

**WYŻSZA SZKOŁA INFORMATYKI STOSOWANEJ I ZARZĄDZANIA
WYDZIAŁ INFORMATYKI**

**WIELODOSTĘPNE SYSTEMY OPERACYJNE 1
(SO1)**

**oraz
UŻYTKOWANIE SYSTEMU UNIX**

**WYKŁADY I LABORATORIUM KOMPUTEROWE
(konspekt)**

Prowadzący wykłady: Lech Kruś

Warszawa

Wielodostępne systemy operacyjne; użytkowanie systemu UNIX

Wykłady, Laboratorium komputerowe

Cel przedmiotu:

Zapoznanie słuchaczy z podstawami budowy i funkcjonowania współczesnych systemów operacyjnych.

Nabycie przez słuchaczy praktycznych umiejętności pracy w systemie UNIX i sieciach komputerowych.

Wymagane przygotowanie słuchaczy:

- w zakresie podstaw informatyki technicznej (cyfrowej reprezentacji informacji, podstaw arytmetyki komputerów, podstaw teorii układów logicznych (A. Skorupski, Podstawy budowy i działania komputerów, WKŁ 1996),
- organizacji i architektury komputerów (P. Metzger, Anatomia PC, Helion 2006),
- pracy w systemie MS Windows.

Zaliczenie przedmiotu:

Laboratorium – Kolokwia/Sprawdziany

Wykład – Sprawdzian końcowy

Zakres tematyczny:

WYKŁADY

Wprowadzenie do wielodostępnych systemów operacyjnych (na przykładzie systemu UNIX).

Podstawowe cechy systemu operacyjnego.

Części składowe systemu operacyjnego.

Struktura systemu operacyjnego

Usługi systemu operacyjnego.

Systemy plików

Pliki i systemy plików w systemie operacyjnym.

Podstawowe typy plików: pliki zwykłe, pliki specjalne (urządzeń), katalogi, potoki nazwane, gniazda.

Struktura systemu plików w systemie UNIX.

Pojęcie i-węzła.

Zawartość i-węzła.

Sposób przechowywania plików na dysku.

Zarządzanie procesami

Pojęcie procesu

Tworzenie, usuwanie, zawieszanie i odwieszanie procesów.

Komunikacja między procesami.

Szeregowanie procesów.

Zarządzanie pamięcią

Pamięć główna

Pamięć pomocnicza na dysku (obszar wymiany „swap”)

Pojęcie pamięci wirtualnej

Procesy wymiany

Procesy stronicowania i segmentacji

Zarządzania operacjami wejścia wyjścia

Podstawowe pojęcia (szyna, port we-wy, sterownik, odpytywanie, system przerwań)

Struktura oprogramowania we-wy w jądrze SO

Urządzenia znakowe, urządzenia blokowe

Pliki specjalne

Tablica rozdzielcza urządzeń

Laboratorium

Podstawy użytkowania systemu

Praca w systemie.

Postać poleceń w systemie UNIX .

Wybrane polecenia podstawowe

Polecenia służące identyfikacji użytkowników w systemie oraz komunikacji między użytkownikami.

System plików

Poruszanie się w systemie plików.

Tworzenie i usuwanie katalogów.

Praca z plikami

Plik i jego atrybuty .

Działania na plikach.

Dowiązywanie plików i katalogów.

Prawa dostępu do plików i katalogów

Podstawowe prawa dostępu

Zmiana praw dostępu.

Podstawy edycji tekstów

Edytory w systemie UNIX
Praca z edytorem vi

Podstawy korzystania z shella

Rodzaje shella
Wykonywanie poleceń w shellu
Środowisko shella

Środowisko użytkownika

Określanie środowiska użytkownika.
Zmienne shella

Różne mechanizmy w shellu

Generowanie nazw plików.
Przeadresowanie wejścia i wyjścia
Potoki
Podstawianie poleceń
Korzystanie z wcześniejszych wydanych poleceń

Wybrane polecenia działania na plikach tekstowych

Nadzorowanie wykonywania procesów

- Informacja o procesach
- Przetwarzanie w pierwszym i drugim planie
- Wysyłanie sygnałów do procesów
- Planowanie wykonywania poleceń

Wybrane polecenia użytkowe związane z archiwizacją plików

- Kompresja danych
- Wyszukiwanie plików
- Tworzenie kopii zapasowych

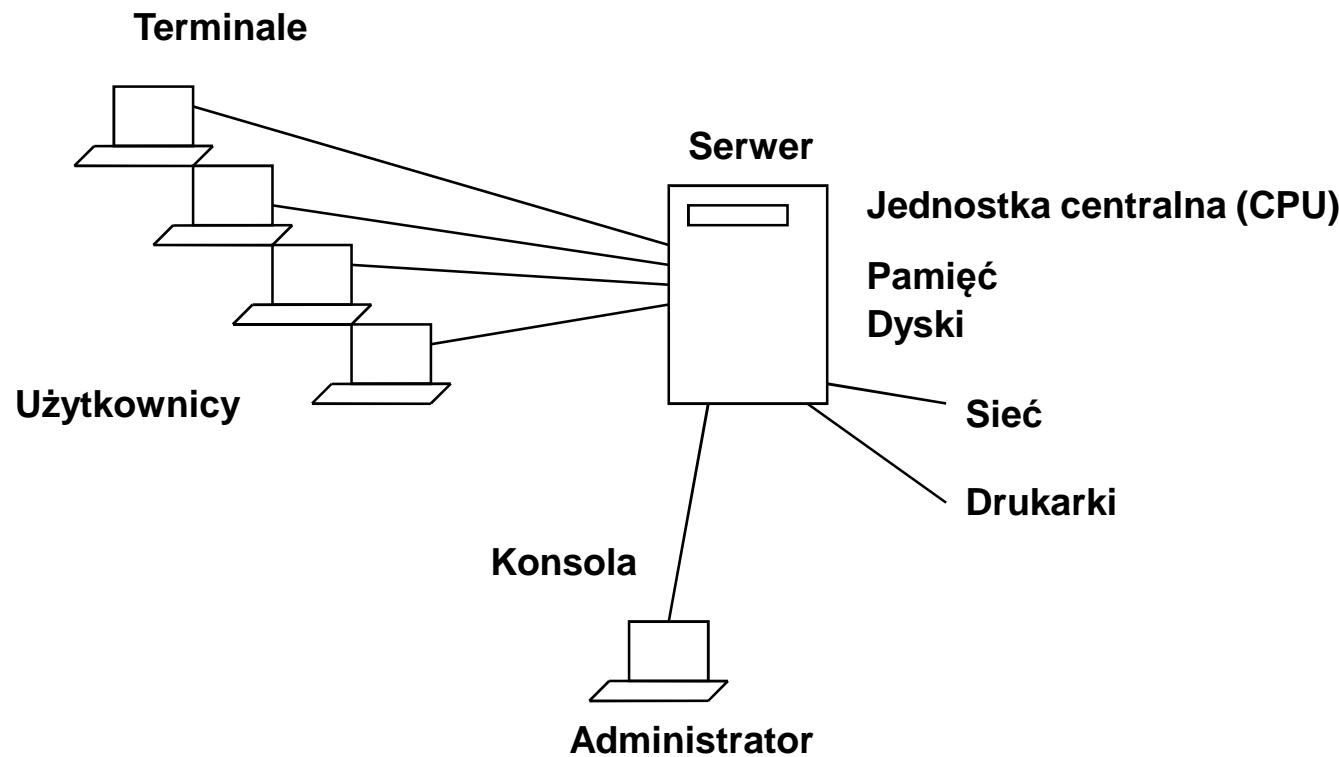
Wprowadzenie do pracy w sieciach komputerowych

- Sieci komputerowe, sieć INTERNET
- Poczta elektroniczna
- Praca na zdalnym komputerze

LITERATURA:

- A. Silberschatz, P.B. Galvin, G. Gaigne; Podstawy systemów operacyjnych, WNT,
Warszawa 2005.**
- M. J. Bach; Budowa systemu operacyjnego UNIX, WNT, Warszawa, 1995.**
- B. Goodheart, J. Cox; Sekrety magicznego ogrodu, Unix system V wersja 4 od środka,
WNT, Warszawa, 2001.**
- B. Goodheart, J. Cox; Sekrety magicznego ogrodu, Unix system V wersja 4 od środka,
- klucz do zadań. WNT, Warszawa, 2001.**
- K. Haviland, D. Gray, B. Salama; UNIX. Programowanie systemowe. Wydawnictwo RM,
Warszawa, 1999.**
- A. Silberschatz, P.B. Galvin, G. Gaign; Operating System Concepts. John Wiley & Sons, Inc.,
2003.**
- A. S. Tanenbaum; Modern Operating Systems, Prentice-Hall, Inc. London, 1992.**
- E. Nemeth, G. Snyder, S. Seebass, T.R. Hein: Przewodnik administratora systemu UNIX.
WNT, Warszawa, 1998.**
- S. Prata, D. Martin; Biblia systemu UNIX V. Polecenia i programy użytkowe, LT&P,
Warszawa, 1994.**
- A. Southerton, E.C. Perkins; Słownik poleceń systemów UNIX i X, Wiley&WNT ,
Warszawa,1995.**
- L. Lamb; Learning the vi editor. O'Reilly&Associates, Inc., Sebastopol, CA, 1996.**
- M.I. Bolsky, D.G. Korn; The Kornshell - command and programming language, Prentice Hall,
1989.**

Przykładowa konfiguracja systemu komputerowego



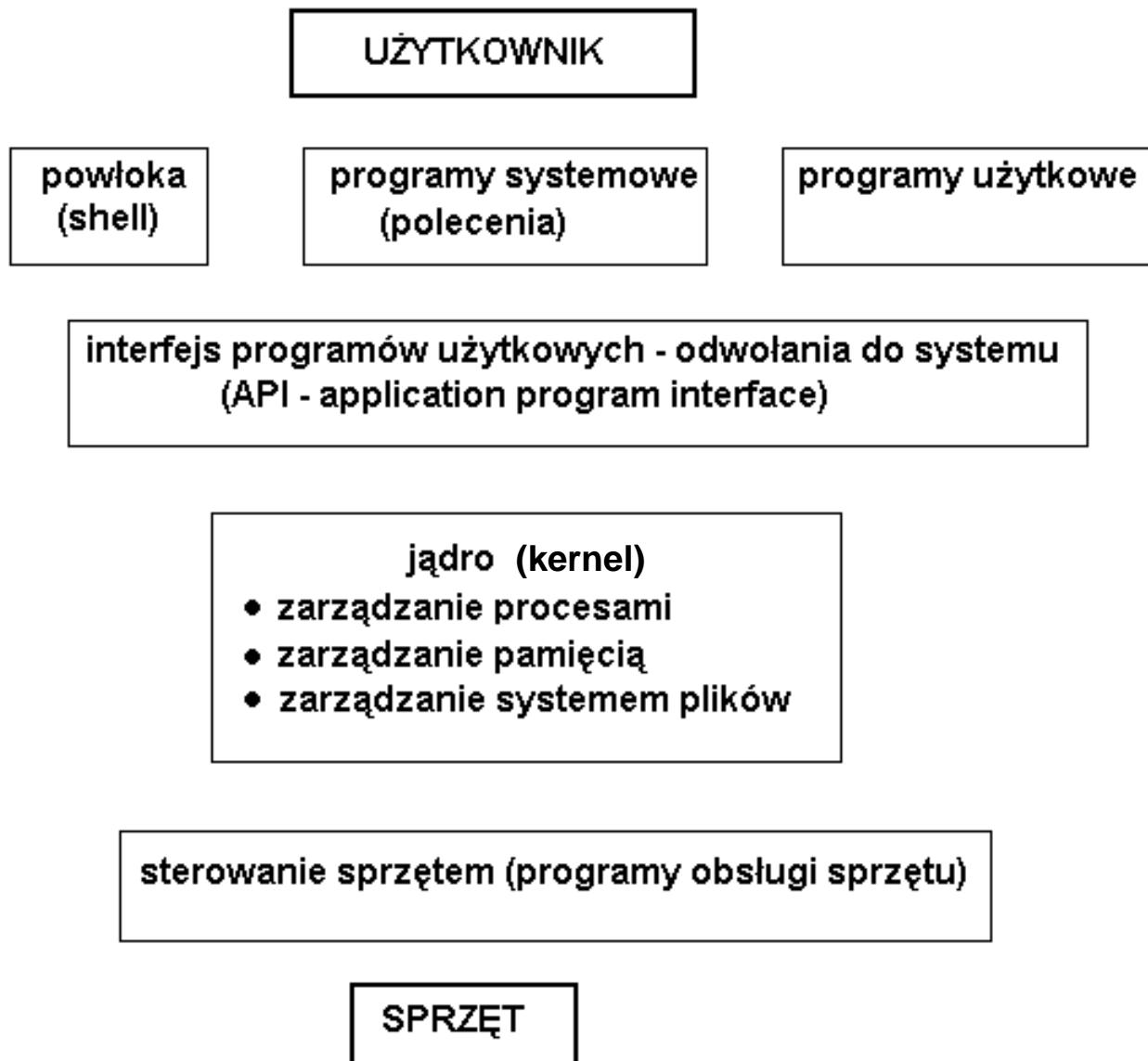
System operacyjny: program zarządzający pracą komputera

Zadania użytkownika wykonywane są jako procesy obsługiwane przez system operacyjny. System przydziela procesom zasoby komputera: dostęp do jednostki centralnej, pamięć, dyski, i inne.

Niezbędny administrator - osoba odpowiedzialna za sprawne działanie i bezpieczeństwo systemu. Również opiekun użytkowników, zakłada konta użytkowników, służy pomocą i radą.

Co to jest system operacyjny ?

Program pośredniczący między sprzętem komputerowym a użytkownikiem.



Krótką historią systemu operacyjnego UNIX

1969

AT&T Bell Labs

Ken Thompson, Ruddy Canaday, Joe Ossana opracowali pierwszy system UNIX

- Założenia: mały, ogólnego przeznaczenia, pracujący z podziałem czasu system operacyjny na minikomputerze PDP 7
- Cel budowy systemu: system operacyjny opracowany przez programistów dla programistów
- Podstawowe założenie: użytkownik wie co robi.

1973

4-ta edycja systemu AT&T, jądro systemu i Shell napisane w języku C opracowanym przez Denisa Ritchie.

1975

Zainteresowanie systemem UNIX na uniwersytetach

Dlaczego UNIX stał się popularny?

- mały, elastyczny, tani, dostępny na komputerach od micro do super komputerów
- właściwości zazwyczaj dostępne tylko w dużych systemach operacyjnych:
- architektura wieloprocesowa, inicjowanie asynchronicznych procesów
- hierarchiczny system plików

1975 Uniwersytet Kalifornijski

Początek prac nad systemem UNIX w Uniwersytecie Kalifornijskim w Berkeley
Berkeley Software Distribution (UNIX BSD)

Iata 80-te

Różne komercyjne wersje Unixa

1985 System V Interface Definition

1986 4.3 BSD

Duże firmy komputerowe zaczęły tworzyć własne systemy, zgodne z systemem UNIX. Systemy te łączyły w sobie cechy systemów AT&T oraz BSD.

Prace nad standardami - założenie Open System Fundation.

Nazwy systemu UNIX w różnych firmach

AT&T	UNIX	Apple	A/UX
Berkeley	BSD	DEC	Ultrix
HP	HP-UX	IBM	AIX
Microsoft	Xenix	SUN	SunOS, Solaris
Santa Cruz	UNIX SCO		

LINUX (Linus Torvalds)

open source linux - Powszechna Licencja Publiczna GNU

Można instalować oprogramowanie na dowolnej liczbie komputerów.

Dowolna liczba użytkowników może używać oprogramowania w tym samym czasie.

Można wykonać dowolną ilość kopii oprogramowania i przekazać je komukolwiek (redystrybucja "otwarta" lub "bez ograniczeń").

Brak ograniczeń w modyfikowaniu oprogramowania (z wyjątkiem zachowania w nietkniętym stanie pewnych uwag).

Nie ma ograniczeń w rozprowadzaniu, a nawet w sprzedaży programowania.

Dystrybucje (przykłady):

NOVEL SUSE, RED HAT, MANDRAKE, DEBIAN, SLACKWARE, ...

Wprowadzanie standardów

Cele standaryzacji

Budowa systemów tzw. "otwartych", charakteryzujących się:

- przenośnością aplikacji (portability),
- możliwością współpracy oprogramowania działającego na różnych maszynach (interoperability),
- skalowalnością (scalability),t.j. możliwością rozbudowy sprzętowej i rozbudowy aplikacji bez konieczności zmian systemu.

Istota standardu: określenie interfejsu, a nie implementacji.

Niektóre standardy i instytucje standaryzujące

SVID - standard firmy AT&T - pierwsza próba standardu interfejsu systemu operacyjnego.

POSIX - standard sponsorowany przez Instytut IEEE (Institute of Electrical and Electronics Engineering). Prace nad tym standardem obejmują: interfejs systemu, programy shell, metody testowania zgodności danego systemu ze standardem, pracę w sieci, zagadnienia ochrony i inne.

OSF (Open System Foundation) stowarzyszenie wiodących firm: HP, IBM, DEC i innych, zajmujące się promocją systemów otwartych. Opracowany został standard OSF/Motif.

Aktualne prace obejmują między innymi: Distributed Computing Environment - przetwarzanie w środowisku rozproszonym, Distributed Management Environment - centralne zarządzanie sieciami heterogenicznymi.

ISO - International Standard Organization - koordynuje tworzenie i stosowanie standardów w skali międzynarodowej. Wprowadził siedmiowarstwowy model odniesienia OSI (Open System Interconnection) dotyczący pracy w sieciach komputerowych.

Cechy systemu UNIX

System wielodostępny
(ang. *multiuser*)

wielu użytkowników pracujących z jednym komputerem,
wrażenie samodzielnej pracy

Praca interakcyjna
(*interactive*)

użytkownik wprowadza polecenie z terminala,
UNIX wykonuje polecenie, wyświetla wyniki i
czeka na nowe polecenie

System wielozadaniowy
(*multitasking*)

każdy użytkownik może uruchomić jednocześnie
wiele programów

Zapewnia bezpieczeństwo
(*security, privacy*)

identyfikacja użytkowników, ochrona dostępu do plików i
katalogów, ochrona procesów

Niezależny od urządzeń system
we/wy (device independent I/O)

każde urządzenie reprezentowane jest przez plik
specjalny systemu

Komunikacja między procesami
(ang. *interprocess communication*)

aplikacje mogą wzajemnie się ze sobą
komunikować

System sieciowy
(*networking*)

wbudowane jest oprogramowanie
umożliwiające pracę w sieci

Polecenie/Program użytkowy
(*commands/utilities*)

polecenie jest programem użytkowym,
można samemu je napisać

Interpretator poleceń
(*shell*)

jest wiele programów shell'a, można go zmienić

Struktura systemu operacyjnego

Podsystemy wykonujące zadania:

Zarządzanie procesami: tworzenie, usuwanie, zawieszanie, odwieszanie procesów, mechanizmy synchronizacji procesów, komunikacja między procesami.

Zarządzanie pamięcią: zarządzanie pamięcią główną, obszarem wymiany (swap), pojęcie pamięci wirtualnej.

