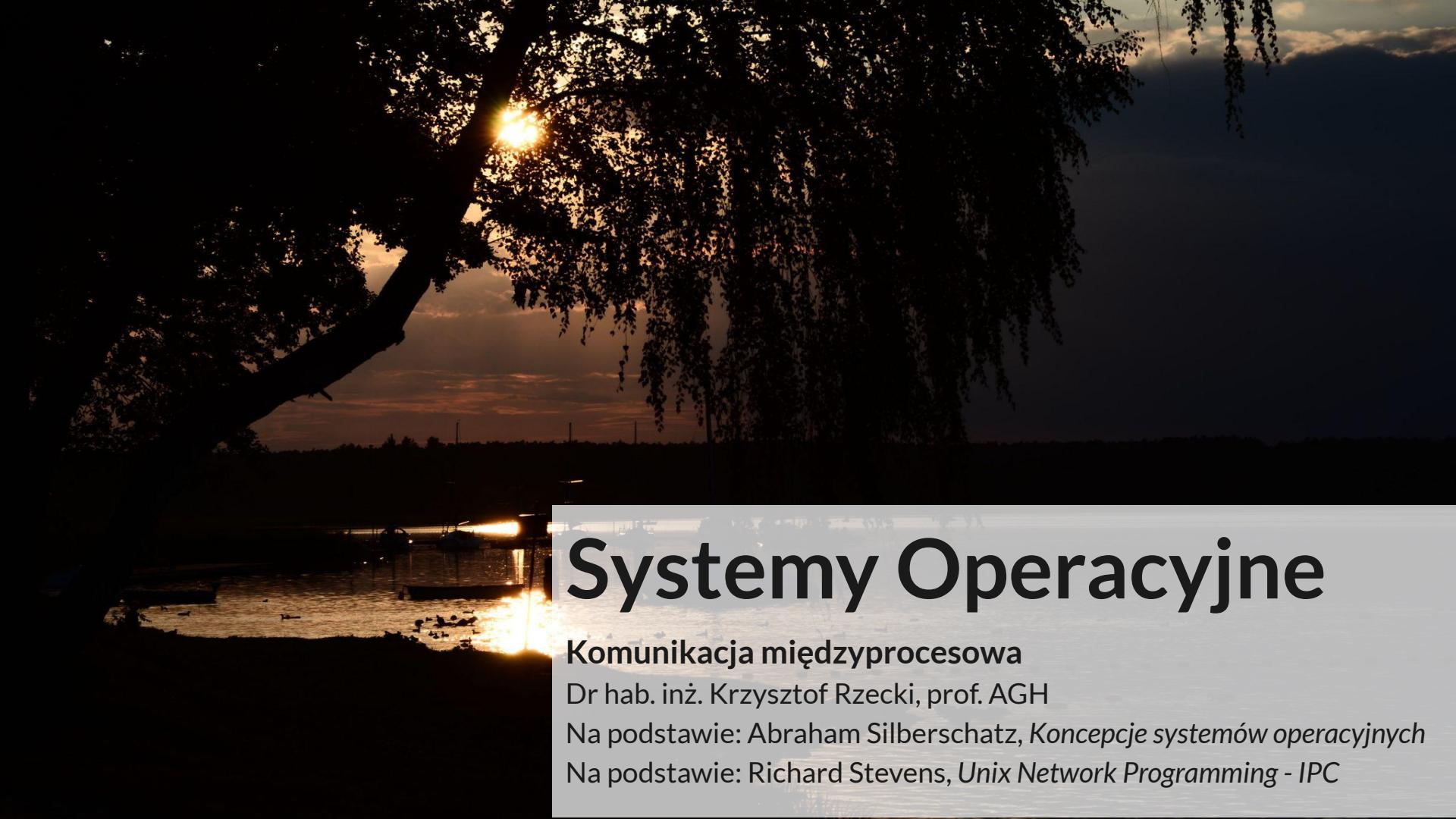
Przechwyci go pierwszy nieblokujący wątek.

Wysyłanie sygnału do wybranego wątku:

```
pthread_kill(pthread_t tid, int signal)
```



Dubrownik / Chorwacja

The background of the slide is a photograph of a sunset or sunrise over a body of water. The sky is filled with warm, orange and yellow hues, transitioning into darker blues and purples at the top. In the foreground, the dark silhouettes of trees and branches frame the scene. The water reflects the light from the sun, creating a bright path across the surface.

Systemy Operacyjne

Komunikacja międzyprocesowa

Dr hab. inż. Krzysztof Rzecki, prof. AGH

Na podstawie: Abraham Silberschatz, Koncepcje systemów operacyjnych

Na podstawie: Richard Stevens, Unix Network Programming - IPC

Podstawy komunikacji międzyprocesowej

Procesy uruchomione jednocześnie mogą być:

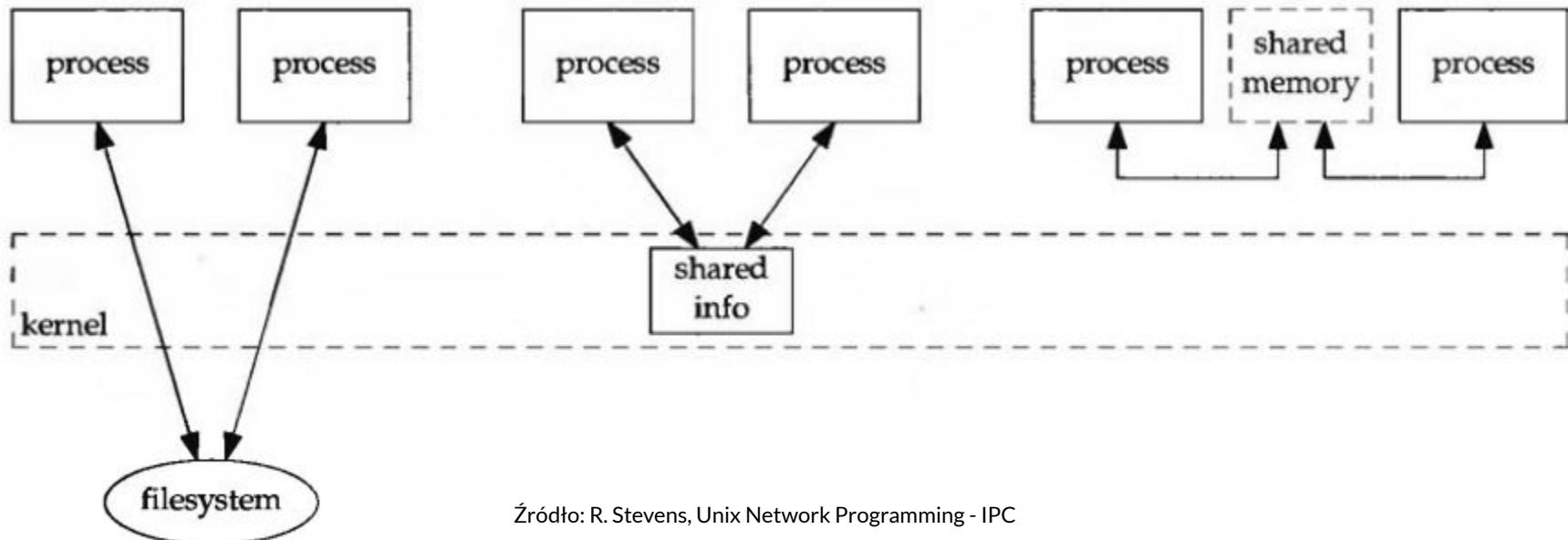
- Niezależne (ang. *independent*) - nie współdzielą danych z żadnym innym wykonywanym procesem w systemie operacyjnym.
- Współpracujące (ang. *cooperating*) - może wpływać lub można na niego wpływać przez inne procesy wykonywane w systemie.

Czy zatem proces odczytujący dane z dysku jest procesem współpracującym, czy niezależnym ?

Zastosowania komunikacji międzyprocesowej

- Współdzielenie informacji
- Przyspieszenie obliczeń
- Modularność oprogramowania

Trzy metody dzielenia informacji

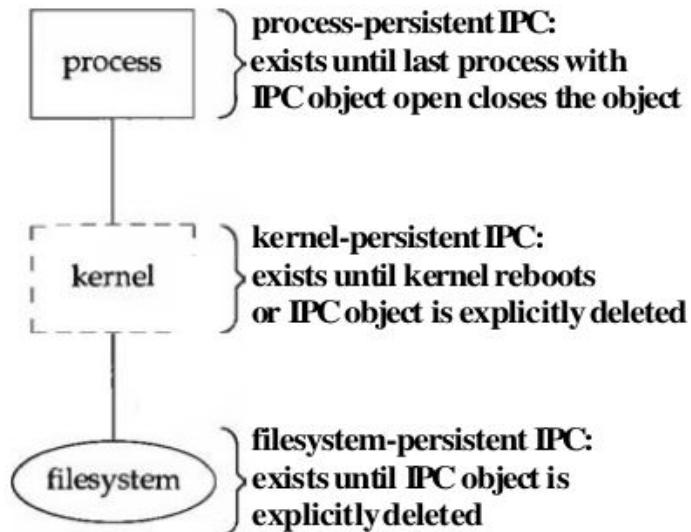


Źródło: R. Stevens, Unix Network Programming - IPC

Implementacje IPC

- Wymiana przez pliki
- Pamięć dzielona
- Sygnały
- Potoki nazwane lub nienazwane
- Semafora
- Kolejki
- Gniazda dziedziny UNIX
- Gniazda udp/tcp
- RPC

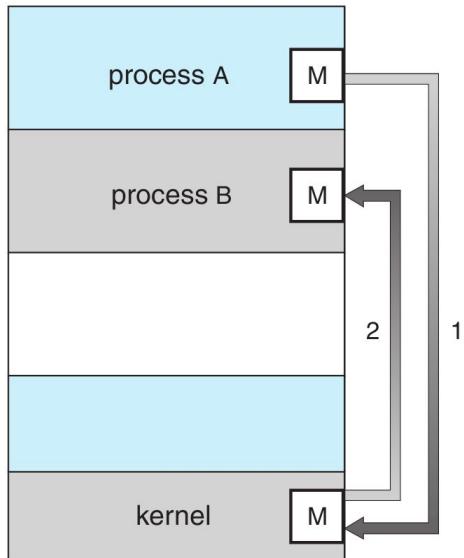
Trwałość obiektów IPC



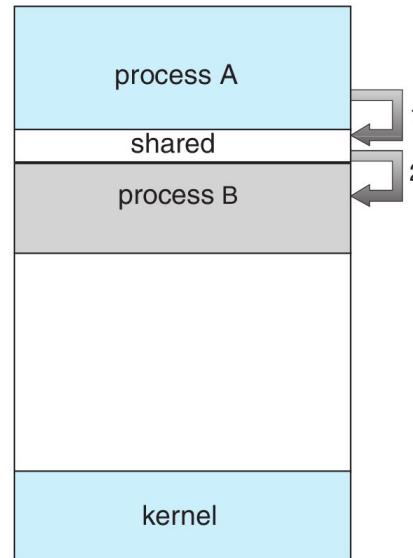
Type of IPC	Persistence
Pipe	process
FIFO	process
Pesix mutex	process
Posix condition variable	process
Posix read-write lock	process
<code>fcntl</code> record locking	process
Posix message queue	kernel
Pesix named semaphore	kernel
Pesix memory-based semaphore	process
Posix shared memory	kernel
System V message queue	kernel
System V semaphore	kernel
System V shared memory	kernel
TCP socket	process
UDP socket	process
Unix domain socket	process

Źródło: R. Stevens, Unix Network Programming - IPC

Modele komunikacji międzyprocesowej (było!)

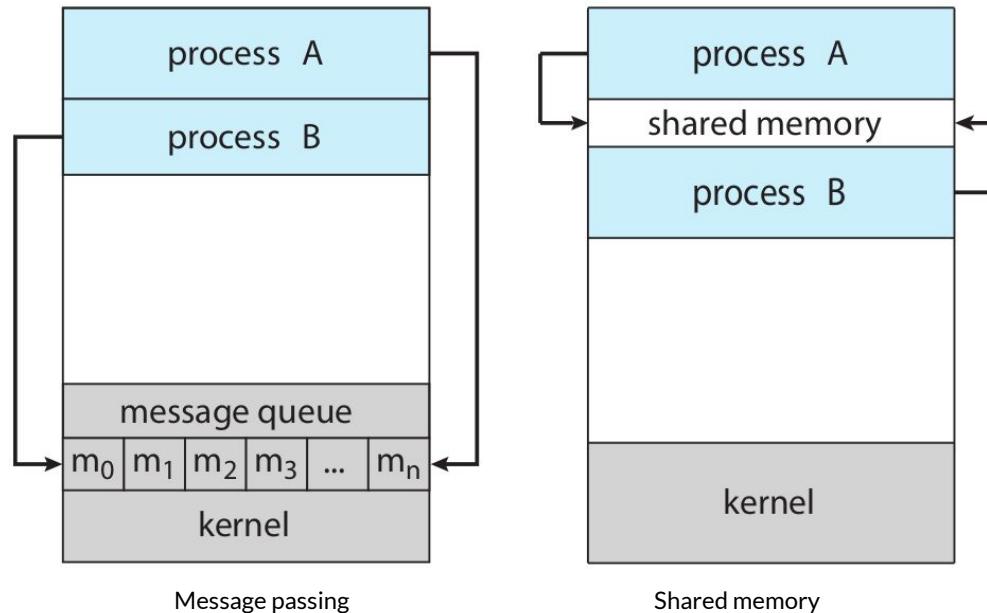


Message passing



Shared memory

Modele komunikacji międzyprocesowej (nowe!)



Message passing

Shared memory

Message-Passing Systems

Message-Passing Systems (systemy przekazywania wiadomości):

- Komunikujące się procesy mogą rezydować na różnych stacjach
- Komunikujące się procesy mogą rezydować na różnych typach i wersjach systemów operacyjnych
- Infrastruktura systemu przekazywania wiadomości obejmuje co najmniej dwie operacje:
 - send (message)
 - receive (message)
- Ze względu na wielkość wiadomości rozróżniamy:
 - System bezpośredni, czyli o stałej długości komunikatów (*straight-forward*) - prosta implementacja w systemie operacyjnym, skomplikowane użytkowanie ze względu na fragmentację,
 - System o zmiennej długości komunikatów (*variable-sized*) - skomplikowana implementacja w systemie operacyjnym, proste użytkowanie.

Communication link

- *Communication link* - logiczne powiązanie między procesami zestawiające kanał komunikacji między nimi. Nie interesuje nas zatem, czy to jest shared memory, szyna sprzętowa, czy sieć.
- Implementacje tego powiązania obejmują zagadnienia:
 - Komunikacji pośredniej i bezpośredniej.
 - Komunikacji synchronicznej i asynchronicznej.
 - Buforowanie automatyczne lub na żądanie.

Komunikacja bezpośrednia

- **Komunikacja bezpośrednią** (ang. *direct communication*) - każdy proces, który uczestniczy w komunikacji musi bezpośrednio wskazać nadawcę, czy odbiorcę:
 - `send(P, message)` - wyślij wiadomość do procesu P.
 - `receive(Q, message)` - odbierz wiadomość od procesu Q.
- Własności:
 - Link jest zestawiany automatycznie, a procesy muszą tylko znać swoją nazwę.
 - Link jest zestawiany dokładnie między dwoma procesami.
 - Między każdą parą komunikujących się procesów zestawiany jest dokładnie jeden link.
 - Występuje symetria w adresacji (P, Q), choć spotykane są warianty asymetryczne (np. P i id).

Komunikacja pośrednia

- **Komunikacja pośrednia** (ang. *indirect communication*) - procesy komunikują się przez skrzynki (ang. *mailbox*) lub porty identyfikowane np. przez liczby całkowite:
 - `send (A, message)` - wyślij wiadomość do skrzynki A.
 - `receive (A, message)` - odbierz wiadomość ze skrzynki A.
- **Własności:**
 - Link jest zestawiany między parą współdzielących skrzynkę procesów.
 - Link jest może zostać zestawiony przez większą liczbę procesów.
 - Między każdą parą komunikujących się procesów mogą istnieć różne linki komunikacyjne.
- **System operacyjny musi obsługiwać następujące mechanizmy:**
 - Utworzenie skrzynki.
 - Wysłanie i odebranie wiadomości do i ze skrzynki.
 - Usunięcie skrzynki.

(Rysunek na tablicy!)

Synchronizacja

- Komunikacja synchroniczna, blokująca:
 - Blokujące wysyłanie - proces wysyłający jest zablokowany na wysyłaniu, aż wysyłana wiadomość zostanie odebrana przez proces odbierający lub skrzynkę.
 - Blokujący odbiór - odbiorca zostaje zablokowany do czasu otrzymania wiadomości.
- Komunikacja asynchroniczna, nieblokująca
 - Nieblokujące wysyłanie - proces wysyłający wysyła wiadomość i nie jest blokowany na metodzie wysyłającej.
 - Nieblokujący odbiór - odbiorca otrzymuje gotową wiadomość lub informację o braku jej dostępności.

W przypadku blokującego nadawcy i odbiorcy mamy do czynienia z ang. *Rendezvous*.

(Rysunek na tablicy!)

Buforowanie

- Pojemność zerowa (ang. *zero capacity*) - maksymalna długość kolejki wynosi zero, czyli link komunikacyjny nie może mieć żadnej wiadomości oczekującej w sobie, co oznacza, że nadawca musi zostać zablokowany do czasu odbioru wiadomości przez odbiorcę.
- Ograniczona pojemność (ang. *bounded capacity*) - kolejka ma określoną, skończoną długość n , czyli co najwyżej n wiadomości może zostać umieszczonych w kolejce. Jeśli kolejka nie jest pełna, można dołożyć wiadomość (nadawca zostaje zablokowany), jeśli kolejka jest pusta, nie można pobrać żadnej wiadomości (odbiorca zostaje zablokowany).
- Nieograniczona pojemność (ang. *unbounded capacity*) - długość kolejki jest potencjalnie nieskończona (nadawca nigdy nie jest blokowany).

(Rysunek na tablicy!)

POSIX Shared Memory

- Tworzenie dowiązania do wspólnej przestrzeni w pamięci:
 - `int fd = shm_open(name, O_CREAT | O_RDWR, 0666);` // pisanie i czytanie
 - `int fd = shm_open(name, O_RDONLY, 0666);` // tylko czytanie
- Ustawianie wielkości pamięci dzielonej:
 - `ftruncate(fd, 4096);`
- Utworzenie miejsca w pamięci:
 - `ptr = (char *) mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);`
- Pisanie do pamięci dzielonej:
 - `sprintf(ptr, "%s", message);` // umieszczenie komunikatu w miejscu 'ptr'
 - `ptr += strlen(message);` // przesunięcie wskazania w pamięci
- Czytanie z pamięci dzielonej:
 - `printf("%s", (char *)ptr);`
- Usunięcie obiektu współdzielonego:
 - `shm_unlink(name);`



POSIX Shared Memory

Description	<code>mq_open</code>	<code>sem_open</code>	<code>shm_open</code>
read-only	<code>O_RDONLY</code>		<code>O_RDONLY</code>
write-only	<code>O_WRONLY</code>		
read-write	<code>O_RDWR</code>		<code>O_RDWR</code>
create if it does not already exist	<code>O_CREAT</code>	<code>O_CREAT</code>	<code>O_CREAT</code>
exclusive create	<code>O_EXCL</code>	<code>O_EXCL</code>	<code>O_EXCL</code>
nonblocking mode	<code>O_NONBLOCK</code>		
truncate if it already exists			<code>O_TRUNC</code>



System V IPC

	Message queues	Semaphores	Shared memory
Header	<code><sys/msg.h></code>	<code><sys/sem.h></code>	<code><sys/shm.h></code>
Function to create or open	<code>msgget</code>	<code>semget</code>	<code>shmget</code>
Function for control operations	<code>msgctl</code>	<code>semctl</code>	<code>shmctl</code>
Functions for IPC operations	<code>msgsnd</code> <code>msgrcv</code>	<code>semop</code>	<code>shmat</code> <code>shmdt</code>

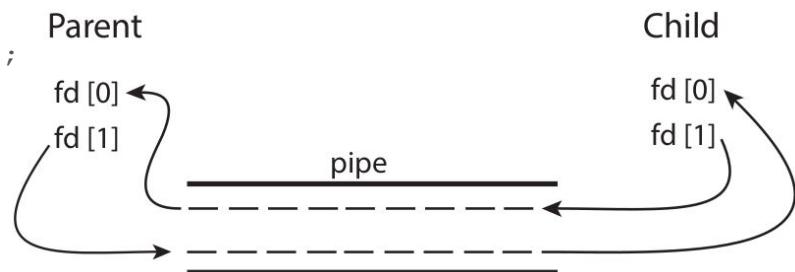
Potok (ang. *pipe*)

- Jedne z pierwszych metod IPC, jedna z najprostszych form komunikacji.
- Przy projektowaniu potoków należy uwzględnić cztery kwestie:
 - Czy potok pozwala na dwukierunkową (ang. *bidirectional*), czy jest to jednokierunkowa (ang. *unidirectional*) komunikacja ?
 - Jeśli możliwa jest komunikacja dwukierunkowa, to czy jest ona half duplex (dane przesyłane są tylko w jednym kierunku w danym momencie), czy full duplex (dane przesyłane są w obu kierunkach w danym momencie) ?
 - Czy jest relacja między komunikującymi się procesami (np. rodzic - dziecko) ?
 - Czy potoki mogą komunikować się poprzez sieć, czy tylko między procesami na tej samej maszynie ?

Potok zwykły (ang. *ordinary pipe*)

- Utworzenie potoku:
 - int fd[2];
 - pipe(int fd[])
- Pisanie do potoku (deskryptor fd[1]):
 - write(fd[WRITE END], write msg, strlen(write msg)+1);
- Czytanie z potoku (deskryptor fd[0]):
 - read(fd[READ END], read msg, BUFFER SIZE);
- Potok w praktyce:
 - ls | less
 - cat file.txt | wc -l

(Przedyskutować efekt użycia fork())

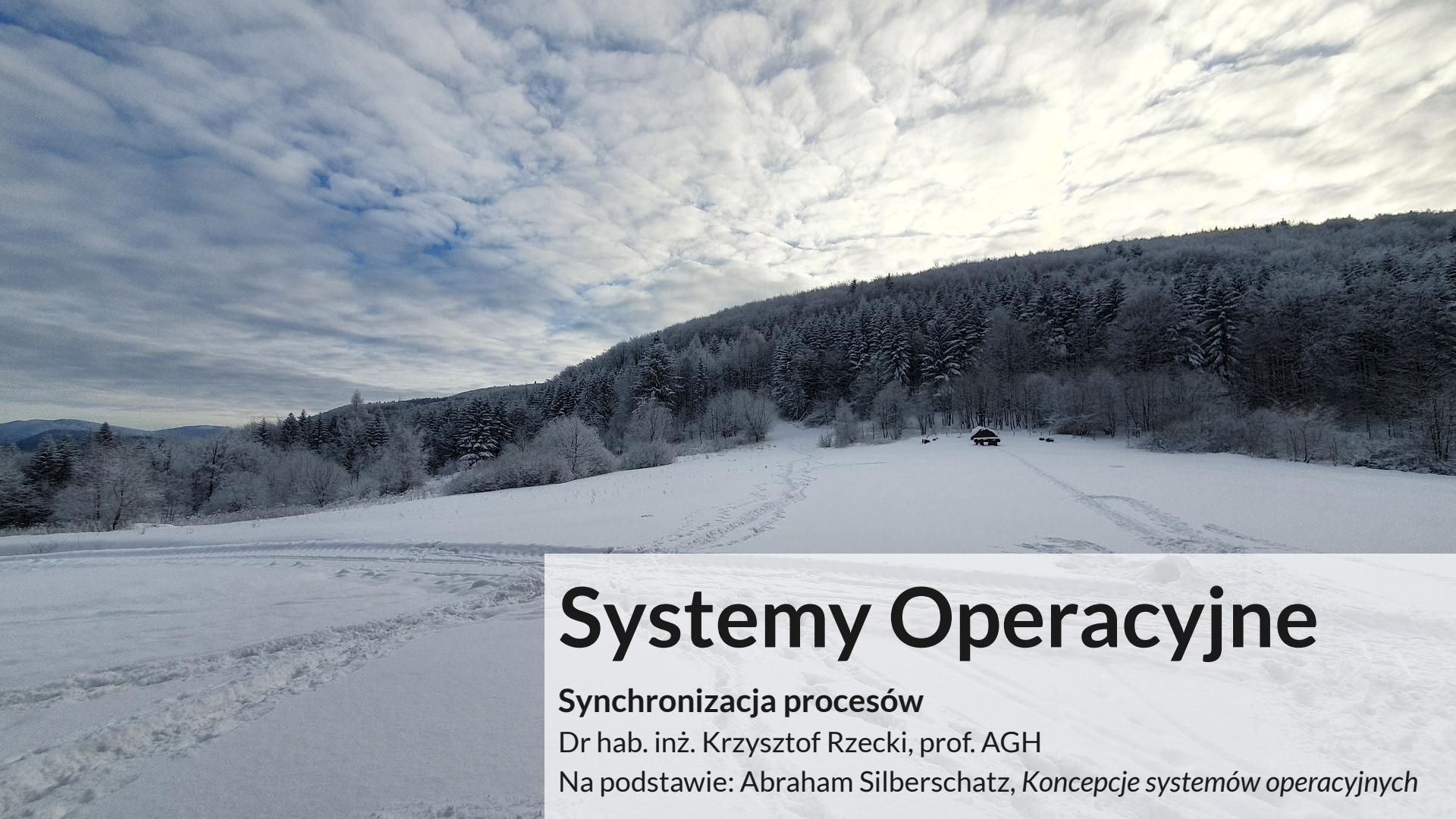


Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Gniazda i komunikacja sieciowa

Osobny wykład.





Systemy Operacyjne

Synchronizacja procesów

Dr hab. inż. Krzysztof Rzecki, prof. AGH

Na podstawie: Abraham Silberschatz, *Koncepcje systemów operacyjnych*

Kooperacja procesów

Sposoby kooperacji:

- Bezpośrednie współdzielenie przestrzeni adresacji (zarówno kod, jak i dane)
- Współdzielenie danych przez system plików lub komunikaty

Skutki kooperacji:

- Utrata spójności danych
- Wzajemne blokowanie

Zobacz: wykład pt. "Procesy"

Zobacz: wykład pt. "Komunikacja międzyprocesowa"

Producent - konsument - pamięć dzielona

- Producent to proces produkujący informację, którą konsumuje konsument
- Przykład: kompilator - asembler, asembler - loader, klient - serwer, etc.
- Dwa typy buforów:
 - Nieskończony - konsument czeka, gdy bufor jest pusty; producent zawsze może umieszczać dane,
 - Skończony - konsument czeka, gdy bufor jest pusty; producent czeka, gdy bufor jest pełny.
- in - następna wolna pozycja w buforze
- out - pierwsza pełna pozycja w buforze
- in == out - bufor jest pusty
- ((in + 1) % BUFFER_SIZE) == out - bufor jest pełny

```
#define BUFFER_SIZE 10

typedef struct {
    . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```



Producent - konsument

```
while (true) {  
    /* produce an item in nextProduced */  
    while (counter == BUFFER_SIZE)  
        ; /* do nothing */  
    buffer[in] = nextProduced;  
    in = (in + 1) % BUFFER_SIZE;  
    counter++;  
}
```

Producent

```
while (true) {  
    while (counter == 0)  
        ; /* do nothing */  
    nextConsumed = buffer[out];  
    out = (out + 1) % BUFFER_SIZE;  
    counter--;  
    /* consume the item in nextConsumed */  
}
```

Konsument

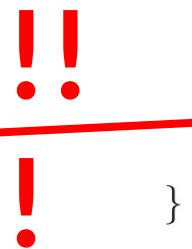
Producent - konsument

```
while (true) {  
    /* produce an item in nextProduced */  
    while (counter == BUFFER_SIZE)  
        ; /* do nothing */  
    buffer[in] = nextProduced;  
    in = (in + 1) % BUFFER_SIZE;  
    counter++;
```

Producent

```
while (true) {  
    while (counter == 0)  
        ; /* do nothing */  
    nextConsumed = buffer[out];  
    out = (out + 1) % BUFFER_SIZE;  
    counter--;  
    /* consume the item in nextConsumed */
```

Konsument



Warunek wyścigu (ang. race condition) to sytuacja, w której dwa lub więcej procesów wykonuje operację na zasobach dzielonych (odczyt lub zapis), a ostateczny wynik tej operacji jest zależny kolejności tego dostępu.

Warunek wyścigu

Niskopoziomowy `count++`

```
register1 = count  
register1 = register1 + 1  
count = register1
```

Niskopoziomowy `count--`