Zarządzanie przestrzenią dyskową: zarządzanie wolną przestrzenią dysków, procesami, zapisywania informacji na dysku, szeregowanie zadań zapisu i odczytu.

Zarządzanie operacjami we/wy: obejmuje podsystem buforowania, interfejs: urządzenia -sterowniki, sterowniki urządzeń.

Zarządzanie plikami: tworzenie, usuwanie plików i katalogów, elementarne operacje z plikami i katalogami.

Podsystem ochrony: ochrona procesów przed działaniem innych procesów, mechanizmy zapewniające, że pliki, segmenty pamięci, CPU, inne zasoby są udostępnione tylko tym procesom, które mają autoryzację systemu operacyjnego. Ogólnie: mechanizmy kontroli dostępu programów, procesów, użytkowników do zasobów systemu komputerowego.

Praca sieciowa: usługi umożliwiające komunikację w sieci, pojęcie systemów rozproszonych.

Usługi systemu operacyjnego

Wykonywanie programów

Operacje we/wy

Operacje obsługi systemu plików

Komunikacja między procesami

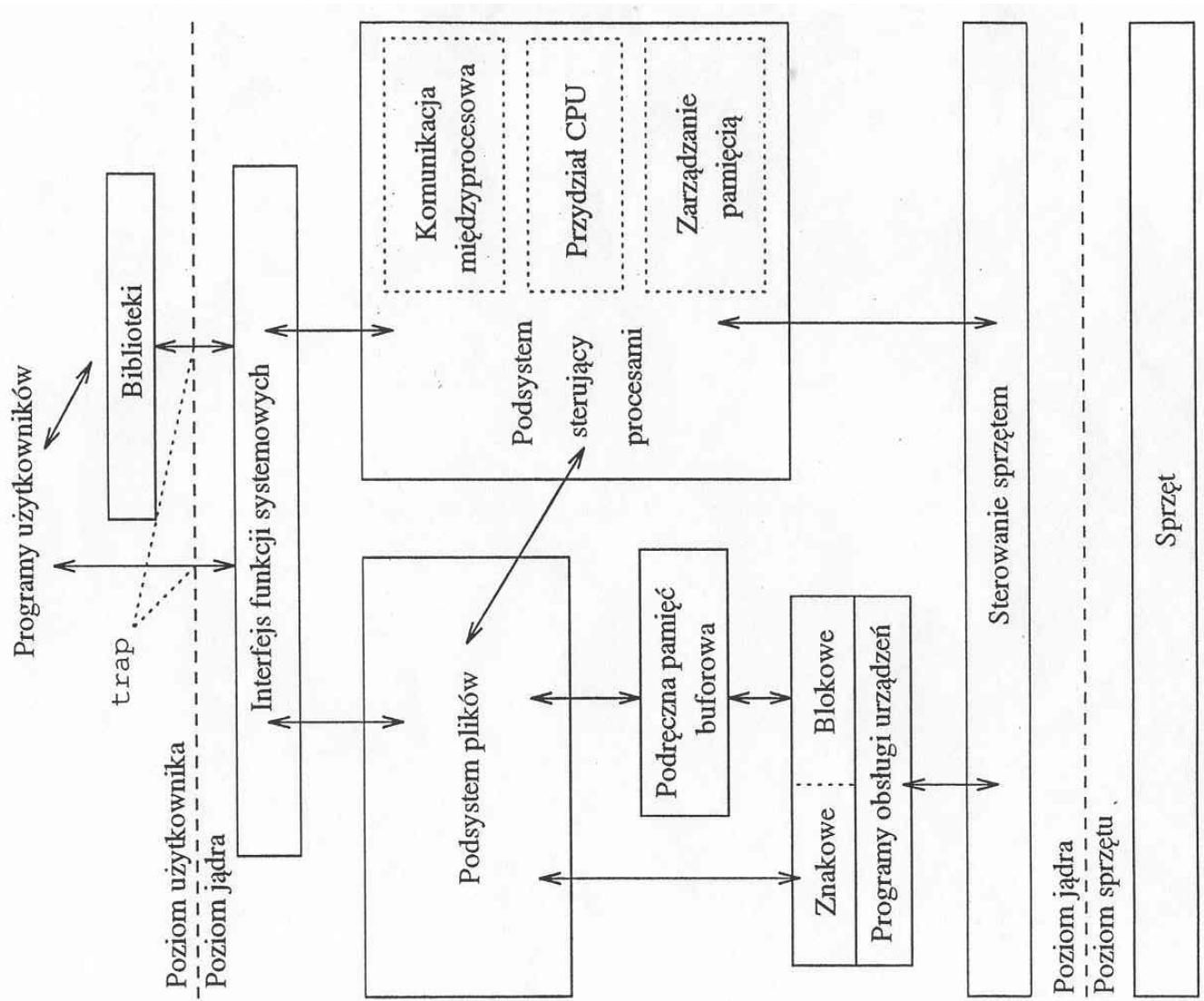
Detekcja błędów

Przydział zasobów

Rozliczanie użytkowników (accounting)

Ochrona

Funkcje systemowe (system calls):
interfejs między procesami i systemem operacyjnym



Systemy plików

Pliki to jednostki logiczne przechowywanej informacji, niezależne od właściwości fizycznych urządzeń pamięciowych. Zwykłe w plikach przechowywane są programy lub dane (tekst, liczby, grafika, itp.).

System plików - zbiór typów danych, struktur danych oraz funkcji systemowych (ang. *system calls*) używanych przez system operacyjny w celu przechowywania informacji w urządzeniach pamięci masowej (głównie na dyskach).

W systemach wielodostępnych systemy plików mają strukturę katalogową (hierarchiczną).

Pliki identyfikuje się za pomocą nazw. W systemie UNIX, w zależności od implementacji, pliki mogą mieć krótkie nazwy, do 14 znaków, lub długie nazwy, do 255 znaków.

Zestaw znaków dopuszczalnych obejmuje:

małe lub duże litery, cyfry, znaki specjalne takie jak „+,-,_,:”

Przykład

Dopuszczalne nazwy plików: .profile .xyz.abcd abc AbC 123..456..78 -a

UWAGA: Tej ostatniej nazwy lepiej jednak nie używać.

UWAGA: W systemie UNIX pliki mogą być identyfikowane za pomocą wielu różnych nazw (dowiązań).

Podstawowe typy plików:

- pliki zwykłe,
- pliki specjalne,
- katalogi,
- dowiązania symboliczne,
- potoki nazwane FIFO (ang. named pipe),
- gniazda (ang. UNIX--domain sockets).

W plikach zwykłych przechowujemy programy, dane, teksty, grafikę, itp. W systemie UNIX pliki zwykłe nie mają ustalonego formatu. Plik zwykły jest po prostu ciągiem bajtów o danej długości. Oczywiście aplikacje mogą tworzyć pliki o ścisłe ustalonym formacie, na przykład plik assembler generuje plik, w którym wyróżniamy: nagłówek, kod wykonywalny, inicjalizowane dane.

Pliki specjalne, nazywane również plikami urządzeń zapewniają łączność z urządzeniami, na przykład z dyskami, terminalami, napędami taśmy, itp. W plikach specjalnych nie przechowuje się żadnych danych. Pliki te charakteryzują sposób działania urządzenia, wskazują miejsce podłączenia urządzeń do systemu oraz zapewniają dostęp do programów obsługi urządzeń („drajwerów”).

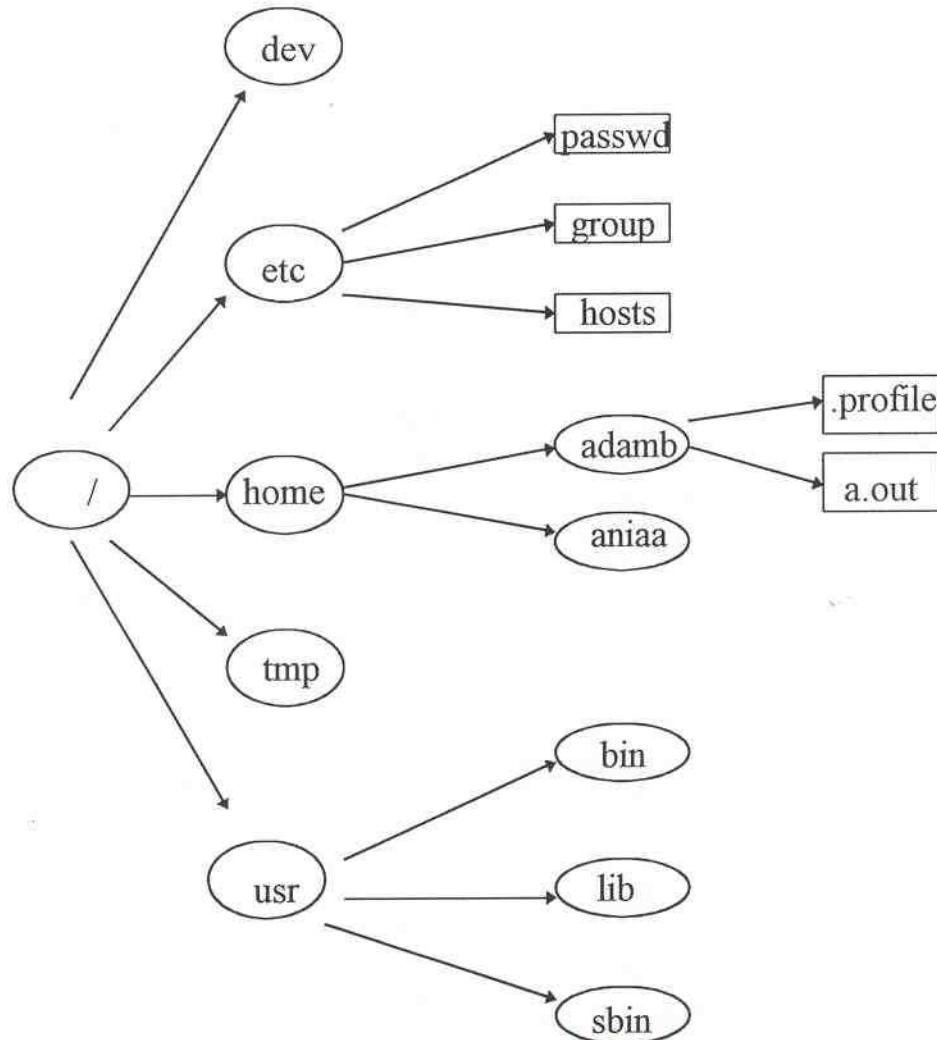
Katalogi służą do powiązania nazw plików z danymi znajdującymi się na dysku. W każdym katalogu może znajdować się pewna liczba plików i innych katalogów (podkatalogów). Katalog jest przechowywany jak plik zwykły i (w uproszczeniu) ma postać tabeli o dwóch kolumnach. Każdy wiersz tej tabeli zawiera nazwę pliku znajdującego się w katalogu (lub podkatalogu) oraz pewien numer, pozwalający na odszukanie atrybutów pliku i danych, które się w nim znajdują.

Dowiązania zwykłe mogą być tworzone w obrębie tego samego systemu plików. **Dowiązań symbolicznych** używa się ponad granicami systemów plików oraz w odniesieniu do katalogów. Przechowywane są w nich ścieżki dostępu do plików lub katalogów, na które dowiązania te wskazują.

Potoki nazwane (FIFO) wykorzystywane są do komunikacji między procesami. Do tworzenia tych potoków wykorzystywane są odpowiednie procedury biblioteczne. Procesy mogą otwierać potoki nazwane do odczytu i zapisu, tak jak otwierają pliki zwykłe.

Gniazda wprowadzone zostały w systemie BSD UNIX. Są wykorzystywane również do komunikacji między procesami. Wykorzystują jednak inne mechanizmy niż potoki nazwane.

Struktura (drzewo) katalogów typowa dla systemu UNIX



Ścieżka dostępu

Jeden z katalogów zawsze służy użytkownikowi pracującemu w systemie UNIX jako katalog bieżący.

Położenie pliku lub katalogu w drzewie katalogów określa ścieżka dostępu do pliku lub katalogu.

Bezwzględna ścieżka dostępu określa położenie pliku lub katalogu względem katalogu głównego (ang. root directory) /, na przykład:

/etc/passwd
/home/adamb/.profile

Względna ścieżka dostępu określa położenie pliku lub katalogu względem katalogu bieżącego, na przykład (jeśli katalogiem bieżącym jest /home):

adamb
adamb/a.out
../usr/lib

Operacje dotyczące katalogów

zmiana katalogu bieżącego,

cd [ścieżka dostępu do katalogu]

ustalenie nazwy katalogu bieżącego,

pwd

sprawdzenie zawartości katalogu,

ls [-opcje] [ścieżka dostępu do katalogu]

tworzenie katalogów,

mkdir [-opcje] [ścieżka dostępu do katalogu]

usuwanie katalogów

rmdir [-opcje] [ścieżka dostępu do katalogu]

utworzenie dowiązania do katalogu:

ln -s stara_nazwa nowa_nazwa

Operacje odnoszące się do plików:

wypisanie zawartości pliku tekstowego: `cat [ściezka dostępu do pliku]`
`more [ściezka dostępu do pliku]`

drukowanie pliku: `lp [-opcje] [ściezka dostępu do pliku]`

wypisanie atrybutów pliku: `ls [-opcje] [ściezka dostępu do pliku]`

kopiowanie pliku: `cp [-opcje] co_kopiujemy dokąd_kopiujemy`

usuwanie pliku: `rm [-opcje] nazwa_pliku_lub_katalogu`

zmiana nazwy pliku lub jego przeniesienie:
`mv [-opcje] stara_nazwa nowa_nazwa`

utworzenie dowiązania do pliku lub katalogu:
`ln [-opcje] stara_nazwa nowa_nazwa`

Atrybuty plików i katalogów:

typ pliku,
prawa dostępu do pliku,
liczba dowiązań do pliku,
identyfikator właściciela,
identyfikator grupy,
rozmiar pliku w bajtach,
czas ostatniej modyfikacji pliku,
czas ostatniego dostępu do pliku,
czas ostatniej zmiany informacji w i-węźle,
nazwa pliku.

Przykład:

```
$ ls -ld /etc
drwxr-xr-x 22 root root 1024 Sep 1995 /etc
```

Każdemu plikowi przyporządkowany jest i-węzeł (ang. *i.node*), który jest rekordem przechowującym większość informacji o pliku.

Zawartość i-węzła:

- typ pliku,
- prawa dostępu do pliku,
- liczba dowiązań do pliku,
- identyfikator właściciela,
- identyfikator grupy,
- rozmiar pliku w bajtach,
- czas ostatniej modyfikacji pliku,
- czas ostatniego dostępu do pliku,
- czas ostatniej zmiany informacji w i.węźle,
- 12 wskaźników zawierających adresy bloków z danymi pliku (bloki bezpośrednio adresowane),
- wskaźnik zawierający adres bloku, w którym przechowywane są adresy bloków z danymi (adresowanie pośrednie jednostopniowe),
- wskaźnik zawierający adresy bloków, w których przechowywane są adresy bloków z adresami bloków z danymi (adresowanie pośrednie dwustopniowe),
- wskaźnik wykorzystywany w adresowaniu pośrednim trzystopniowym.

i-węzły są tworzone wtedy, gdy tworzony jest system plików.

Liczba i-węzłów w systemie plików zależy od jego rozmiaru oraz założonego średniego rozmiaru pliku (np. 2kB lub 6kB).

Każdy i-węzeł zajmuje 128 bajtów. i-węzły tworzą tablicę i-węzłów.

Poszczególne i-węzły identyfikowane są przez numery, określające ich położenie w tablicy i-węzłów.

Aby sprawdzić, jaki i-węzeł został przyporządkowany danemu plikowi, należy w poleceniu ls użyć opcji -i.

Przykład:

```
$ ls -lid /etc
```

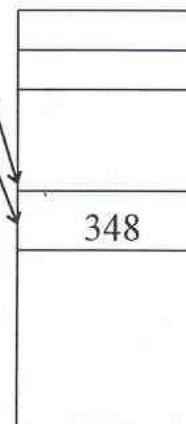
```
77 drwxr-xr-x 22 root root 1024 Sep 1995 /etc
```

Nazwy plików są przechowywane w katalogach, łącznie z numerami odpowiadającymi tym plikom i-węzłów. Dzięki temu możliwe jest odczytanie atrybutów pliku oraz odszukanie przechowywanych w nim danych.

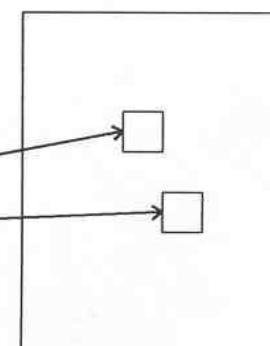
Katalog

234	.
112	..
348	plik1
0	plik2
355	plik3
348	plika

Tablica i-węzłów



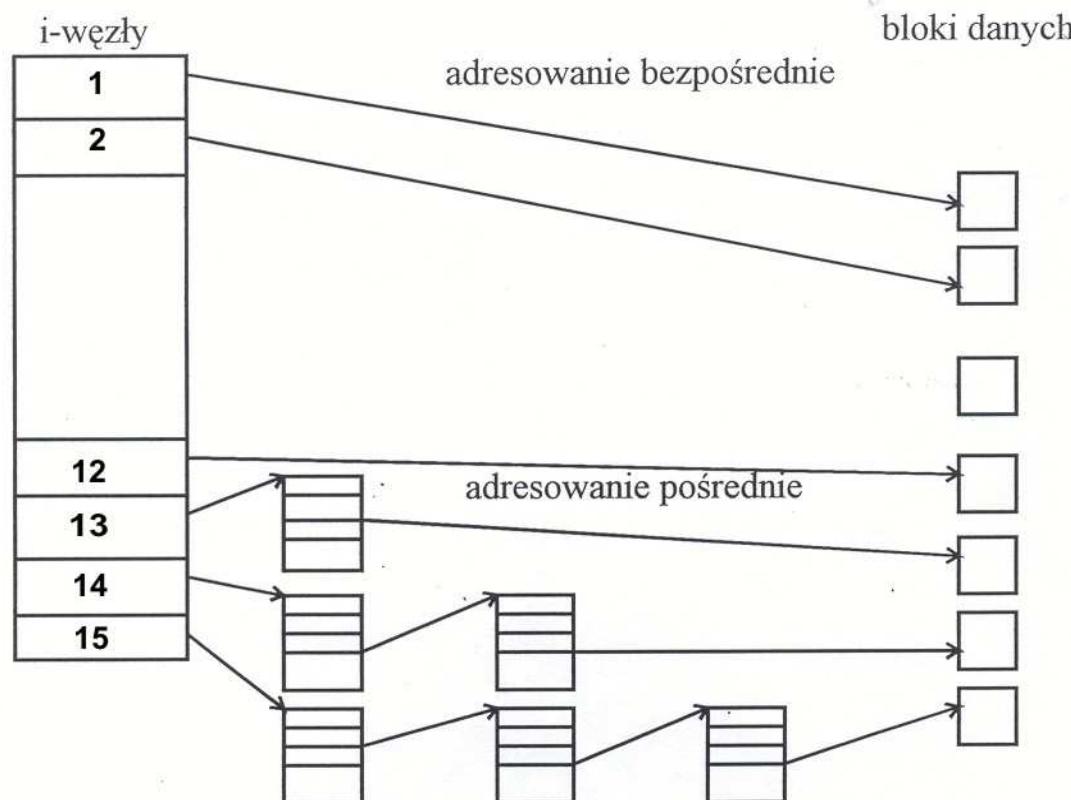
Dysk



Blok identyfikacyjny (ang. Superblock) zawiera między innymi:

1. rozmiar systemu plików,
2. liczbę wolnych bloków w systemie plików,
3. listę wolnych bloków dostępnych w systemie plików,
4. indeks następnego wolnego bloku na liście wolnych bloków,
5. rozmiar tablicy i-węzłów,
6. liczbę wolnych i-węzłów w systemie plików,
7. listę wolnych i-węzłów w systemie plików,
8. indeks następnego wolnego i-węzła na liście wolnych i-węzłów.

Adresowanie bloków danych



Adresowanie tylko bezpośrednie nie jest efektywne i znacznie ograniczyłoby rozmiary plików.

W systemie UNIX zastosowano następujące reguły:

- tablica adresów przechowywana w i-węźle ma 15 elementów (wskaźników) i każdy zajmuje 4 bajty,
- 12 pierwszych wskaźników zawiera adresy bloków z danymi,
- następny, 13 wskaźnik zawiera adres bloku, w którym znajdują się adresy bloków z danymi,
- 14 wskaźnik to adres bloku, w którym umieszczane są adresy bloków zawierających adresy bloków z danymi,
- 15 wskaźnik to adres bloku, w którym umieszczane są adresy bloków przeznaczonych na adresy kolejnych bloków zawierających adresy bloków z danymi.

Przykład: jeśli blok zajmuje 4 kB, to

- adresowanie bezpośrednie pozwala na zaadresowanie danych plików o rozmiarach nie przekraczających: 48 kB,
- adresowanie pośrednie pozwala na zaadresowanie danych plików o rozmiarach nie przekraczających: $48 \text{ kB} + 1024 * 4 \text{ kB} = 4144 \text{ kB}$,
- podwójne adresowanie pośrednie pozwala na zaadresowanie danych plików o rozmiarach nie przekraczających: $48 \text{ kB} + 1024 * 4 \text{ kB} + 1024 * 1024 * 4 \text{ kB} = 4198448 \text{ kB}$,
- adresowanie pośrednie pozwala na zaadresowanie danych plików o rozmiarach nie przekraczających: $48 \text{ kB} + 1024 * 4 \text{ kB} + 1024 * 1024 * 4 \text{ kB} + 1024 * 1024 * 1024 * 4 \text{ kB}$,

Sposób przechowywania plików na dysku

Do przechowywania danych na dysku system UNIX używa bloków i fragmentów. Fragment jest najmniejszą jednostką przestrzeni dyskowej zajmowanej przez plik. Rozmiar bloku jest całkowitą wielokrotnością rozmiaru fragmentu (stosunek rozmiaru bloku do rozmiaru fragmentu nie może jednak przekraczać 8): np. rozmiar bloku wynosi 8 kB a rozmiar fragmentu 2kB.

Reguły przydzielania bloków i fragmentów:

1. Jeśli rozmiar pliku jest mniejszy niż rozmiar fragmentu, plikowi temu przydzielany jest pierwszy wolny fragment.
2. Jeśli rozmiar pliku jest większy niż rozmiar fragmentu, ale mniejszy niż rozmiar bloku, plikowi temu przydzielane są kolejne fragmenty należące do tego samego bloku.
3. Jeśli rozmiar pliku jest większy niż rozmiar bloku, to plikowi przydzielana jest odpowiednia liczba bloków, niekoniecznie znajdujących się obok siebie, o łącznym rozmiarze nie przekraczającym rozmiaru pliku.

Pozostała część pliku umieszczana jest zgodnie z regułami 1 oraz 2.

Do przechowywania informacji o stanie wolnych bloków i fragmentów można wykorzystać mapę bitową, na przykład dla bloków 8 kB i fragmentów 2 kB :

1110	1110	0100	0011	0000	0001
blok 0	blok 1	blok 2	blok 3	blok 4	blok 5 ... -.

0 - fragment wolny

1 - fragment zajęty

Struktura systemu plików

Blok identyfikacyjny

Tablica i-węzłów

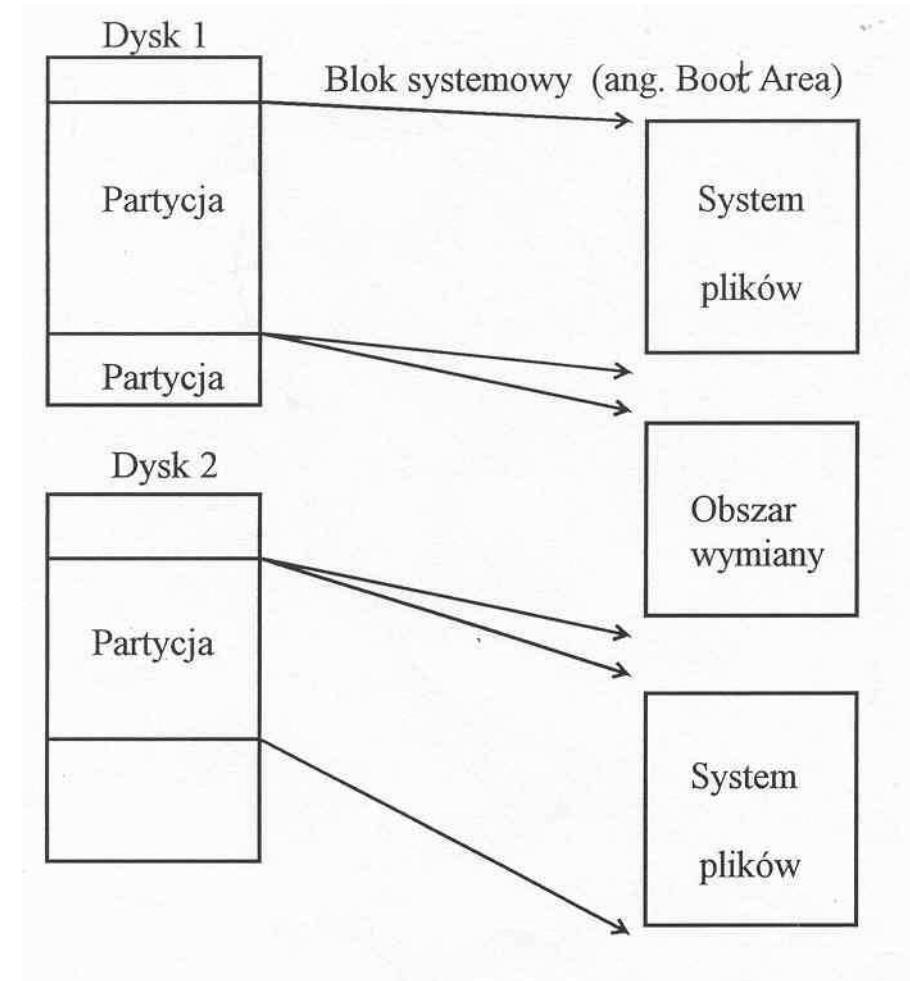
Bloki danych

Podział dysków na partycje

Dyski bywają dzielone na sekcje fizyczne. Taki podział ma szereg wad. Do najważniejszych należą:

- nieefektywne wykorzystanie miejsca na dysku,
- rozmiar sekcji (a w wyniku rozmiar systemu plików nie większy niż rozmiar dysku),
- nie można zmienić rozmiarów sekcji (systemu plików)

Niektóre implementacje systemu UNIX wykorzystują partycje logiczne. LVM. Każda partycja (sekcja) jest traktowana jako niezależny wirtualny dysk. Jest reprezentowana przez plik specjalny (urządzenia), podobnie jak każdy dysk

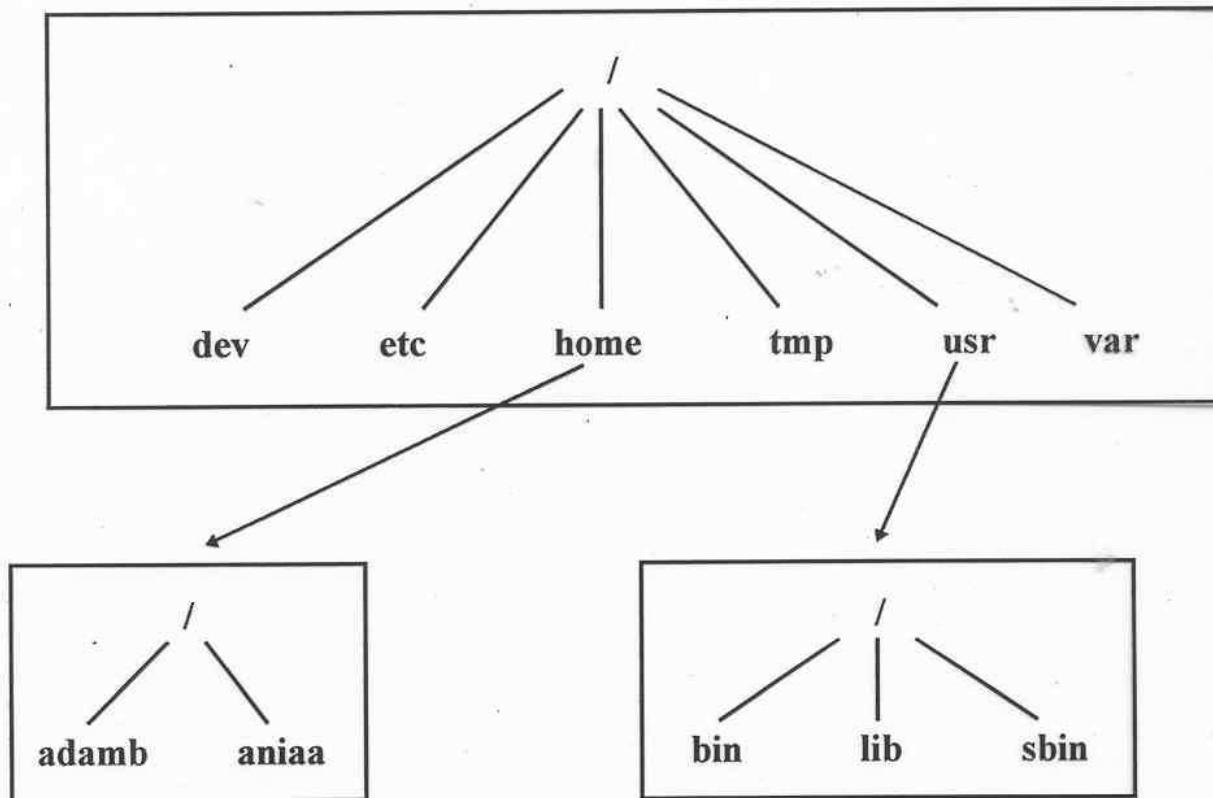


W partycjach tworzone są systemy plików. Można je również wykorzystać jako obszary wymiany (swap).

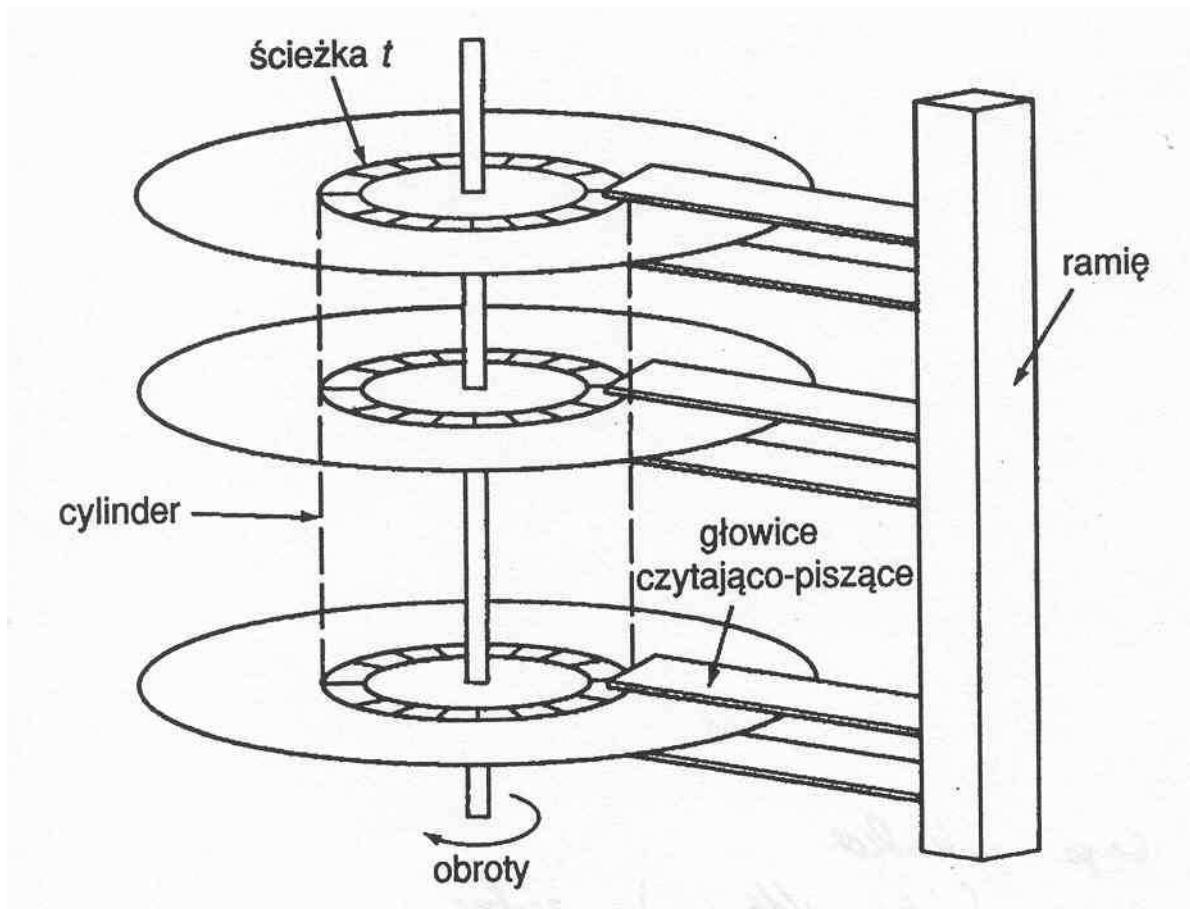
Systemy plików utworzone w poszczególnych partycjach muszą być dowiązane do głównego systemu plików:

mount plik_specjalny_reprezentujący_partycję katalog

Główny system plików (ang. Root file System)



Operacja odwrotna: **umount plik_specjalny_reprezentujący_partycję
umount katalog**