```
register2 = count  
register2 = register2 - 1  
count = register2
```

Niech `count == 5`

T_0 :	<i>producer</i>	<code>execute</code>	$register_1 = count$	$\{register_1 = 5\}$
T_1 :	<i>producer</i>	<code>execute</code>	$register_1 = register_1 + 1$	$\{register_1 = 6\}$
T_2 :	<i>consumer</i>	<code>execute</code>	$register_2 = count$	$\{register_2 = 5\}$
T_3 :	<i>consumer</i>	<code>execute</code>	$register_2 = register_2 - 1$	$\{register_2 = 4\}$
T_4 :	<i>producer</i>	<code>execute</code>	$count = register_1$	$\{count = 6\}$
T_5 :	<i>consumer</i>	<code>execute</code>	$count = register_2$	$\{count = 4\}$

Sekcja krytyczna

Sekcja krytyczna (ang. *critical section*) to segment kodu, w którym proces może zmieniać wartości zmiennych, aktualizować tabele, pisać do pliku, etc. Podstawową własnością sekcji krytycznej jest to, że w tym samym czasie żaden inny proces nie może realizować swojej sekcji krytycznej (obejmującej te same zasoby).

- **Sekcja wejścia** (ang. *entry section*) - segment kodu, w którym zgłaszane jest żądanie dostępu do zasobu celem realizacji wzajemnego wykluczenia.
- **Sekcja krytyczna** (ang. *critical section*)
- **Sekcja wyjścia** (ang. *exit section*) - segment kodu, w którym zgłaszane jest zwolnienie zasobu.
- **Sekcja pozostałego kodu** (ang. *remainder section*) - nie związana z obsługą współdzielenia zasobów część pozostała część kodu.

Sekcje kodu

W ramce oznaczone zostały sekcje sterujące przebywaniem w sekcji krytycznej.

do {

entry section

critical section

exit section

remainder section

} while (TRUE);

Wymagania dot. rozwiązań sekcji krytycznej

Rozwiązanie problemu sekcji krytycznej musi spełniać następujące wymagania:

- **Wzajemne wykluczenie** (ang. *mutual exclusion*) oznacza, że jeśli jeden proces wykonuje swoją sekcję krytyczną, to żaden inny proces nie może wykonać swojej sekcji krytycznej.
- **Postęp** (ang. *progress*) oznacza, że jeśli żaden proces nie jest w sekci krytycznej i jakiś proces chciałby wejść do swojej sekcji krytycznej, to tylko procesy nierealizujące swojej sekcji kodu pozostałego mogą brać udział w decydowaniu, który z nich wejdzie do swojej sekcji krytycznej.
- **Skończony czas oczekiwania** (ang. *bounded waiting*) oznacza, że czas oczekiwania na wejście do sekcji krytycznej dla każdego procesu powinien być ograniczony.

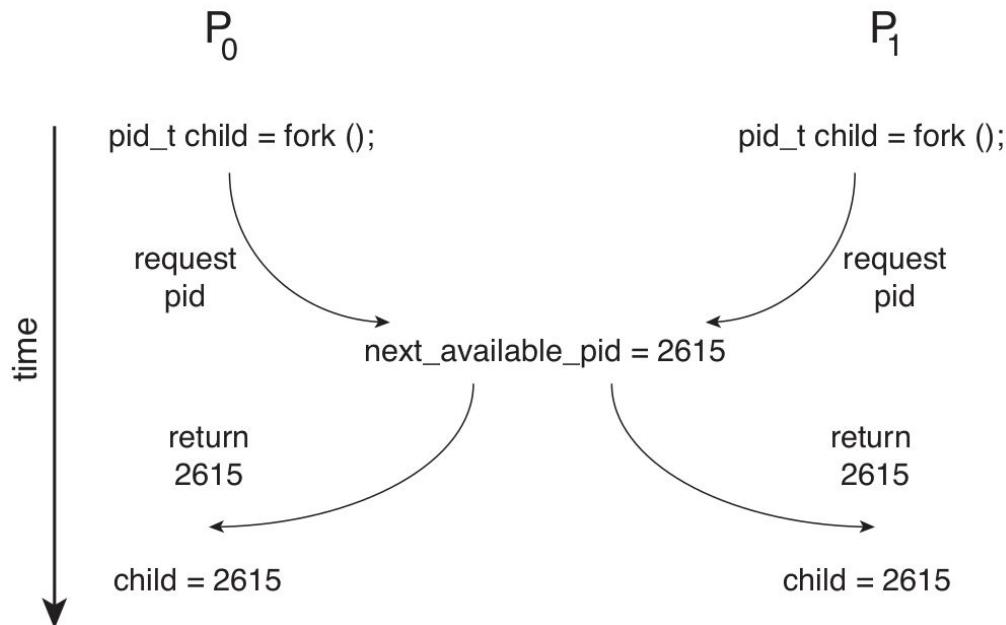
Zakleszczenie, ang. *deadlock*

Zakleszczeniem jest sytuacja, kiedy dwa procesy wzajemnie czekają na zwolnienie zasobów.

P_0	P_1
wait(S);	wait(Q);
wait(Q);	wait(S);
.	.
.	.
signal(S);	signal(Q);
signal(Q);	signal(S);

Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Sekcja krytyczna - przydzielanie PID po fork()



Wywłaszczenie jądra (ang. *Kernel preemption*)

Wywłaszczenie - technika, w której planista (algorytm szeregujący zadania, ang. *dispatcher*) może wstrzymać aktualnie wykonywane zadanie, aby umożliwić wykonywanie innemu zadaniu. Zawieszenie (np. zapętlenie) zadania nie powoduje zawieszenia całego systemu.

Wywłaszczenie jądra - jądro pozwala na wywłaszczenie własnego kodu, co oznacza, że w wykonywanie jego kodu może zostać przerwane na czas wykonywania przez procesor innego zadania.

Wywłaszczanie przerwań - przerwania są zwykle niewywłaszczalne, dlatego powinny być krótkie.

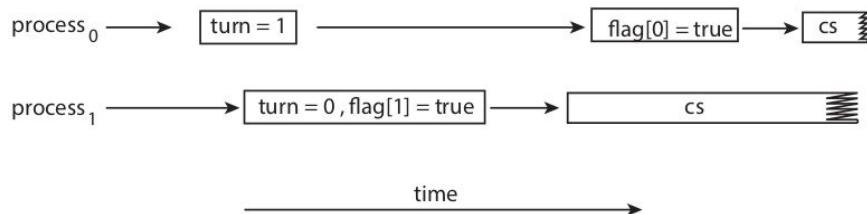
Cecha	Jądro wywłaszczające	Jądro niewywłaszczające
Definicja	Pozwala na usuwanie i podmianę procesu wykonywanego w trybie kernela, a w efekcie wykonywane jest zadanie o najwyższym priorytecie.	Pozwala na wywłaszczenie procesu wykonywanego w trybie kernela, co oznacza konieczność czekania na jego zakończenie.
Warunek wyścigu	Występuje - wiele procesów jest aktywnych	Nie występuje - jeden proces jest aktywny
Responsywność	Większa i deterministyczna responsywność	Mniejsza i niedeterministyczna responsywność
Implementacja	Skomplikowany projekt i implementacja	Mniej skomplikowany projekt i implementacja
Bezpieczeństwo	Większa stabilność pracy i użyteczność	Mniejsza stabilność pracy i użyteczność
Semafora	Nie wymaga użycia semaforów	Dane współdzielone wymagają semaforów
Programowanie RT	Większa użyteczność w programowaniu RT	Mniejsza użyteczność w programowaniu RT
Wywłaszczanie	Jest	Brak
Przykłady	Linux od 2.6, IRIX, Solaris, NetBSD od v5 Mikrokernele: Windows NT, Vista, 7 i 10	Windows XP, Windows 2000, Linux do 2.4

Wywłaszczający - ang. *preemptive*

Algorytm Peterson'a

- Dwa procesy P_0 oraz P_1
- Dwa procesy współdzielą:

```
int turn;
boolean flag[2];
```
- Zmienna `turn` wskazuje, którego procesu jest kolej na wejście do sekcji krytycznej.
- Tablica `flag` wskazuje, czy proces jest gotowy na wejście do sekcji krytycznej.



```
do {
```

```
    flag[i] = TRUE;
    turn = j;
    while (flag[j] && turn == j);
```

critical section

```
    flag[i] = FALSE;
```

remainder section

```
} while (TRUE);
```

Synchronizacja sprzętowa

Mechanizmy:

- Blokowanie przerwań
- Test and set lock - TSL
- Swap
- TLS + czas oczekiwania

```
do {  
    acquire lock  
    critical section  
    release lock  
    remainder section  
} while (TRUE);
```

Test and set lock - TSL

- Wymagane wsparcie procesora do realizacji instrukcji atomowych
- Realizacja sekwencyjna instrukcji atomowych (także w przypadku SMP)
- Instrukcja atomowa: TestAndSet()

```
boolean TestAndSet(boolean *target) {  
    boolean rv = *target;  
    *target = TRUE;  
    return rv;  
}  
  
do {  
    while (TestAndSet(&lock))  
        ; // do nothing  
  
    // critical section  
  
    lock = FALSE;  
  
    // remainder section  
} while (TRUE);
```

Swap

- Wymagane wsparcie procesora do realizacji instrukcji atomowych
- Realizacja sekwencyjna instrukcji atomowych (także w przypadku SMP)
- Instrukcja atomowa: Swap()
- Inicjalizacja globalnych zmiennych:

```
boolean waiting[n];
boolean lock;
```

```
void Swap(boolean *a, boolean *b) {
    boolean temp = *a;
    *a = *b;
    *b = temp;
}

do {
    key = TRUE;
    while (key == TRUE)
        Swap(&lock, &key);
    // critical section

    lock = FALSE;
    // remainder section
} while (TRUE);
```

TLS + czas oczekiwania

Test and set lock oraz Swap:

- Spełniają wymaganie wzajemnego wykluczenia, ale
- Nie spełniają wymagania dot. skońzonego czasu oczekiwania
- Obok: algorytm spełniający wszystkie wymagania dot. sekcji krytycznej

```
do {
    waiting[i] = TRUE;
    key = TRUE;
    while (waiting[i] && key)
        key = TestAndSet(&lock);
    waiting[i] = FALSE;

    // critical section

    j = (i + 1) % n;
    while ((j != i) && !waiting[j])
        j = (j + 1) % n;

    if (j == i)
        lock = FALSE;
    else
        waiting[j] = FALSE;

    // remainder section
} while (TRUE);
```

Semafora

- Semafor S to zmienna całkowita
- Semafor można modyfikować tylko w:
 - wait()
 - signal()
- Kiedy jeden proces modyfikuje semafor, żaden inny nie może tego robić
- Testowanie warunku $S \leq 0$ oraz inkrementacja $S--$ musi wykonać się bez przerwania

```
wait(S) {  
    while S <= 0  
        ; // no-op  
    S--;  
}  
  
signal(S) {  
    S++;  
}
```

Typy semaforów

- Semafor binarny (ang. *binary semaphore*)
= **mutex lock** od: *mutual exclusion*, czyli
wzajemne wykluczenie
Przy zastosowaniu do sekcji krytycznej:
 - procesy współdzielą semafor, mutex=1,
 - każdy proces działa jak na listingu obok.
- Semafor zliczający (ang. *counting semaphore*)
Zastosowanie do sekcji krytycznej, kiedy
dany zasób ma wiele instancji.

```
wait(S) {  
    while S <= 0  
        ; // no-op  
    S--;  
}
```

```
signal(S) {  
    S++;  
}
```

```
do {  
    wait(mutex);  
  
    // critical section  
  
    signal(mutex);  
  
    // remainder section  
} while (TRUE);
```

Problemy synchronizacyjne

- Problem ograniczonego bufora
- Problem czytelników i pisarzy
- Problem uczących filozofów



Problem ograniczonego bufora

Założenia:

- Pula buforów, rozmiar puli wynosi n
- Każdy bufor może zawierać jeden obiekt
- Zmienna mutex jest semaforem b. do puli
- Na początku $mutex = 1$
- Semaforы $empty$ i $full$ to liczność pustych/pełnych buforów
- Na początku $empty = n$, $full = 0$

```
do {  
    . . .  
    // produce an item in nextp  
    . . .  
    wait(empty);  
    wait(mutex);  
    . . .  
    // add nextp to buffer  
    . . .  
    signal(mutex);  
    signal(full);  
} while (TRUE);
```

Producent

```
do {  
    wait(full);  
    wait(mutex);  
    . . .  
    // remove an item from buffer to nextc  
    . . .  
    signal(mutex);  
    signal(empty);  
    . . .  
    // consume the item in nextc  
    . . .  
} while (TRUE);
```

Konsument

Problem czytelników i pisarzy - definicja

Założenia:

- Istnieje współdzielona baza danych
- Dwa typy procesów: piszące i czytające
- Procesy czytające mogą w dowolnej liczbie osiągać dostęp do bazy
- Jeśli jeden proces piszący ma dostęp, w tym czasie żaden inny proces (ani piszący, ani czytający) nie może mieć dostępu

Warianty:

- I. Żaden proces czytający nie czeka na dostęp, chyba, że proces piszący go uzyskał. Ryzyko zagłodzenia pisarzy.
- II. Jeśli pisarz oczekuje na dostęp, żaden czytelnik nie może rozpocząć czytania. Może dojść do zagłodzenia czytelników.

Problem czytelników i pisarzy - rozwiązanie I

- Czytelnicy współdzielą:

```
semaphore mutex, wrt;      // init: 1
int readcount;              // init: 0


- wrt jest wspólny także dla pisarzy
- wrt jest muteksem obsługującym pisarzy
- wrt jest także dla pierwszego i ostatniego czytelnika w sekcji krytycznej,
- mutex obsługuje zmienną readcount
- readcount - liczba aktualnie czytających

```

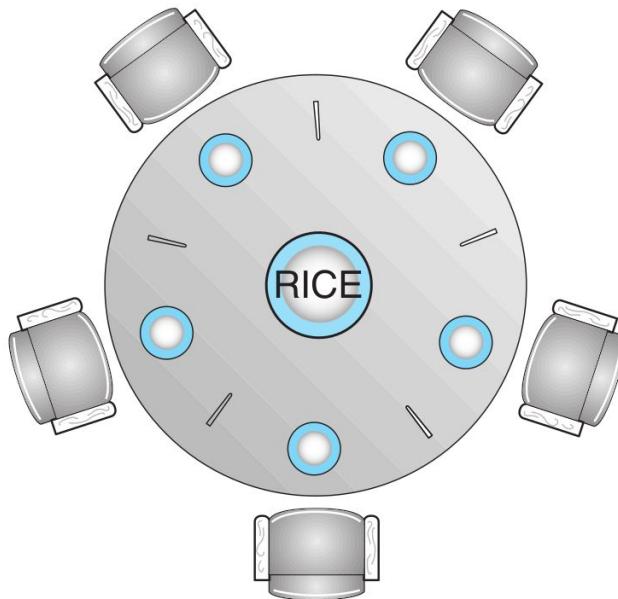
```
do {
    wait(wrt);
    . . .
    // writing is performed
    . . .
    signal(wrt);
} while (TRUE);
```

Pisarz

```
do {
    wait(mutex);
    readcount++;
    if (readcount == 1)
        wait(wrt);
    signal(mutex);
    . . .
    // reading is performed
    . . .
    wait(mutex);
    readcount--;
    if (readcount == 0)
        signal(wrt);
    signal(mutex);
} while (TRUE);
```

Czytelnik

Problem ucztujących filozofów - definicja



- Rozważmy pięciu filozofów siedzących przy stole jak na obrazku obok
- Na środku stołu jest miska ryżu, wokół pięć talerzy, po jednym dla każdego filozofa
- Pomiędzy talerzami jest pięć sztućców
- Od czasu do czasu dany filozof chce zjeść
- Aby zjeść muszą być wolne dwa sztućce
- Jedząc filozof ma wyłączność na 2 sztućce
- Po skończeniu jedzenia zwalnia sztućce

Problem ucztujących filozofów - rozwiązanie

- Filozof próbuje wziąć sztućce wywołując:
`wait()`
- Filozof odkłada sztućce wywołując:
`signal()`
- Filozofowie wspólnie dzielą sztućce:
`semaphore chopstick[5]; // init: 1`
 - Rozwiązania:
 - 1. Max 4-ch filozofów przy stole
 - 2. Można podnieść sztućce tylko wtedy, jeśli oba są wolne (podnieść w sekcji krytycznej)
 - 3. Parzyści filozofowie podnoszą najpierw lewy, potem prawy sztuciec, a nieparzyści odwrotnie: najpierw prawy, potem lewy