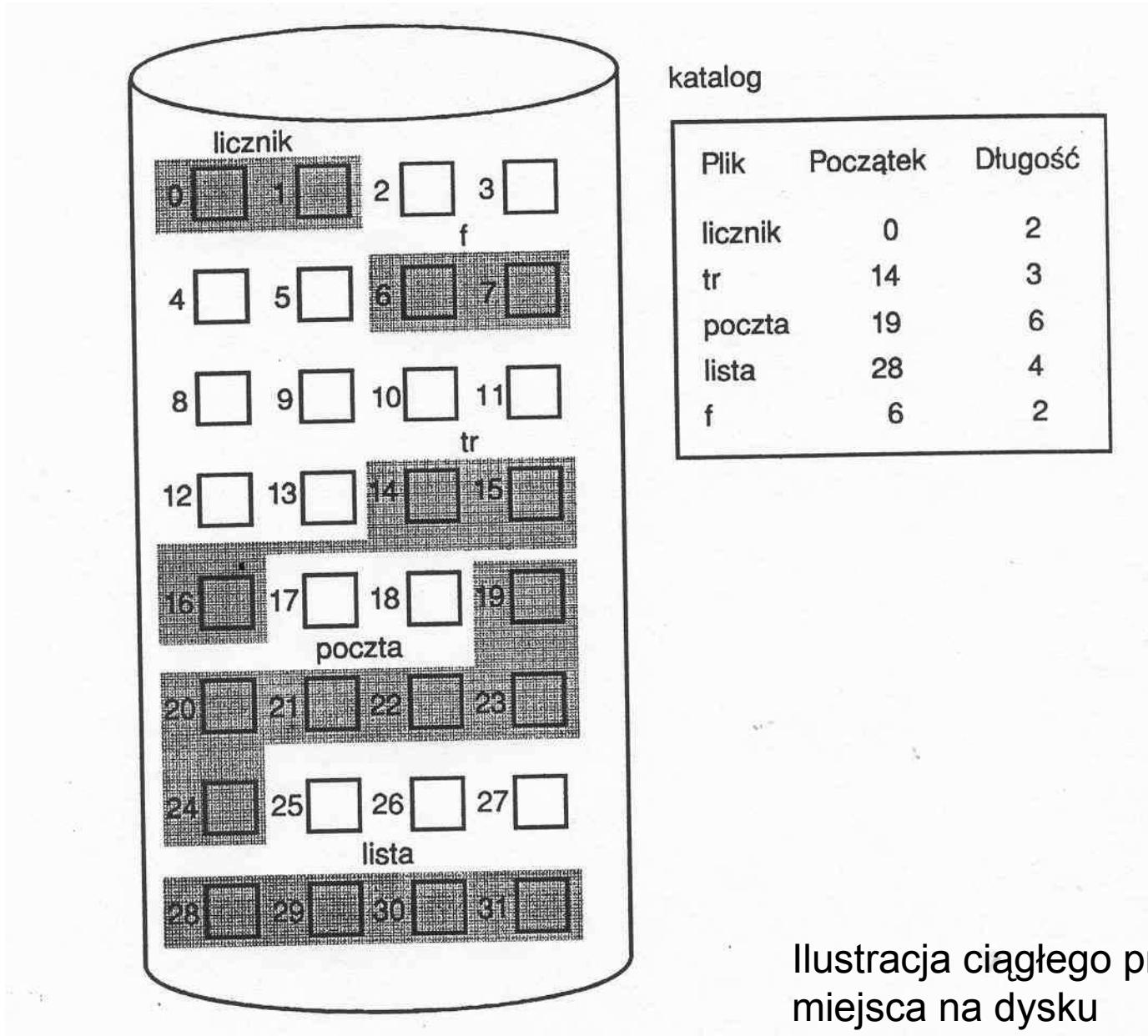
Mechanizm dysku z ruchomymi głowicami

Różne sposoby przydzielu miejsca na dysku

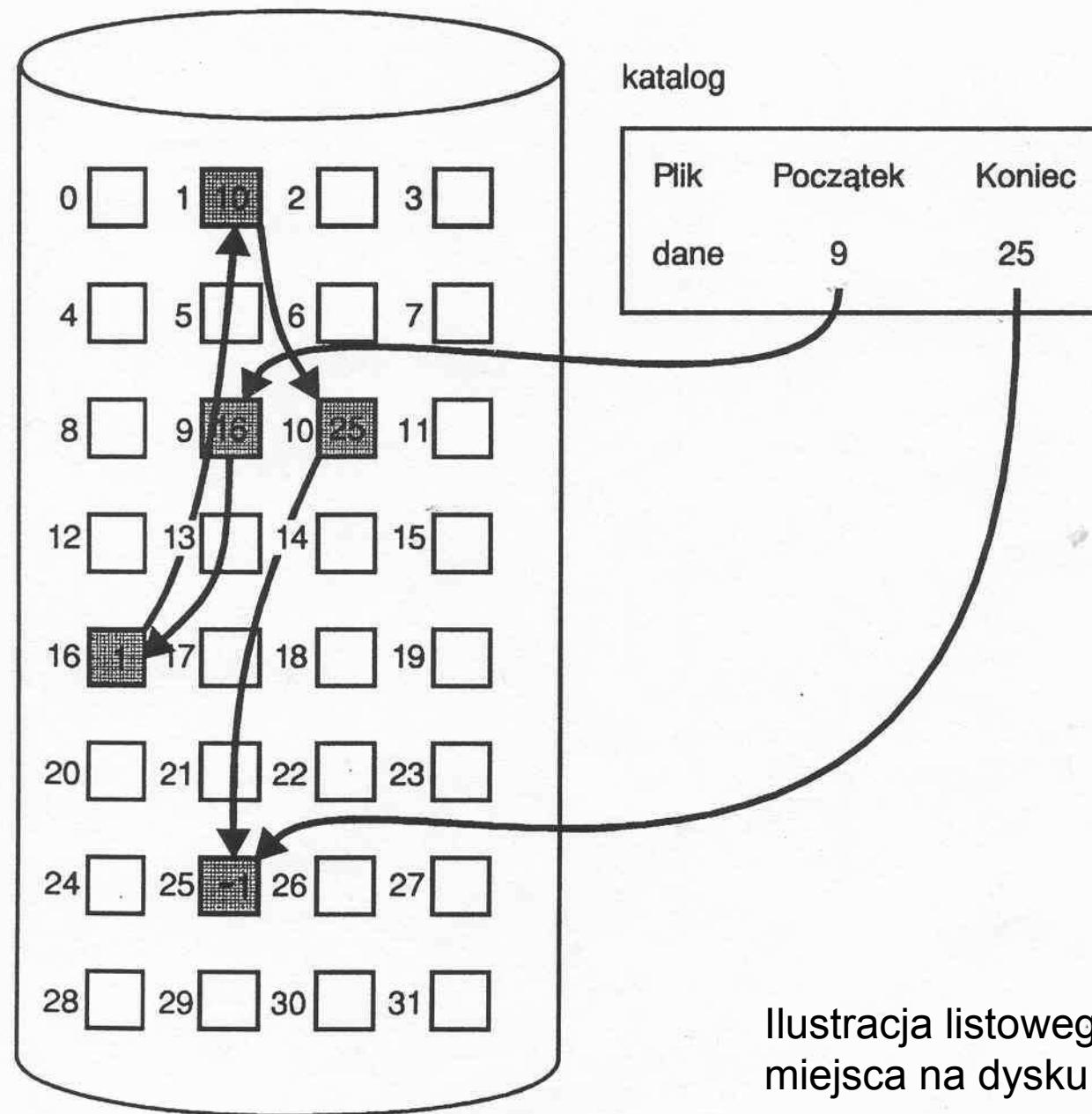
- Przydział ciągły
- Przydział listowy
- Przydział indeksowy

Metody dostępu do informacji pliku

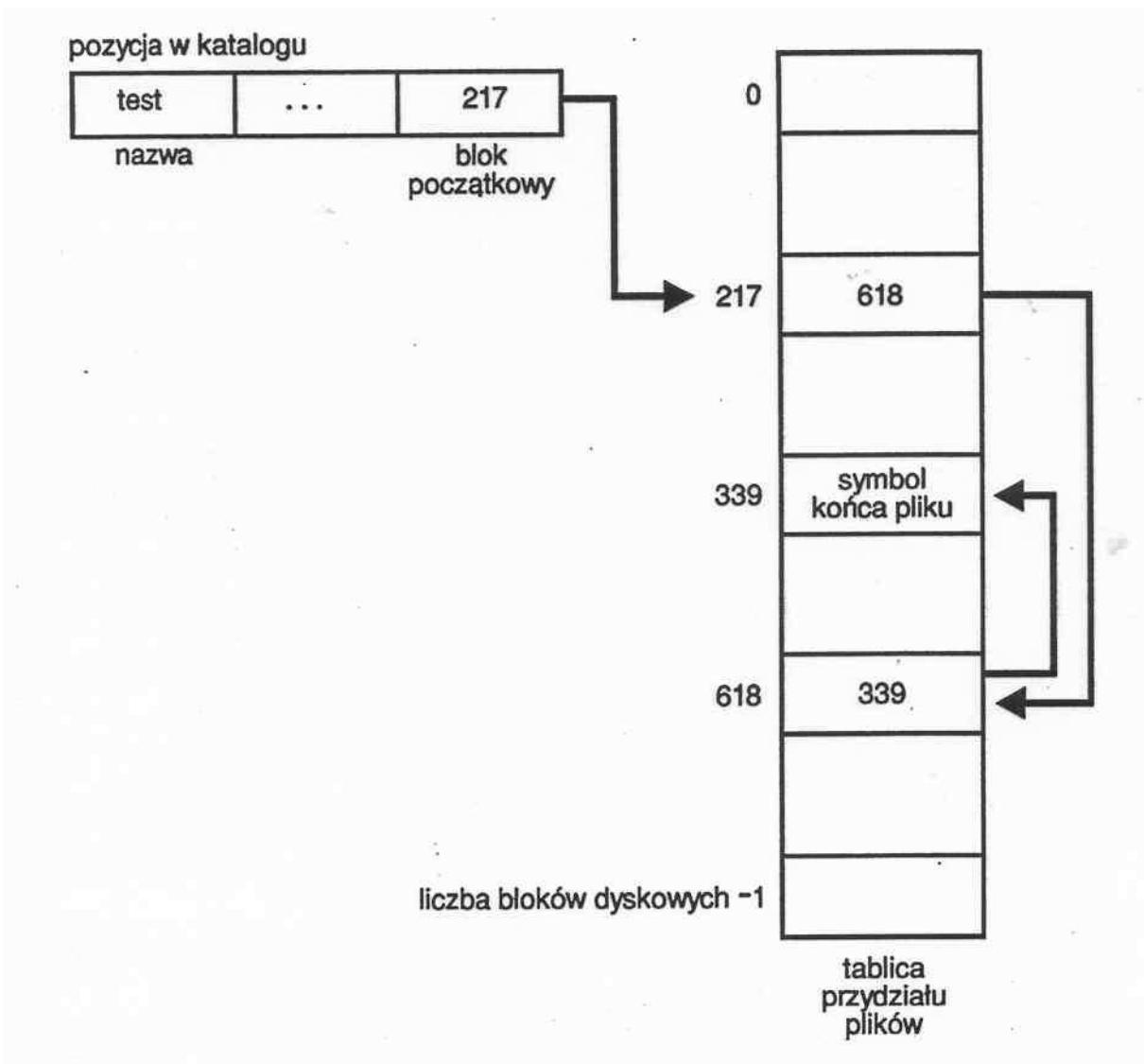
- Dostęp sekwencyjny
- Dostęp bezpośredni (swobodny)



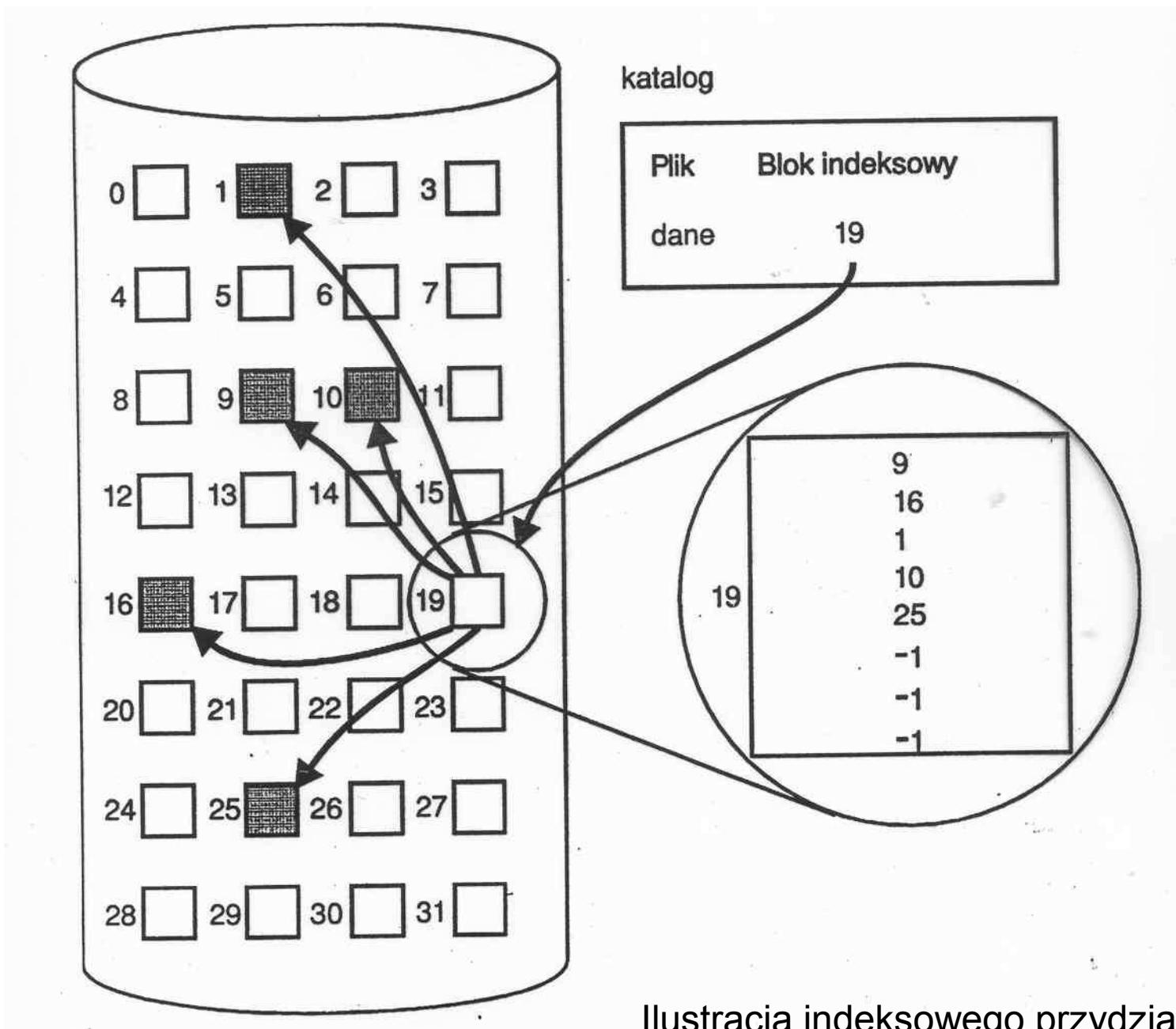
Ilustracja ciągłego przydziału
miejsc na dysku



Ilustracja listowego przydziału
miejsc na dysku



Tablica przydziału plików (File Allocation Table)



Ilustracja indeksowego przydziału miejsca na dysku

Zarządzanie procesami

(omawiane zagadnienia)

Pojęcie procesu

Stany procesu

Blok kontrolny procesu

Tworzenie procesu

Sygnały

Kończenie wykonania procesu

Działanie interpretatora poleceń (shell-a)

Koncepcja wątków

Planowanie - szeregowanie procesów (scheduling)

Diagram kolejek

Kryteria planowania

Przykłady algorytmów planowania (szeregowania)

Szeregowanie procesów w systemie UNIX

Priorytety procesów

Ładowanie systemu operacyjnego (bootstrap) w systemie UNIX.

Otwieranie sesji użytkownika

Zamykanie systemu (shutdown)

Pojęcie procesu

Proces jest to wykonywany program.

Składa się z kodu programu i przydzielonego mu obszaru pamięci.

Program, zawarty w pliku przechowywanym na dysku, ma charakter bierny.

Proces z licznikiem rozkazów określającym kolejny rozkaz do wykonania ma charakter aktywny. Ten sam program może być wykonywany jako kilka niezależnych procesów, np. program vi wywołany przez dwóch użytkowników.

W systemie wielodostępnym, pracującym z podziałem czasu jest jednocześnie wiele procesów. Procesy działają współbieżnie co oznacza konieczność podziału zdolności obliczeniowej procesora (przełączanie) między poszczególne procesy, ponieważ w każdej chwili czasu wykonywany jest tylko jeden proces.

Proces wykonywany jest w sposób sekwencyjny tzn. w każdej chwili na zamówienie procesu wykonywany jest jeden rozkaz kodu programu.

Procesy można podzielić na:

procesy systemu operacyjnego wykonujące program (zadania) systemu,
procesy użytkowe wykonujące programy użytkowników.

Procesy wykonywane są w trybie:

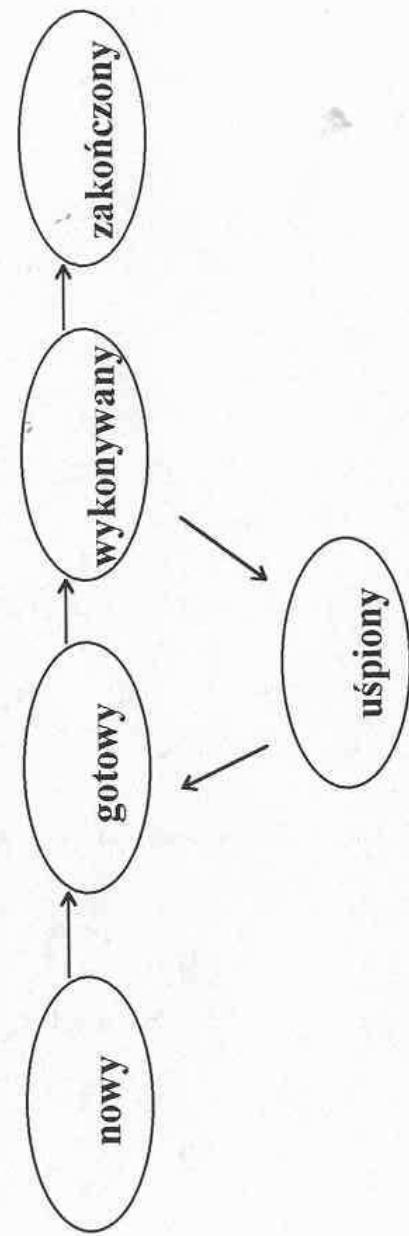
użytkownika
jądra systemu.

Proces obejmuje:

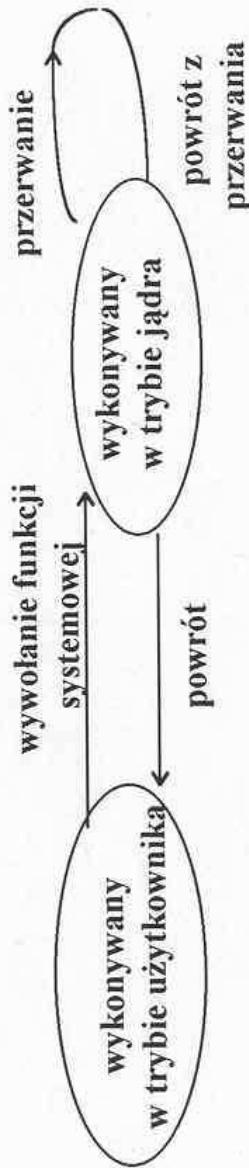
- zakodowany program (tzw. sekcja tekstu),
- bieżącą czynność - reprezentowaną przez aktualną wartość licznika rozkazów,
- stos procesu, w którym przechowywane są dane tymczasowe (parametry procedur, adresy powrotne, zmienne tymczasowe)
- sekcja danych, w której przechowywane są zmienne

Stany procesu

- **nowy**: proces jest tworzony,
- **wykonywany**: w trybie użytkownika - wykonywane są instrukcje,
w trybie jądra - wykonywane są instrukcje,
- **gotowy do wykonywania** - proces czeka na przydział procesora przez program szeregujący, w tym stanie jest zwykle wiele procesów,
- **czekający (uśpiony)** - proces czeka na wystąpienie zdarzenia niezbędnego do jego dalszego wykonywania, np. na zakończenie operacji we/wy.,
- **zakończony** - proces zakończył wykonywanie.



Schemat przejścia między stanami procesów



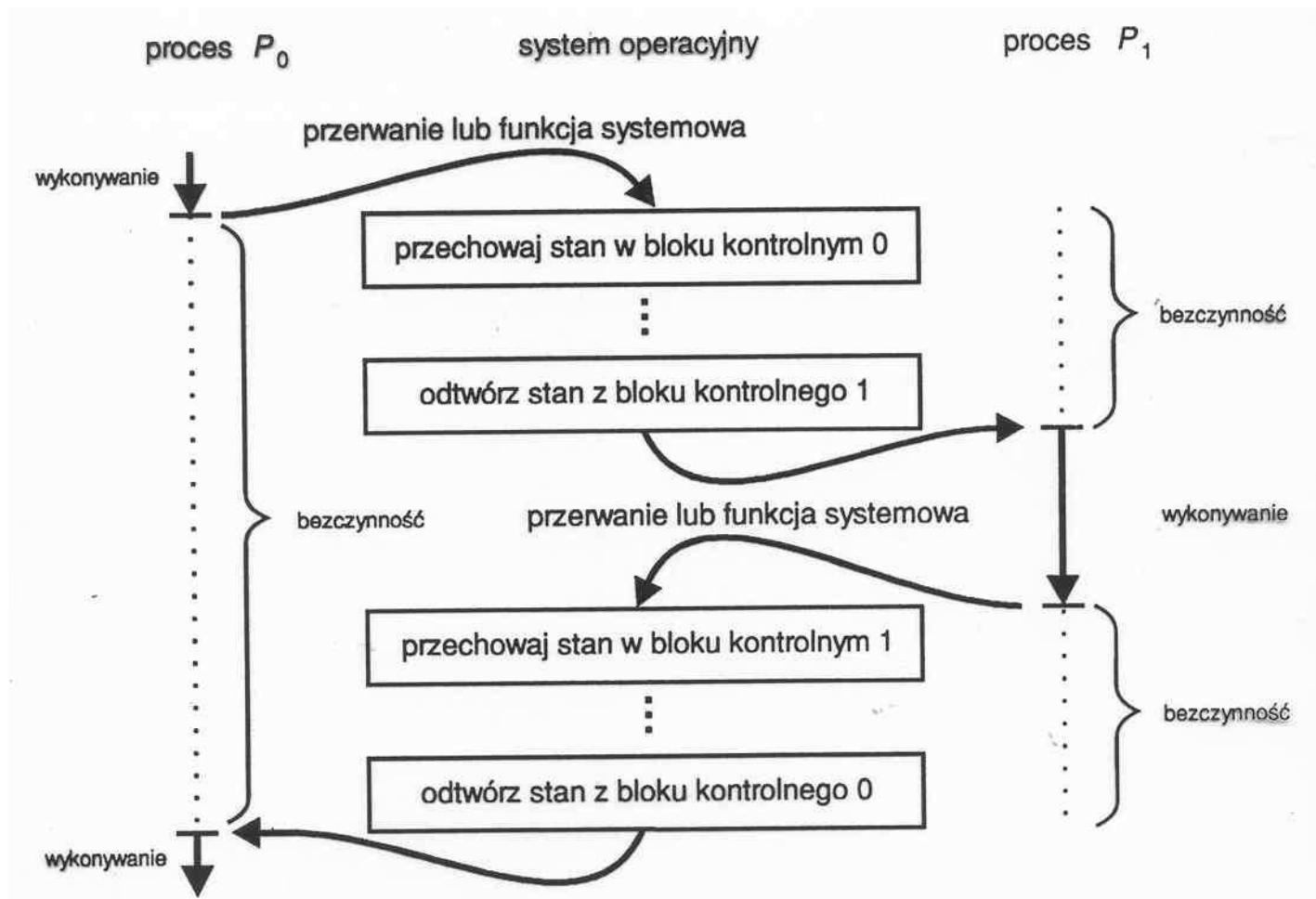
Schemat przejścia między stanami wykonywanego procesu

Blok kontrolny procesu

Zawiera informacje obejmujące:

- Stan procesu.
- Licznik rozkazów - wskazuje adres następnej instrukcji do wykonania.
- Rejestry procesora - w zależności od architektury komputera: akumulatory, rejestr indeksowe, wskaźniki stosu, rejestr ogólnego przeznaczenia, rejestr warunków. Blok kontrolny zawiera informacje pamiętane w rejestrach w celu kontynuacji wykonywania procesu po przerwaniu.
- Informacje związane z planowaniem przydziału czasu procesora – priorytet procesu, wskaźniki do kolejek porządkujących, inne parametry.
- Informacje dotyczące zarządzania pamięcią
- Informacje do rozliczeń - zużyty czas procesora, czas rzeczywisty, ograniczenia czasu, numery konta, zadań, procesów.
- Informacje o stanie operacji wejścia wyjścia - lista otwartych plików, wykaz urządzeń przydzielonych do procesu, informacja o niezrealizowanych zamówieniach na operacje we/wy.

Przykład przełączania procesora między procesami P_0 i P_1



Tworzenie procesu

- Nowy proces tworzony jest przez wywołanie funkcji `fork`
- Proces wywołujący tę funkcję zwany jest procesem macierzystym, a proces utworzony - potomkiem.
- Potomek uzyskuje pozycję w tablicy procesów.
- Uzyskuje identyfikator (**PID**).
- Tworzona jest kopia kontekstu procesu macierzystego (nowy proces zawiera kopię przestrzeni adresowej procesu macierzystego).
- Uaktualnione zostają liczniki odwołań do plików i i-węzłów, z którymi związany jest potomek.
- Identyfikator potomka przekazywany jest procesowi macierzystemu,
- Potomek dziedziczy prawa dostępu do otwartych plików, a także dzieli dostęp do plików. Procesy mają identyczne kopie segmentów instrukcji, danych i stosu.
- Po wywołaniu funkcji `fork` wywoływana jest zwykle funkcja `exec`.
- Funkcja `exec` służy do wykonania określonego, innego programu. Umieszcza w obszarze pamięci procesu kopię pliku wykonywalnego i rozpoczyna jego wykonywanie. Zawartość kontekstu (obraz pamięci) procesu wywołującego funkcję `exec` zostaje zamazana.

Proces

- Jednostka wykonywania w systemie Unix.
- Zawiera instrukcje programu oraz jego środowisko.
- Kiedy użytkownik rozpoczyna sesję, jądro przydziela mu proces interpretatora poleceń - login shell.
- Proces może powołać do życia inny proces - proces potomny.
- Proces, który utworzył dany proces jest dla niego procesem macierzystym lub procesem przodkiem.
- Proces ma jeden proces macierzysty, lecz może mieć wiele procesów potomnych.
- Jądro identyfikuje proces za pomocą numeru nazywanego identyfikatorem procesu PID.
- Identyfikator procesu przodka danego procesu oznaczany jest w skrócie PPID.
- Proces `init`, którego PID jest równe 1 jest przodkiem wielu innych procesów, np. interpretatora poleceń przydzielanego użytkownikom w momencie rozpoczętia sesji.
- Proces demon jest procesem systemowym wykonywanym w tle - nie ma przydzielonego terminala, ani nie jest związany z żadnym shelliem.

Tworzenie procesu przy wykonywaniu polecenia zewnętrznego

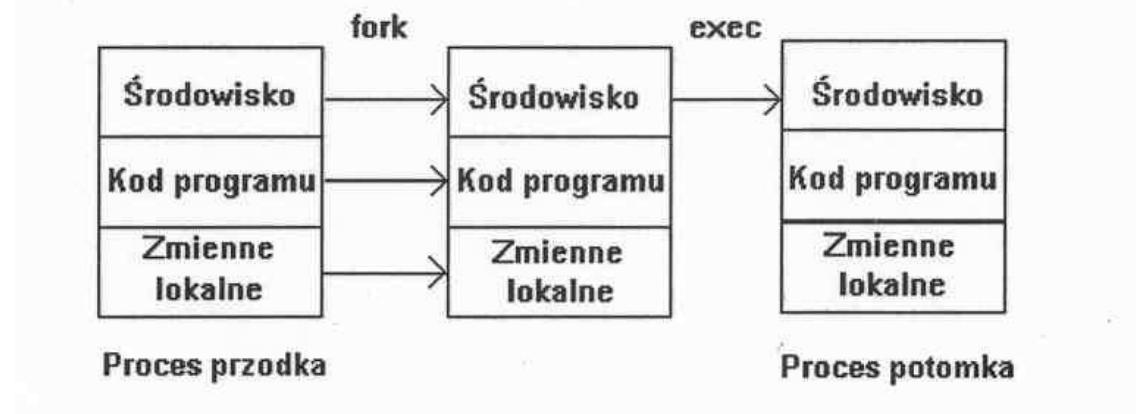
Proces to wykonywany program, tzn. program, który został rozpoczęty i jeszcze nie zakończył się.

Proces składa się z kodu programu i przydzielonego mu obszaru pamięci.

Polecenie wykonywane jest dwuetapowo:

1. tworzony jest nowy proces (proces potomka), który ma wykonać polecenie; jest to kopia bieżącego procesu; mechanizm ten nosi nazwę rozwidlania (ang. *fork, forking a process*)
2. nowy proces jest przekształcany w proces wykonujący polecenie; mechanizm nosi nazwę wykonywania (ang. *executing a command*).

Dziedziczone przez proces potomny środowisko obejmuje informacje o właścielcu i grupie, o standardowych strumieniach, do których przyłączony jest terminal, otwartych plikach, bieżącym katalogu i zmiennych globalnych.



Sygnały

Sygnały służą do informowania procesów o zdarzeniach asynchronicznych.

Sygnały związane są z:

- zakończeniem procesu (gdy proces wykonuje exit)
- wyjątkowymi sytuacjami spowodowanymi przez proces - próbą dostępu do niedozwolonego obszaru pamięci, próbą wykonania niedozwolonej instrukcji, błędami sprzętowymi,
- niespodziewanymi błędami przy wykonywaniu funkcji systemowej, np. pisaniu do potoku, który nie czyta, są także sygnały:
- wysyłane przez proces w trybie użytkownika, związane z interakcją z terminalem.

Użytkownicy mogą wysyłać sygnały stosując polecenie kill,

składnia: kill [-s nazwa sygnału | numer sygnału] PID [PID],

gdzie PID jest identyfikatorem procesu do którego wysyła się sygnał.

Niektóre sygnały mogą być wysyłane z klawiatury.

Obsługa sygnałów jest następująca: proces, który otrzymał sygnał, wykonuje exit, albo ignoruje sygnał, albo wykonuje określoną funkcję użytkownika.

Sygnał nr 9 powoduje bezwarunkowe zakończenie procesu. Sygnały obsługiwane są przez jądro.

W przypadku procesu wykonywanego w trybie jądra, odbywa się to wtedy, gdy proces wraca z trybu jądra do trybu użytkownika.

W przypadku procesu w trybie użytkownika, jądro obsługuje sygnał bezpośrednio po obsłużeniu przerwania, które go wywołało.

Kończenie wykonywania procesów

Proces można przerwać wysyłając do niego odpowiedni sygnał. Można w tym celu skorzystać z polecenia `kill`, które powoduje wysyłanie sygnałów do procesów.
Składnia:

`kill [-s nazwa sygnału | -numer sygnału] PID [PID ...]`

gdzie PID to identyfikator procesu.

Polecenie `kill`, w którym pominięto nazwę i numer sygnału wysyła do wskazanego procesu sygnał o numerze 15 (SIGTERM). Taki sygnał może być przechwycony, zignorowany lub zablokowany.

Sygnałem, którego nie można przechwycić, zignorować, ani zablokować jest sygnał o numerze 9 (SIGKILL).

Polecenie:

`kill -9 PID`

zawsze zakończy proces o podanym PID.

Przykłady sygnałów:

- | | |
|---------|---|
| 1 HUP | zerwanie łączności |
| 2 INT | przerwanie (wysyłany również z klawiatury: Ctrl + C) |
| 3 QUIT | zakończenie (wysyłany również z klawiatury: Ctrl + \) |
| 9 KILL | bezwarunkowe zakończenie procesu |
| 15 TERM | programowe zakończenie procesu |

Kończenie wykonania procesu

Proces kończy się w naturalny sposób, gdy wykona ostatnią instrukcję. Proces kończąc się wywołuje funkcję `exit`.

Proces przechodzi do stanu zombi, zwalnia swoje zasoby i oczyszcza kontekst, za wyjątkiem pozycji w tablicy procesów.

Proces macierzysty czekający na zakończenie potomka wykonuje funkcję `wait`.

Jądro szuka potomków takiego procesu, będących w stanie zombi. Jeśli znajdzie, to przekazuje PID potomka oraz parametr funkcji `exit` i dostarcza do procesu macierzystego.

Proces macierzysty uzyskuje informację, który z wielu możliwych potomków zakończył pracę.

Jądro zwalnia następnie pozycję procesu potomnego w tablicy procesów.

Podglądarkanie procesów

Polecenie `ps` (process status) pozwala uzyskać wiele informacji o procesach (w zależności od opcji),

np. `ps -ef` wyświetla informacje: identyfikator użytkownika, identyfikator procesu, identyfikator procesu (PID), identyfikator przodka (PPID), czas uruchomienia procesu i jego wykonywania i inne.

Podglądarkanie procesów - polecenie ps (process status)

Składnia: ps [opcje]

Polecenie ps bez opcji wyświetla skróconą informację o procesach związanych z używanym terminaliem.

Opcje pozwalają uzyskać

- informację o różnych atrybutach procesów,
- informację o innych procesach.

Wybrane opcje:

- e podaje listę wszystkich procesów w systemie
- f podaje pełną informację o procesach
- t terminal podaje informacje o procesach związanych z danym terminaliem
- u użytkownik podaje informacje o procesach, których właścicielem jest użytkownik

Przykład 1

\$ ps

PID	TTY	TIME	COMMAND
401	ttyOp2	0:00	ksh
428	ttyOp2	0:00	ps

gdzie:

- PID identyfikator procesu
- TTY nazwa terminala związanego z procesem
- TIME czas wykonywania procesu (minuty:sekundy)
- COMMAND nazwa polecenia

Przykład 2

```
$ ps -ef
UID  PID  PPID C STIME      TTY      TIME COMMAND
root  1      0  O Mar  8      ?      0:12  init
root 666      1  O 10:23:45  ttyOp5  0:00 /etc/getty  ttyOp5
as1  505      1  O 11:22:33  ttyOp1  0:00 -ksh
```

W tym przykładzie pokazano jedynie fragment długiej listy wszystkich procesów, w której:

UID	identyfikator użytkownika
PID	identyfikator procesu
PPID	identyfikator procesu przodka
C	parametr określający wykorzystanie procesora
STIME	czas uruchomienia procesu
TTY	nazwa terminala związanego z procesem
TIME	czas wykonywania procesu (minuty:sekundy)
COMMAND	nazwa polecenia

Wykonywanie polecenia

Shell dopuszcza trzy typy poleceń:

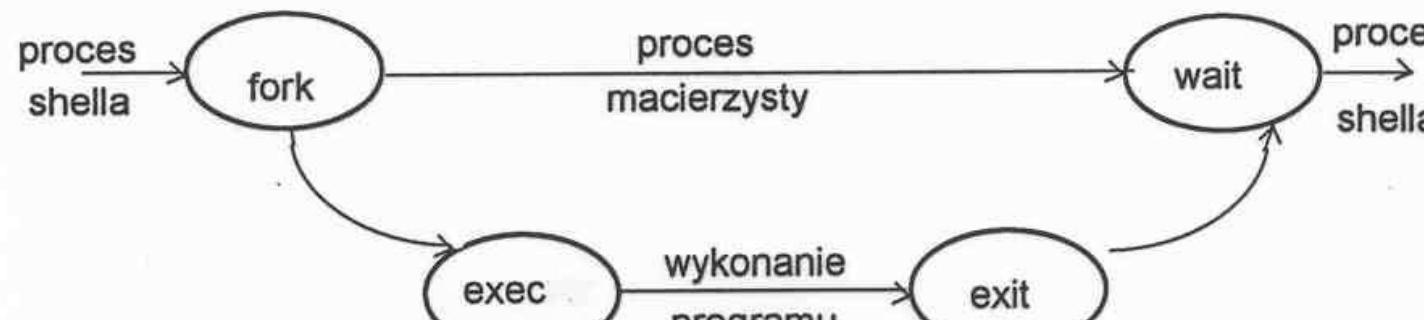
plik wykonywalny, który zawiera kod wynikowy skompilowanego programu

plik wykonywalny, który zawiera ciąg poleceń dla shella (nazywany skryptem)
wewnętrzne polecenia shella

Przed wykonaniem polecenia shell musi wiedzieć

- kto wykonuje polecenie (czy ma prawo do wykonania)
- gdzie umieszczone jest polecenie do wykonania

Schemat wykonywania poleceń w systemie Unix

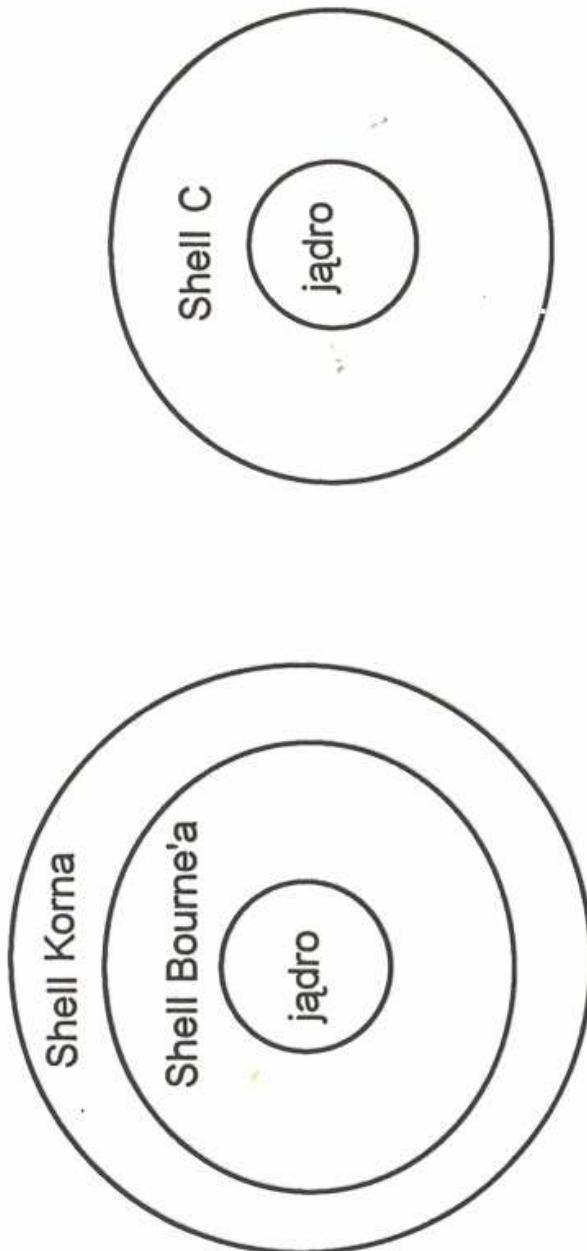


Co to jest Shell?

Jest to program, który pośredniczy między jądrem (*kernel*), systemem plików (*file system*) i programami usługowymi (*utilities*).



Podstawowe shelle



Podstawowe funkcje shella

- Przekazywanie sterowania do programu wybranego poleciem użytkownika
- Wykonywanie wbudowanych poleceń
- Dostarczenie języka do pisania skryptów
- Ustawianie środowiska pracy
- Przywoływanie i edycja uprzednio wydanych poleceń
- Przeadresowywanie wejścia - wyjścia poleceń
- Generowanie nazw plików
- Umożliwienie łączenia poleceń w potok
- Umożliwienie przetwarzania w drugim planie (nie interakcyjnie)

Shell jako interpretator polecień

Shell interpretuje pierwsze słowo w wierszu polecenia jako nazwę polecenia.

\$ ls -l plik? "test?"

1. Shell wczytuje polecenie do wewnętrznego bufora.

ls -l plik? "test?"

2. Polecenie jest dzielone na części nazywane słowami. Shell określa znaczenie każdego słowa.

ls -l plik? "test?"

3. Shell szuka i przetwarza znaki specjalne.

ls -l plik? "test?"

- Cudzysłów kończący
- Metaznak ujęty w cudzysłowy nie ma specjalnego znaczenia
- Cudzysłów początkowy
- Metaznak bez cudzysłowa ma znaczenie specjalne

4. Shell zleca wykonanie polecenia o postaci:

ls -l plik1 plik2 plik3 test?

5. "Uśpiony" czeka na zakończenie wykonywania polecenia.