```
do {  
    wait(chopstick[i]);  
    wait(chopstick[(i+1) % 5]);  
    . . .  
    // eat  
    . . .  
    signal(chopstick[i]);  
    signal(chopstick[(i+1) % 5]);  
    . . .  
    // think  
    . . .  
} while (TRUE);
```



Kudłacz



Systemy Operacyjne

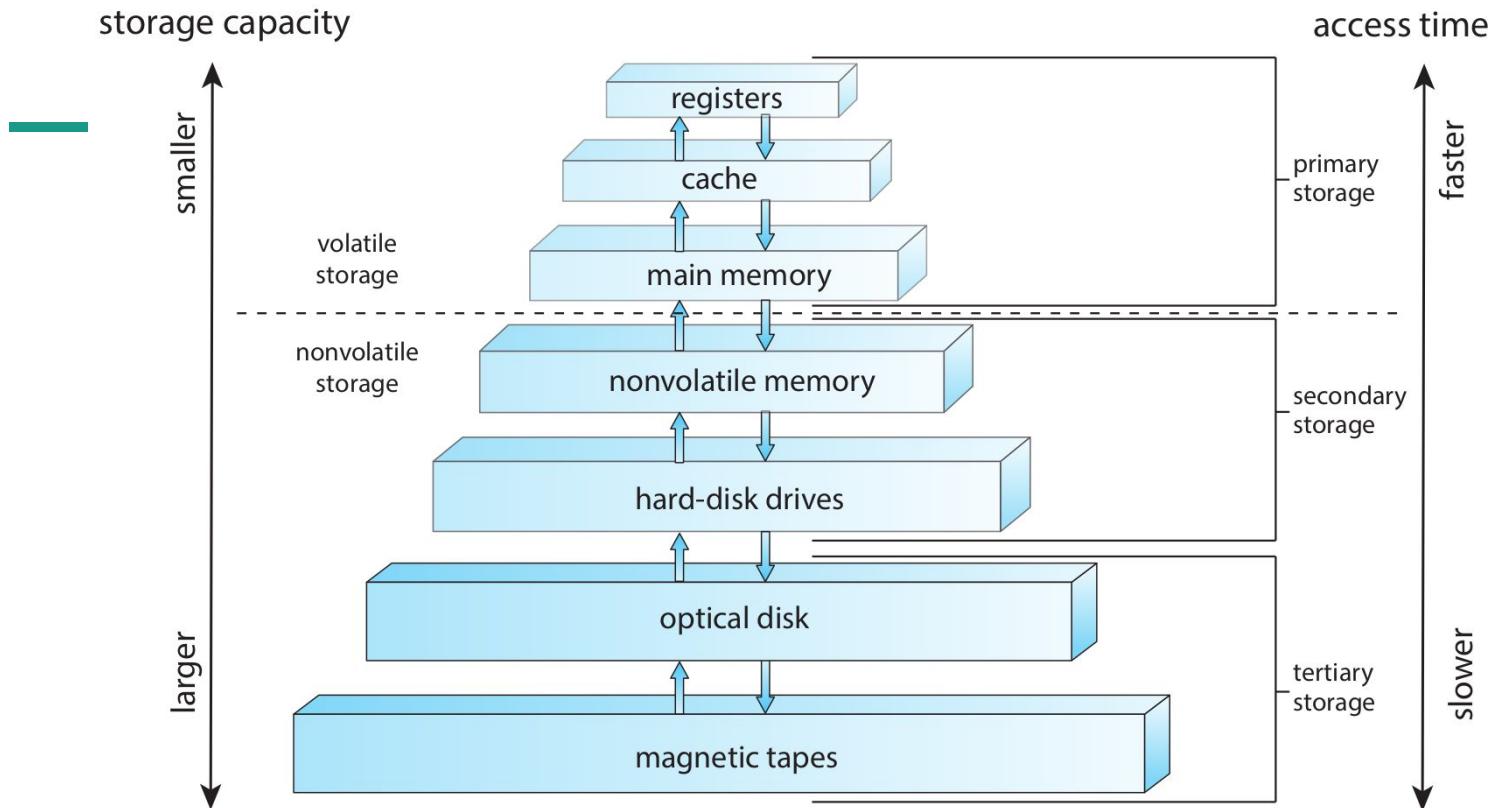
Pamięć główna

Dr hab. inż. Krzysztof Rzecki, prof. AGH

Na podstawie: Abraham Silberschatz, *Koncepcje systemów operacyjnych*

Współdzielenie zasobów

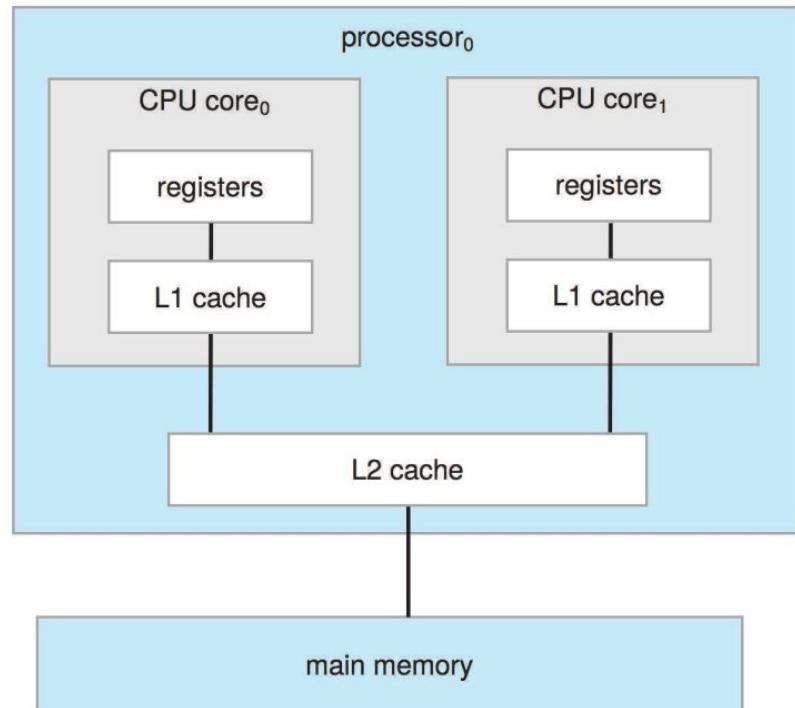
- CPU: współdzielenie w czasie
- MEM: współdzielenie w ilości
- HDD: współdzielenie w dostępie
- NET: współdzielenie w czasie



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Procesor + pamięć

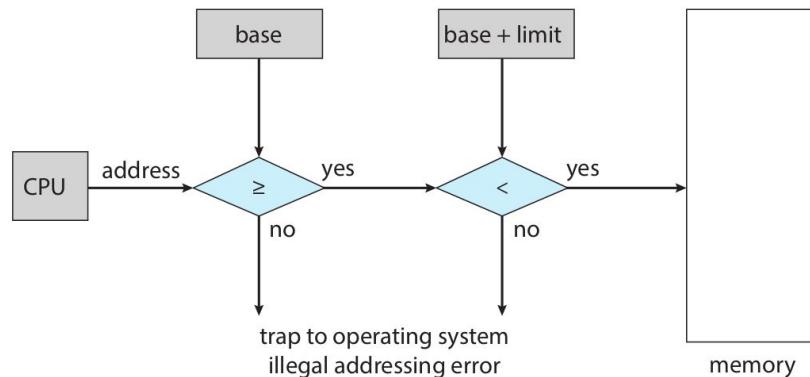
- Procesor ładuje instrukcje tylko z pamięci głównej, więc każdy program musi być do niej najpierw załadowany.
- Pamięć główna (ang. *main memory*, RAM - ang. *random-access memory*) wykonana jest w technologii półprzewodnikowej zwanej DRAM - ang. *dynamic random-access memory*.
- Nie wszystko mieści się w pamięci RAM oraz pamięć ta jest ulotna, stąd wymagana jest pamięć dodatkowa, tj. ang. *secondary storage* (ang. *hard-disk drives* - HDDs lub ang. *nonvolatile memory* - NVM).



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Podstawy adresowania

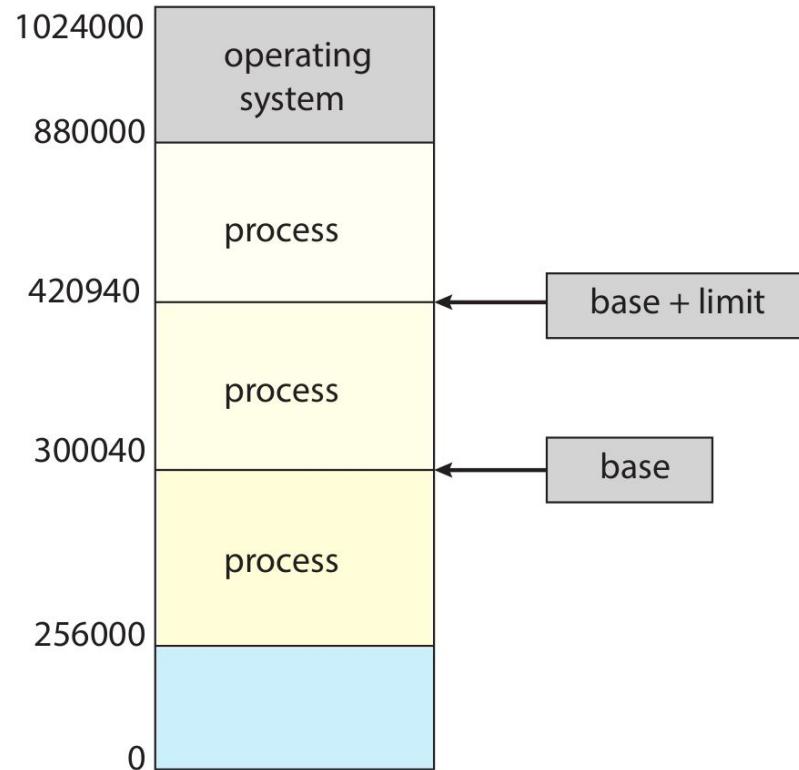
- Pamięć główna oraz rejestrów wbudowane w CPU to jedyna pamięć, którą CPU może adresować bezpośrednio.
- Jeśli dana, na której mają być wykonywane operacje znajduje się na którejkolwiek z pozostałych pamięci, musi zostać najpierw skopiowana w obszar o bezpośredniim dostępie.
- Adresowanie rejestrów, w odróżnieniu od adresowania pamięci głównej, zazwyczaj odbywa się w jednym takcie procesora.
- Aby w trakcie uzyskiwania dostępu do pamięci głównej procesor nie marnował cykli, wykorzystywany jest cache.



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Izolacja pamięci

- Izolacja pamięci dla każdego procesu gwarantuje indywidualną przestrzeń.
- Pamięć przydzieloną procesowi wyznaczają dwa rejesty: baza i limit.
- Proces użytkownika (*user mode*) może zapisywać i odczytywać tylko pamięć w przydzielonym zakresie.
- *Dlaczego ?*
- System operacyjny (*kernel mode*) może swobodnie operować po całej pamięci, w szczególności zmieniać bazę i limit.
- *Jakie to ma zastosowanie ?*



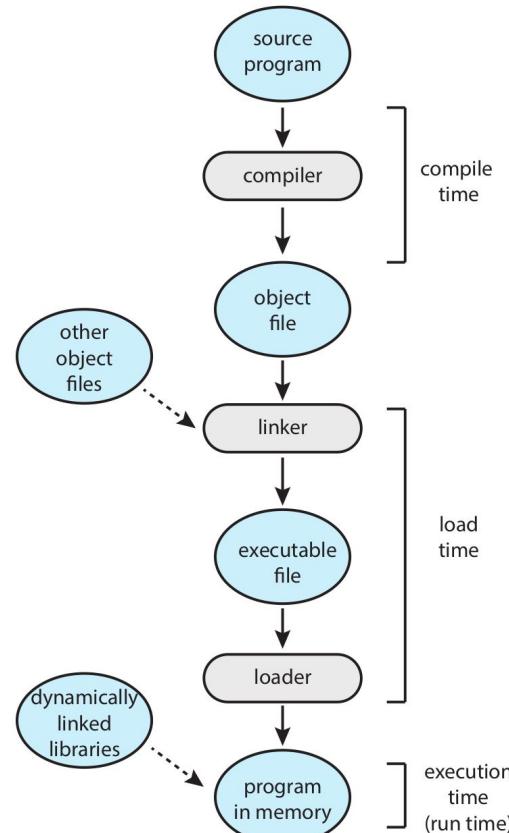
Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Wiązanie adresu

- Adres w kodzie programu może być symboliczny, np. nazwa zmiennej.
- Kompilator wiąże w/w adres symboliczny do adresu relokowalnego (względem danego modułu).
- Linker lub loader zamienia adres relokowalny w bezwzględny.

Wiązanie może wystąpić na każdym z etapów:

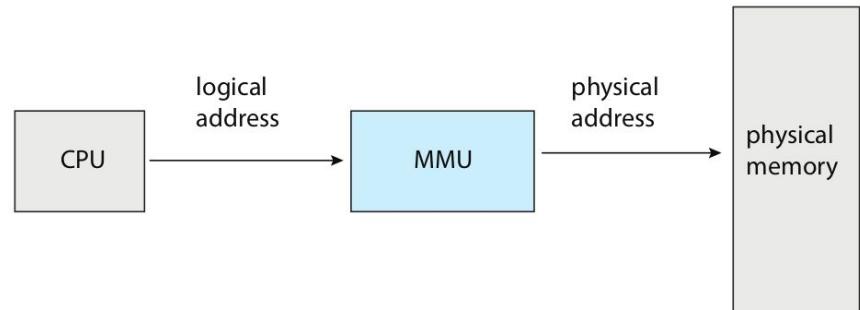
- Kompilacja (stare systemy),
- Ładowanie (MS-DOS),
- Wykonywanie - najczęściej (obecnie).



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Logiczna i fizyczna przestrzeń adresowa

- Adres logiczny - adres widziany przez proces i dla niego dostępny.
- Adres fizyczny - adres na szynie adresowej.
- Translację adresów zajmuje się MMU - ang. *Memory management unit*.
- Wiązanie adresów w czasie:
 - komplikacji -> adres logiczny == fizyczny,
 - ładowania -> adres logiczny == fizyczny,
 - wykonywania -> adres logiczny != fizyczny.
- Adres wirtualny = adres logiczny w sytuacji wiązania w czasie wykonywania.



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Ćw.: w C/C++ statyczna/dynamiczna rezerwacja pamięci, odczyt adresu zmiennej, kilkukrotne uruchomienie programu.

Dynamiczne ładowanie funkcjonalności

- Po stronie programisty leży konstrukcja programu z dynamicznie ładowanymi funkcjonalnościami.
- Funkcjonalności ładowane są dopiero w momencie ich wywołania.
- Kiedy któraś funkcjonalność wywołuje inną, a ta nie jest załadowana do pamięci, następuje jej załadowanie.
- Zaletą jest ładowanie funkcjonalności tylko wtedy, kiedy są potrzebne.

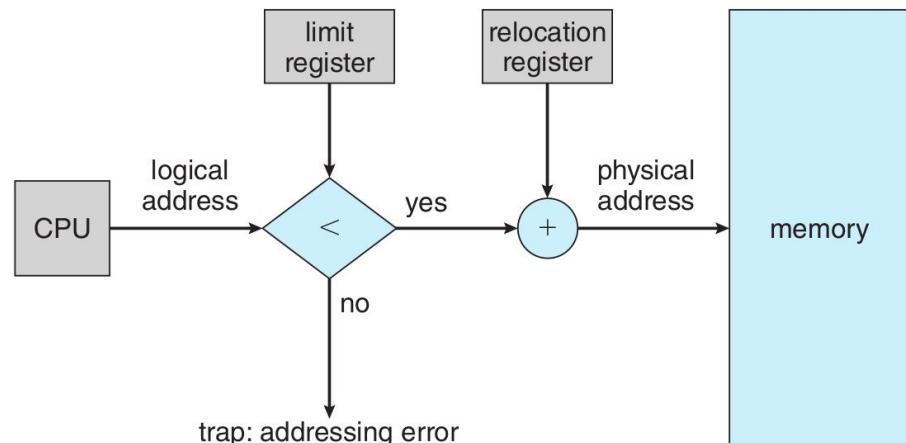
Linkowanie dynamiczne i statyczne

- DLL - ang. *Dynamic-Link Library* - biblioteka systemowa linkowana do programu w momencie jego uruchomienia (wymaga wsparcia ze strony systemu operacyjnego).
- Zalety DLL:
 - nie trzeba kopiować/implementować istniejących już modułów programu (oszczędność pamięci),
 - biblioteka może zostać jednokrotnie załadowana do pamięci, a wiele procesów może z niej korzystać,
 - biblioteki mogą być aktualizowane jednokrotnie, jakkolwiek każdy program może używać wskazanej wersji.
- Linkowanie statyczne - włączanie biblioteki systemowej do obrazu binarnego programu.

Pytanie C/C++: jak linkować statycznie/dynamicznie ?

Ochrona pamięci

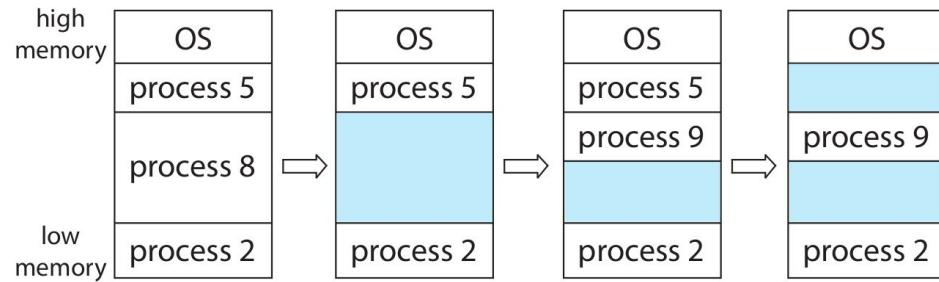
- Relocation register (rejestr bazowy, rejestr relokacji) - początkowy adres fizyczny obszaru przeznaczonego dla procesu.
- Limit register - zawiera zakres adresów logicznych.
- Każdy proces wskazany przez scheduler CPU do uruchomienia sprawdzany jest względem w/w rejestrów.
- Dzięki temu można chronić system operacyjny i inne programy przed modyfikacją przez ten program.



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Alokacja pamięci

- Metoda *variable-partition* - przypisanie procesu do zakresu pamięci i pamiętanie przez SO, które części pamięci są zajęte, a które wolne.
- Jeśli nie ma możliwości umieścić proces w pamięci: a/ proces nie jest uruchamiany, b/ proces oczekuje w kolejce.
- Zarządzanie wolną przestrzenią (ang. *hole*) obejmuje jej dzielenie i łączenie i nazywane jest ang. *dynamic storage allocation problem*:
 - First fit - pierwszy pasujący
 - Best fit - najlepiej pasujący
 - Worst fit - największy wolny



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

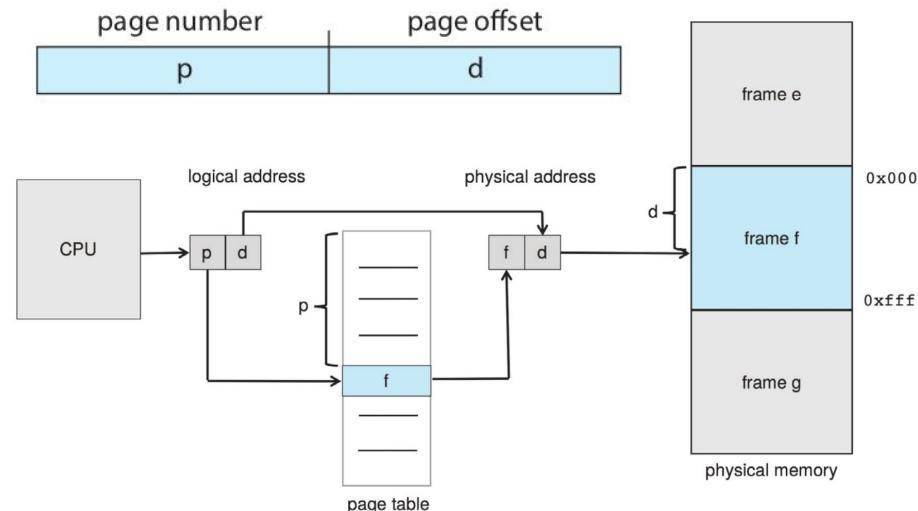


Fragmentacja

- W przypadku strategii first-fit i best-fit szybko dochodzi do tzw. fragmentacji zewnętrznej.
- Najgorszy przypadek fragmentacji: między każdą dwójką procesów jest wolna przestrzeń.
- Oprócz doboru strategii znaczenie ma też położenie nowego procesu (na początku czy na końcu wolnego miejsca?).
- Statystycznie, po pewnym czasie strategii first-fit aż 50% bloków zmarnowanych będzie na fragmentację.
- W przypadku podzielenia pamięci na bloki przydzielona pamięć dla danego procesu może być większa niż proces wymaga - różnica między wielkością przydzieloną a wymaganą to fragmentacja wewnętrzna.
- Rozwiązanie: kompaktowanie (Java: garbage collection) - możliwe tylko w czasie działania.
- Rozwiązanie: stronicowanie (następny slajd).

Stronicowanie, ang. *paging*

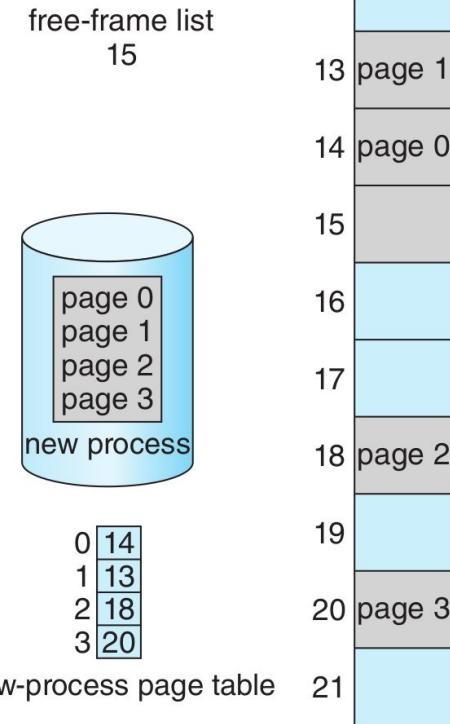
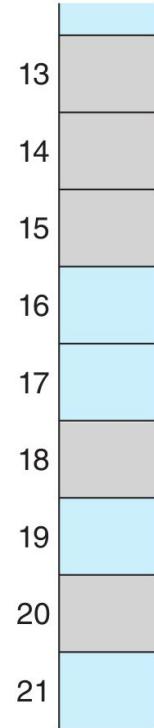
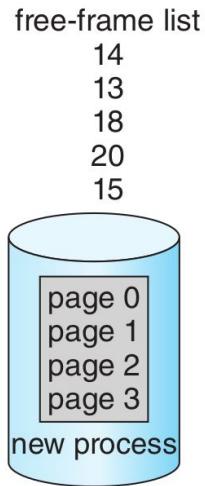
- Stronicowanie - logiczna przestrzeń adresowa jest nieciągła.
- Pamięć fizyczna dzielona jest na bloki, ang. *frames*.
- Pamięć logiczna dzielona jest na bloki tej samej wielkości, ang. *pages*.
- Kiedy proces ma być wykonywany, jego strony ładowane są do ramek dostępnej pamięci.
- Każdy proces ma swoją tablicę stron.
- Rozmiar strony zależny jest od sprzętu i wynosi od 4KB do 1GB i ze względu na adresację jest potęgą 2 (getconf PAGESIZE).



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Alokacja pamięci

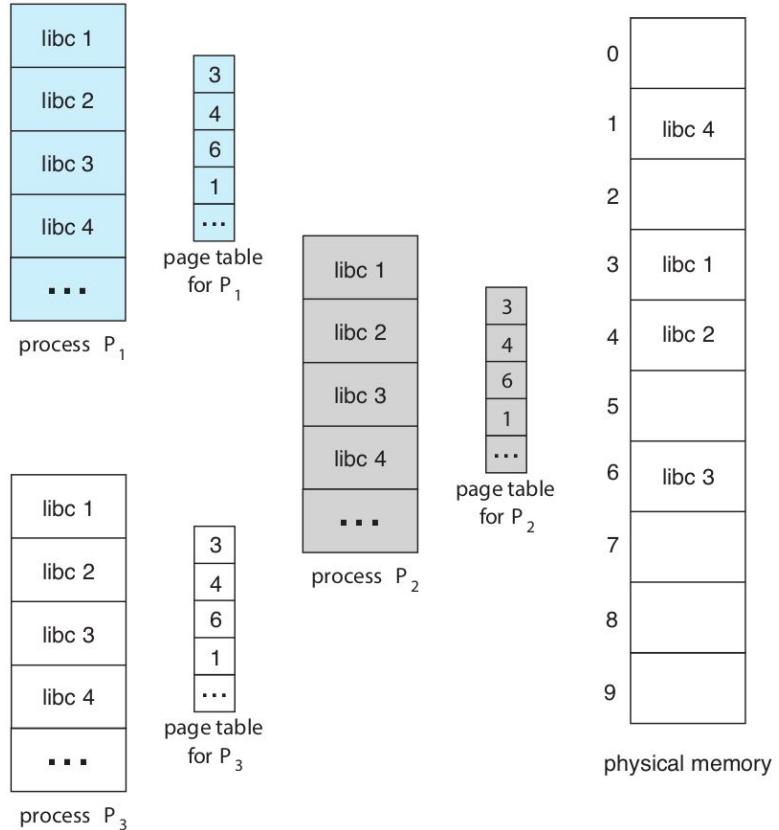
- Programista jest zupełnie odseparowany od widoku na pamięć fizyczną.
- Program może być rozrzucony po obszarach pamięci.
- Tablicą stronicowania oraz tablicą ramek zarządza system operacyjny.



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

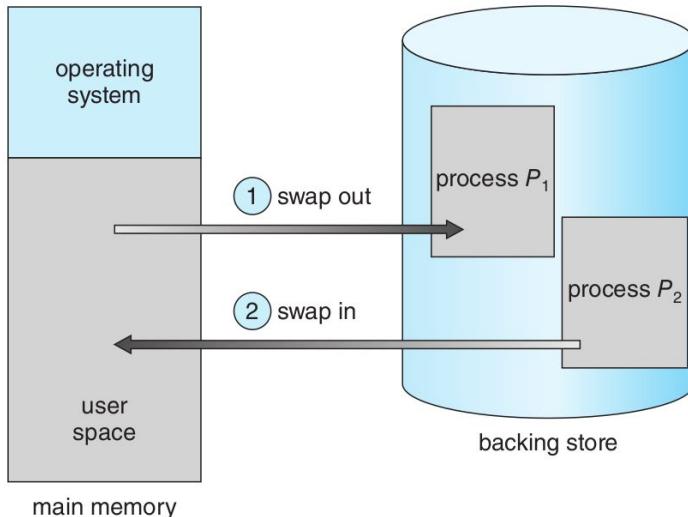
Współdzielenie

Przykład kodu wtórnego: biblioteka libc



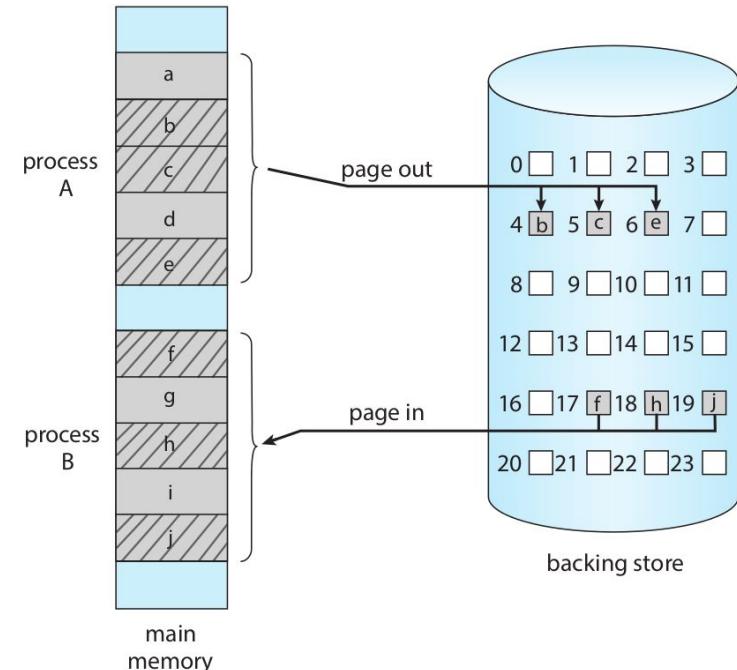
Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Swap



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Kiedyś



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Obecnie

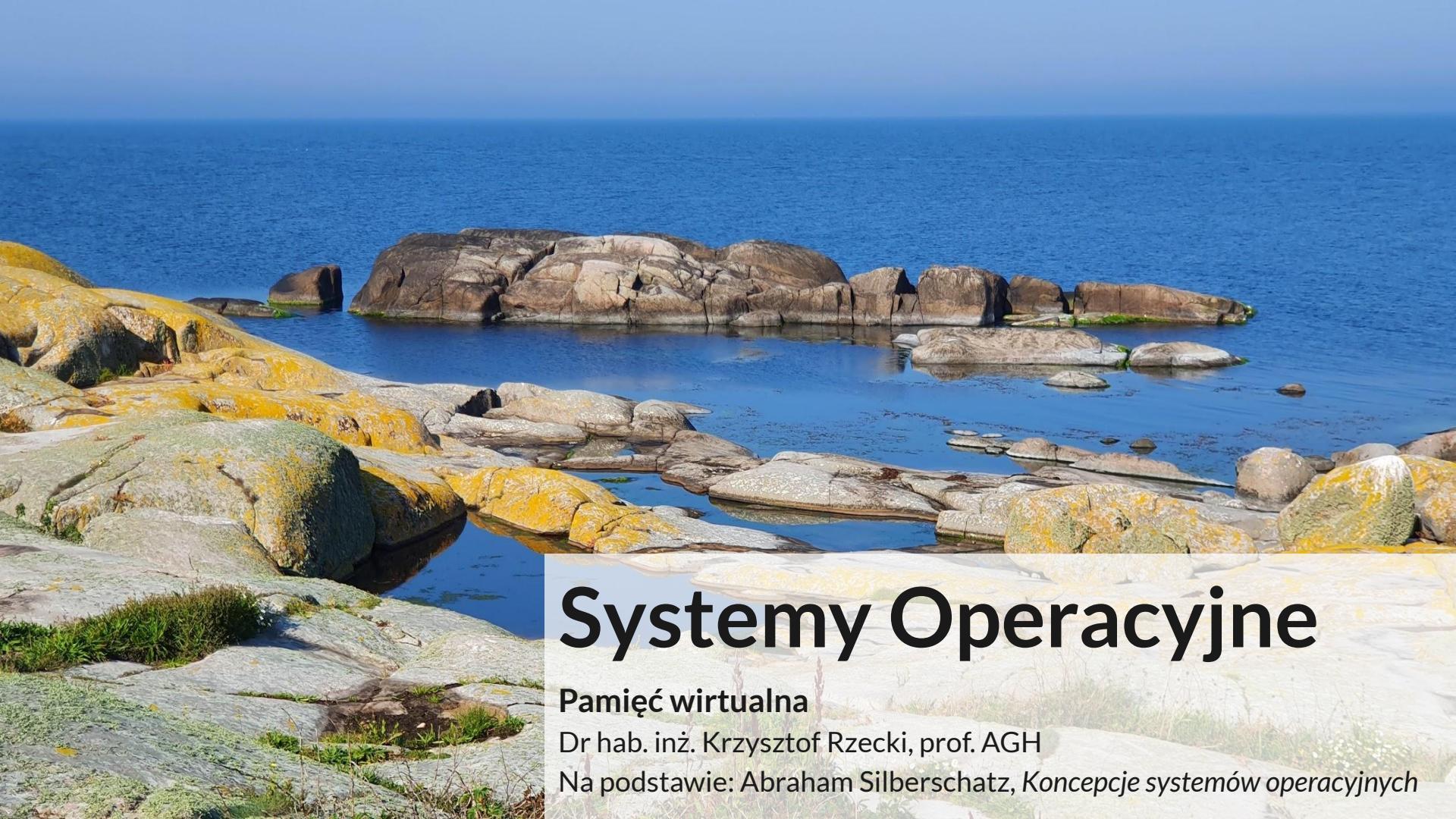
Zadanie

Zakładając logiczną przestrzeń adresową składającą się z 64 stron o rozmiarze 1024 słów zmapowaną na fizyczną przestrzeń adresową o rozmiarze 32 ramek:

- Jaki rozmiar ma adres logiczny ?
- Jaki rozmiar ma adres fizyczny ?



Skrzyczne



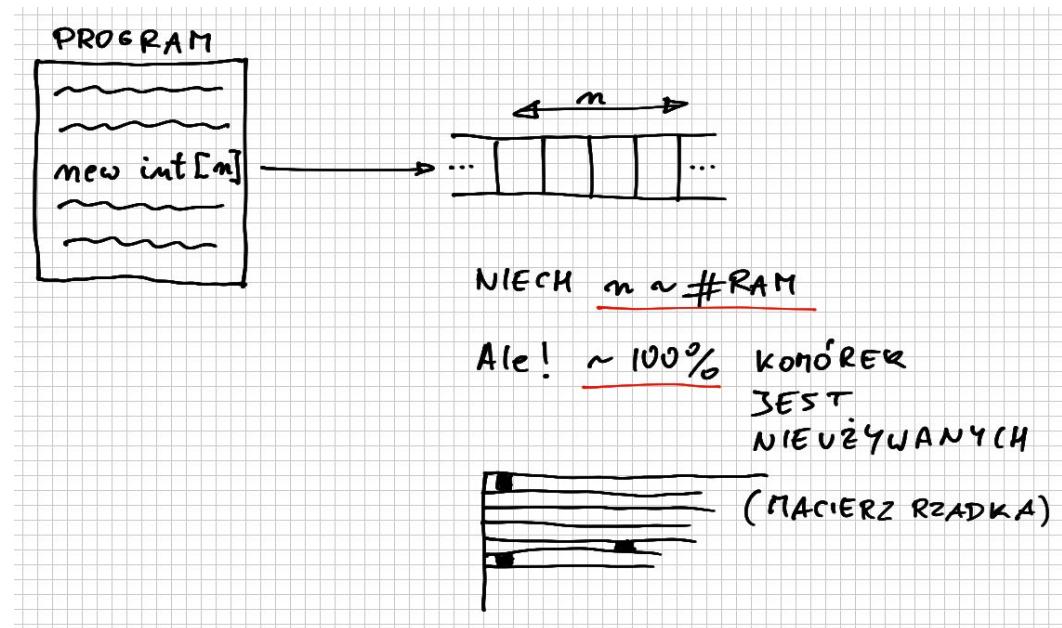
Systemy Operacyjne

Pamięć wirtualna

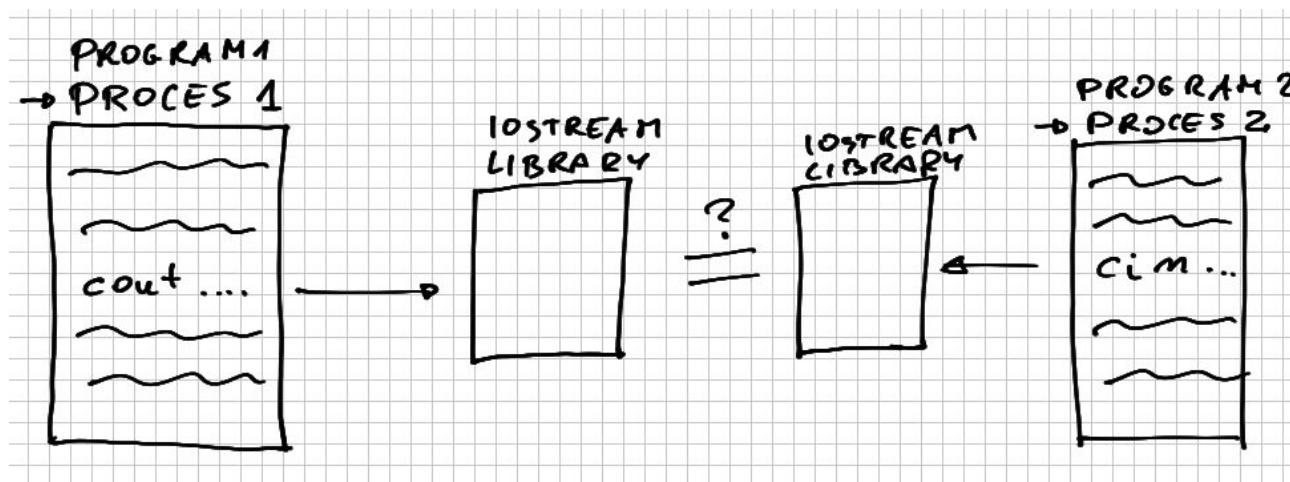
Dr hab. inż. Krzysztof Rzecki, prof. AGH

Na podstawie: Abraham Silberschatz, *Koncepcje systemów operacyjnych*

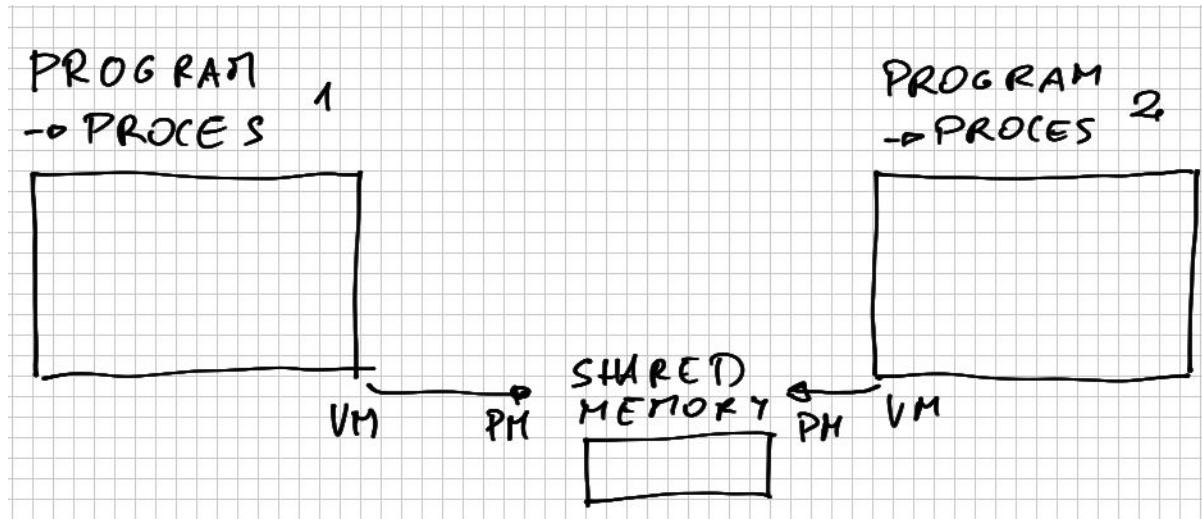
Przypadek 1



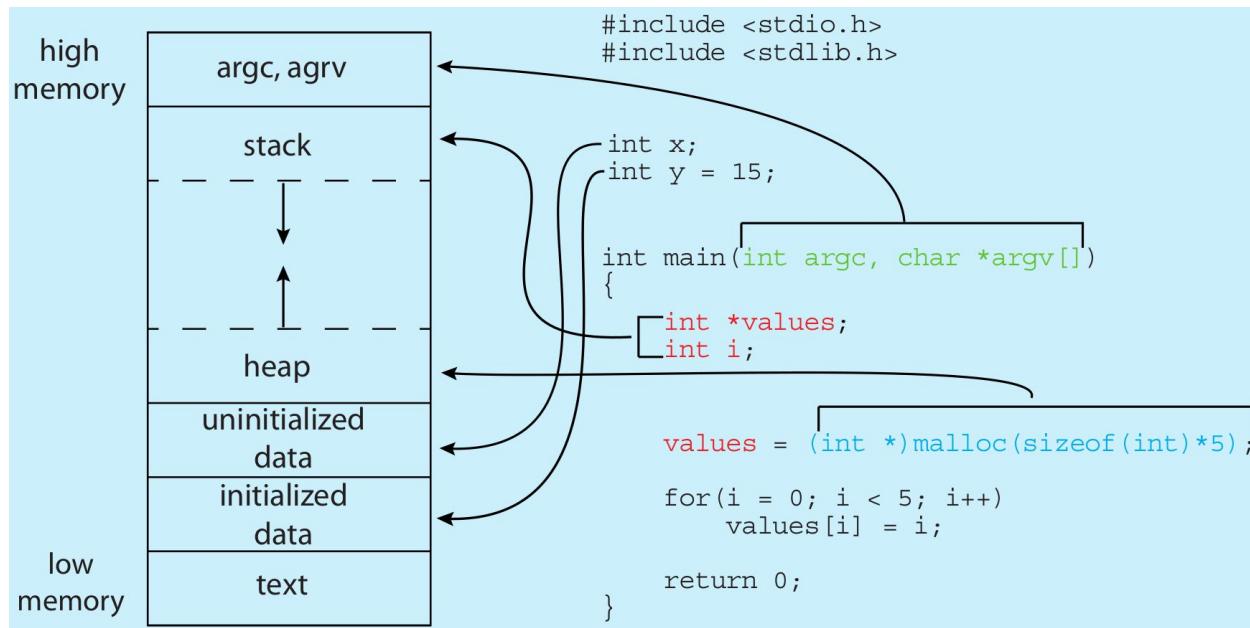
Przypadek 2



Przypadek 3



Program w C - przypomnienie

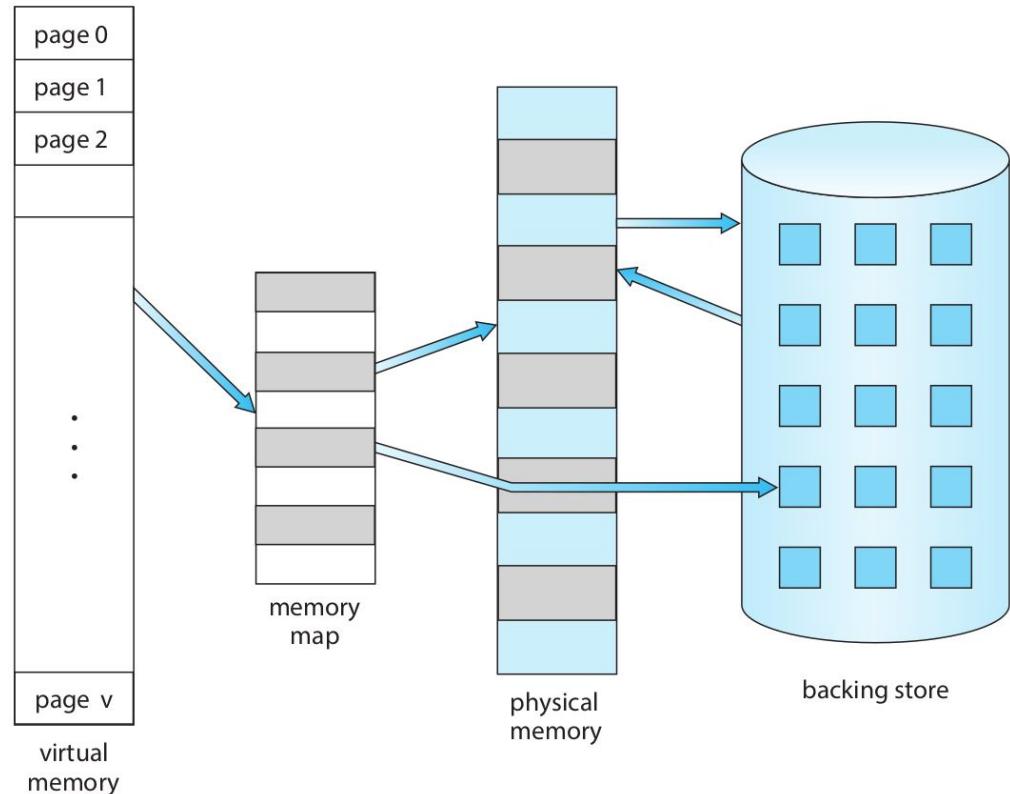


Pamięć wirtualna

- Pamięć wirtualna to technika pozwalająca na wykonywanie procesów, które nie są całkowicie w pamięci (m.in. np. z powodu ich rozmiaru).
- Pamięć wirtualna jest abstraktem pamięci głównej jako bardzo dużej macierzy, oddzielając pamięć logiczną przeznaczoną dla programisty od pamięci fizycznej.
- Pamięć wirtualna pozwala procesom na współdzielenie plików, bibliotek oraz implementację pamięci dzielonej.
- Implementacja pamięci wirtualnej nie jest prosta, a nieostrożne korzystanie z niej wpływa znacząco na wydajność procesów.

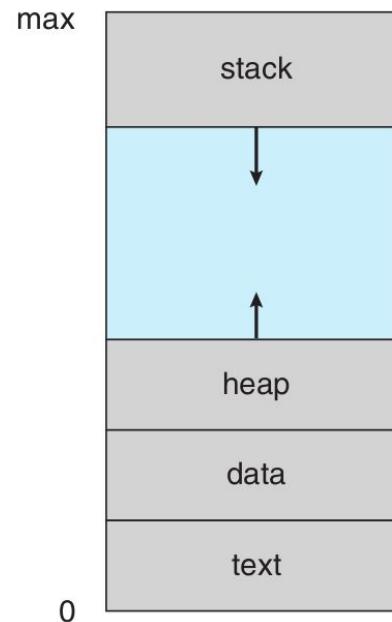
Pamięć wirtualna

- Pamięć wirtualna oddzielona jest od pamięci fizycznej.
- Dzięki temu możliwe jest stosowanie olbrzymiej pamięci wirtualnej przy niewielkiej pamięci fizycznej.
- Programista nie musi zajmować się obsługą i ilością pamięci fizycznej.
- Pamięć zapasowa (ang. *backing store*, ang. *spare space*) - przestrzeń poza pamięcią główną.



Wirtualna przestrzeń adresowa

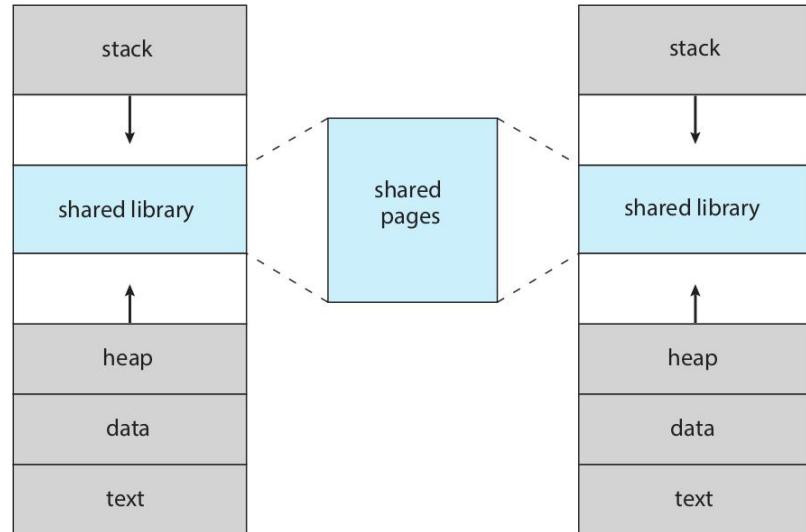
- Stos - ang. *Stack* - miejsce, w którym dane umieszczane są w sposób uporządkowany, w tym miejscu odkładane są dane dotyczące wywołań funkcji, zmiennych (statyczne w tym globalne, automatyczne w tym lokalne). Miejscem tym zarządza program.
- Sterta - ang. *Heap* - miejsce, w którym dane umieszczane są swobodnie wg zapotrzebowania, zwykle przez wywołanie malloc/new, są to zmienne (dynamiczne). Miejscem tym zarządza programista.



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Współdzielenie przestrzeni pamięci fizycznej

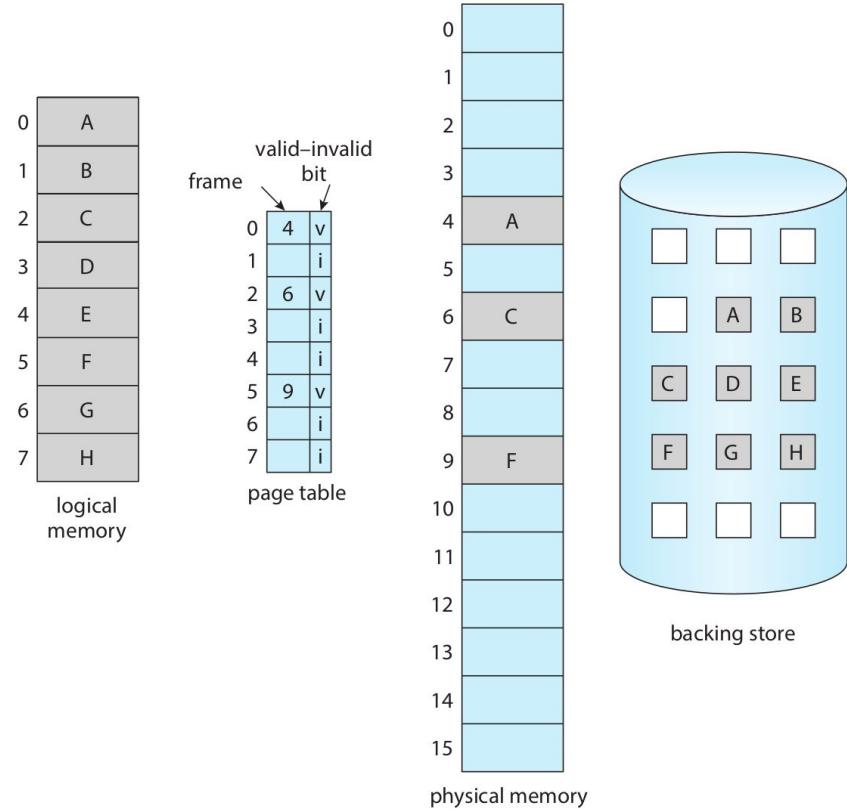
- Adresy wirtualne bibliotek systemowych mogą być mapowane z różnych procesów (w trybie tylko do odczytu) na wspólną część w przestrzeni adresów fizycznych.
- Procesy mogą współdzielić obszar pamięci (ang. *shared memory*) celem wymiany komunikatów i ją też muszą adresować w przestrzeni adresów wirtualnych.



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Stronicowanie na żądanie

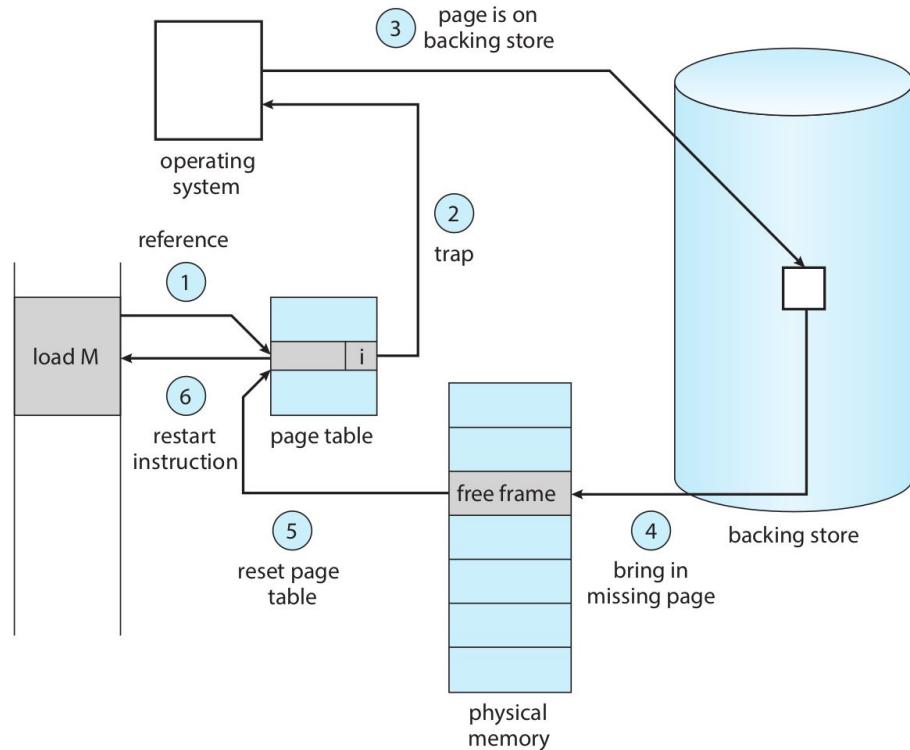
- Strona jest wprowadzana do pamięci wtedy, gdy jest potrzebna.
- Tablica stron zawiera dodatkowo bity poprawności odwołania:
 - 1 - strona znajduje się w pamięci głównej (ang. *valid*)
 - 0 - strona znajduje się poza pamięcią główną (ang. *invalid*)
- Odwołanie do strony z bitem == 0:
 - Błąd braku strony (ang. *page fault*)
 - Realizacja procedury z następnego slajdu.



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Obsługa błędu brak strony

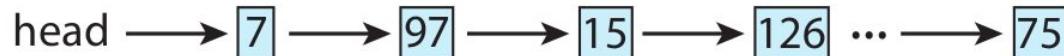
1. Sprawdzenie tablicy stron (valid vs. invalid).
2. Jeśli referencja ma stan *invalid*, to:
 - a. Brak strony w ogóle -> terminowanie procesu.
 - b. Brak strony w pamięci głównej, ale jest w pamięci zapasowej, przejdź do (3).
3. Zgłoszenie do SO zapotrzebowania na wczytanie strony z pamięci zapasowej.
4. Odnajdowanie na liście wolnej ramki i wczytanie strony do ramki.
5. Aktualizacja tablicy stron.
6. Restart instrukcji, która wywołała błąd.



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Lista wolnych ramek (ang. *Free-Frame List*)

- W momencie wystąpienia błędu strony, system operacyjny musi dostarczyć daną stronę z pamięci zapasowej do pamięci głównej.
- W tym celu stosowana jest lista wolnych ramek:



- System operacyjny zwykle stosuje technikę ang. *zero-fill-on-demand*, która zeruje ramki przed użyciem (względy bezpieczeństwa).
- W momencie startu systemu, cała dostępna pamięć umieszczana jest na liście wolnych ramek.

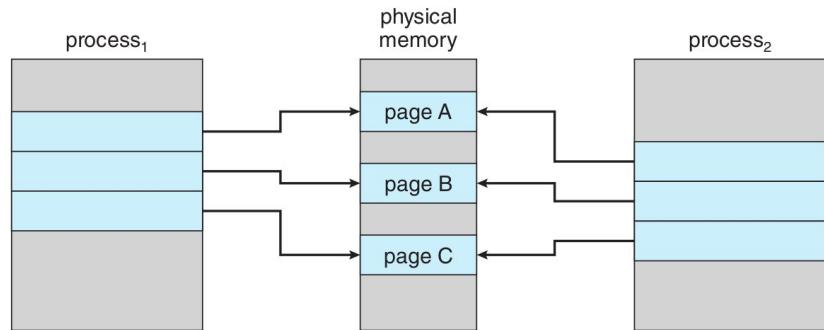
Przebieg stronicowania na żądanie

1. Odwołanie/przerwanie do systemu operacyjnego.
2. Zapisz stan rejestrów oraz procesu.
3. Stwierdź, że przerwanie wywołane było przez błąd strony.
4. Sprawdź, czy odwołanie jest właściwe i określ położenie strony w pamięci zapasowej.
5. Wywołaj odczyt z pamięci zapasowej do wolnej ramki:
 - a. Czekaj w kolejce, aż żądanie odczytu zostanie obsłużone.
 - b. Odczekaj czas działania urządzenia.
 - c. Rozpocznij transfer strony do wolnej ramki.
6. Podczas oczekiwania, przekaż CPU innemu procesowi.
7. Przechwyć przerwanie z podsystemu I/O (zakończenie wczytywania).
8. Zapisz rejestyry oraz stan innego procesu (jeśli krok 6 jest wykonany).
9. Określ, że przerwanie było z pamięci zapasowej.
10. Zaktualizuj tablicę stron, aby wskazać, że określona strona jest teraz w pamięci.
11. Czekaj, aż CPU zostanie przypisany znów temu procesowi.
12. Wznów rejestyry, stan procesu oraz nową tablicę stron i wznowdź działanie procesu.

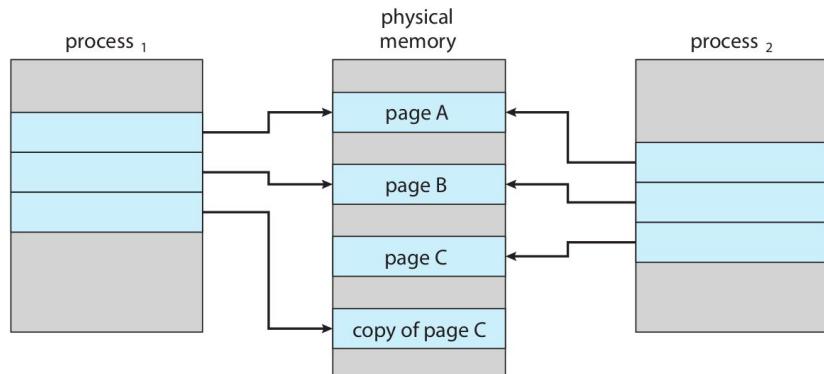
Czasochłonne!

Copy-on-Write

- COW w technice fork() pozwala na współdzielenie tych samych stron w pamięci.
- Tylko ta strona, która jest modyfikowana wymaga kopiowania.
- Często po fork() występuje exec() i wtedy okazuje się, że kopiowanie w ogóle nie jest potrzebne.

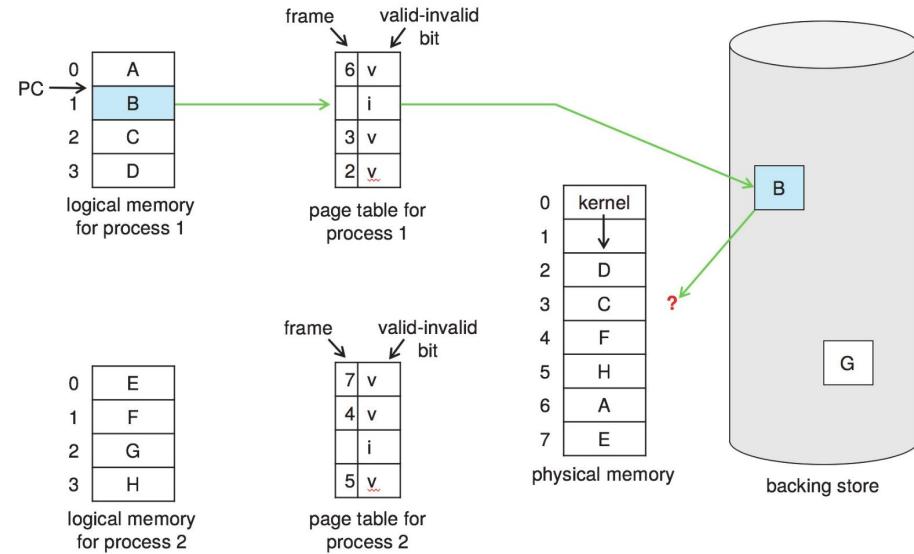


Modyfikacja strony C



Over-allocation

- W trakcie wykonywania procesu, występuje błąd strony.
- System operacyjny odnajduje stronę w pamięci zapasowej, ale stwierdza, że nie ma wolnych ramek - cała pamięć jest zajęta.
- Jednym z rozwiązań jest terminowanie procesu... ale to nie jest rozwiązanie.



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Zastępowanie stron

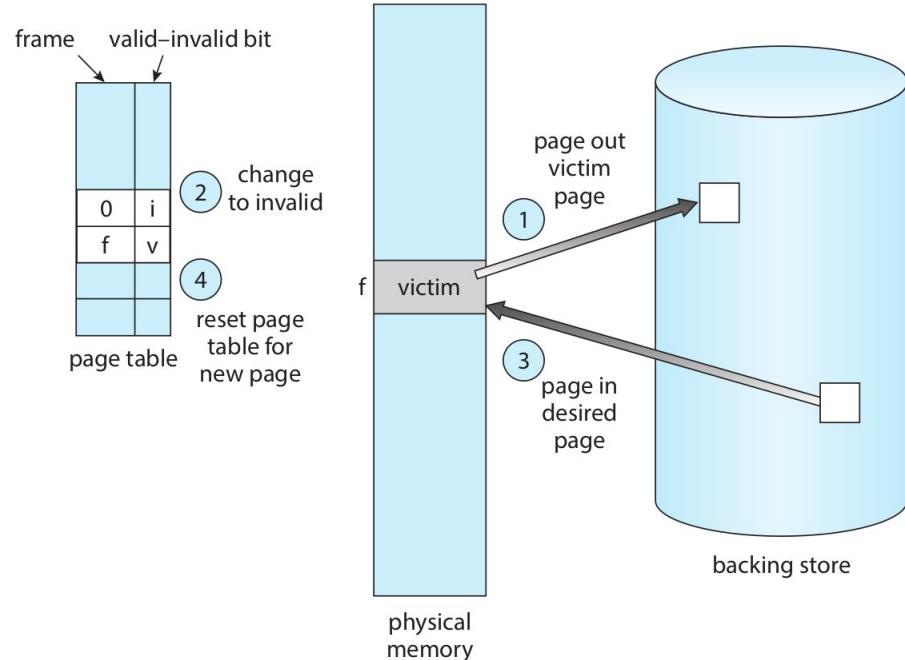
- Co kiedy wolne ramki się skończą ?
- Co w sytuacji, kiedy nie wszystkie strony są aktualnie w użyciu ?
- Jak zastępować strony nieużywane ?
- Czy ładować wszystkie strony, czy tylko przewidziane do użycia ?
- Co w sytuacji, kiedy odwołania do stron są w kilku miejscach programu ?
- Czy pozwalać na uruchomienie programów, dla których nie ma dostępnych ramek ?
- Jak wybierać ramki do zastąpienia ?
- Jak informować procesy o zastąpieniu ramki ?

Uwaga: adresacja logiczna = strony, adresacja fizyczna = ramki.

Zastępowanie stron

1. Znajdź stronę w pamięci zapasowej.
2. Szukaj wolnej ramki:
 - o Jeśli jest, użyj jej.
 - o Jeśli brak, użyj algorytmu wyboru ramki podlegającej wymianie (ang. *victim frame*).
 - o Zapisz jej zawartość do pamięci zapasowej, zaktualizuj tablice ramek i stron.
3. Wczytaj oczekiwana stronę do zwolnionej ramki. Zaktualizuj tablice ramek i stron.
4. Kontynuuj działanie procesu.

Celem optymalizacji stosuje się bit modyfikacji, tzn. brudny bit (ang. *dirty bit*). Np. strony z kodem programu zasadniczo są read-only.



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Algorytmy zastępowania stron

- Algorytm FIFO:
 - najstarsza z umieszczonych w pamięci stron (głowa) jest zastępowana, nowe strony umieszczane są na końcu kolejki (ogon),
 - wada: z jednej strony strona zastępowana może być czymś, co było dawno temu umieszczone w pamięci i jest już nieużywane, ale może to być też często używana zmieniająca od początku procesu.
- Optymalne zastępowanie stron:
 - zastąp stronę, która nie będzie używana przez najdłuższy czas.
 - wada: trzeba znać przyszłość ;-)
- Algorytm LRU - ang. *least recently used*
 - zastąp stronę, która najdłużej nie była używana,
 - implementacje: liczniki (timestamp ostatniego użycia), stos (przekładanie na wierzch użytej strony),
 - dalsze modyfikacje i optymalizacje.

Alokacja ramek

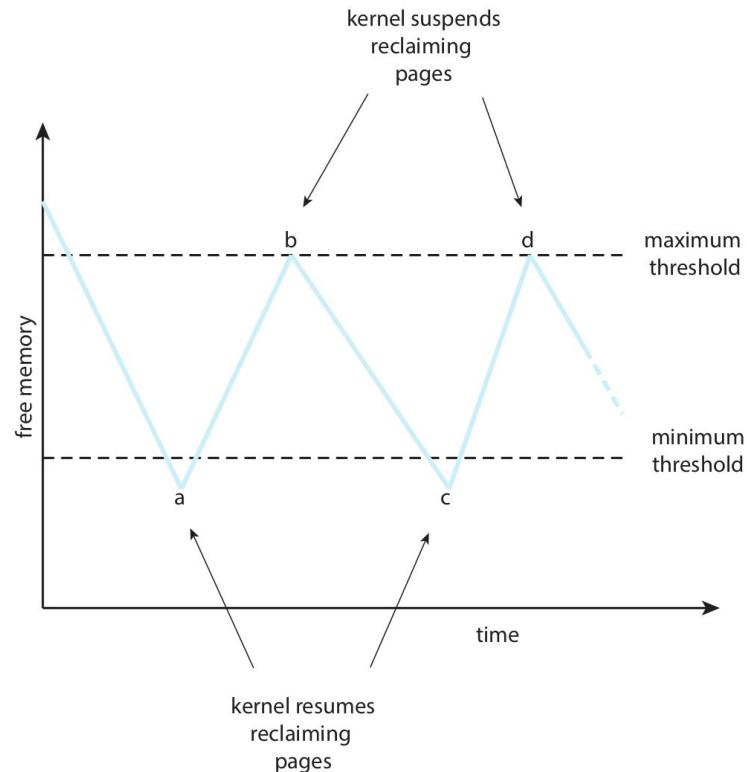
- **Przykład:** system ma 128 ramek pamięci fizycznej, OS zajmuje 35, a 93 ramki są dla procesów.
- W przypadku czystego stronicowania na żądanie (proces w całości ładowany jest do pamięci zapasowej), proces użytkownika wywoła 93 razy błąd strony, a przy żądaniu 94-tej strony zadziała algorytm zastępowania stron. Po zakończeniu procesu zwolnią się wszystkie ramki.
- **Strategie:**
- Minimalna liczba ramek - określona jest minimalna liczba ramek, które muszą zostać zaalokowane. Powód: im mniej zaalokowanych ramek, tym bardziej spada wydajność wykonywanych procesów.
- Równa alokacja - ramki po równo podzielone są między procesy, np. 93 ramki podzielić między 5 procesów oznacza, że każdy z nich otrzyma 18 ramek. Pozostałe 3 ramki trafią do puli wolnych.
- Proporcjonalna alokacja - ramki przydzielane są proporcjonalnie do wielkości procesów.

Globalna i lokalna alokacja

- Alokacja globalna / zastępowanie globalne - realizacja procesu wymaga zastępowania ramek pośród wszystkich ramek, także tych zaalokowanych do innego procesu.
- Alokacja lokalna / zastępowanie lokalne - realizacja procesu wymaga zastępowania ramek tylko pośród zaalokowanych do procesu.

Alokacja globalna ma szczególne zastosowanie w przypadku priorytetyzowania procesów.

Rysunek obok: strategia utrzymania wolnej pamięci w alokacji globalnej.



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Major and minor page faults

- Major page fault - występuje wtedy, gdy strona jest wywoływana, a nie znajduje się w pamięci. Obsługa tego błędu wymaga odczytania wskazanej strony z pamięci zapasowej do wolnej ramki i aktualizacji tablicy stron. Stronicowanie na żądanie zwykle generuje znaczną liczbę tych błędów.
- Minor page fault - występuje wtedy, gdy proces nie ma logicznego mapowania do strony, ale ta strona jest w pamięci. Błąd ten występuje w jednym z przypadków:
 - Proces odwołuje się do biblioteki dzielonej, która jest w pamięci, ale proces nie ma do niej mapowania. W tym przypadku wystarczy zaktualizować tablicę stron.
 - Proces utracił stronę, która trafiła na listę wolnych ramek, ale nie została jeszcze wyzerowana. W tym przypadku strona jest ponownie przypisana do procesu i usunięta z listy wolnych ramek.

Na laboratorium:

```
ps -eo min_flt, maj_flt, cmd
```



Wyspa Utklippan

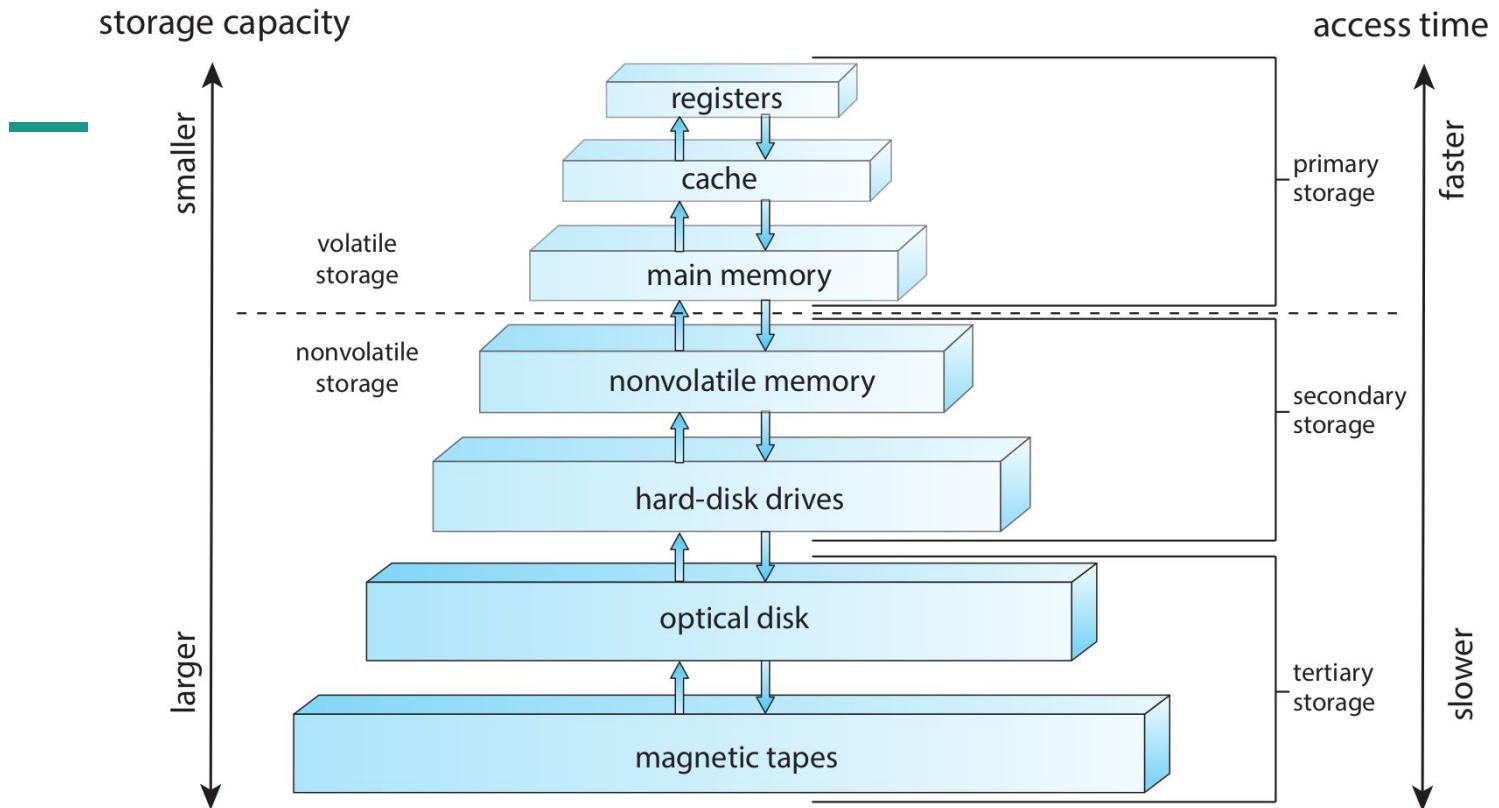


Systemy Operacyjne

Pamięć masowa

Dr hab. inż. Krzysztof Rzecki, prof. AGH

Na podstawie: Abraham Silberschatz, *Koncepcje systemów operacyjnych*



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Pamięć masowa

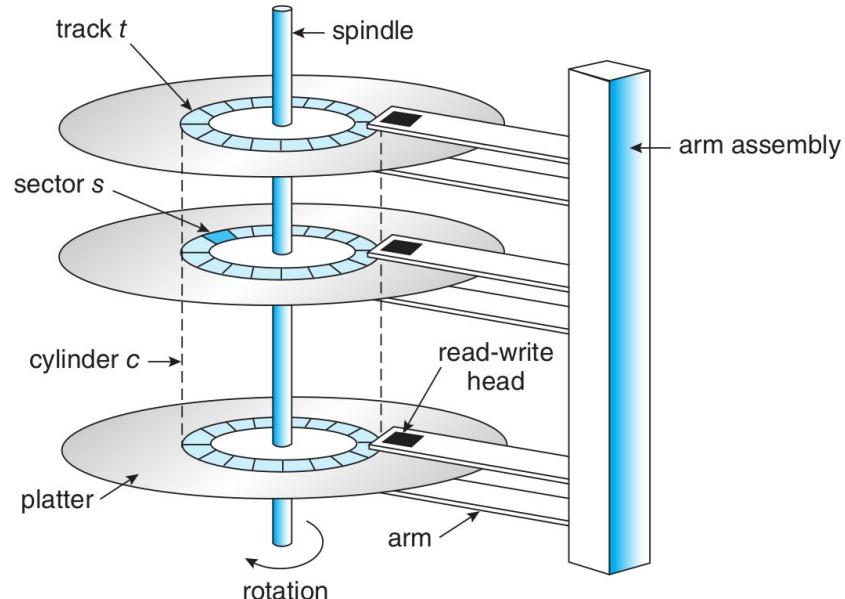
- Pamięć masowa to trwały nośnik pamięci, który w odróżnieniu do pamięci operacyjnej, może przechowywać (bez podtrzymywania elektrycznego) informacje przez długi czas.
- Adresowanie pamięci masowej zależy od jej typu, ale nie jest to adresowanie bezpośrednie, jak w przypadku pamięci operacyjnej.
- Rodzaje:
 - Pamięć magnetyczna: dysk twardy (HDD, ang. *hard disk drive*), ale też dyskietka (ang. *floppy disk drive*), taśma magnetyczna (ang. *magnetic tape*).
 - Pamięć optyczna: dysk CD (ang. *compact disk*), dysk DVD (ang. *Digital Video Disc lub Digital Versatile Disc*), dysk BD (ang. *Blu-ray Disc*).
 - Pamięć półprzewodnikowa: dysk SSD (ang. *Solid State Drive*), NVM (ang. *Non-volatile memory*): pamięć typu *flash*, pamięć USB, karty pamięci.

Pamięć masowa

- Brak możliwości bezpośredniego wykonywania programów.
- Cechy charakterystyczne (SSD/HDD):
 - Wielkość: 250 GB - 6+TB
 - Niska cena: ~ 0,05 - 0,10 zł / 1 GB
- Dane trwale zapisane bez podłączonego zasilania.
- Czas dostępu:
 - HDD: 20 ms
 - SSD: ~20 us
- Trwałość MTBF:
 - HDD: 1-2,5 mln godzin
 - SSD: 1,5-3 mln godzin

Budowa dysku twardego

- Dysk twardy składa się z:
 - Talerz (ang. platter): 1.8-3.5" - dwustronna powierzchnia magnetyczna.
 - Główica czytająco-pisząca (ang. head).
 - Ramię z głowicami (ang. arm).
- Powierzchnia talerza:
 - Ścieżki (ang. tracks), które podzielone są na:
 - Sektor (ang. sectors), których zbiór przy danej pozycji ramienia to:
 - Cylinder (ang. cylinder).
- Każdy sektor posiada stały rozmiar i jest to najmniejsza jednostka transferu (obecnie najczęściej 4KB).
- Prędkość obrotowa: 5'400, 7'200, 10'000, 15'000.



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*



LBA - logical block addressing

$LBA = (\text{numer cylindra } C * \text{liczba głowic na cylinder HPC} + \text{numer głowicy } H)$

* liczba sektorów na ścieżkę SPT + numer sektora S - 1

$$C = LBA / (HPC * SPT)$$

$$H = (LBA / SPT) \bmod HPC$$

$$S = (LBA \bmod SPT) + 1$$

Interfejsy pamięci masowej

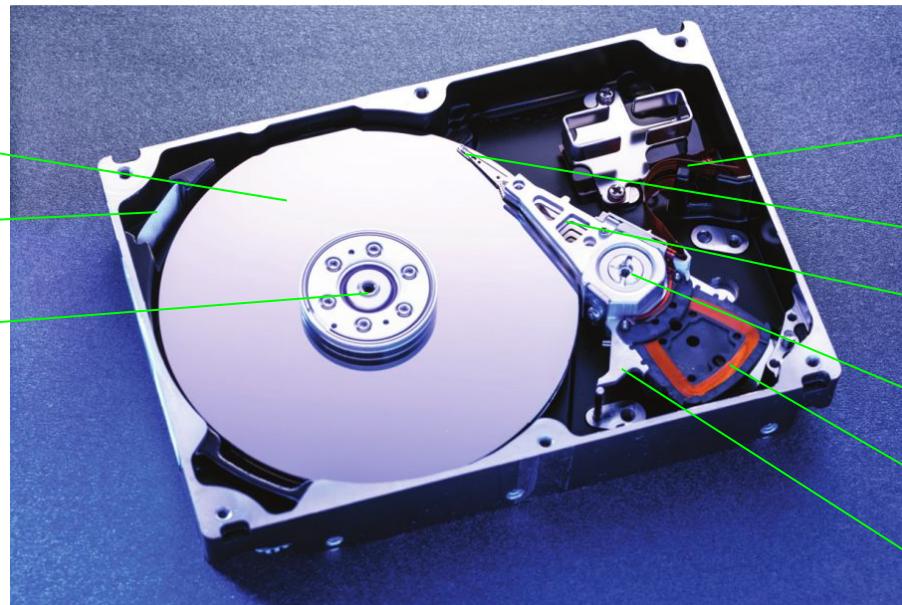
- ATA, ang. *Advanced Technology Attachment*
== IDE, ang. *Integrated Device Electronics*
- ATAPI, ang. *Advanced Technology Attachment Packet Interface*
- PATA, ang. *Parallel ATA* (ATA + kontroler)
- SATA, ang. *Serial ATA*
- eSATA, ang. *External SATA*
- SCSI, ang. *Small Computer Systems Interface*
- SAS, ang. *Serial Attached SCSI* - częściowo kompatybilny z SATA
- USB, ang. *Universal Serial Bus*
- M.2 (NGFF, ang. *Next Generation Form Factor*) i PCI Express dla urządzeń NVMe, ang. *Non-volatile Memory Express*

Dysk twardy HDD 3.5" (cale) - podobnie 2.5"

Talerz dysku
twardego

Filtr

Piasta silnika
(łożysko kulkowe)



Podłączenie
elektroniki

Główica

Ramię

Pozycjoner

Cewka

Magnesy

Dysk półprzewodnikowy SSD 3.5"



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Formaty dysków SSD (ang. *Solid-state drive*)



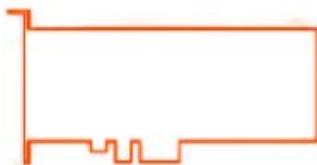
2,5"



mSATA



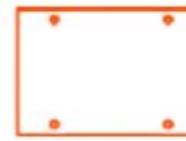
M.2



HHHL - Half Height, Half Length
(zwana także kartą rozszerzeń)



M.2
(wspiera wersję NVMe)



U.2
(dostępne tylko w NVMe)

Rozmiar sektora - dyski HDD i SSD

- Początkowo 512 bajtów, obecnie najczęściej 4KiB (4 kibibity).
- W systemie plików:

```
$ cat /sys/class/block/sdX/queue/physical_block_size => 4096
```

```
$ cat /sys/class/block/sdX/queue/logical_block_size => 512
```

- Programy:

```
$ LC_ALL=C fdisk -l /dev/sdX | grep 'Sector size'
```

```
$ sudo smartctl -a /dev/sdX | grep 'Sector Size'
```

```
$ sudo hdparm -I /dev/sda | grep 'Sector size:'
```

Rozmiar sektora - dyski NVMe

- W systemie plików:

```
$ sudo smartctl -a /dev/nvme0n1
```

Supported LBA Sizes (NSID 0x1)

Id	Fmt	Data	Metadt	Rel_Perf
----	-----	------	--------	----------

0 +	512	0	0	
-----	-----	---	---	--

Odczyt własności dysku twardego: HDD

```
# gdisk -l /dev/sdb
```

```
Disk /dev/sdb: 3907029168 sectors, 1.8 TiB
Model: ST2000DM008-2FR1
Sector size (logical/physical): 512/4096 bytes
Disk identifier (GUID): AB46240F-CCFF-4768-9379-8C1FAC47CD7B
Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
First usable sector is 34, last usable sector is 3907029134
Partitions will be aligned on 2048-sector boundaries
Total free space is 3907029101 sectors (1.8 TiB)
```

```
# hdparm -g /dev/sdb
```

```
geometry      = 243201/255/63, sectors = 3907029168, start = 0
```

Odczyt własności dysku twardego: SSD

```
# gdisk -l /dev/sda
```

```
Disk /dev/sda: 937703088 sectors, 447.1 GiB
Model: KINGSTON SUV500M
Sector size (logical/physical): 512/4096 bytes
Disk identifier (GUID): 2025DC72-437D-425C-A561-4D4EEB1FDB61
Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
First usable sector is 34, last usable sector is 937703054
Partitions will be aligned on 2048-sector boundaries
Total free space is 9139 sectors (4.5 MiB)
```

```
# hdparm -g /dev/sda
```

```
geometry      = 58369/255/63, sectors = 937703088, start = 0
```

Odczyt własności dysku twardego: NVMe

```
# gdisk -l /dev/nvme0n1

Disk /dev/nvme0n1: 1953525168 sectors, 931.5 GiB
Model: KINGSTON SA2000M81000G
Sector size (logical/physical): 512/512 bytes
Disk identifier (GUID): 15F70755-2100-47E8-B0DF-F5D42D92599B
Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
First usable sector is 34, last usable sector is 1953525134
Partitions will be aligned on 2048-sector boundaries
Total free space is 28013 sectors (13.7 MiB)
```

```
# hdparm -g /dev/nvme0n1

geometry      = 953869/64/32, sectors = 1953525168, start = 0
```

Informacje diagnostyczne o dysku twardym

```
# smartctl -a /dev/sdb
```

- Reallocated_Sector_Ct- liczba realokowanych sektorów w miejsce zapasowe.
- Reallocated_Event_Count- liczba operacji realokowania (po kilka sektorów).
- Current_Pending_Sector- liczba sektorów oczekujących na remapowanie.
- Offline_Uncorrectable- liczba błędów, których nie można naprawić.

Mapowanie adresów

- Pamięć masowa adresowana jest jako jednowymiarowa macierz **bloków logicznych**.
- Blok logiczny to najmniejsza jednostka transferu.
- Każdy blok logiczny mapowany jest do sektora fizycznego lub strony pamięci półprzewodnikowej.
- W przypadku HDD logiczny sektor 0 to np. fizycznie pierwszy sektor najbardziej zewnętrzny cylindra. Adresowanie jest wzdłuż tego cylindra, a potem schodzi do cylindrów wewnętrznych.
- W przypadku NVM adresacja zaczyna się od pierwszych: chip, blok, strona.
- Na ogół używa się adresacji przez LBA, ang. *Logical Block Address*.

MBR, ang. *Master Boot Record*

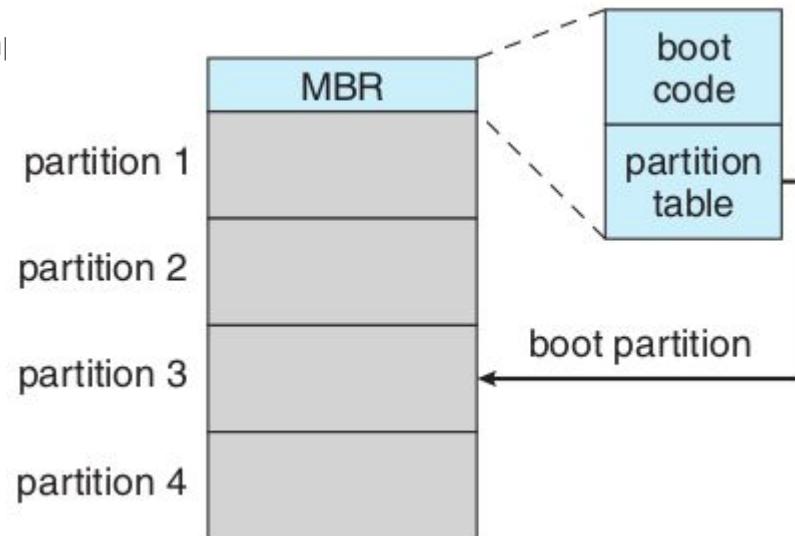
- Ładowanie systemu zaczyna się od programu **bootstrap**.
- 446 bytes - Bootstrap.
- 64 bytes - Partition table.
- 2 bytes - Signature.

Kopia:

```
$ dd if=/dev/sdX of=file.img bs=512 count=1  
+ odzyskiwanie
```

Dla partycji rozszerzonych:

```
$ sfdisk -d /dev/sdX > /tmp/sdX.sfdisk  
$ sfdisk /dev/sdX < /tmp/sdX.sfdisk
```



Monitorowanie urządzeń we/wy

Narzędzia do monitorowania:

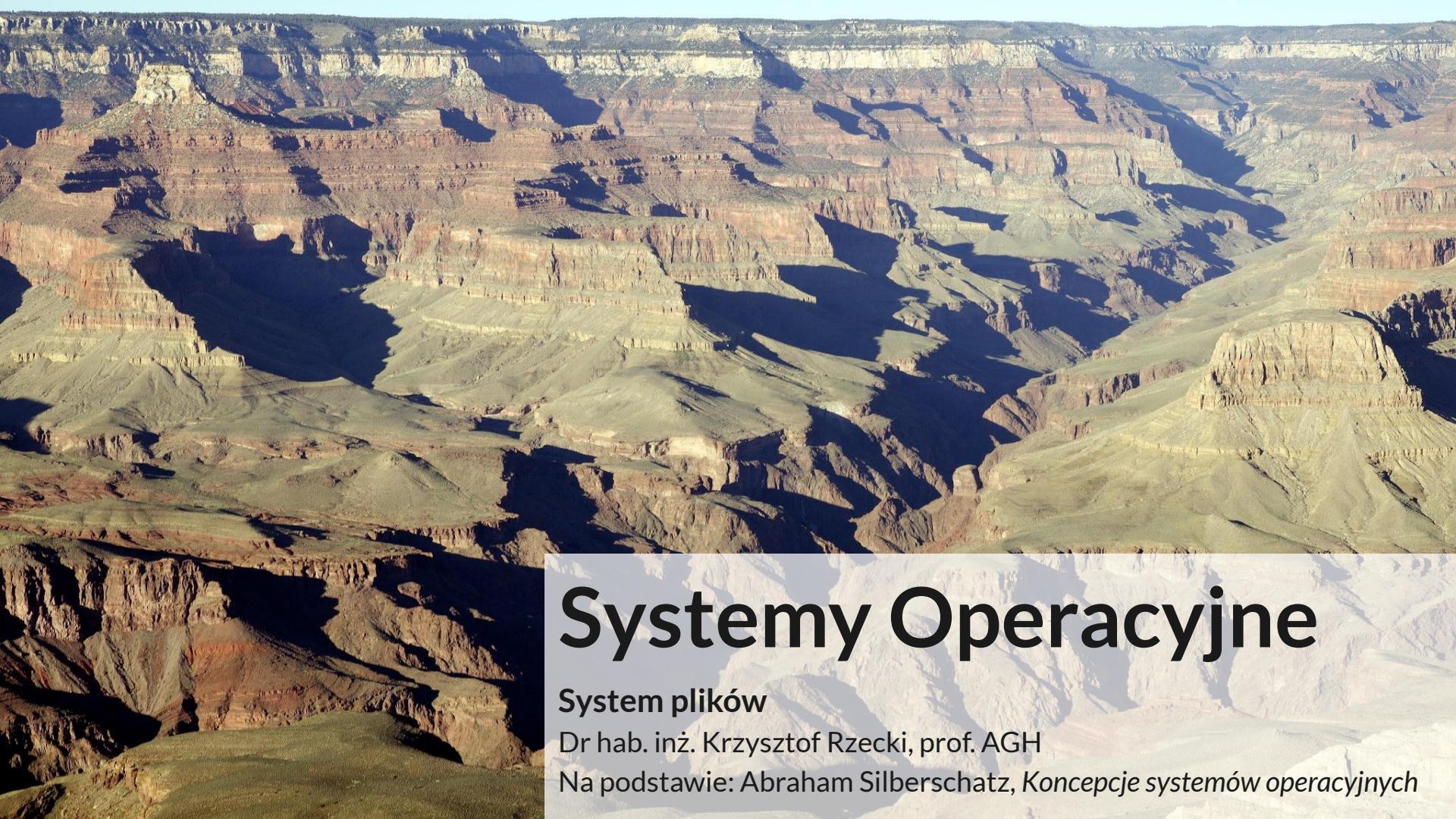
- iotop + watch
- iostat + watch

Przepływ danych:

- dd if=/dev/{zero,random}
- cp
- sync
- pidstat -p `pidof ...`



Madahora



Systemy Operacyjne

System plików

Dr hab. inż. Krzysztof Rzecki, prof. AGH

Na podstawie: Abraham Silberschatz, *Koncepcje systemów operacyjnych*

System plików - wprowadzenie

- System plików umożliwia na dostęp do przechowywanych:
 - programów i danych,
 - dla systemu operacyjnego i użytkowników.
- System plików składa się z dwóch części:
 - kolekcji plików do bezpośredniego przechowywania danych,
 - struktury katalogów, która organizuje pliki i przechowuje informacje o nich.
- Zwykle system plików znajduje się na urządzeniach z pamięcią nieulotną, ale w szczególnych przypadkach może to być np. pamięć RAM.

Koncepcja pliku

- Komputer może przechowywać informacje na różnych nośnikach: dysk twardy, taśma magnetyczna, dysk optyczny, pamięć USB, urządzenie NVM.
- System operacyjny dostarcza mechanizm nadający logiczny widok na przechowywane informacje - tworzy abstrakt między właściwościami fizycznymi urządzenia do przechowywania, a logiczną strukturą pliku. W ten sposób pliki są mapowane przez system operacyjny na urządzenie fizyczne.
- Dla użytkownika plik to kolekcja informacji zapisana w pamięci zapasowej, która nie może być zapisana dopóki nie jest zawarta do postaci pliku.
- Pliki zwykle reprezentują programy (zarówno źródłowe, jak i binarne) oraz dane.
- Pliki mogą być w formie luźnej, jak pliki tekstowe, albo ściśle ustrukturyzowane (źródła, binaria).

Atrybuty plików

- Nazwa pliku (kwestia wielkości liter, kwestia rozszerzenia).
- Plik jest bytem niezależnym od procesu, użytkownika, a nawet systemu operacyjnego plików.
- Atrybuty plików zależą od systemu plików, zwykle są to:
 - Nazwa - symboliczna nazwa pliku, jedyna informacja o pliku w zrozumiałym dla człowieka zapisie.
 - Identyfikator - unikalne oznaczenie, zwykle liczba, która identyfikuje plik w systemie plików.
 - Typ - rozróżnienie pliku.
 - Położenie - informacja o położeniu pliku na dysku.
 - Rozmiar - bieżący rozmiar pliku (w bajtach, słowach lub blokach).
 - Zabezpieczenia - informacje o kontroli dostępu dotyczące czytania, pisania i uruchamiania (itp.).
 - Znacznik czasu - informacje o znacznikach czasu mogą obejmować utworzenie, ostatnią modyfikację oraz ostatnie użycie pliku.
 - Identyfikator użytkownika i grupy - informacje dot. właściciela pliku.
- Informacje o plikach przechowywane są w strukturze katalogów (w pliku katalogu).

Operacje na plikach

- Tworzenie pliku - dwa kroki: rezerwacja miejsca, wpis w katalogu.
- Otwieranie - sprawdzenie uprawnień, itp. i wywołanie `open()`.
- Zapis - wywołanie systemowe z danymi do zapisu oraz `open()`. Zapis odbywa się sekwencyjnie, więc system musi przechowywać informację o wskaźniku miejsca do zapisu. Przypadek nadpisywania i dopisywania.
- Czytanie - wywołanie systemowe z wskaźnikiem do pliku oraz informacją, gdzie do pamięci zawartość pliku ma zostać wczytana. Podobnie, jak przy zapisie, system potrzebuje wskaźnika odczytu (i zwykle jest to ten sam wskaźnik).
- Pozycjonowanie w pliku - wskaźnik położenia w pliku, ang. `seek`.
- Usuwanie pliku - znalezienie pliku w katalogu, zwolnienie miejsca, oznaczenie w katalogu (wymazanie wpisu). Niektóre systemy plików: kwestia twardych dowiązań.
- Obcinanie pliku - np. usuwanie zawartości, ale zachowanie atrybutów.

Typy plików

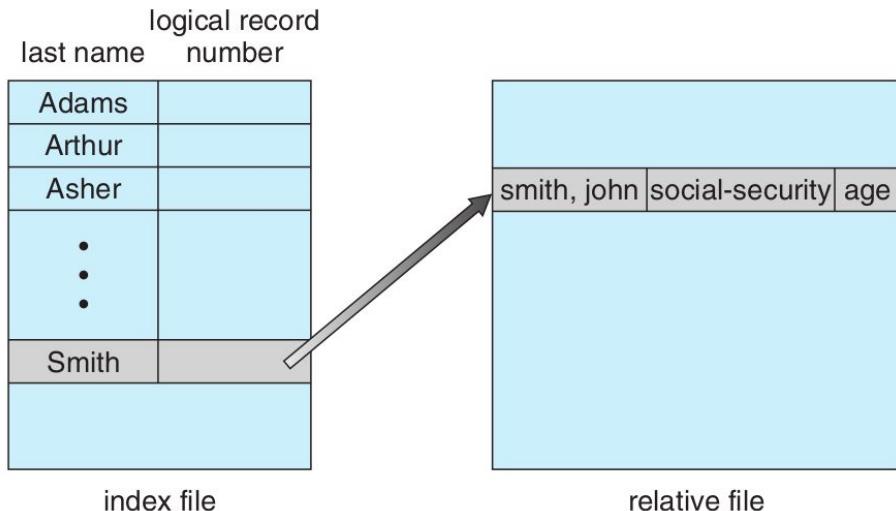
- Typ pliku rozpoznawany po rozszerzeniu nie jest tym samym, co typ pliku rozpoznawany w atrybutach pliku.

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

Struktura katalogu

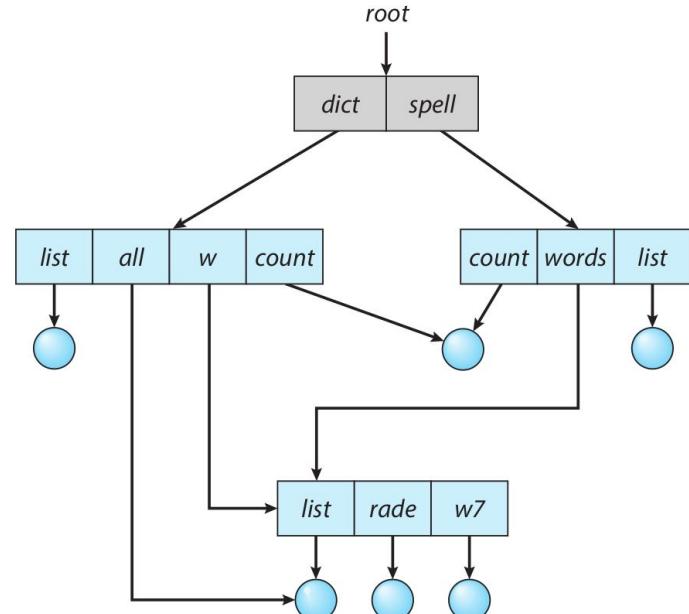
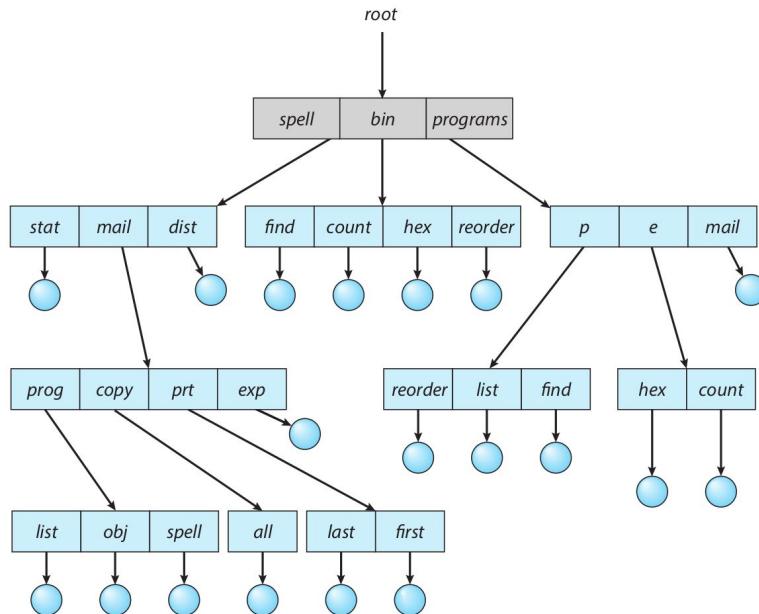
Operacje:

- Szukanie pliku.
- Tworzenie pliku.
- Usuwanie pliku.
- Listowanie zawartości katalogu.
- Zmiana nazwy pliku.
- Trawers systemu plików.



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Zaawansowane struktury katalogów



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Rodzaje dostępu

Podstawowe:

- **Read** - odczyt.
- **Write** - zapis.
- **Execute** - wykonywanie.
- **Append** - dopisywanie na końcu.
- **Delete** - usuwanie.
- **List** - listowanie nazwy i atrybutów pliku.
- **Attribute change** - zmiana wartości atrybutów.

A także:

- **Renaming** - zmiana nazwy.
- **Copying** - kopiowanie.
- **Editing** - edycja.

Uprawnienia do plików w Linux ext

```
krz@zinc:~/abc$ ls -al
razem 44
drwxrwxr-x  2 krz krz  4096 paź  9 18:40 .
drwx----- 98 krz krz 28672 paź  9 18:40 ..
-rw-rw-r--  1 krz krz     0 paź  9 18:40 file.txt
krz@zinc:~/abc$ 
```

- Katalog bieżący: ~/abc oraz .
- Katalog nadrzędny: ..

Ustawianie uprawnień:

```
$ chmod uprawnienia plik
$ chmod 644 file.txt
$ chmod a+rwx,a-w directory
```

Uprawnienia, przykład:

```
drwxr-x---
```

0123456789

Pozycja 0: d (dir), l (link), b (block), c (character)

Pozycja 1, 2 i 3: uprawnienia właściciela 'u'

Pozycja 4, 5 i 6: uprawnienia grupy 'g'

Pozycja 7, 8 i 9: uprawnienia pozostałych 'o'

Pozycje 1..9: uprawnienia wszystkich 'a'

rwx - read, write, eXecute

421 - zapis numeryczny

np. r-x = 5, rw- = 6, r-- = 4.

Uprawnienia do plików w Linux ext

0	---	brak uprawnień	blokada
1	--x	wykonywanie	nieprzydatne
2	-w-	zapis	zbieranie sekretnych logów (plik)
3	-wx	zapis i wykonywanie	zbieranie sekretnych logów (dir)
4	r--	odczyt	stała konfiguracja
5	r-x	odczyt i uruchamianie	pliki wykonywalne, katalogi
6	rw-	odczyt i zapis	pliki edytowalne
7	rwx	odczyt, zapis i uruchamianie	skrypty i katalogi usera :-(
	??s	bit suid	programy specjalne
	??t	sticky bit	katalog specjalny

Struktura systemu plików

Założenia:

1. Dysk może być nadpisywany w miejscu; możliwy jest odczyt bloku, modyfikacja bloku zapis bloku w to samo miejsce.
2. Możliwe jest sięgnięcie w dowolny blok (sektorów) na dysku, czyli dostęp do pliku może być realizowany sekwencyjnie lub swobodnie.

Transfer danych z/na dysk zorganizowany jest w bloki wielkości zwykle 512 lub 4096 bajtów.

application programs



logical file system



file-organization module



basic file system



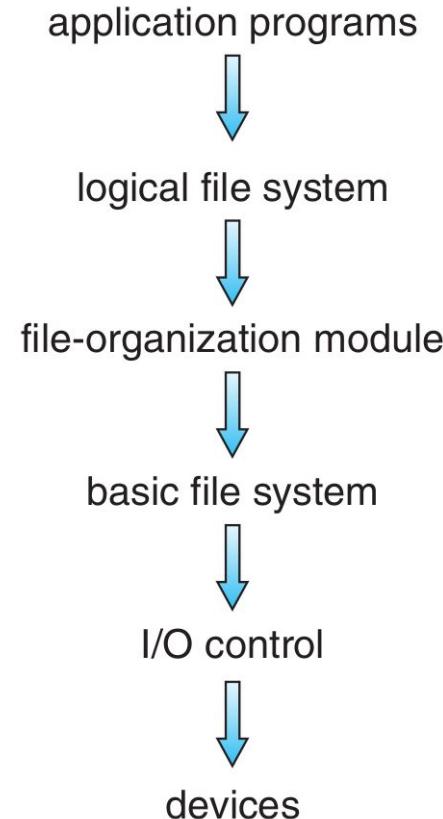
I/O control



devices

Struktura systemu plików

- I/O - sterownik urządzenia oraz przerwań do transferu danych.
- Podstawowy system plików - blokowy subsystem I/O w Linux.
- Moduł organizacji plików - posiada wiedzę na temat plików oraz logicznych bloków, a także wolnego miejsca.
- Logiczny system plików - zarządza metadanymi, strukturą katalogów, wykorzystuje FCB (file control block) w np. Linux poprzez inode.

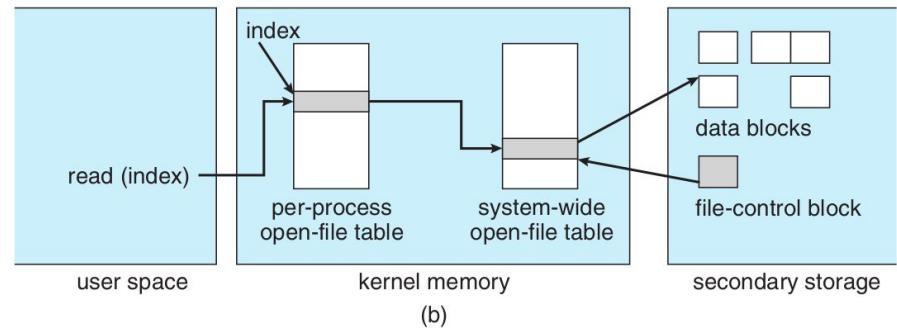
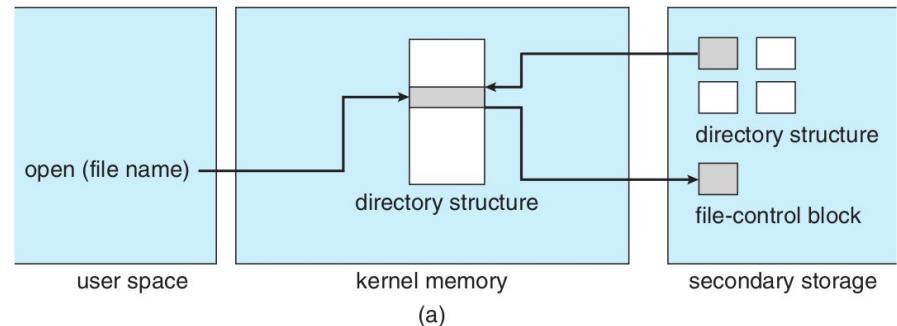


Operacje systemu plików

- Blok kontrolny ładowania (ang. *Boot control block*), Główny rekord rozruchowy (ang. *Master Boot Record*) - informacje zapisane w pierwszym sektorze dysku (CHS = 0,0,1), zajmuje jeden sektor, czyli 512 bajtów i zawiera (w przypadku Linux) program rozruchowy systemu operacyjnego (446 bajtów) oraz tablicę partycji.
Polecenie podglądu MBR: `hd -n 512 /dev/sda`
Polecenie podglądu partycji: `cat /proc/partitions`
- GPT, ang. *Globally Unique Partition Table* - nowszy typ rekordu ładowania.
- Blok kontrolny wolumenu (ang. *Volume control block*) - informacje o partycji/wolumenie, jak liczbę bloków w wolumenie, rozmiar bloków, liczba wolnych bloków, itp.
- Struktura katalogów zawierająca informacje o organizacji plików.
- Blok kontrolny pliku (ang. *File Control Block*) - zawiera wiele informacji o pliku. Posiada unikalny identyfikator.

Otwarcie/odczyt pliku

- (a) Otwarcie pliku
- (b) Odczyt pliku



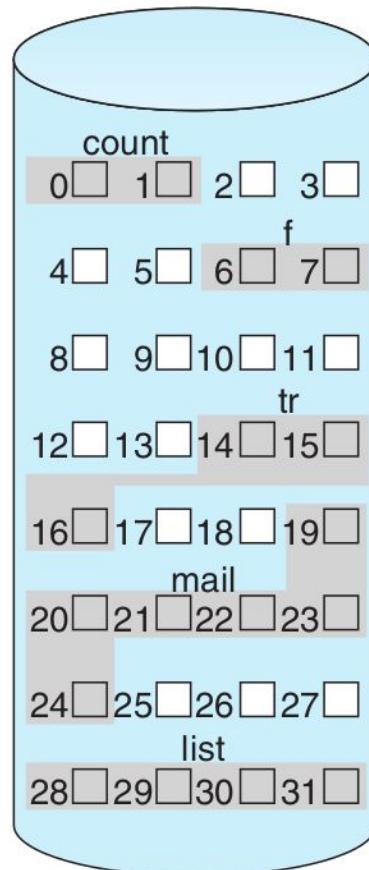
Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Metody alokacji plików

Na kolejnych slajdach:

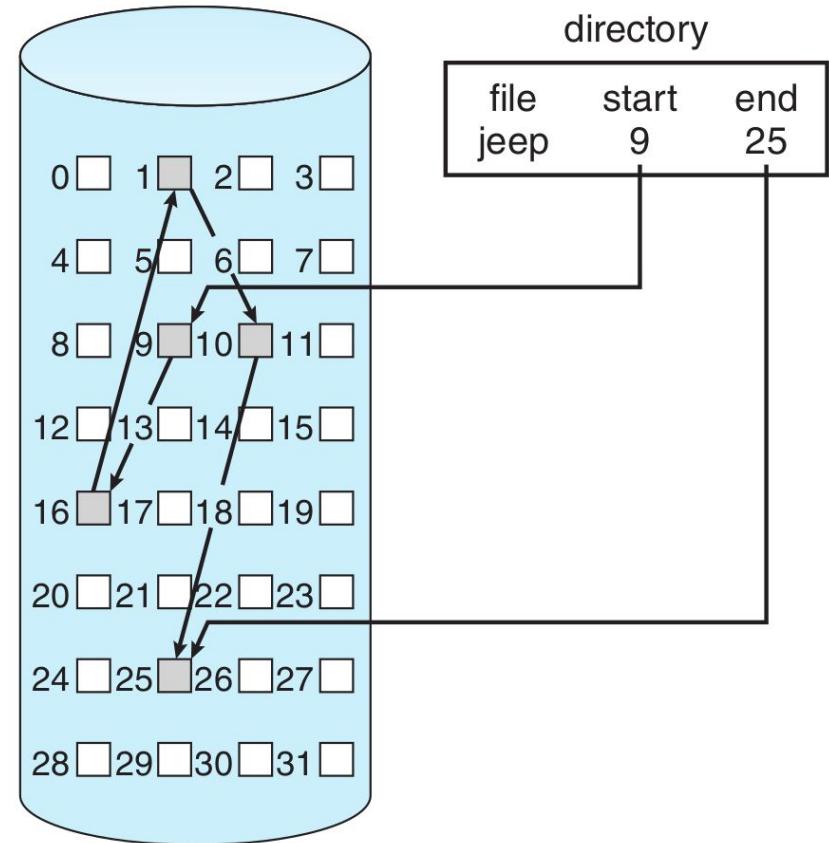
- Alokacja ciągła - ang. *Contiguous Allocation*
- Alokacja połączona - ang. *Linked Allocation*
- Alokacja indeksowana - ang. *Indexed Allocation*

Alokacja ciągła

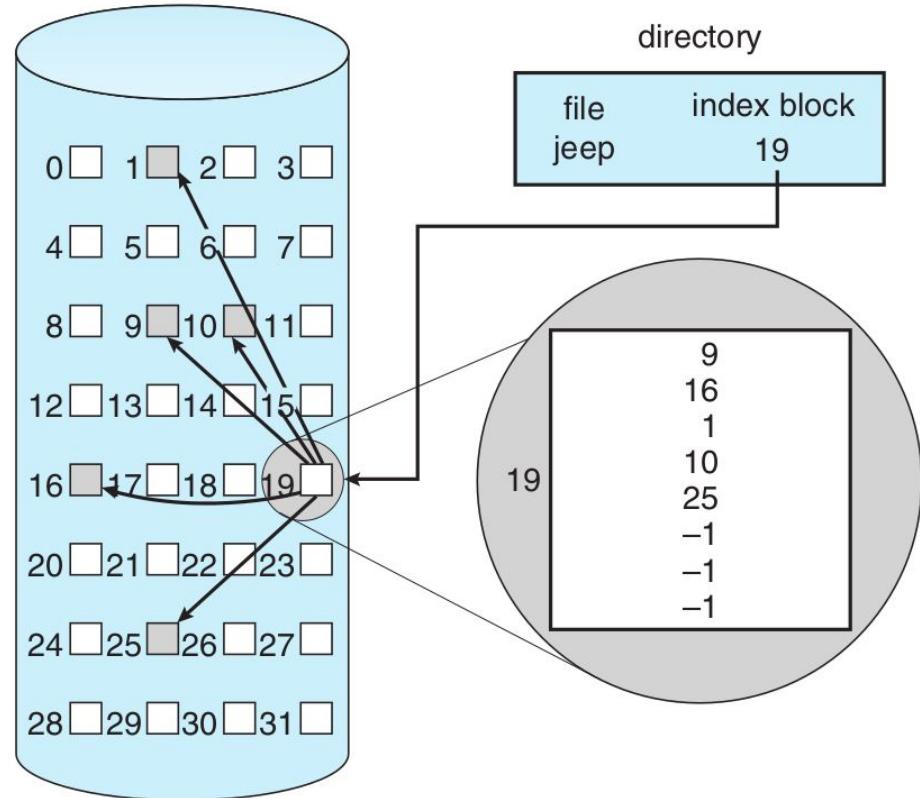


directory		
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Alokacja połączona



Alokacja indeksowana



Tworzenie pliku na urządzenie blokowe

- Utworzenie pustego pliku (100 MB), z którego zrobimy urządzenie blokowe:

```
$ dd if=/dev/zero of=disc0 bs=1M count=100
```

```
$ file -s disc0
```

```
disc0: data
```

```
$ ls -sh disc0
```

```
101M disc0
```

Tworzenie urządzenia blokowego z pliku

- Znajdź pierwsze wolne urządzenie blokowe:

```
$ losetup -f
```

np.:

```
/dev/loop12
```

```
$ sudo file -s `losetup -f`
```

```
/dev/loop12: empty
```

- Utworzenie z pliku urządzenia blokowego:

```
$ losetup `losetup -f` disc0
```

```
$ file -s /dev/loop12
```

```
/dev/loop12: data
```

Zakładanie partycji na urządzeniu blokowym

- Podział tak utworzonego urządzenia na 2 partycje po 50 MB:

```
$ sudo fdisk /dev/loop12
n, [Enter], p, 3 x [Enter], +50M, Enter
n + [Enter], p, 4 x [Enter], p, [Enter], w, [Enter]
```

- Synchronizacja z jądrem systemu operacyjnego:

```
$ sudo partprobe /dev/loop12
```

```
$ sudo file -s /dev/loop12
/dev/loop12: DOS/MBR boot sector; partition 1 : ID=0x83, start-CHS
(0x0,32,33), end-CHS (0x6,127,57), startsector 2048, 102400 sectors;
partition 2 : ID=0x83, start-CHS (0x6,127,58), end-CHS (0xc,190,50),
startsector 104448, 100352 sectors
```

```
$ sudo file -s /dev/loop12p1
/dev/loop12p1: data
```

Formatowanie (zakładanie systemu plików)

- Utworzenie systemu plików na pierwszej z partycji:

```
$ mkfs.ext4 /dev/loop12p1
```

```
$ file -s /dev/loop12p1
/dev/loop0p1: Linux rev 1.0 ext4 filesystem data,
UUID=c22af2f0-449d-4119-a117-45edfa32e287 (extents) (64bit)
(large files) (huge files)
```

- Narzędzia do serwisowania systemu plików:

- \$ e2fsck - sprawdzanie i/lub naprawa systemu plików
- \$ fsck.ext4 - alias do j.w.
- \$ tune2fs - zmiana konfiguracji systemu plików
- \$ resize2fs - zmiana wielkości systemu plików
- \$ debugfs - debugging, w tym przeglądanie i edycja systemu plików

Zarządzanie systemem plików

- Podłączenie pliku jako urządzenia blokowego:

```
$ losetup /dev/loop12 disc0 // już jest z poprzednich slajdów
```

- Podmontowanie systemu plików z pierwszej partycji:

```
$ mkdir mntdir
```

```
$ sudo mount /dev/loop12p1 mntdir
```

```
$ df -h mntdir
```

```
/dev/loop12p1      43M    24K    40M    1% /home/krz/mntdir
```

- Odmontowanie systemu plików:

```
$ sudo umount mntdir
```

- Odłączanie urządzenia loopback

```
$ losetup -d /dev/loop12
```

Szyfrowanie partycji urządzenia blokowego

- Podłączenie pliku jako urządzenia blokowego (jak poprzednio), czyli :

```
$ losetup `losetup -f` disc0 // założmy, że to znów loop12
```

- Zaszyfrowanie urządzenia blokowego (druga z partycji):

```
$ sudo cryptsetup luksFormat /dev/loop12p2
```

```
$ sudo file -s /dev/loop12p2
```

```
/dev/loop12p2: LUKS encrypted file, ver 2 [, sha256] UUID:
```

```
5bbf82ca-613b-4a32-8c1d-b18b2ad6f520
```

- Podłączenie zaszyfrowanej partycji, jako urządzenie blokowe:

```
$ sudo cryptsetup luksOpen /dev/loop12p2 encrypted_partition
```

```
$ ls -l /dev/mapper/encrypted_partition
```

```
/dev/mapper/encrypted_partition -> ../../dm-0
```

Formatowanie zaszyfrowanej partycji

- Podłączenie pliku jako urządzenia blokowego (jak poprzednio), czyli :

```
$ losetup `losetup -f` disc0 // założmy, że to znów loop12
```

Podłączenie zaszyfrowanej partycji jako urządzenie blokowe (jak poprzednio):

```
$ sudo cryptsetup luksOpen /dev/loop12p2 encrypted_partition
```

- Utworzenie systemu plików na zaszyfrowanej partycji:

```
$ mkfs.ext4 /dev/mapper/encrypted_partition
```

- Montowanie zaszyfrowanej partycji:

```
$ mkdir mntdir_enc
```

```
$ sudo mount /dev/mapper/encrypted_partition mntdir_enc
```

```
$ df -h mntdir
```

/dev/mapper/encrypted_partition	27M	24K	25M	1%	/home/krz/mntdir_enc
---------------------------------	-----	-----	-----	----	----------------------

Odłączanie zaszyfrowanej partycji

- Odmontowanie:
`$ sudo umount mntdir_enc`
- Odłączanie zmapowania kryptograficznego:
`$ sudo cryptsetup luksClose encrypted_partition`
- Odłączanie urządzenia blokowego:
`$ losetup -d /dev/loop12`

Zadanie dodatkowe

- Zaprojektować i napisać program, który we wskazanych miejscach urządzenia blokowego odczyta zawartość bloków, zabezpieczy i nadpisze celem utworzenia sztucznego uszkodzenia.
- Następnie, oprogramowaniem do naprawy systemu plików naprawić uszkodzenie.
- Porównać systemy plików pod względem odporności na uszkodzenia.
- Opracować wynik w formie sprawozdania.



Grand Canyon



Systemy Operacyjne

Gniazda i komunikacja sieciowa

Dr hab. inż. Krzysztof Rzecki, prof. AGH

Komunikacja między procesami

- System komputerowy: IPC (ang. *Interprocess Communication*):
 - Standardy: POSIX, System V
 - Rodzaje: pipe, kolejki, pamięć wspólna, semafory, rygle, etc.
- Sieci komputerowe:
 - Warstwa łącza: **Ethernet**, SLIP, PPP
 - Protokoły sieciowe: **IPv4**, IPv6, ICMP
 - Protokoły transportowe: **TCP**, **UDP**
 - Protokoły aplikacji: DNS, HTTP, SSH, etc.

Programowanie usług sieciowych

... czyli aplikacji:

- wieloprocesowych,
- komunikujących się przez interfejs sieciowy,
- tworzących formę (architekturę) klient-serwer.

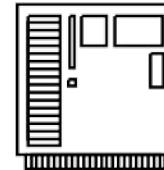
Zapotrzebowanie



Komputer
użytkownika
klient

Zasadniczo słabo wyposażony komputer
w własnym systemem operacyjnym
oprzygodowaniem komunikacyjnym.

Pojęcie: cienki klient (ang. thin client)



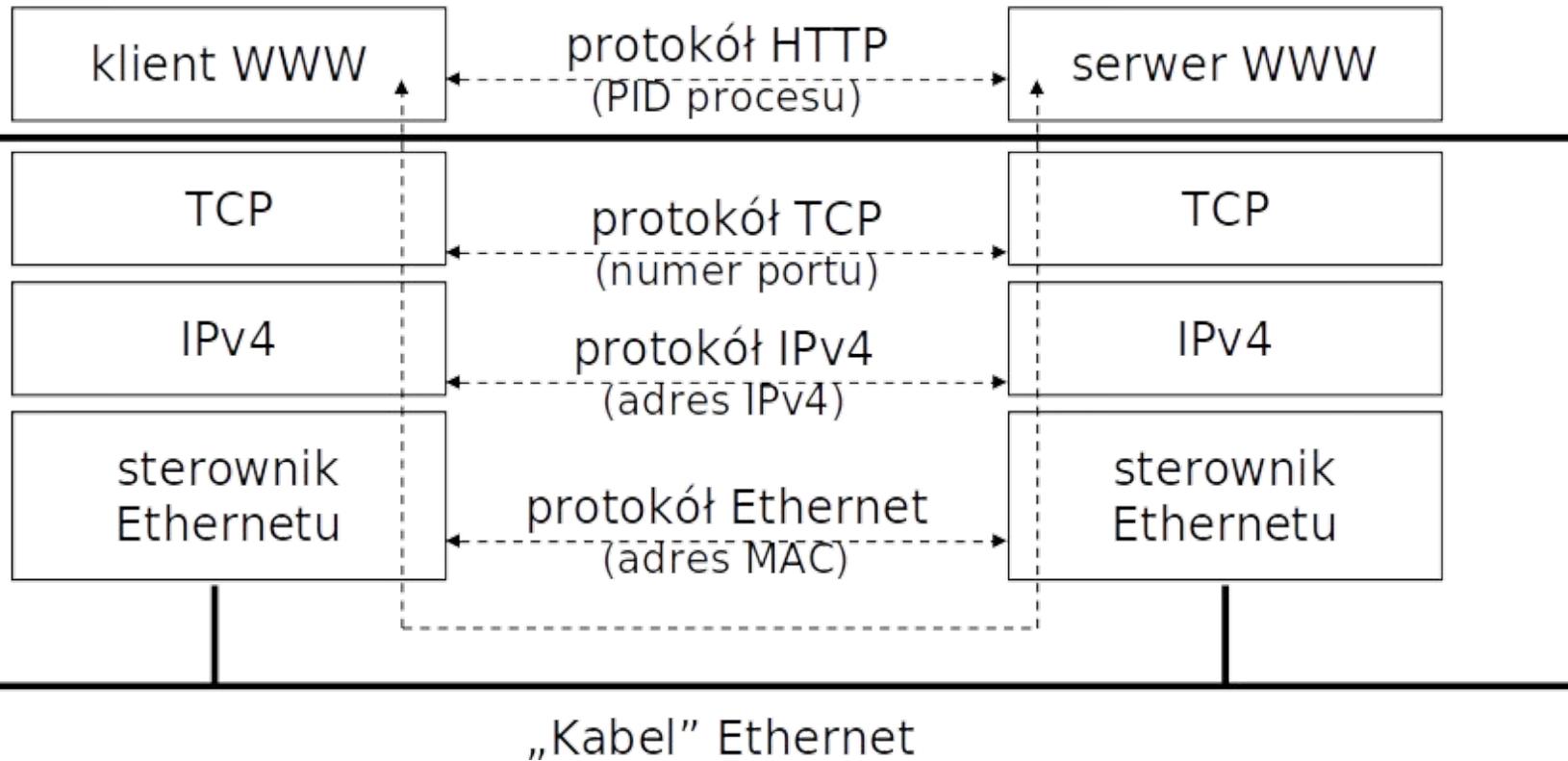
Komputer
zasobami
serwer

Zasoby (ang. resources):

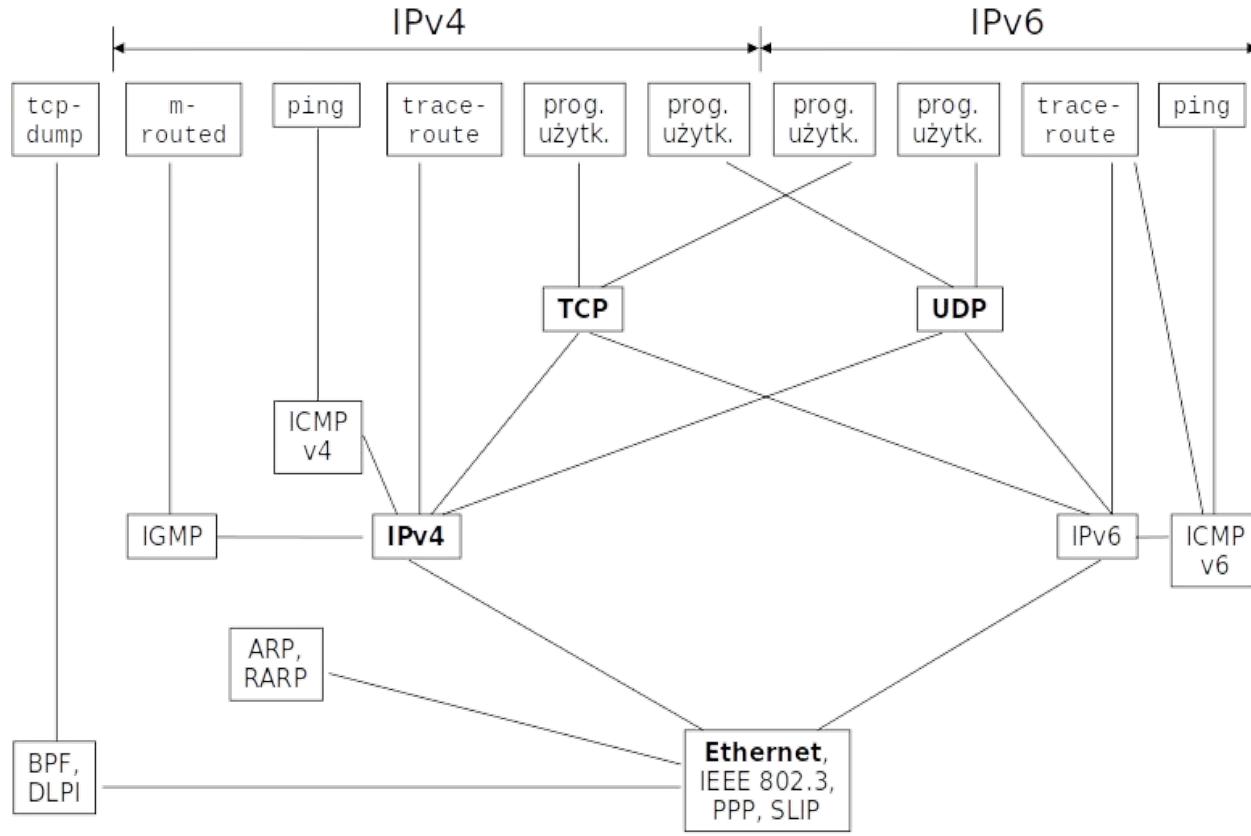
- przestrzeń dyskowa:
 - pobieranie danych,
 - przechowywanie danych,
- czas procesora,
- pamięć operacyjna,
- dostęp do innych urządzeń,
- łącze / brama (ang. gateway),

Strona kliencka

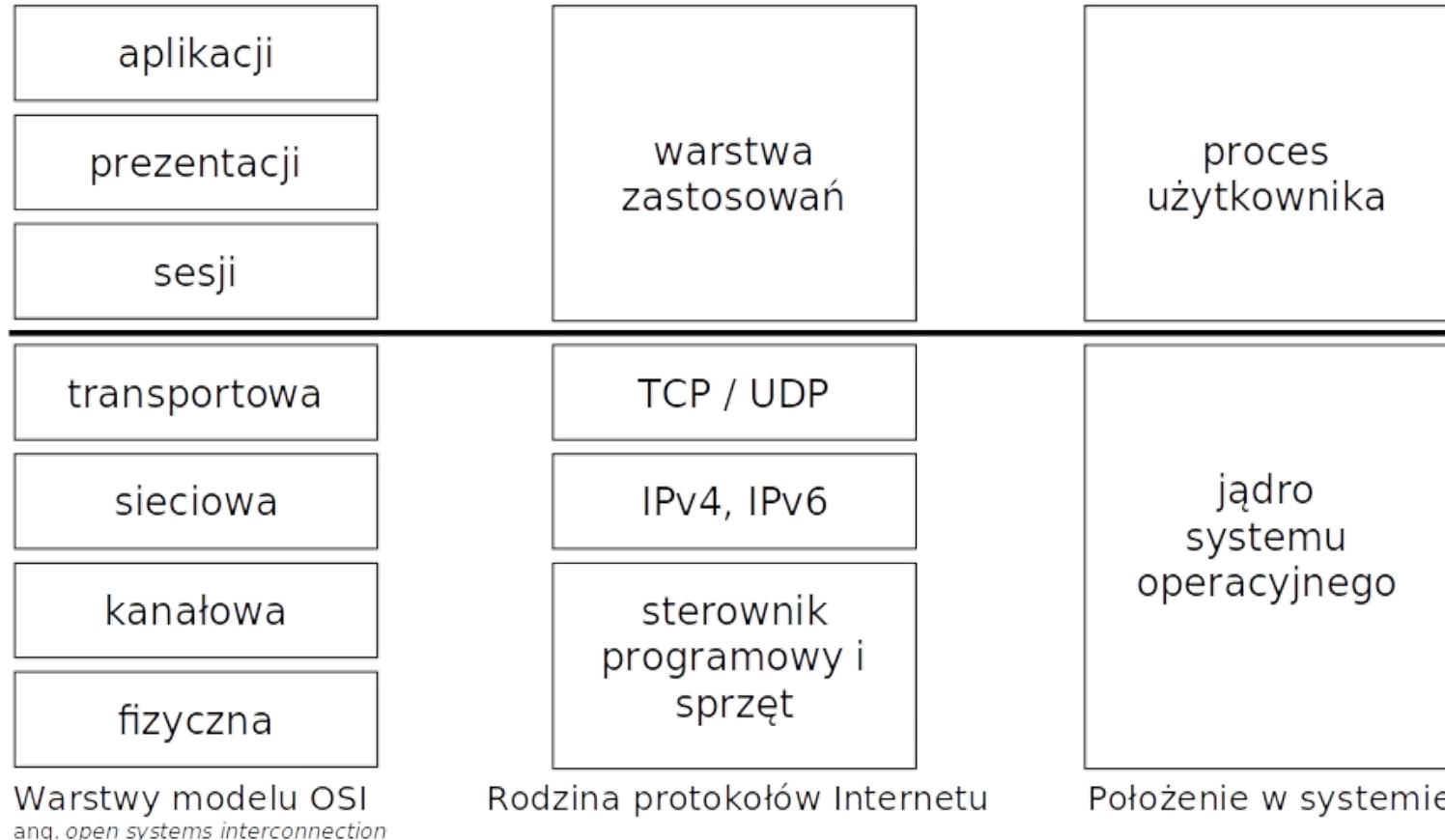
Strona serwera



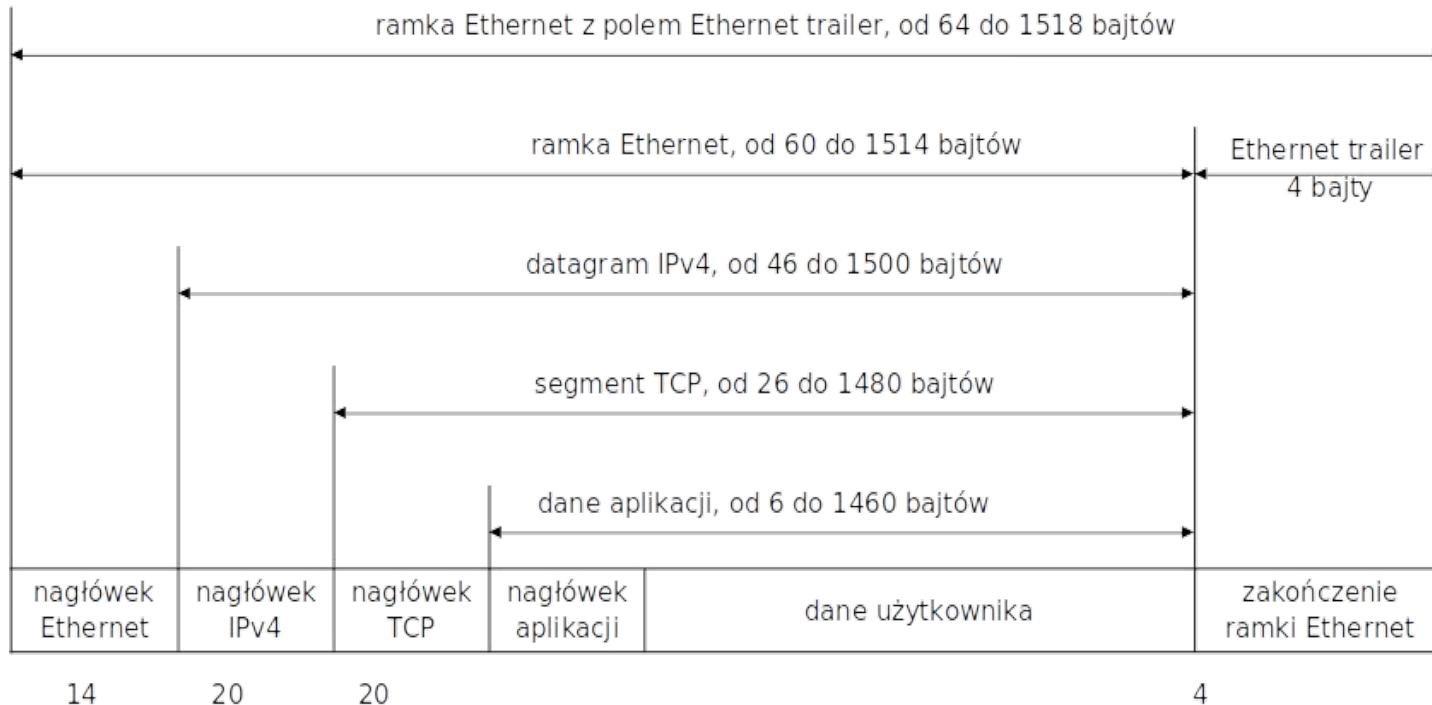
Przegląd protokołów



Model i rodzina

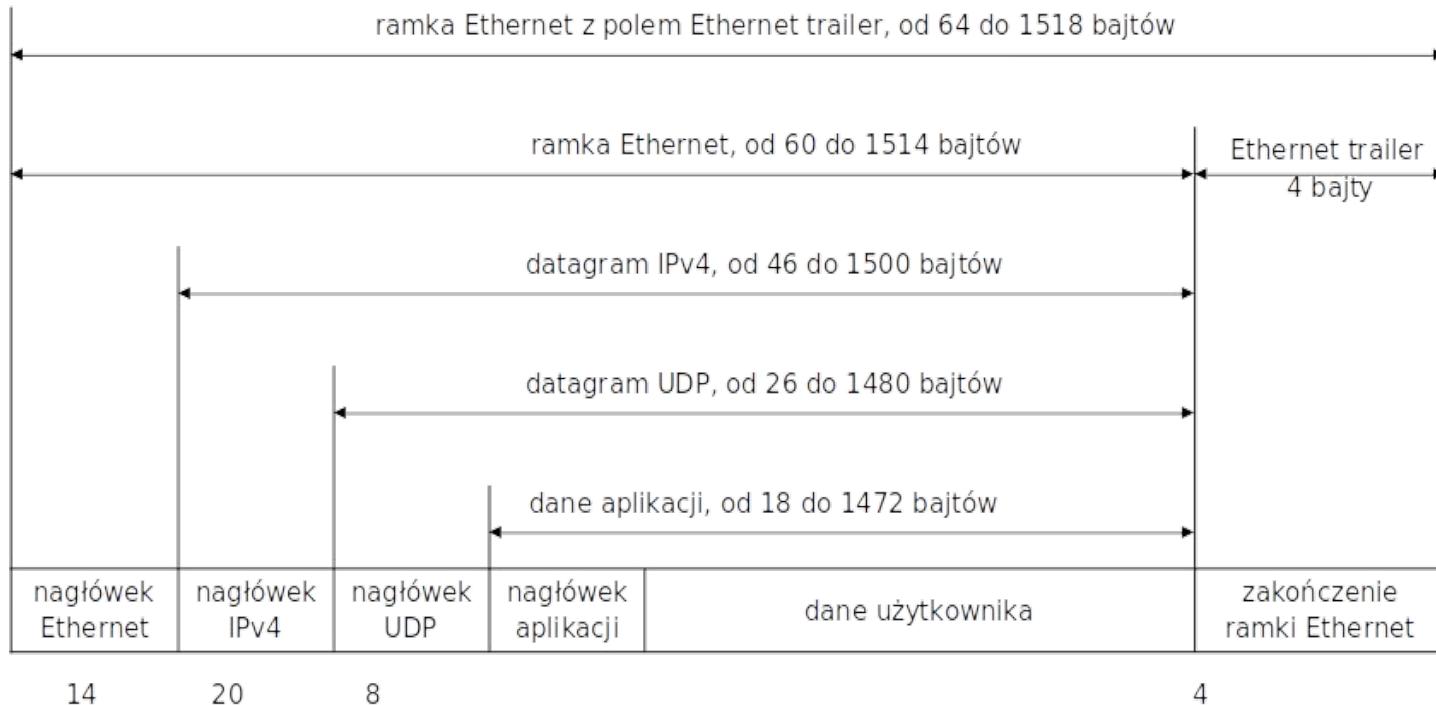


Enkapsulacja TCP/IPv4/Ethernet



Wprowadzone ograniczenia wywodzą się z MTU
TCP strumieniuje - nie ma ograniczeń

Enkapsulacja UDP/IPv4/Ethernet



Wprowadzone ograniczenia wywodzą się z MTU
UDP ogranicza host i pole określające wielkość

Internet Protocol i adres IPv4

wersja 4 bity	długość nagłówka 4 bity	typ usługi (TOS) 8 bitów	długość całkowita 16 bitów (zapisana w bajtach)			
identyfikacja 16 bitów		znaczniki 3 bity	usunięcie fragmentacji 13 bitów			
czas życia (TTL) 8 bitów	protokół 8 bitów		suma kontrolna nagłówka 16 bitów			
adres źródłowy 32 bity						
adres docelowy 32 bity						
opcje (jeśli są)						
dane						



Port

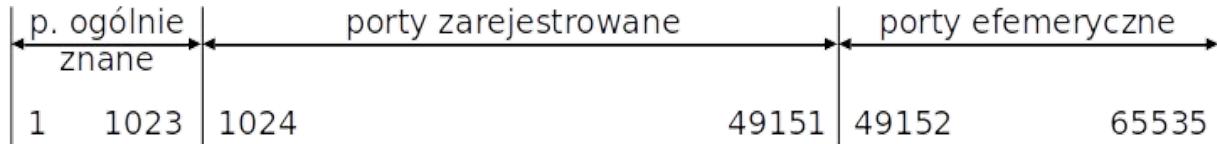
Port to 16-bitowa liczba całkowita z zakresu 1...65535.

Trzy grupy (IANA):

- strona serwera: porty ogólnie znane (ang. well-known port),
- strona serwera: porty zarezerwowane (ang. reserved port),
- strona klienta: porty efemeryczne (ang. ephemeral port).

Port

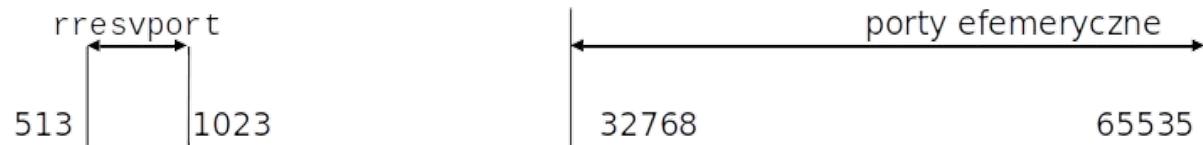
- Konwencje **przyjęte** przez IANA
(ang. *Internet Assigned Numbers Authority*)



- Konwencje BSD



- Konwencje Solaris



Protokół użytkowy

- Protokół użytkowy to format komunikacji między procesami umieszczony najwyżej w hierarchii rodziny protokołów Internetu.
- Protokół użytkowy obejmuje warstwę sesji, prezentacji oraz zastosowań.
- Przykłady:
 - HTTP (ang. Hyper Text Transfer Protocol),
 - FTP (ang. File Transfer Protocol),
 - SSH (ang. Secure SHell).
- Skojarzenie port <-> protokół użytkowy nie musi być jednoznaczne!

Podsumowanie wprowadzenia

- Protokół Ethernet i adres MAC -> bezpośrednie przekazywanie danych.
- Protokół IP oraz adres IPv4 (IPv6) -> przekazywanie danych poza sieć lokalną (ang. routing).
- Protokół TCP/UDP -> kontrola przesyłanych danych między aplikacjami.
- Rola systemu operacyjnego to przekazywanie:
 - danych od aplikacji do wysłania,
 - dane otrzymanych dla aplikacji.
- System operacyjny utrzymuje skojarzenie: numer portu i PID procesu
- Protokół użytkowy: format komunikacji między aplikacjami.

Standardy systemu Unix

- IEEE (ang. Institute for Electrical and Electronics Engineers, Inc.), której standardy są normami ISO/IEC:
 - Posix – Przenośny Interfejs Systemu Operacyjnego (ang. Portable Operating System Interface) => Posix.1g,
 - ISO (ang. International Organization for Standardization),
 - IEC (ang. International Electrotechnical Commission),
- Open Group:
 - powstała z X/Open Company oraz OSF (ang. Open Software Foundation),
 - X/Open Portability Guide,
 - Single Unix Specification,
 - CDE – Common Desktop Environment,
- IETF (ang. Internet Engineering Task Force):
 - dokumenty RFC,
 - dokumenty Internet-Drafts.

Przykłady standardów RFC

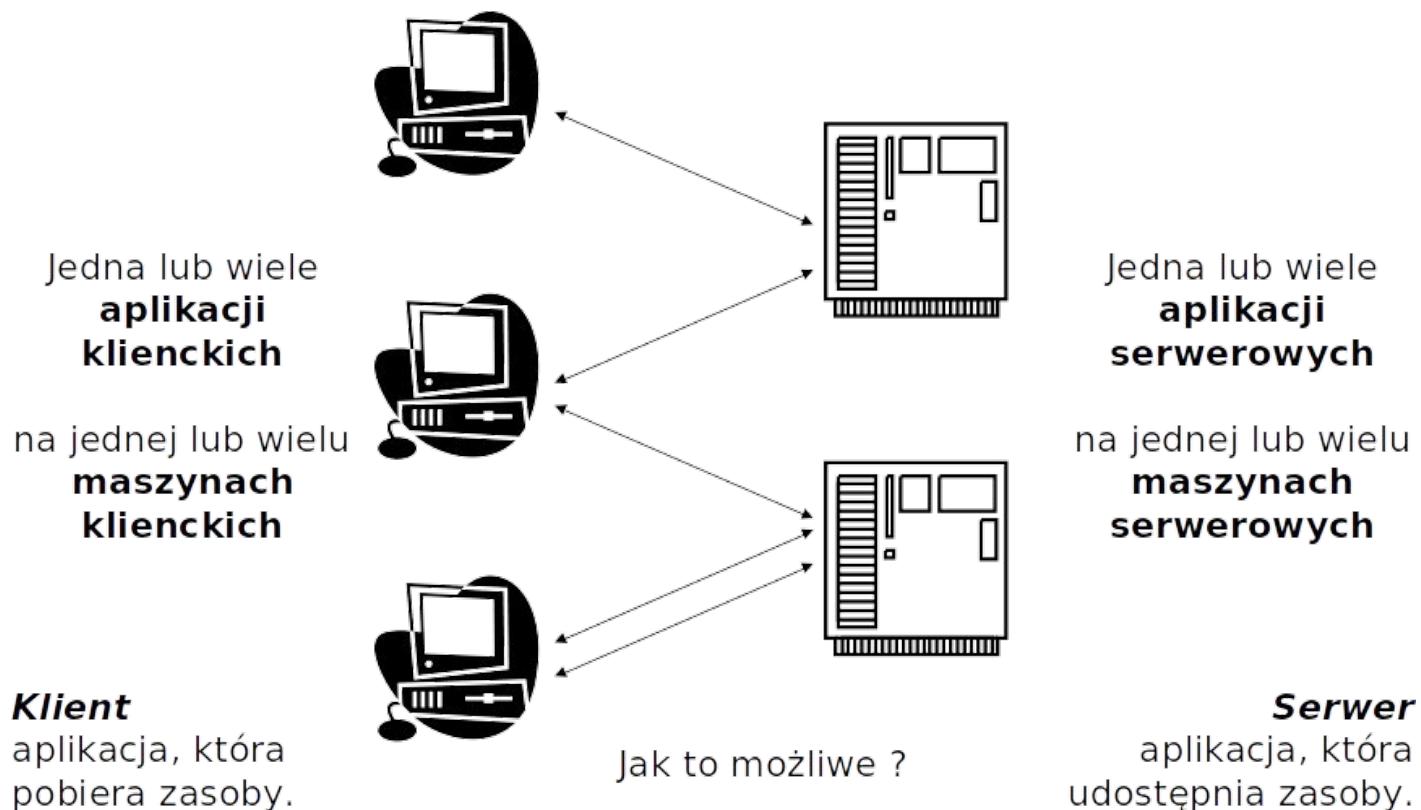
- Adres e-mail i format wiadomości:
 - RFC 822 Standard for the Format of ARPA Internet Text Messages, 1982r.
 - Rozszerzenia i aktualizacje: RFC 2822, RFC 4021, RFC 5280, RFC 5322, RFC 6854, etc.
- DNS:
 - RFC 882: DOMAIN NAMES - CONCEPTS and FACILITIES, 1983r.
 - RFC 883: DOMAIN NAMES - IMPLEMENTATION and SPECIFICATION, 1983r.
 - RFC 2929: Domain Name System (DNS) IANA Considerations, 2000r.
- IRC: RFC 1459: Internet Relay Chat Protocol, 1993r.
- Sieci lokalne: RFC 1918: Address Allocation for Private Internets
- VoIP: RFC 2543: SIP: Session Initiation Protocol
- HTTP: RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content
- Video: RFC 8216: HTTP Live Streaming

Przykłady standardów RFC

Badania:

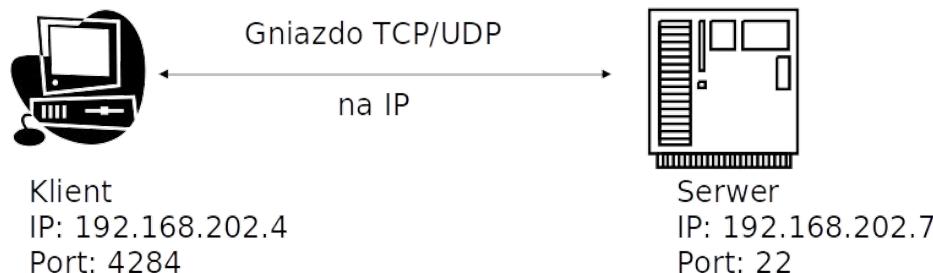
- RFC 5106: The Extensible Authentication Protocol-Internet Key Exchange Protocol version 2 (EAP-IKEv2) Method
- RFC 5222: LoST: A Location-to-Service Translation Protocol

Klient - Serwer



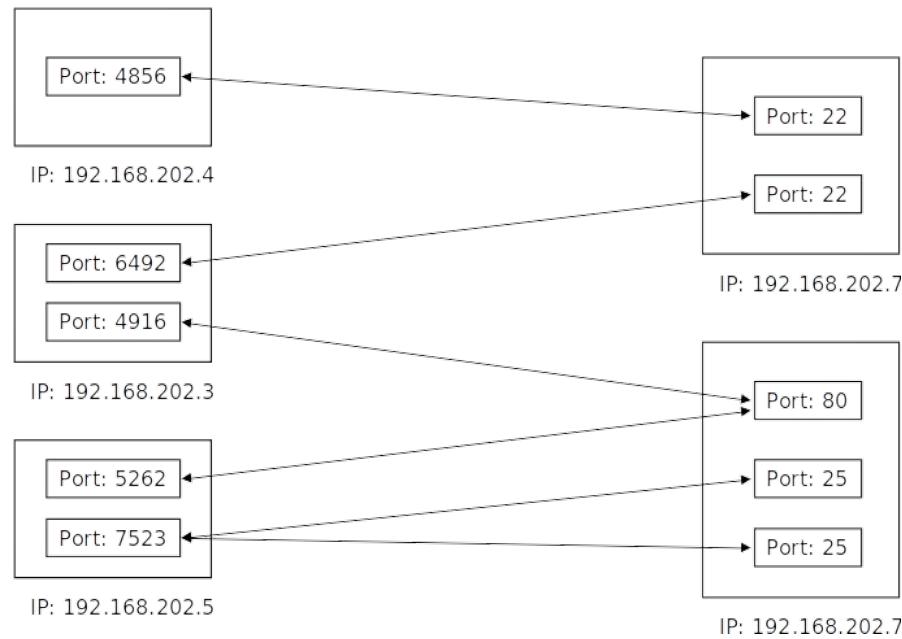
Gniazdo (ang. *socket*)

- **Gniazdo** to unikalna struktura dwóch par liczb (czyli czterech liczb):
 - adres IP i port klienta,
 - adres IP i port serwera.

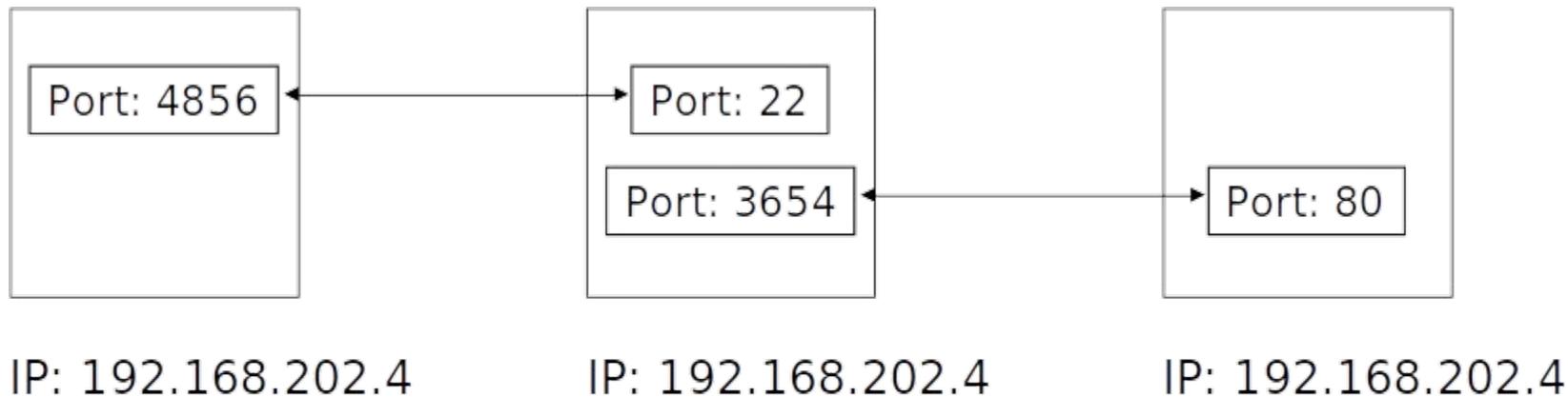


Klient / Serwer to aplikacje czy komputery ?

Architektura klient-serwer



Serwer to czy klient ?



?

Ogólna gniazdowa struktura adresowa

<sys/socket.h>

// plik nagłówkowy