6. Po zakończeniu wykonywania polecenia zgłasza gotowość przyjęcia nowego polecenia.

Polecenia wbudowane (wewnętrzne) i zewnętrzne

Polecenie wewnętrzne ksh
cd
exit

Polecenie zewnętrzne Unixa
ls
more

Polecenia wbudowane

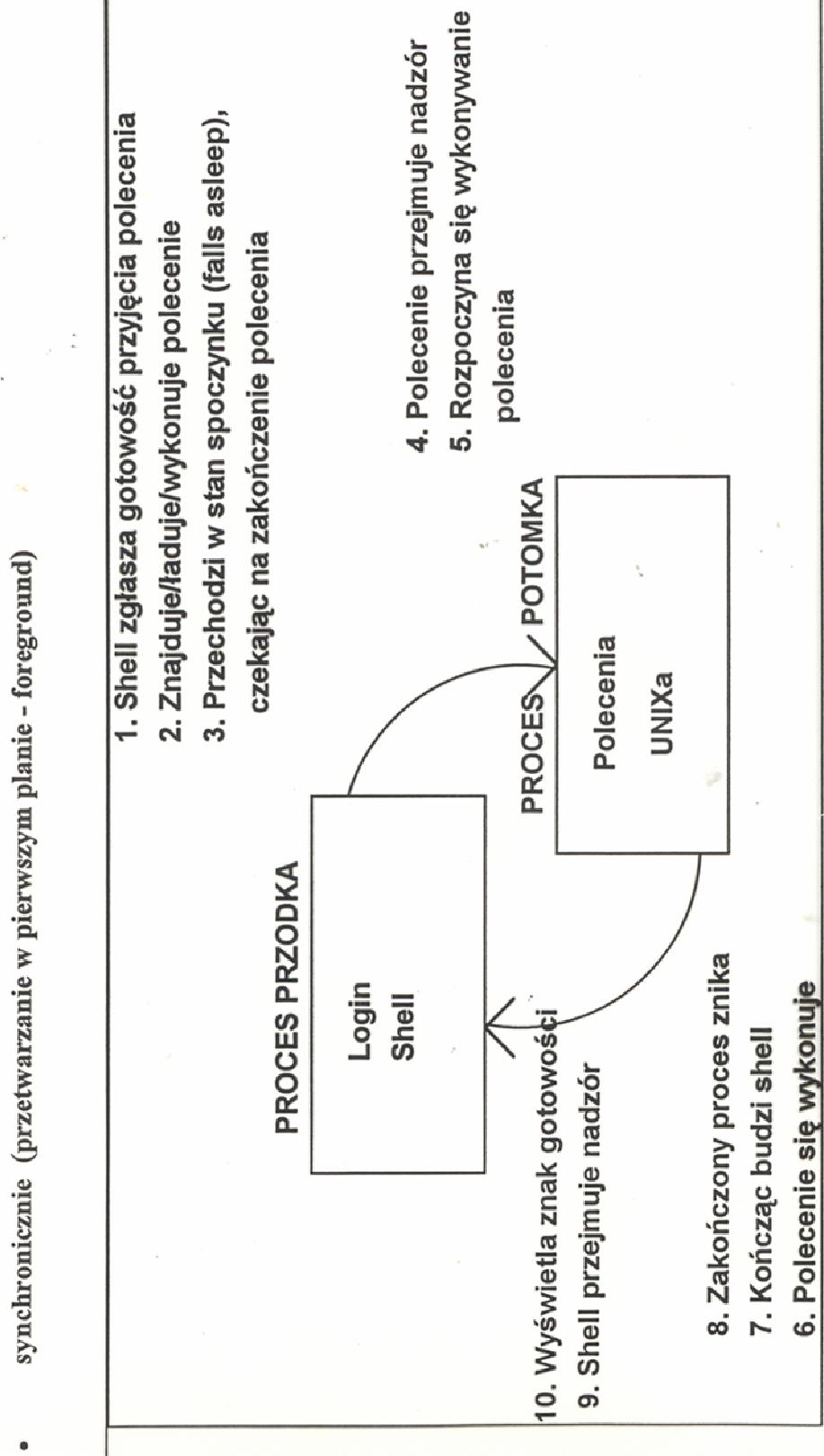
ich zestaw zależy od shella (inne w /bin/ksh i w /bin/csh)
wykonywane są wewnątrz shella - nie tworzą nowego procesu

Polecenia zewnętrzne

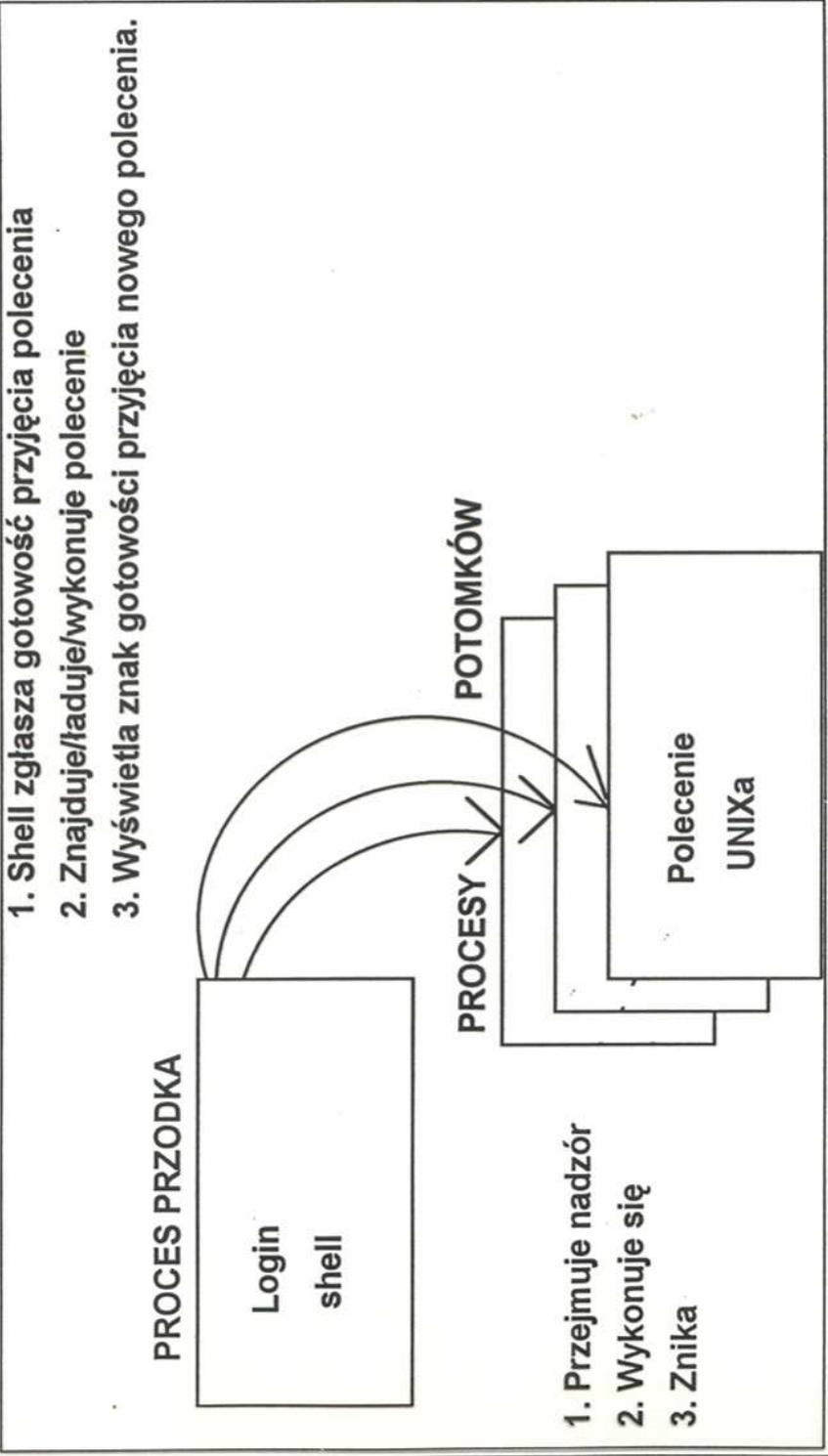
realizowane przez oddzielne programy
wymagają utworzenia nowego procesu
wymagają określenia katalogu, w którym są umieszczone

Skrypty

plik skryptu musi mieć prawo do czytania i wykonywania



- asynchronicznie (przetwarzanie w drugim planie - background)



Działanie interpretatora poleceń: shell-a

Interpretator wczytuje wiersz poleceń z pliku standardowego (terminala) i interpretuje go.

Polecenia proste

np. **who**, **ls**.

Interpretator (shell) wykonuje rozwidlenie (fork), a następnie funkcję (wait). Wykonywana jest funkcja exec w odniesieniu do procesu potomnego i proces ten wykonuje polecenie (program) umieszczone w wierszu polecenia. W tym czasie interpretator czeka na zakończenia procesu potomnego. Po jego zakończeniu wczytuje kolejne polecenie.

Uwaga: w przypadku poleceń wbudowanych (**cd**, **for**, **while**) interpretator wykonuje je bezpośrednio bez tworzenia nowych procesów.

Polecenie ze zmianą standardowego pliku wyjściowego

np. **ls -l > lista**

Proces potomny tworzy plik lista. Jeśli próba utworzenia pliku nie powiedzie się, to proces potomny kończy działanie. Jeśli plik zostanie utworzony, to proces potomny zamyka standardowy plik wyjściowy i powiela deskryptor nowego. Po zapisaniu informacji, zamyka plik wyjściowy i kończy działanie.

Działanie interpretatora poleceń: shell-a (kontynuacja)

Wykonanie polecenia w tle

np. `grep user * > lista &`

Interpretator rozpoznaje w wierszu polecenia znak & (przetwarzanie w tle). Ustawia odpowiednią zmienną lokalną. Pod koniec pętli sprawdza jej wartość i jeśli jest ustawiona to nie wykonuje funkcji **wait**, ale przechodzi do początku pętli i wczytuje następne polecenie.

Polecenie z potokiem

np. `ls -l | wc`

Interpretator wykonuje funkcję **fork** i tworzy proces potomny (wc).

Proces potomny tworzy potok (**pipe**) i wywołuje funkcję **fork** tworząc swojego potomka.

Proces ten (wnuk interpretatora) wykonuje pierwszą część polecenia (ls). W celu pisania do potoku, zamyka deskryptor pliku standardowego, powiela deskryptor potoku do pisania, a następnie zamyka go.

Proces (wc) zamyka swój standardowy plik wejściowy, a powiela deskryptor potoku do czytania. Po przeczytaniu zamyka deskryptor potoku i wykonuje polecenie wc.

Interpretator wraca do początku pętli.

Koncepcja wątków

Została wprowadzona jako realizacja idei współbieżnego korzystania przez procesy z ich zasobów.

Wątek zwany też procesem lekkim stanowi podstawową jednostkę wykorzystującą procesor. Obejmuje licznik rozkazów, stan rejestrów i obszar stosu.

Dzieli z innymi wątkami w ramach zadania sekcję tekstu, danych i zasoby systemu operacyjnego np. otwarte pliki i sygnały.

Tradycyjny (ciężki) proces posiadający oddzielny obszar przestrzeni adresowej odpowiada zadaniu z pojedynczym wątkiem. Dzielenie wspólnych zasobów w ramach określonego zadania powoduje, że tworzenie wątków i ich przełączanie jest znacznie mniej kosztowne niż w przypadku procesów "ciężkich".

Wątki mogą być w analogicznych stanach jak procesy tradycyjne :

(nowy, gotowy, wykonywany, czekający, zakończony).

Tylko jeden wątek w danej chwili czasu może być wykonywany (przez jeden procesor). Każdy ma swój własny stos i licznik rozkazów. Może tworzyć wątki potomne.

W odróżnieniu od procesów wątki nie są niezależne, ponieważ mają dostęp do wspólnych adresów pamięci w ramach zadania. Wątki wykonywane w ramach jednego zadania nie wymagają jednak takiej ochrony jak procesy pochodzące od różnych użytkowników.

Wątki wykonują się znacznie efektywniej niż tradycyjne procesy.

Przykłady systemów realizujących idee wątków: OS2, Solaris 2, HP-UX 10.10, . . .

Planowanie - szeregowanie procesów (scheduling)

Planowanie polega na określaniu w jakiej kolejności procesy uzyskują dostęp do zasobów komputera: procesora, pamięci operacyjnej.

Celem planowania jest zwykle zapewnienie jak najlepszego wykorzystania procesora.

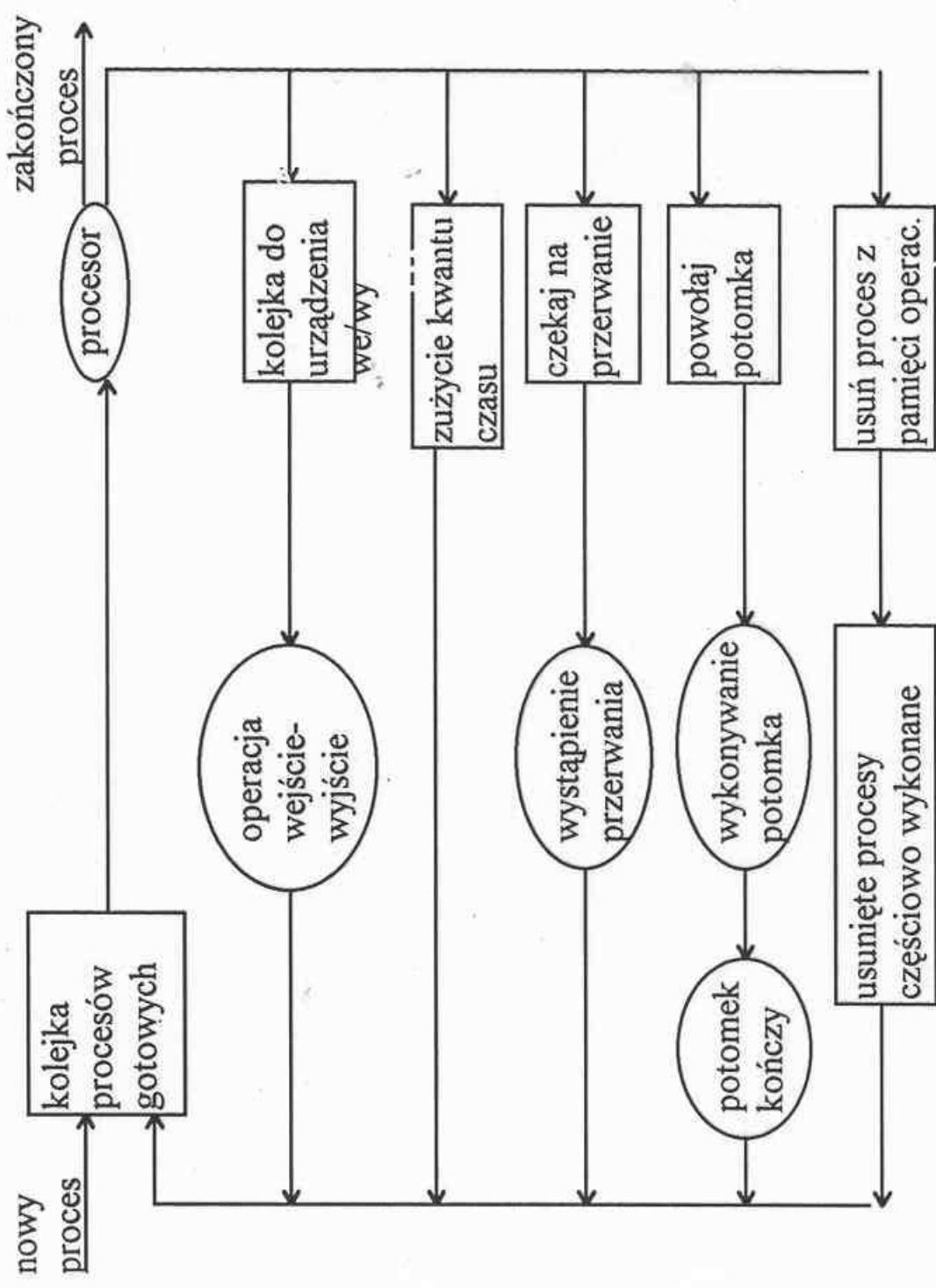
Procesy starając się o dostęp do procesora, czy konkretnego urządzenia czekają w kolejkach:

- kolejka zadań,
- kolejka procesów gotowych do procesora,
- kolejki do urządzeń we/wy.

Wykonywany proces może:

- . zażądać operacji we/wy - zostanie wtedy umieszczony w kolejce do urządzenia we/wy,
- . może utworzyć nowy proces i czekać na jego ukończenie,
- . może wykorzystać cały kwant czasu, lub w wyniku przerwania zostać przeniesiony do kolejki procesów gotowych.

Diagram kolejek



Kryteria planowania

1. Wykorzystanie procesora, w % czasu.
2. Przepustowość - liczba procesów kończonych w jednostce czasu.
3. Czas cyklu przetwarzania - średni czas przetwarzania procesu od chwili utworzenia do zakończenia.
4. Czas oczekiwania - średni czas oczekiwania w kolejkach.
5. Czas odpowiedzi - w systemach interakcyjnych - czas między zgłoszeniem zamówienia przez użytkownika, a pierwszą odpowiedzią.

Przykłady algorytmów planowania (szeregowania)

FCFS (First Come, First Served) - pierwszy nadszedł - pierwszy obsłużony. Realizowany za pomocą kolejki FIFO. Wady: efekt konwoju, kłopotliwy w systemach z podziałem czasu. Jest algorytmem niewylaşczającym

SJF (Shortest Job First) - najpierw najkrótsze zadanie. Procesor przydzielany jest procesowi o najkrótszej następnej fazie procesora. Teoretycznie daje minimalny średni czas oczekiwania. Wymaga jednak dokładnego oszacowania czasu przyszłej fazy procesora dla każdego procesu. Szacowanie to wykonywane jest zwykle na podstawie pomiarów czasu faz poprzednich.

Może być wywłaszczający lub nie.

Planowanie priorytetowe - każdemu procesowi jest przydzielany pewien priorytet.

Wybierany jest proces o najwyższym priorytecie. Algorytm SJF jest szczególnym przypadkiem.

Może być wywłaszczającym lub nie.

Planowanie rotacyjne - zaprojektowany dla systemów z podziałem czasu. Ustalony jest kwant czasu (10-100 ms). Kolejka procesów ma charakter cykliczny. Kolejnym procesom przydzielany jest co najwyżej kwant czasu.

Przełączanie kontekstu obejmuje czynności związane z przechowaniem stanu (*informacji wskazanych w bloku kontrolnym*) starego procesu i załadowania stanu procesu nowego. Czas przełączania kontekstu (rzędu $1-100\mu s$) istotnie wpływa wydajność systemu.

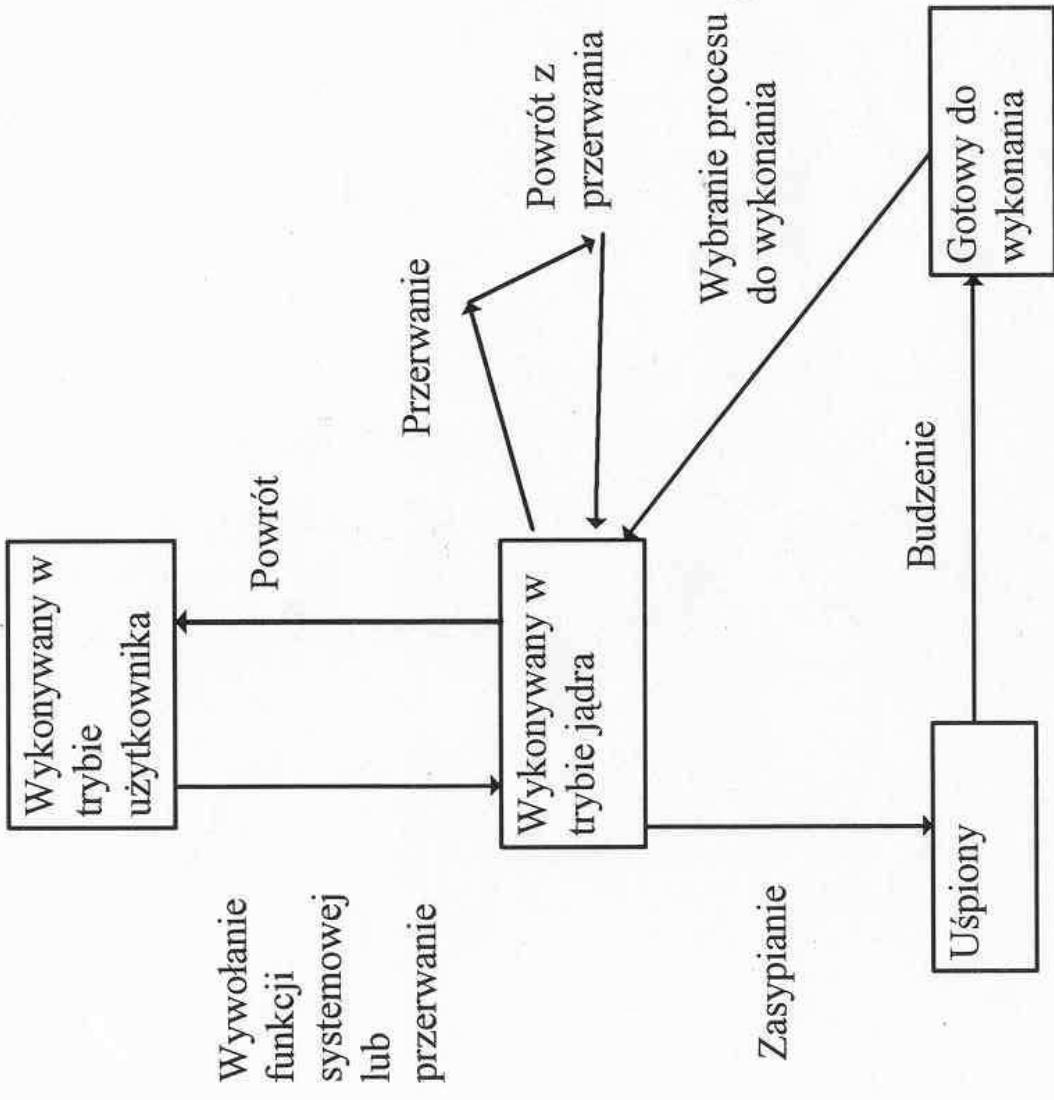
Szeregowanie procesów w systemie UNIX

- System z podziałem czasu - jądro przydziela procesor gotowemu procesowi na jeden kwant czasu, zgodnie z algorytmem rotacyjnym.
- Po upływie tego czasu wywłaszcza proces i przydziela procesor procesowi następnemu w kolejce. Wybiera proces załadowany do pamięci o najwyższym priorytecie.
- Jądro okresowo przelicza i modyfikuje priorytety wszystkich procesów gotowych.
- Wywłaszczony proces jest umieszczany w jednej z kolejek priorytetowych i gdy przyjdzie na niego kolej, jest wznowiany od miejsca, w którym nastąpiło zawieszenie wykonywania.

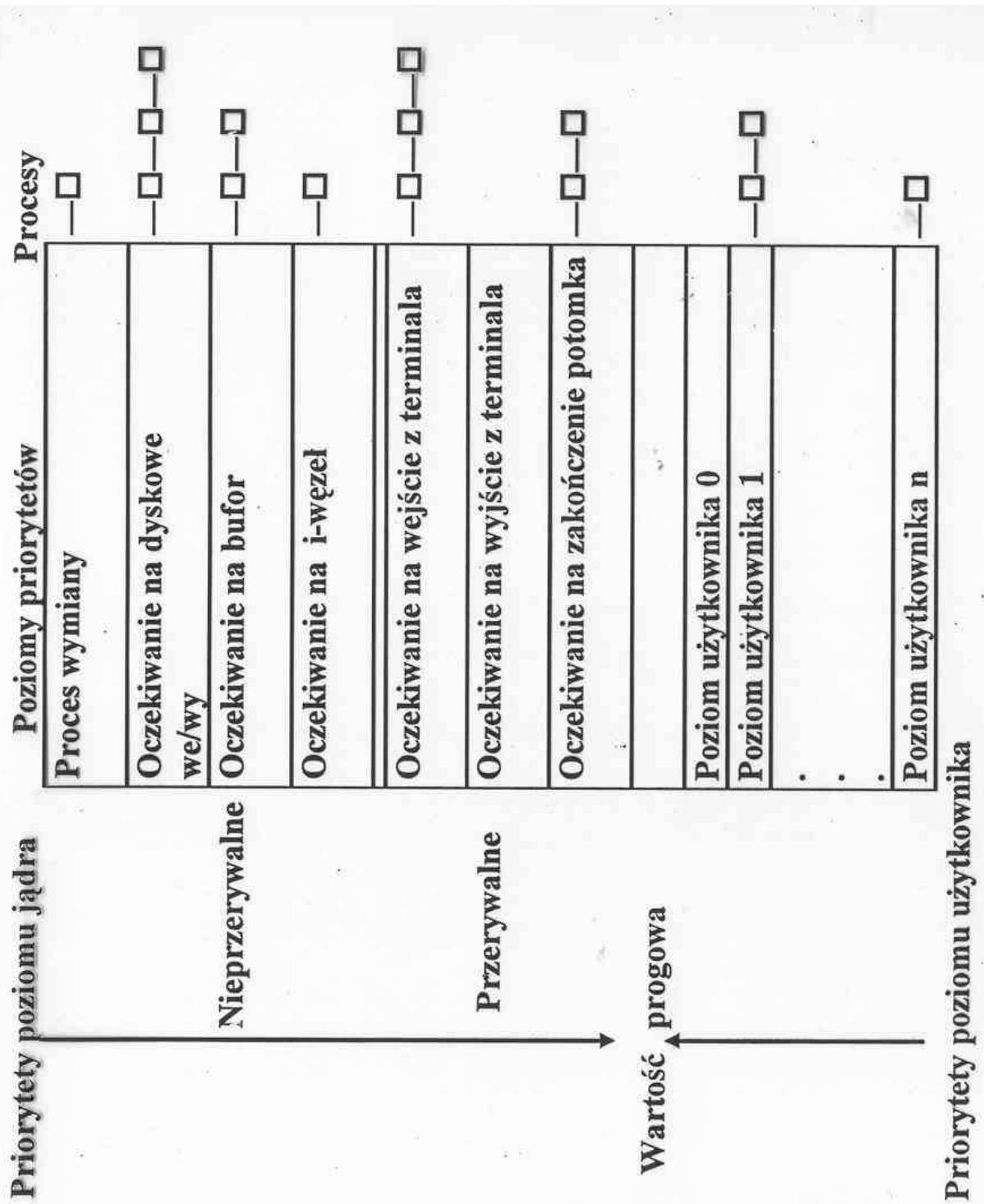
Algorytm szeregowania

1. Wybierz proces o najwyższym priorytecie spośród procesów gotowych do wykonania i załadowanych do pamięci operacyjnej.
2. Jeśli jest kilka o równym priorytecie wybierz proces najdłużej oczekujący w stanie gotowym do wykonania.
3. Jeśli nie ma procesów gotowych do wykonania, czekaj do następnego przerwania (najpóźniej do kolejnego taktu zegara). Powróć do kroku 1.

Schemat przejścia między stanami procesu



Zakres priorytetów:



Priorytety procesów (c.d.)

Priorytety poziomu jądra przydzielane są procesom wykonującym operację **sleep** (przechodzącym w stan uśpienia), w zależności od powodu przejścia do tego stanu.

Priorytety poziomu użytkownika nadawane są (modyfikowane) procesom powracającym z **trybu jądra** do **trybu użytkownika**.

Podprogram obsługi zegara co np. 1-ą sekundę (System V) przelicza zużycie czasu procesora przez wszystkie procesy i wyznacza priorytety:

priorytet = ostatnie_zużycieCPU/2 + priorytet_bazowy,
gdzie **priorytet_bazowy** przyjmuje wartość progową.

Uwagi:

Niska wartość zmiennej **priorytet** oznacza wysoki priorytet szeregowania.

Funkcja systemowa **nice** (wartość) pozwala zmienić **priorytet** wykonywanego procesu:

priorytet = zużycie_CPU/2 + priorytet_bazowy + wartość_nice.

Administrator systemu może podać wartość **nice** niższe od wartości progowej.

Użytkownicy - tylko wyższe od tej wartości. Wykorzystuje się do tego polecenie przedrostkowe **nice** o składni: **nice[-n] wiersz_polecenia**

gdzie n jest liczbą całkowitą w przedziale 1- 19.

Procesy potomne dziedziczą wartość **nice** przodka.

Proces nie może zmienić wartości **nice** innego procesu.

Ładowanie systemu operacyjnego (bootstrap) w systemie UNIX.

Celem procesu ładowania jest umieszczenie systemu operacyjnego w pamięci operacyjnej i rozpoczęcie jego wykonywania. Składa się z kilku etapów:

1. Inicjalizacja i testowanie sprzętu.
2. Wczytanie i umieszczenie w pamięci bloku systemowego (blok **0**) z dysku.
3. Program zawarty w bloku systemowym ładuje jądro systemu operacyjnego z pliku systemowego (np. /stand/vmunix) do pamięci. Przekazuje sterowanie do pierwszej instrukcji jądra. Program jądra systemu zaczyna się wykonywać.
4. Wykrywanie i konfiguracja urządzeń, odnalezienie głównego katalogu plików.
5. Przygotowanie środowiska procesu **0**. Wykonywanie programu systemu jako procesu **0** wykonywanego w trybie jądra. Utworzenie procesów jądra, np. procesów zarządzania pamięcią.
6. Rozwidlenie procesu **0** (wywołanie funkcji **fork** z jądra). Utworzony proces **1** tworzy kontekst poziomu użytkownika i przechodzi do trybu użytkownika.
7. Proces **1** wywołuje funkcję systemową **exec** wykonując program **/sbin/init**
8. Proces **init** wczytuje plik **/etc/inittab** i rozmnaża procesy.
9. Inicjacja wewnętrznych struktur danych jądra, tworzenie np. listy wolnych buforów, i-węzłów, kolejek; inicjacja struktur segmentów i tablic stron pamięci
10. Sprawdzenie głównego i pozostałych systemów plików (ew. uruchomienie fsck).
11. Wywoływane są procesy **getty** monitorujące konsolę i terminale systemu komputerowego, zgodnie z deklaracjami w pliku **inittab**, a proces **init** wykonuje funkcję systemową **wait** monitorując zakończenie procesów potomnych. Proces **init** tworzy również procesy demony.

Otwieranie sesji użytkownika

1. Proces getty otwiera terminal i wywołuje proces login
2. Proces login sprawdza prawidłowe rozpoczęcie pracy przez użytkownika i wywołuje początkowy interpretator poleceń (shell).
3. Proces shell uruchamia systemowy skrypt startowy `/etc/profile`, a następnie lokalny skrypt `$HOME/.profile`.
4. Shell wysyła znak zachęty \$, interpretuje wprowadzane polecenia, tworzy procesy potomne.

Zamykanie systemu (shutdown)

1. Zakończenie wszystkich procesów.
2. Synchronizacja i odmontowanie systemów plików.
3. Zatrzymanie procesora.
4. Można wyłączyć zasilanie.

Uwagi:

Wszyscy użytkownicy powinni zostać zawiadomieni o planowanym zatrzymaniu systemu.

Polecenie shutdown jest podstawowym poleceniem realizującym operację zatrzymania systemu: np. po wydaniu polecenia:

```
# shutdown -y -h 300
```

operacja zatrzymywania systemu rozpocznie się za 5 minut, a wszyscy pracujący w danej chwili użytkownicy otrzymają komunikat systemowy o planowanym za 5 minut zatrzymaniu systemu.

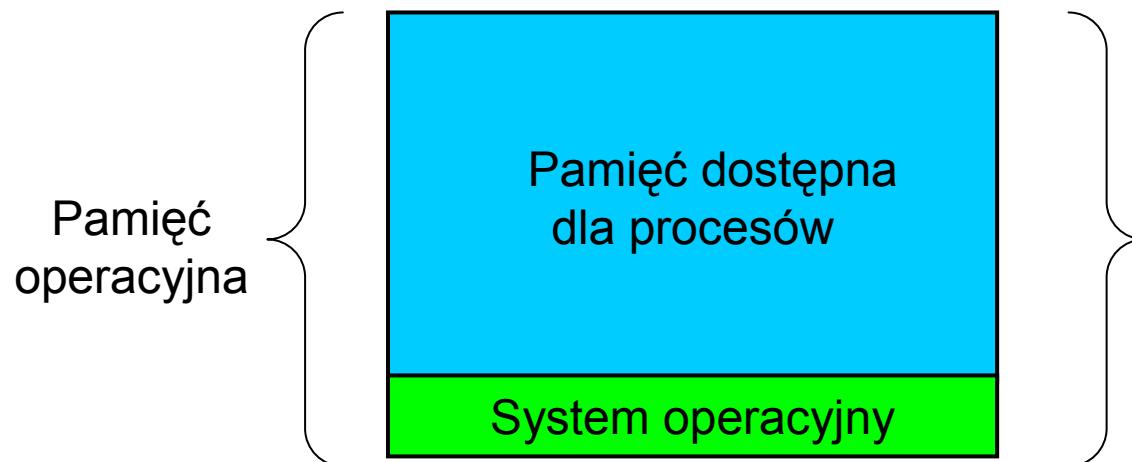
Zarządzanie pamięcią

Literatura: A. Silberschatz, J. Peterson, P. Galvin, Podstawy systemów operacyjnych, rozdz. 7, 8,
M. Bach, Budowa systemu operacyjnego UNIX, rozdz. 9.

Zainstalowana pamięć (operacyjna) nazywana jest pamięcią fizyczną.
Można ją interpretować jako tablicę bajtów, w której każdy element ma
przyporządkowany jednoznaczny adres.

Wykonywane programy wraz danymi (procesy) przechowywane są, przynajmniej
częściowo w pamięci.

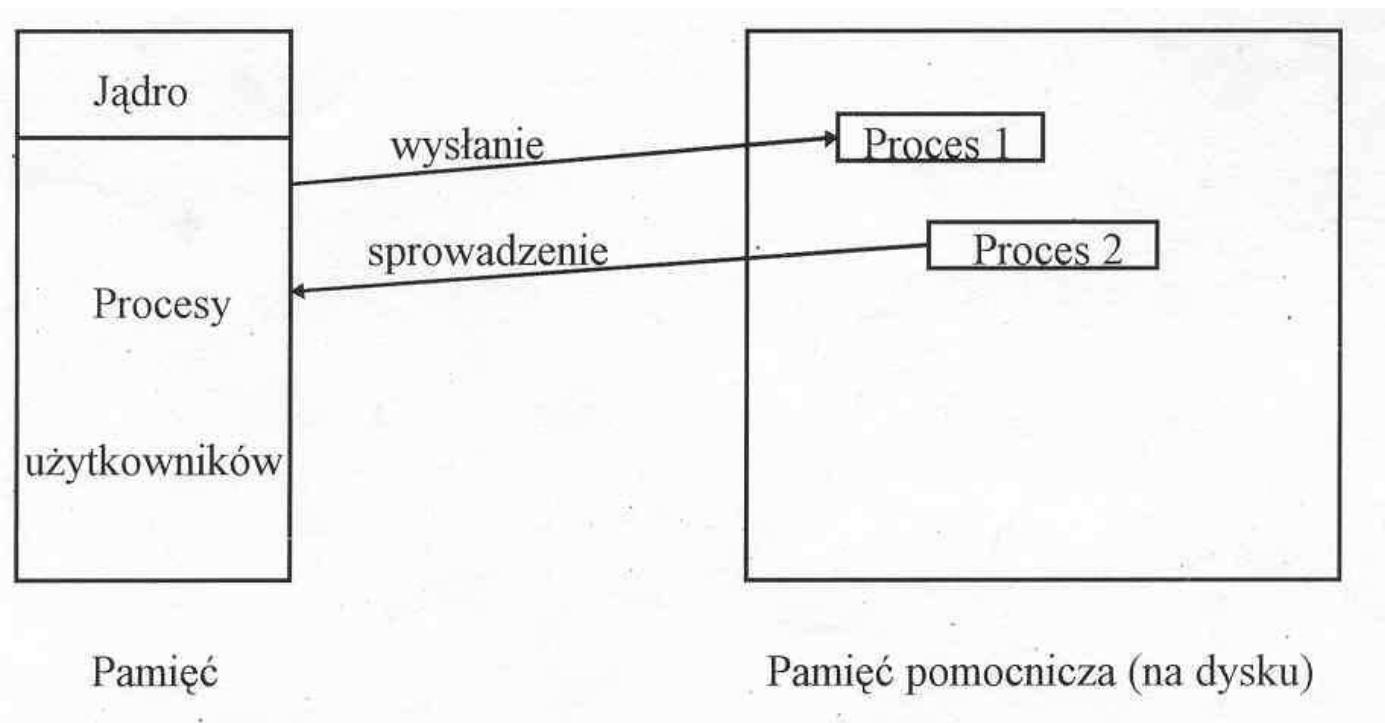
Część pamięci fizycznej, w której umieszczane są procesy
nazywana jest pamięcią dostępną dla procesów.



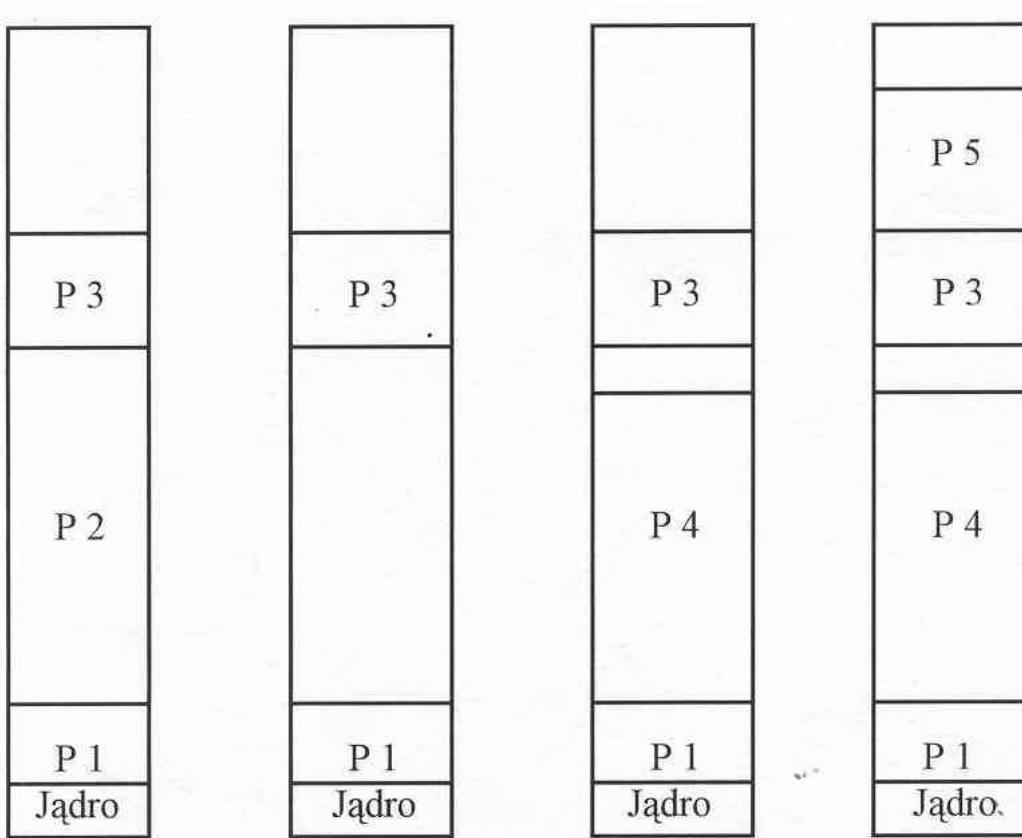
Metody przydziału pamięci:

- . metoda nakładek (ang. overlaying),
- . wymiana (ang. swapping),
- . stronicowanie (ang. paging).