```
struct sockaddr {           // uogólniona gniazdowa SA
    uint8_t          sin_len;   // długość struktury, 16 bajtów (?)
    sa_family_t      sin_family; // AF_XXX
    char             sa_data[14]; // adres właściwy dla protokołu
};
```

Gniazdowa SA dla IPv4

Gniazdowe struktury adresowe

IPv4	IPv6	Dziedzina Unix	Warstwa kanał.																										
<p>sockaddr_in</p> <table border="1"><tr><td>długość</td><td>AF_INET</td></tr><tr><td>16-bit numer portu</td><td></td></tr><tr><td>32-bitowy adres IPv4</td><td></td></tr><tr><td>nieużywane</td><td></td></tr></table> <p>16 bajtów</p>	długość	AF_INET	16-bit numer portu		32-bitowy adres IPv4		nieużywane		<p>sockaddr_in6</p> <table border="1"><tr><td>długość</td><td>AF_INET6</td></tr><tr><td>16-bit numer portu</td><td></td></tr><tr><td>32-bitowa etykieta przepływu</td><td></td></tr><tr><td>128-bitowy adres IPv6</td><td></td></tr></table> <p>24 bajty</p>	długość	AF_INET6	16-bit numer portu		32-bitowa etykieta przepływu		128-bitowy adres IPv6		<p>sockaddr_un</p> <table border="1"><tr><td>długość</td><td>AF_LOCAL</td></tr></table> <p>nazwa ścieżkowa (do 104 bajtów)</p>	długość	AF_LOCAL	<p>sockaddr_dl</p> <table border="1"><tr><td>długość</td><td>AF_LINK</td></tr><tr><td>indeks interfejsu</td><td></td></tr><tr><td>typ</td><td>dł. nazwy</td></tr><tr><td>dł. adr.</td><td>dł. sel.</td></tr></table> <p>nazwa interfejsu oraz adres warstwy kanałowej</p>	długość	AF_LINK	indeks interfejsu		typ	dł. nazwy	dł. adr.	dł. sel.
długość	AF_INET																												
16-bit numer portu																													
32-bitowy adres IPv4																													
nieużywane																													
długość	AF_INET6																												
16-bit numer portu																													
32-bitowa etykieta przepływu																													
128-bitowy adres IPv6																													
długość	AF_LOCAL																												
długość	AF_LINK																												
indeks interfejsu																													
typ	dł. nazwy																												
dł. adr.	dł. sel.																												

A jak wygląda ogólna gniazdowa struktura adresowa?

Operacje na SA

- Przesyłane parametry to co najmniej:
 - deskryptor gniazda (int),
 - dowolna gniazdowa struktura adresowa zrzutowana na ogólną gniazdową strukturę adresową (struct sockaddr *),
 - długość struktury (int lub int *).
- Sposób przekazywania parametrów z procesu do jądra:
`int function (int, struct sockaddr *, int);`
Przykład: bind, connect, sendto.
- Sposób pobierania parametrów z jądra do procesu^(*):
`int function (int, struct sockaddr *, int *);`
Przykład: accept, recvfrom, getsockname, getpeername

^(*) często dana funkcja przesyła parametry w obie strony

Kolejność bajtów

Jeśli zapiszemy daną liczbę w jednym modelu, a odczytamy tak, jakby to był drugi z modeli, wówczas otrzymamy liczbę o przestawionej kolejności bajtów (nie bitów!):

3232287249 <-> 298494144

192.168.202.17 <-> 17.202.168.192

5432 <-> 14357

Kolejność bajtów

- Systemowa kolejność bajtów
(ang. host byte order) to kolejność przechowywania bajtów w pamięci komputera:
 - Linux: little-endian,
 - HP-UX: big-endian.
- Sieciowa kolejność bajtów
(ang. network byte order) to kolejność przechowywania bajtów w protokołach sieciowych – tylko big-endian.

Funkcje konwertujące

Oznaczenia:

'n' = network oraz 'h' = host

's' = short oraz 'l' = long

Funkcje host -> network

```
uint16_t htons (uint16_t host16bitvalue);  
uint32_t htonl (uint32_t host32bitvalue);
```

Funkcje network -> host

```
uint16_t ntohs (uint16_t net16bitvalue);  
uint32_t ntohl (uint32_t net32bitvalue);
```



presentation <-> numeric

presentation (network) -> numeric (host)

```
int inet_aton (const char *strptr, struct in_addr *addrptr);
```

```
in_addr_t inet_addr (const char *strptr);
```

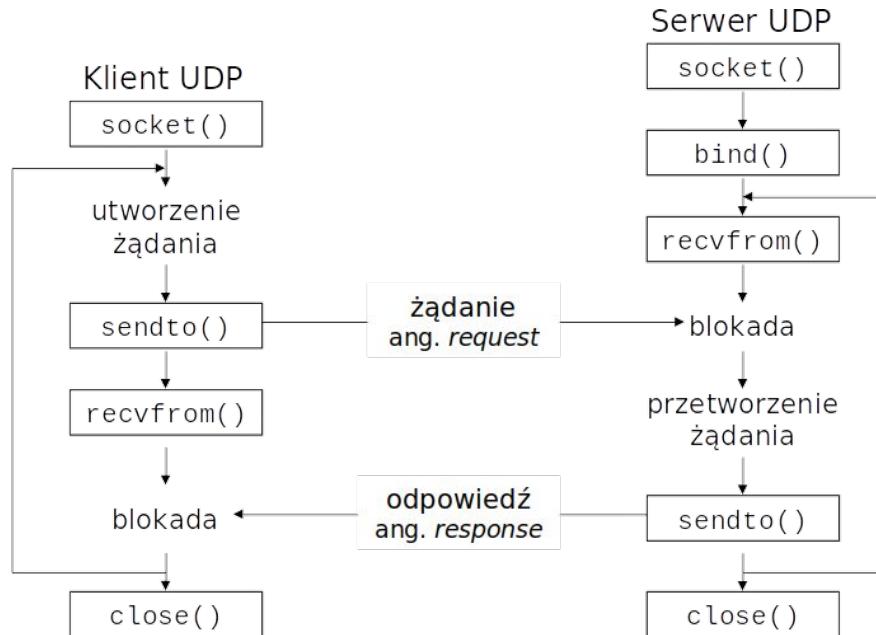
```
int inet_pton (int family, const char *strptr, void *addrptr);
```

numeric (host) -> presentation (network)

```
char *inet_ntoa (struct in_addr inaddr);
```

```
const char *inet_ntop (int family, const void *addrptr, char *strptr, size_t len);
```

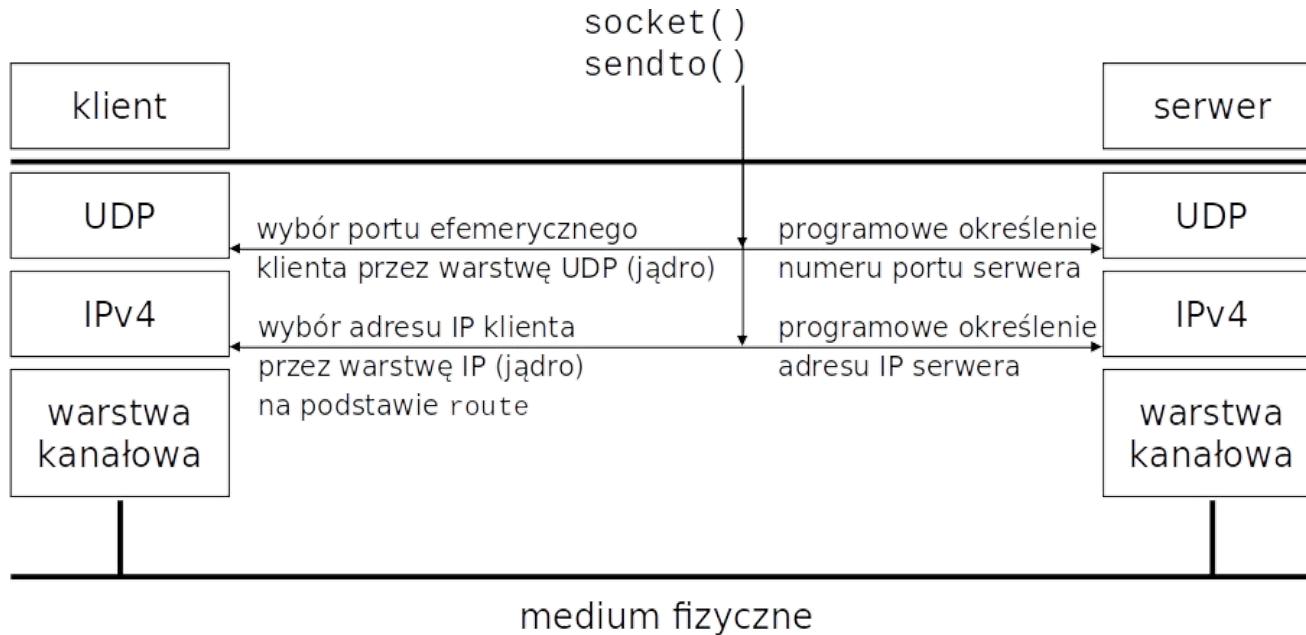
Algorytm iteracyjny dla UDP



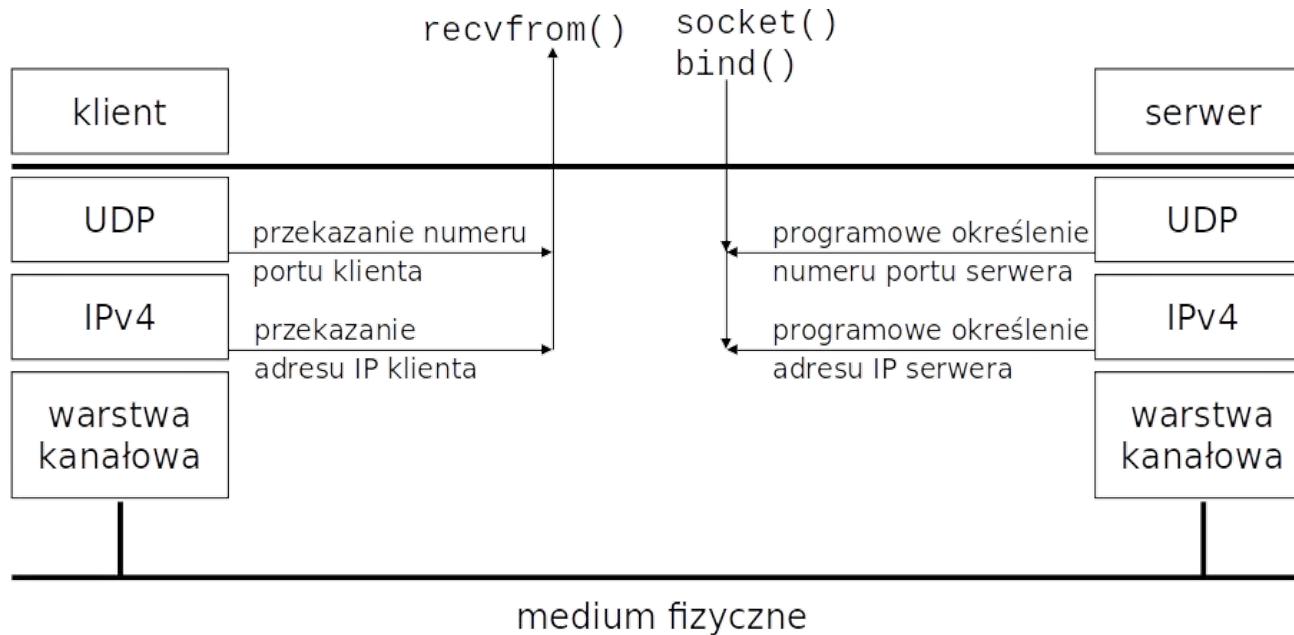
Algorytm iteracyjny dla UDP

- Nie ma fazy akceptacji połączenia – kto nadeśle datagram, ten będzie przyjęty (nie oznacza to, że każdy klient musi być obsłużony).
- Serwer pracuje w **nieskończonej pętli**, gdyż nie wiadomo kiedy klient zakończy swoje działanie (brak fazy zakończenia połączenia) => w jednej pętli może występować tylko jedna wymiana komunikatów, która musi zapewnić pełne obsłużenie klienta.
- Serwer działa w trybie **iteracyjnym** – serwer współbieżny, ze względu na wcześniejszy powód, mógłby nigdy się nie zakończyć => nieskończone mnożenie się procesów.
- W skończonym buforze obowiązuje kolejka **FIFO**.
- Funkcja **close()** odnosi się do socket'u, a nie do połączenia.

Analiza z punktu widzenia klienta



Analiza z punktu widzenia serwera



Implementacja: serwer UDP

```
struct sockaddr_in servaddr, cliaddr;      // struktury adresowe serwer/klient
memset (&servaddr, 0, sizeof(servaddr)); // wyzerowanie str.adr. serwera
servaddr.sin_family      = AF_INET;
servaddr.sin_addr.s_addr = htonl (INADDR_ANY);
servaddr.sin_port        = htons (9100);

// utworzenie socket'u i przypisanie adresu/portu lokalnego:
int sockfd = socket (AF_INET, SOCK_DGRAM, 0);
int bindresult = bind (sockfd, (struct sockaddr *) &servaddr, sizeof (servaddr));

int msglen;
const size_t MAXLINE = 100;
char msg[MAXLINE];

while ( 1 )
{
    // odbierz datagram
    socklen_t cliaddrrlen = sizeof(cliaddr);
    msglen = recvfrom (sockfd, msg, MAXLINE, 0,
                      (struct sockaddr *) &cliaddr, &cliaddrrlen);
    printf ("msg: %s msglen: %d\n", msg, msglen);
    // wyslij datagram
    sendto (sockfd, msg, msglen, 0, (struct sockaddr *) &cliaddr, cliaddrrlen);
}
```

Implementacja: klient UDP

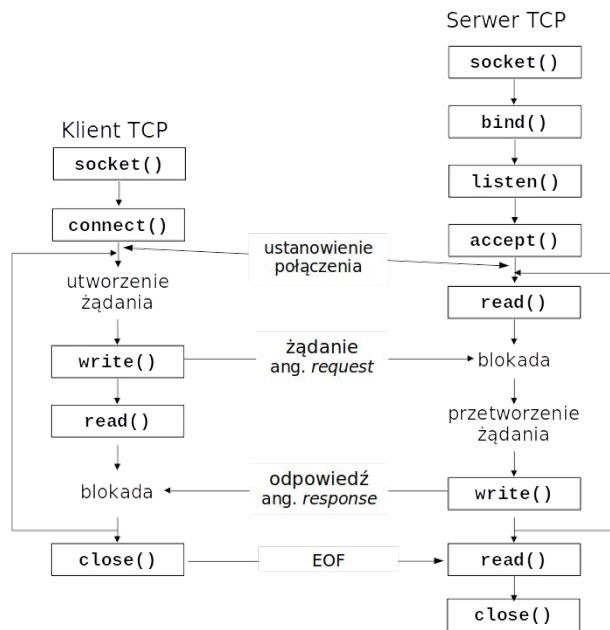
```
struct sockaddr_in servaddr;           // struktura adresowa serwera
memset (&servaddr, 0, sizeof(servaddr)); // wyzerowanie str.adr. serwera
servaddr.sin_family      = AF_INET;
inet_pton (AF_INET, "127.0.0.1", &servaddr.sin_addr);
servaddr.sin_port        = htons (9100);

// utworzenie socket'u
int sockfd = socket (AF_INET, SOCK_DGRAM, 0);

int msglen;
const size_t MAXLINE = 100;
char msg[MAXLINE];

while ( 1 )
{
    // wyslij datagram
    socklen_t servaddrlen = sizeof(servaddr);
    fgets (msg, MAXLINE-1, stdin);
    sendto (sockfd, msg, strlen(msg), 0,
            (struct sockaddr *) &servaddr, servaddrlen);
    // odbierz datagram
    msglen = recvfrom (sockfd, msg, MAXLINE, 0, NULL, NULL);
    printf ("msg: %s msglen: %d\n", msg, msglen);
}
```

Algorytm iteracyjny dla TCP



Uzgodnienie trójfazowe (ang. *three-way handshake*)

Otwarcie bierne (ang. passive open) – przygotowanie serwera na przyjęcie połączenia:

- serwer C/C++: `socket()`, `bind()`, `listen()`, `accept()` (blokada)

Otwarcie aktywne (ang. active open) – klient nawiązuje połączenie wysyłając pakiet synchronizujący (ang. synchronize):

- klient C/C++: `connect()` (blokada)
- tcpdump: SYN

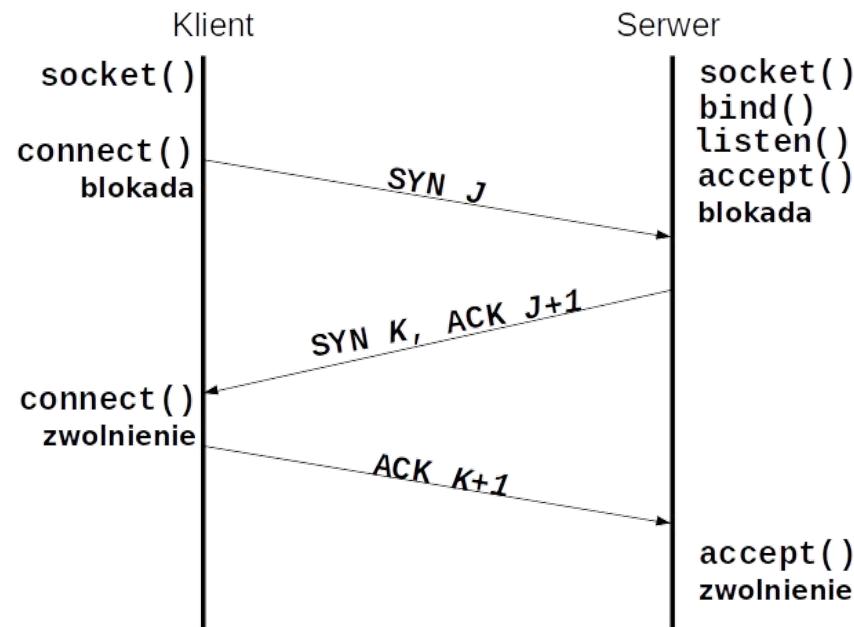
Potwierdzenie (ang. acknowledgment) ze strony serwera:

- tcpdump: SYN, ACK
- klient C/C++: `connect()` (zwolnienie)

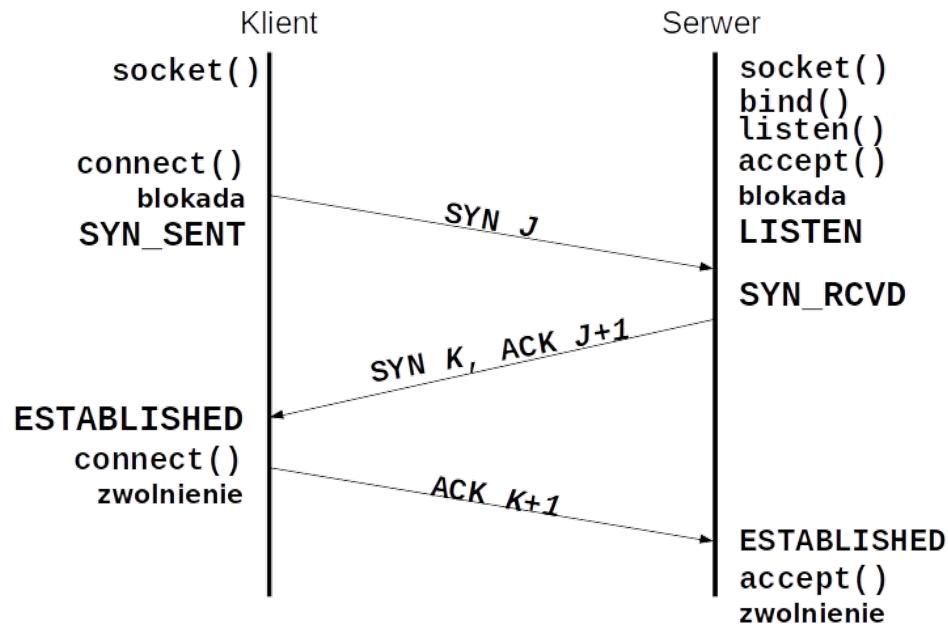
Potwierdzenie ze strony klienta:

- tcpdump: ACK
- serwer C/C++: `accept()` (zwolnienie)

Uzgodnienie trójfazowe



Stany TCP - nawiązanie połączenia



Przesyłanie danych

Serwer oczekuje na przyjęcie danych:

- serwer C/C++: read() (blokada)

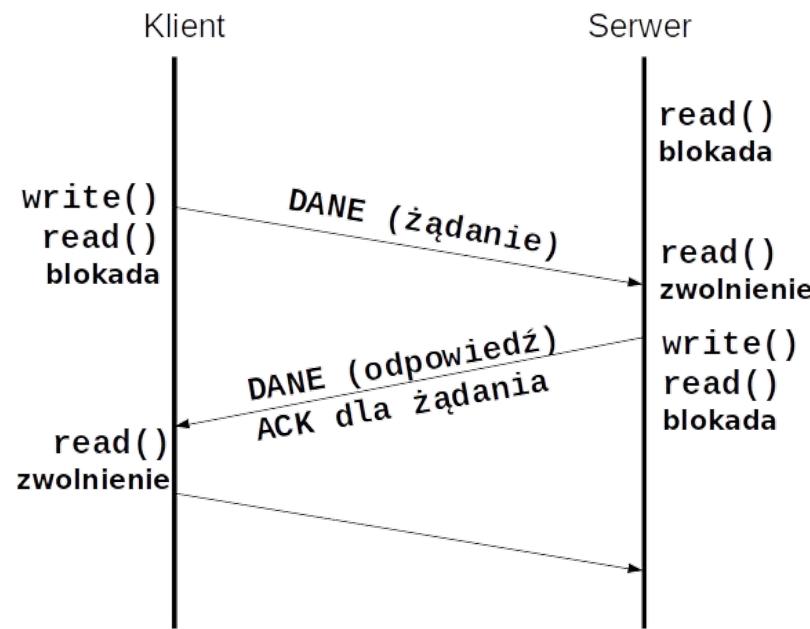
Wysłanie żądania, oczekiwanie na odpowiedź:

- klient C/C++: write(), read() (blokada)
- serwer C/C++: read() (zwolnienie)

Przetworzenie żądania, odpowiedź:

- serwer C/C++: write(), read() (blokada)
- klient C/C++: read() (zwolnienie)
- tcpdump: ACK dla żądania, ACK dla odpowiedzi.

Przesyłanie danych



Zakończenie połączenia TCP

Zamknięcie aktywne (ang. active close):

- C/C++: close()
- tcpdump: FIN

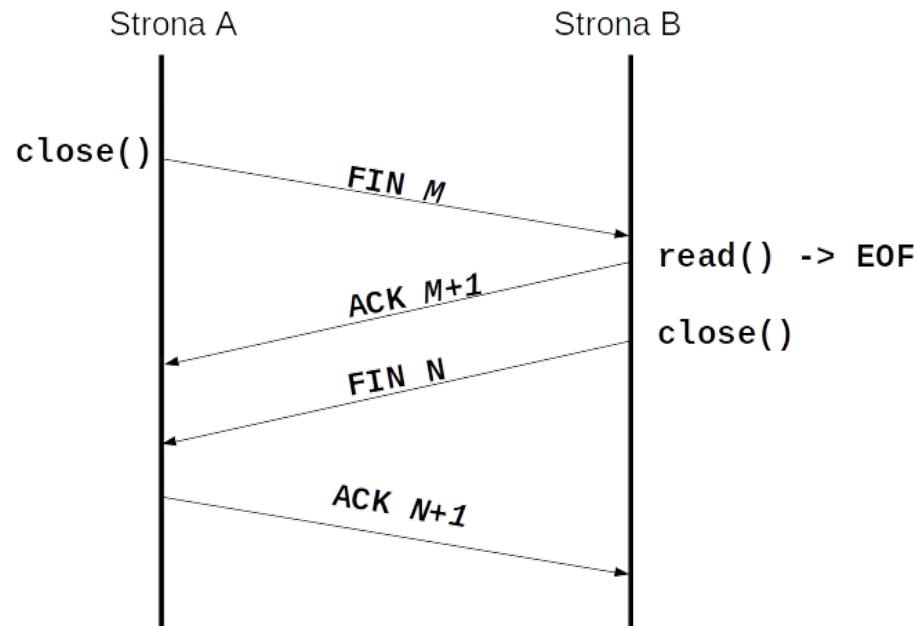
Zamknięcie bierne (ang. passive close):

- tcpdump: ACK
- C/C++: read() -> EOF
- ... odstęp czasu ...
- C/C++: close()
- tcpdump: FIN

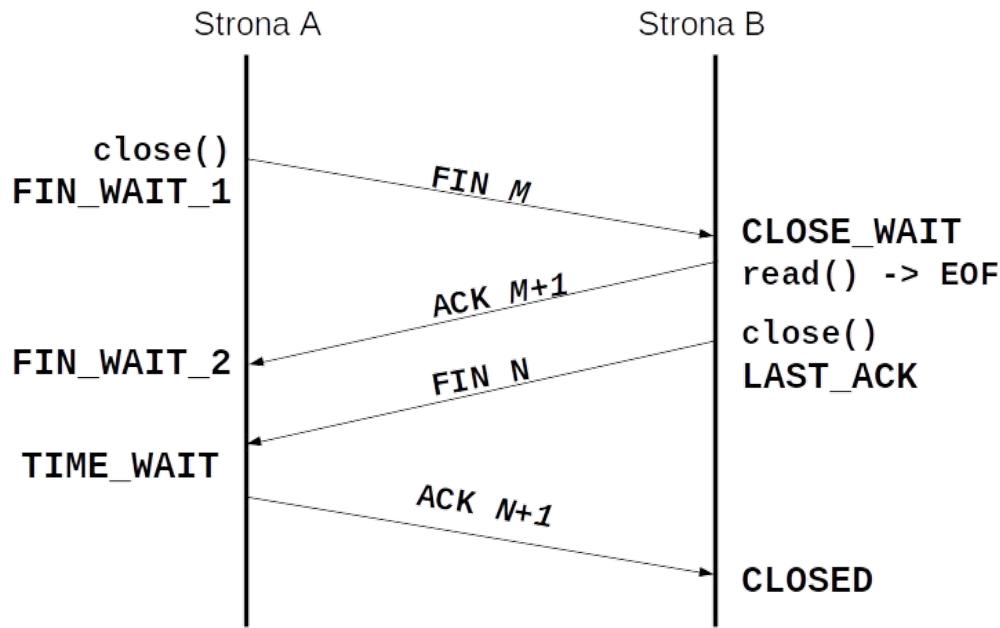
Potwierdzenie zamknięcia:

- tcpdump: ACK

Zakończenie połączenia



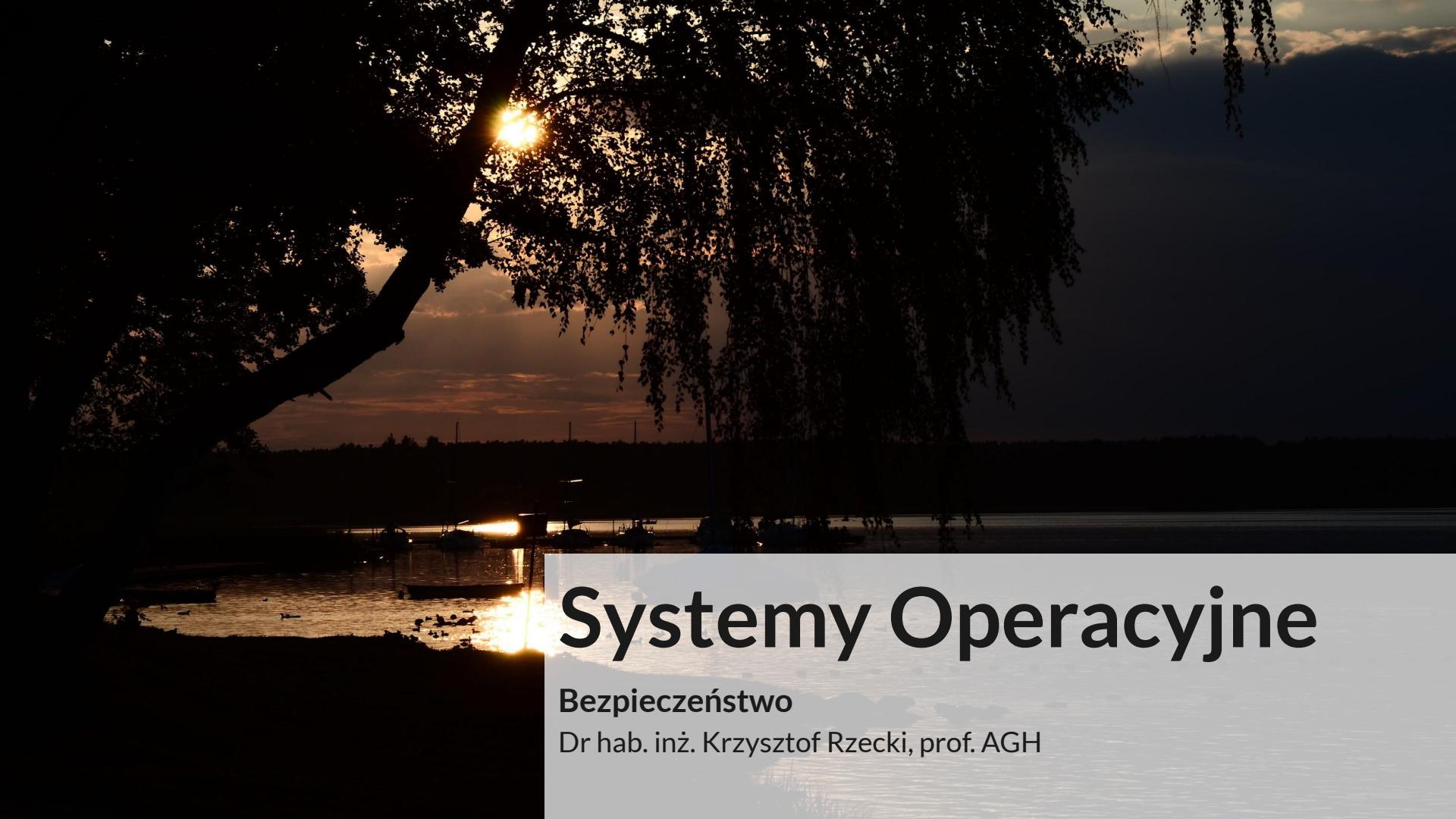
Zakończenie połączenia



Dziękuję za uwagę ;)



Tatry

The background of the slide is a photograph of a sunset or sunrise over a body of water. In the foreground, the dark silhouettes of tree branches and leaves frame the scene. The sky is filled with warm, orange, and yellow hues from the setting sun, which is partially obscured by the trees. In the distance, a small pier or dock with some structures is visible on the water.

Systemy Operacyjne

Bezpieczeństwo

Dr hab. inż. Krzysztof Rzecki, prof. AGH

Bezpieczeństwo

- **Bezpieczeństwo** jest miarą ufności, że integralność systemu i danych zostanie zachowana.
- Cztery najważniejsze funkcje bezpieczeństwa:
 - Uwierzytelnianie / Autentyczność
 - Poufność
 - Integralność
 - Rozliczalność / Niezaprzeczalność.
- Zasoby systemu komputerowego:
 - Informacje (dane, kod)
 - Czas procesora
 - Pamięć główna, pamięć zapasowa
 - Dostęp do sieci komputerowej
 - etc.

Aspekty biznesowe

- Utrata informacji przez jej bezpowrotnie usunięcie.
- Kompromitacja informacji przez jej ujawnienie.
- Wykorzystanie zasobów przedsiębiorstwa, np do rozsyłania spamu, tzw. kopania kryptowalut lub ataku na inne systemy komputerowe.

Problem bezpieczeństwa

- System jest **bezpieczny**, kiedy dostęp do jego zasobów oraz ich wykorzystanie odbywa się zgodnie z ustalonym przeznaczeniem.
- Z jednej strony można wprowadzać coraz mocniejsze zabezpieczenia.
- Z drugiej strony, zabezpieczenia nie mogą powodować uciążliwości w korzystaniu z systemu (patrz: odłączenie od sieci internet).
- Dobór zabezpieczeń (i ich koszt) musi być dopasowany do wartości przechowywanych informacji, czy krytyczności działania systemu oraz potencjalnych zagrożeń (nie ma sensu wprowadzać firewall do systemu bez dostępu do internetu)

Zarządzanie informacją

- Pozyskiwanie
- Parsowanie
- Oczyszczanie
- Transformacja
- Przetwarzanie
- Przechowywanie
- Backup
- Archiwizacja
- Przesyłanie
- Dodawanie
- Zmiana
- Usuwanie

Pozyskiwanie

- Ang. *acquiring* – pobieranie do systemu danych ze źródeł zewnętrznych.
- Źródła pozyskiwania danych:
 - Skaner dokumentów papierowych
 - Sieć komputerowa (inne hosty)
 - Urządzenie pomiarowe cyfrowe
 - Urządzenia akwizycji sygnałów analogowych
 - itp.



Parsowanie

- Ang. *parsing* – proces wyuskiwania interesujących nas danych z ciągów znaków, obrazów, sygnałów, itp.
- Parsowanie tekstu realizowane jest najczęściej w technice wyrażeń regularnych (ang. regular expression).
- Przykład: Dziś jest piątek trzynastego Jeśli parsowanie ma wyuskać dni tygodnia to z tego ciągu otrzymamy ‘piątek’.

Oczyszczanie

- Ang. *cleaning* – proces markowania danych, które są błędne.
- Oczyszczanie realizowane jest podobnie do parsowania.
- Przykład: Skanning laserowy prowadzony z samolotu-awionetki w rozdzielczości 12 punktów na 1 m² wykazał na obszarze oznaczonym jako jezdnia pewne osoby. Jeśli celem jest stworzenie ortofotomapy, to dane związane z rozpoznanymi osobami oznaczyć należy jako błędne.

Transformacja

- Ang. *transforming* – proces przekształcania danych między formatami.
- ETL – ang. Extract Transform Load – ciąg trzech procesów wyciągania danych z bazy danych źródłowych, transformacji do innego formatu i załadowanie do bazy danych docelowych.
- Przykład: Pobranie danych ze strony WWW MPK, przekształcenie na strefę czasową UTC i załadowanie do bazy MySQL.

Przetwarzanie

- Ang. *processing* – obróbka danych za pomocą algorytmów charakterystycznych dla danego problemu.

Przechowywanie

- Ang. *storing* – przechowywanie danych na nośnikach o dostępie bezpośrednim (ang. direct access).
- Przechowywanie w warstwie:
 - W warstwie fizycznej (dane binarne)
 - W warstwie logicznej systemu plików
 - W warstwie logicznej bazy danych

Backup i archiwizacja

- Ang. *backuping* – sposób utworzenia kopii danych w celu szybkiego ich odtworzenia na wypadek wystąpienia awarii.
- Ang. *archiving* – sposób utworzenia kopii danych w celu długotrwałego przechowywania.
- Archiwizacja:
 - Pełna (ang. *mirroring*),
 - Przyrostowa (ang. *incremental*).

Przesyłanie

- Ang. *transferring* – tworzenie kopii danych w innym systemie komputerowym za pomocą sieci komputerowej.
- Szybkość przesyłania liczona jest w bitach (kilobitach, megabitach, gigabitach) na sekundę.
- Przykłady:
 - $1 \text{ b/s} = 1 \text{ bps}$
 - $1024 \text{ b/s} = 1 \text{ Kbps}$
 - $1048576 \text{ b/s} = 1 \text{ Mbps}$

Zarządzanie bezpieczeństwem

- Poufność
- Integralność
- Dostępność
- Rozliczalność
- Identyfikacja / Uwierzytelnianie
- Autoryzacja
- Awaria / Niezawodność
- Anonimowość
- Zagrożenie
- Ryzyko
- Podatność
- Zabezpieczenie
- Monitorowanie
- Odtwarzanie

Poufność

- Ang. *confidentiality* –ochrona danych przed ich ujawnieniem osobom i/lub procesom.
- Poufność realizowana jest przez szyfrowanie danych i/lub kanałów komunikacji.
- Szyfrowanie: symetryczne i asymetryczne.

Integralność

- Ang. integrity - rozpoznanie zmiany, dodania lub usunięcia danych.
- Narzędziem do badania integralności są sumy kontrolne, kody korekcyjne CRC, kody MAC.
- Zaawansowane badanie integralności opiera się o podpis cyfrowy.

Dostępność

- Ang. *availability* – własność danej polegająca na tym, że jest ona dostępna dla osoby / procesu w zadanym przedziale czasu i zadanym miejscu.
- Dostępność to także stosunek czasu bezawaryjnego działania danej usługi w odniesieniu do całości założonego czasu.

Rozliczalność / Niezaprzeczalność

- Ang. *accountability* / ang. *nonrepudiation* – zapewnienie, że aktywność danej osoby / procesu może zostać bezsprzecznie stwierdzona.
- Rozliczalność realizowana jest za pomocą logowania (rejestrowanie zdarzeń).

Identyfikacja / Uwierzytelnianie

- Ang. *authenticity* – weryfikacja osoby / procesu.
- Weryfikację poprzedza identyfikacja (ang. *identification*).
- Weryfikacja realizowana jest za pomocą loginu i hasła, haseł jednorazowych, albo technik biometrycznych.

Zobacz: <https://doi.org/10.1016/j.ins.2017.05.041>

Autoryzacja

- Ang. *authorization* - przyznanie uprawnień.

Awaria

- Ang. *malfunction* – stan niesprawności systemu uniemożliwiający jego normalne użytkowanie i działanie.
- Awaria definiowana jest zwykle jako nagła i nieprzewidywalna (choć mogą wystąpić oznaki wskazujące zbliżającą się awarię).
- Awaria dotyczyć może całego systemu, lub niektórych funkcjonalności (zwykle kluczowych).

Niezawodność

- Ang. reliability – własność obiektu / systemu stwierdzająca prawdopodobieństwo nie wystąpienia awarii:
 - $R(t) = P\{t \geq r\}$
 - $R(t)$ – niezawodność po czasie t
 - r – założony czas pracy
- $\lim_{t \rightarrow \infty} R(t) = 0$
- Niezawodność $R(1h) = 90\%$ oznacza, że w pierwszej godzinie wystąpi 10% awarii.
- Niezawodność realizowana jest przez redundancję i nadmiarowość.

Anonimowość

- Ang. *anonymity* – własność związana z brakiem możliwości identyfikacji osoby / procesu, czy też powiązania zdarzenia, utworu z osobą / procesem.
- Anonimowość w sieci realizowana jest przez serwery proxy, NAT, sieć 'cebulkową'.
- Anonimowość technicznie nie jest możliwa – ale jest możliwa prawnie.

- Anonimizacja danych - jednokierunkowa zmiana wartości identyfikacyjnych.

Zagrożenie i ryzyko

- Ang. *threat* – stan obniżonego bezpieczeństwa.
 - Ang. *risk* – prawdopodobieństwo, że zagrożenie zmieni stan osoby / procesu / zasobu / systemu z pozytywnego w negatywny (w tym w awarię).
-
- Zagrożeniem jest włamanie, a z uwagi na hasła słownikowe ryzyko jest wysokie.
 - Zagrożeniem jest uszkodzenie dysku (awaria), ale redundancja obniża ryzyko.

Podatność i zabezpieczenie

- Ang. *susceptibility* – wysokie ryzyko zmiany zagrożenia w awarię.
 - Ang. *protection* – obniżanie ryzyka zmaterializowania zagrożenia (awarii).
-
- Podatny system na włamania można zabezpieczyć uruchamiając firewall.

Monitorowanie

- Ang. *monitoring* – realizacja procesów obserwacji osoby / procesu / systemu o charakterze ciągłym i długotrwałym.
- Monitoring realizowany jest przez ciągłe analizowanie logów (rozliczalność), odpytywanie o stan, obserwację bezinwazyjną.

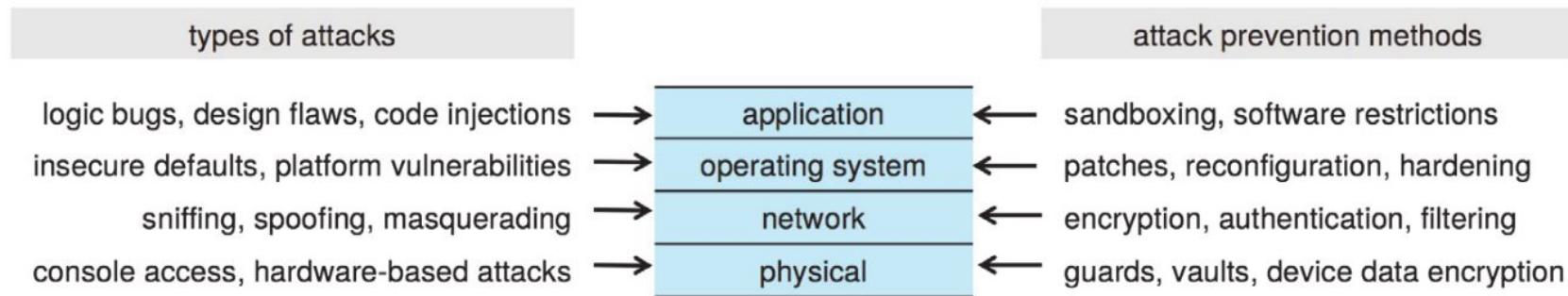
Odtwarzanie

- Przywracanie stanu systemu sprzed awarii.
- Przywracanie może być pełne lub częściowe.
- Przywracanie może wyłączać system z użycia lub trwać w trakcie jego pracy.

Podstawowe typy ataków

- Naruszenie poufności, integralności, czy dostępności do danych.
- Kradzież usługi, np. przez przechwycenie danych uwierzytelniających.
- Odmowa usługi (ang. *Denial-of-service*, DOS) oraz DDOS (ang. *distributed denial of service*).
- Powtarzanie operacji (ang. *reply attack*), czyli wykonanie powtórne tej samej operacji (np. przelew).
- Atak ang. *man-in-the-middle* polegający na umieszczeniu atakującego pomiędzy stronami.
- Atak ang. *session hijacking* polegający na przechwyceniu sesji (może poprzedzać powyższy atak).
- Eskalacja uprawnień (ang. *privilege escalation*) - przekazywanie i rozszerzanie uprawnień.

Cztery poziomy bezpieczeństwa



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Jednak najsłabszym ogniwem zwykle jest: **człowiek** (socjotechnika, ang. *social engineering*).

Oprogramowanie naruszające bezpieczeństwo

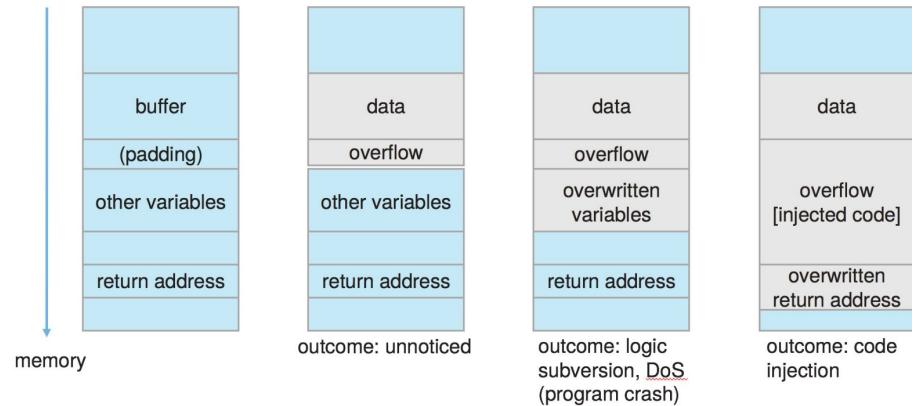
- **Malware** - oprogramowanie przeznaczone do wykorzystania, zablokowania lub uszkodzenia systemu komputerowego. Podstawa działania: uruchamianie z uprawnieniami innego użytkownika.
- **Koń trojański** - oprogramowanie realizujące w ukryty sposób szkodliwą funkcjonalność, na przykład: pobieranie informacji o danych do logowania, kontakty użytkownika, itp.
- **Spyware** - odmiana konia trojańskiego, którego funkcjonalność polega na wykorzystaniu informacji o użytkowniku, np. w celu dobrania właściwych reklam.
- **Ransomware** - jego działanie polega na szyfrowaniu danych użytkownika celem wyłudzenia od niego opłaty za odzyskanie danych.
- **Back door** - celowo pozostawione przez twórców oprogramowania luki w zabezpieczeniach pozwalające na nieuprawniony dostęp (w tym logowanie klawiszy, ang. *keystroke logger*).
- **Logic bomb** - rodzaj luki w oprogramowaniu, która aktywnia się pod specjalnymi warunkami.

Podstawowy sposób obrony

Zasada minimalnych uprawnień.

Wstrzykiwanie kodu

- **Wstrzykiwanie kodu** (ang. *code-injection attack*) - rodzaj ataku polegający na zmianie lub rozszerzeniu kodu uruchamialnego.
- Wstrzykiwanie kodu zwykle jest wynikiem wadliwego stosowania paradygmatów programowania w językach niskiego poziomu, np. C/C++, które umożliwiają swobodne poruszanie się po pamięci.



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*

Wirusy i robaki

Wirus - fragment kodu dołączony do programu, który potrafi sam się replikować infekując inne programy. Działanie wirusa może prowadzić w lekkiej postaci tylko do rozprzestrzeniania, a w przypadku kodu złośliwego, do niszczenia danych. Wirusy są problemem głównie dla systemów z rodziny Windows. Wirusy roznoszone są między systemami przez e-mail (w tym ataki phishingu), pobieranie zawirusowanego oprogramowania.

Robak - rodzaj oprogramowania “wędrującego” po sieci internet.

Zagrożenia systemowe i sieciowe

- Model 'secure by default', czyli wszystko, co nie jest dozwolone, jest zabronione.
- **Zombie system** - opanowany przez atakującego system, z którego prowadzony jest atak.
- **Sniffing** - podsłuchiwanie.
- **Spoofing** - podszywanie.
- **Scanning** - skanowanie.

Podstawy kryptografii

- M – Tekst jawny (ang. *plaintext, cleartext*)
- $E()$ – Szyfrowanie (ang. *encryption*) oparty o algorytm kryptograficzny (ang. *cipher*)
- $C = E(M)$ – Kryptogram (ang. *ciphertext*)
- $M = D(C)$ – Deszyfrowanie (ang. *decryption*) => $D(E(M)) = M$
- Kryptografia – nauka zajmująca się zabezpieczaniem informacji.
- Kryptoanaliza – nauka zajmująca się łamaniem kryptogramów.
- Kryptologia = Kryptografia + Kryptoanaliza.
- $H()$ – Funkcja skrótu (ang. *hash function*)
- $h = H(M)$ – Skrót (ang. *hash*)
- $S()$ – Podpisywanie cyfrowe (ang. *digital signing*)
- $V()$ – Weryfikacja podpisu (ang. *digital verification*): $V(S(M)) = M$

Podstawy szyfrowania

- Poufność algorytmu – jeśli bezpieczeństwo zaszyfrowanej wiadomości oparte jest o siłę algorytmu (skomplikowany algorytm).
- Poufność klucza – jeśli bezpieczeństwo zaszyfrowanej wiadomości oparte jest o siłę klucza (długość):
 - Algorytm symetryczny (np. DES – *Data Encryption Standard*):
 - $D_k(E_k(M)) = M$
 - Algorytm asymetryczny (np. RSA – *Rivest Shamir Adleman*):
 - $D_{k1}(E_{k2}(M)) = M$
 - $D_{k2}(E_{k1}(M)) = M$

Funkcja skrótu

- Operuje na dowolnej długości wiadomości wejściowej M . Zwraca wartość hash o stałej długości h .
$$h=H(M)$$
, gdzie h ma długość m
- Własności:
 - Łatwo obliczalna: mając M łatwo obliczyć h .
 - Jednokierunkowa: mając h trudno wyznaczyć źródłowe M : $H(M)=h$.
- Wolna od kolizji, ale suriekcja: istnieją takie dwie różne M i M' , że $H(M)=H(M')$.
- Jednoznaczna: dla każdego M : $H(M)=H(M)$.
- Dyfuzja, czyli cecha powodująca rozsianie bitów wiadomości jawniej w skrócie.
- Konfuzja, czyli cecha ukrywająca powiązanie pomiędzy wiadomością jawną, a skrótem.

Podpis odręczny

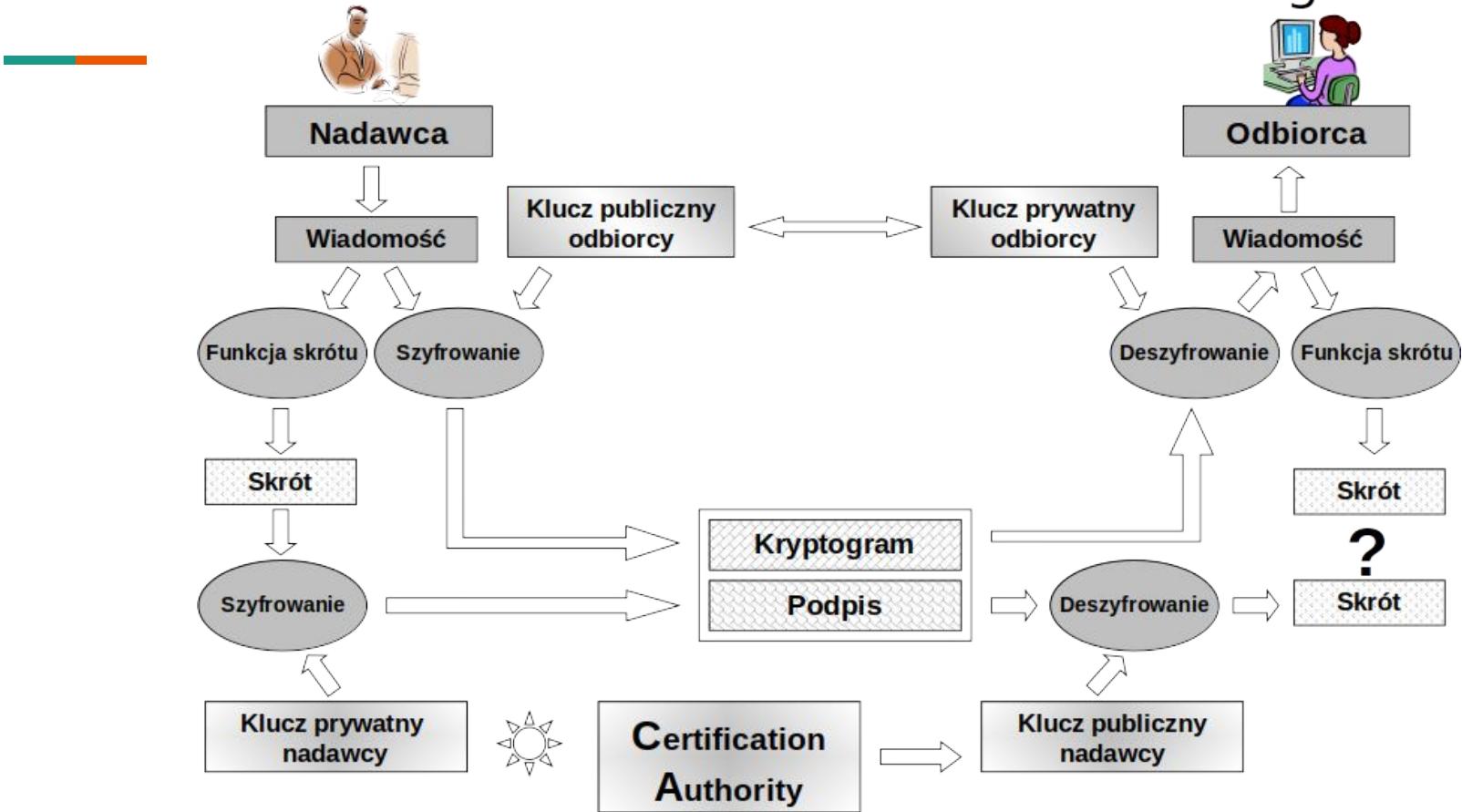
- Trudny do podrobienia.
- Łatwy do zweryfikowania.
- Nieprzenoszalny na inny dokument.
- Dokumentu nie można zmienić.
- Podpisu nie można się wyprzeć.

(Nie zawsze możliwe)

Podpis cyfrowy

- k_1 – klucz prywatny, k_2 – klucz publiczny
- $S_{k_1}(M) = E_{k_1}(H(M))$ – podpis cyfrowy M
- $V_{k_2}(S_{k_1}(M), M)$ – weryfikacja podpisu
 $D_{k_2}(S_{k_1}(M)) = ? = H(M)$
 $D_{k_2}(E_{k_1}(H(M))) = ? = H(M)$
 $H(M) = ? = H(M)$

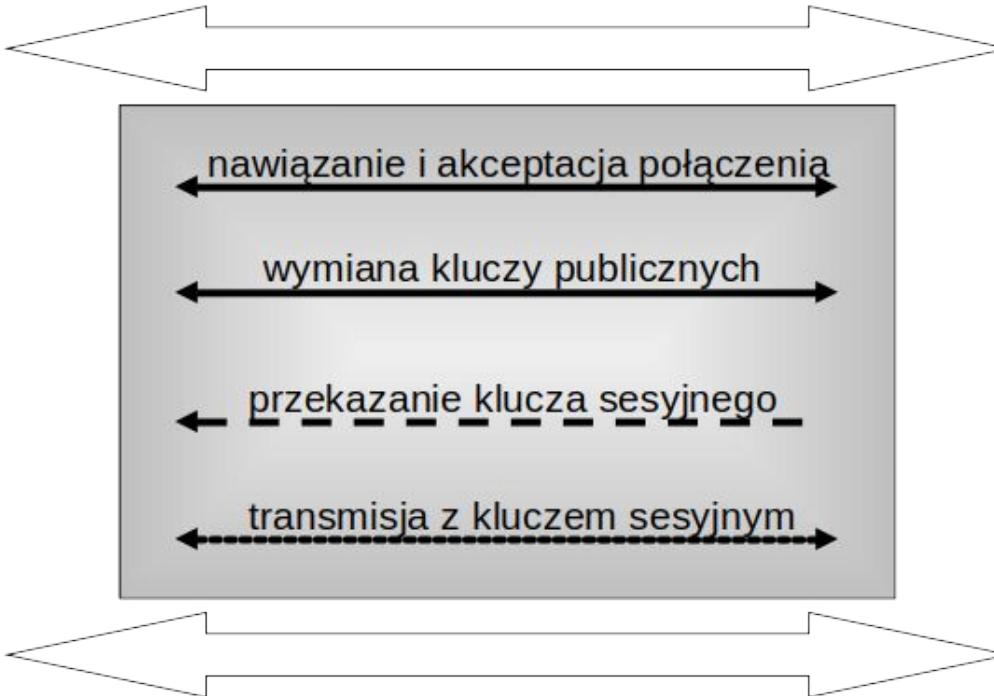
Infrastruktura Klucza Publicznego



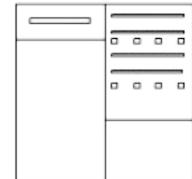
Secure Socket Layer



Klient



HTTPS



Serwer

POP3S
SMTPS



Niesulice