Wymiana:



Przydział pamięci wielu procesom:



Strategie wyboru "dziur":

- Pierwsza pasująca (najszybsza),
- Najlepiej pasującą (wybór najmniejszej spośród dostatecznie dużych), ""
- Najgorzej pasującą (przydział .największej).

Wszystkie te algorytmy powodują zewnętrzna fragmentację, wolna pamięć dostępna jest wtedy w wielu drobnych kawałkach. Aby uniknąć zewnętrznej fragmentacji stosuje się upakowanie pamięci.

Stronicowanie

Procesy dzielone są na strony o stałej długości, np. 4 kB.

Pamięć dzieli się na bloki o stałej długości, nazywane ramkami.

Tablica ramek stron zawiera informacje o stanie ramek.

Strony mogą być dynamicznie ładowane do pamięci, na żądanie.

Strony mogą być wysyłane do pamięci pomocniczej.

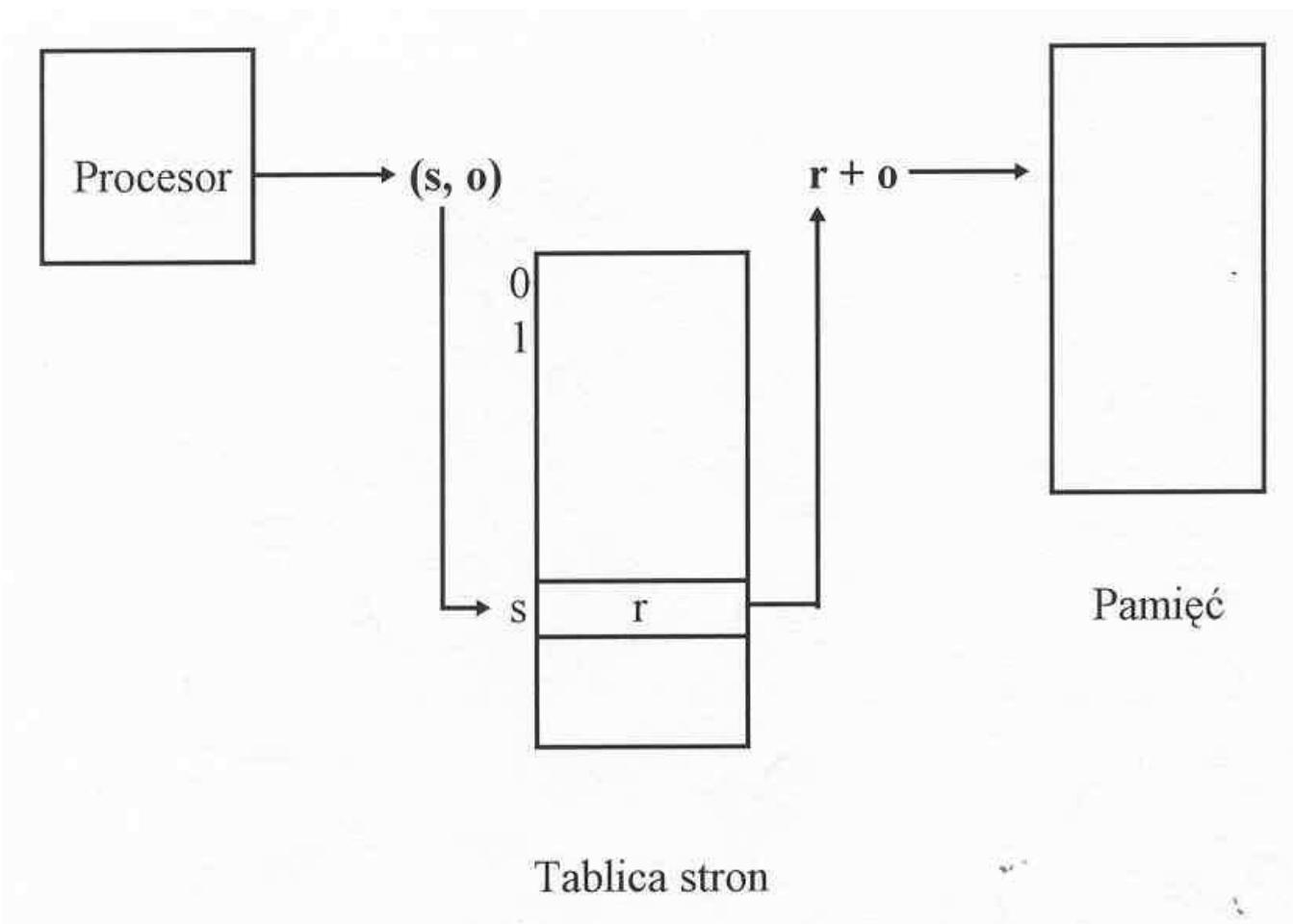
Strony o stałej długości nie powodują powstawania "dziur".

Adresy generowane są w postaci par: s - numer strony,
 o - odległość od początku strony.

Tablica stron zawiera adresy bazowe ramek, przydzielonych stronom.

Numer strony używany jest jako indeks w tablicy stron. .

Wyznaczanie adresu w mechanizmach stronicowania



Prosty model stronicowania

Strona 0
Strona 1
Strona 2
Strona 3

Obraz procesu

0	1
1	4
2	3
3	7

Tablica
stron

0
1
2
3
4
5
6
7

Obraz pamięci

Stronicowanie w przypadku kodów współdzielonych

Polecenie: file nazwa_pliku

wyświetla informację o typie wskazanego pliku, np. commands text, directory, ascii text
np.:

file /usr/bin/vi

/usr/bin/vi: s800 shared executable dinamically linked

Przykład dwóch procesów edytora vi

vi 1
vi 2
vi 3
dane 1

Proces P1

vi 1
vi 2
vi 3
dane 2

Proces P2

0	3
1	4
2	6
3	1

tablica stron P1

0	3
1	4
2	6
3	7

tablica stron P2

0
1
2
3
4
5
6
7
8

Pamięć

Implementacja tablicy stron

Cała tablica stron jest przechowywana w pamięci operacyjnej.

Dodatkowo wykorzystuje się szybką pamięć sprzętową, tzw. rejesty asocjacyjne (ang. Translation Look-aside Buffers, TLB). Zawierają one numery ustalonej liczby stron oraz odpowiadających im ramek.

Segmentacja

Segmente to semantycznie określone fragmenty programu, np. program główny, podprogramy i biblioteki, tablica symboli, dane, stos.

Do podstawowych zalet segmentacji należą: możliwość powiązania ochrony pamięci z wybranymi segmentami oraz współdzielenie wybranych segmentów przez różne procesy.

Segmentacja umożliwia na przykład ustawienie bitu ochrony dla segmentów kodu (tylko do odczytu) lub współdzielenie kodu edytora (np. vi) przez procesy edycji wielu jednocześnie pracujących użytkowników. W tym przypadku, obrazy procesów użytkowników zawierają (w uproszczeniu) segmenty danych i stosu oraz wskaźniki do właściwego miejsca w segmencie programu.

Aby odwołać się do odpowiedniego segmentu, procesy korzystają z tablicy segmentów. Każdy element tej tablicy można przedstawić w postaci pary, adres bazowy danego segmentu oraz jego rozmiar. Tablicę segmentów przechowuje się zwykle w pamięci głównej, a tylko pewną liczbę elementów tej tablicy przechowuje się w rejestrach.

We współczesnych systemach operacyjnych stosuje się stronicowanie segmentów.

Stronicowanie na żądanie

Wykorzystuje zasadę lokalności odwołań. Do pamięci przesyłane są strony niezbędne w danej chwili lub te, które niebawem mogą się okazać niezbędne.

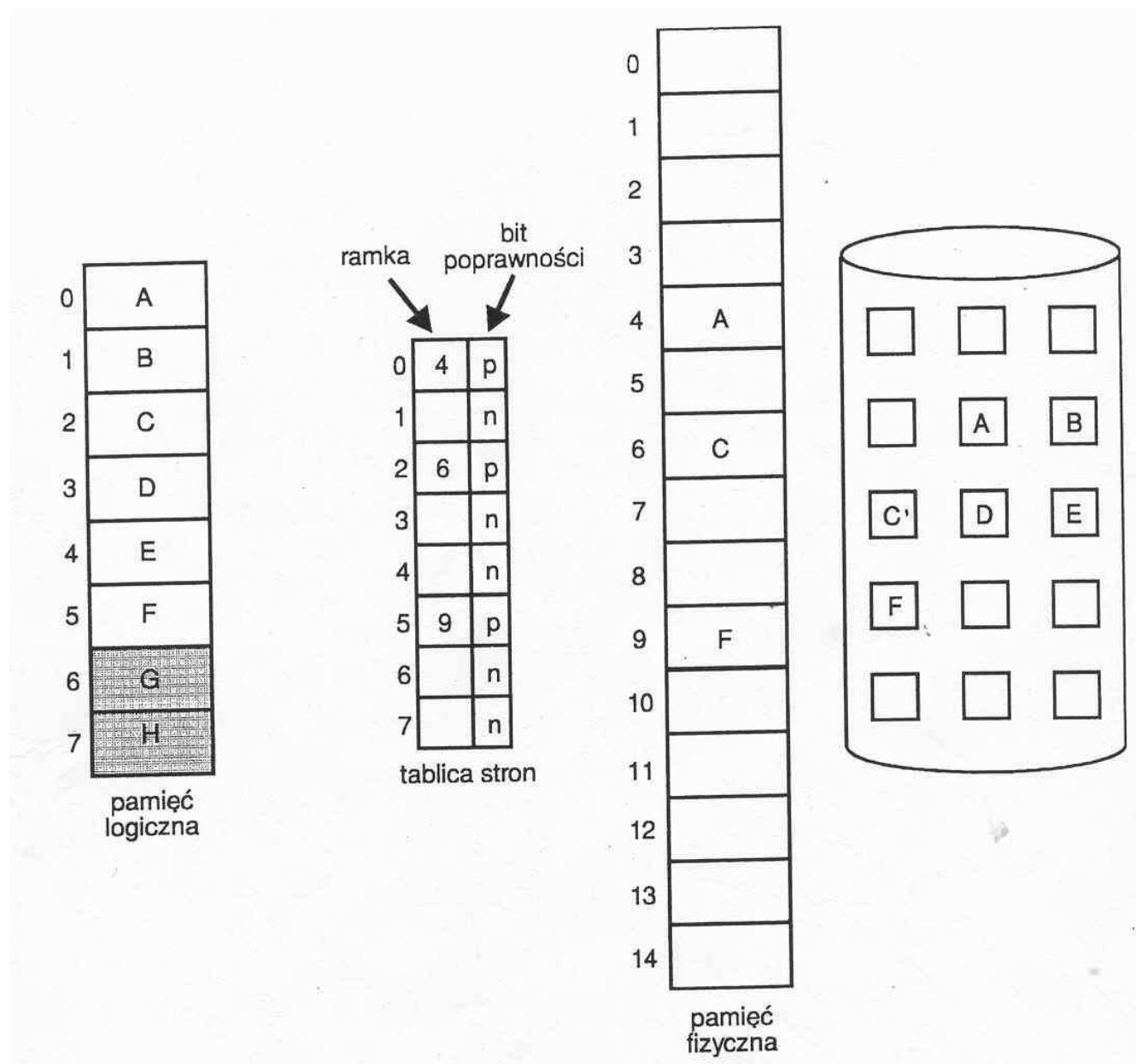
Strony mogą więc znajdować się na dysku, w pamięci głównej lub w obszarze wymiany.

Jeśli niezbędna jest strona znajdująca się na dysku, generowany jest błąd strony i odpowiednie przerwanie. Wykonywanie procesu jest wstrzymywane, po czym odnajdywana jest wolna ramka, do której przepisywana jest potrzebna strona.

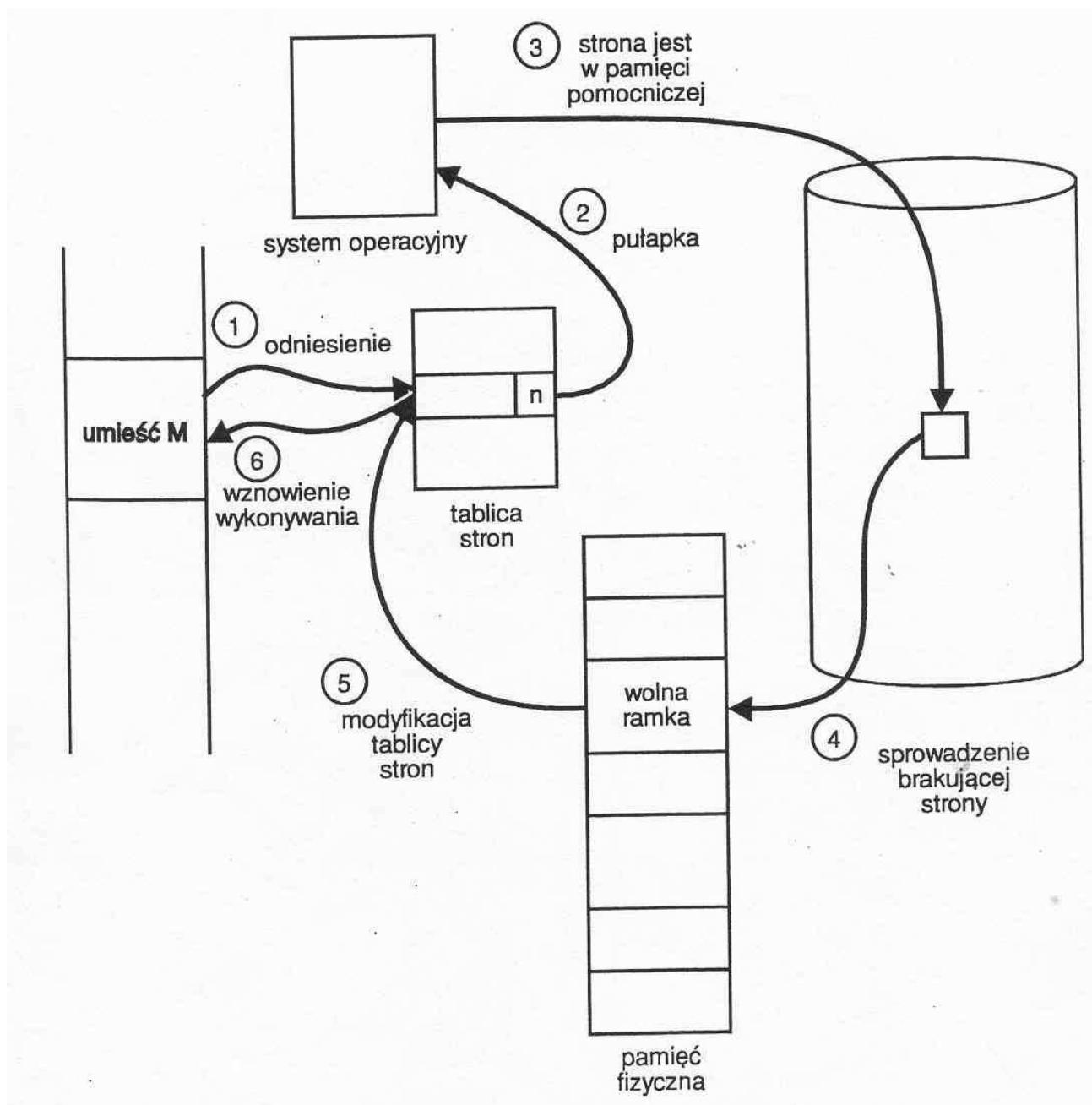
Zwykle wymaga to zwolnienia ramki przez stronę innego procesu (wymiana stron, ang. paging) - niezbędne więc są odpowiednie algorytmy zastępowania stron.

Zbiór roboczy to zbiór stron procesu jednocześnie znajdujących się w pamięci. Brak strony w zbiorze roboczym generuje błąd strony.

Stronicowanie na żądanie, w zbiorze roboczym są tylko niektóre strony:



Obsługa błędu strony:



Przykłady algorytmów stronicowania

(A. Silberschatz, J. Peterson, P. Galvin, Podstawy systemów operacyjnych, rozdz. 8)

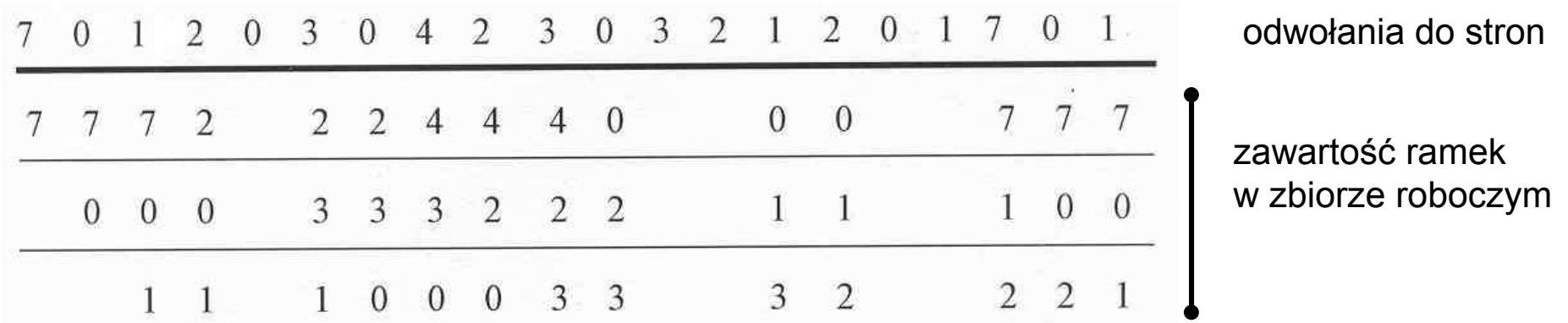
Założymy, że wielkość zbioru roboczego jest równa 3 oraz dany jest również następujący ciąg odwołań do stron: **7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1**

Algorytm FIFO

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2		2	2	4	4	4	0		0	0		7	7	7		
0	0	0		3	3	3	2	2	2		1	1		1	0	0			
1	1		1	0	0	0	3	3		3	2		2	2	1				

odwołania do stron

zawartość ramek
w zbiorze roboczym



Liczba błędów = 15

Uwaga:

Algorytm FIFO charakteryzuje anomalia Belady'ego, współczynnik błędów stron może wzrastać wraz z wielkością zbioru roboczego.

Można to pokazać posługując się ciągiem odwołań do stron:

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

oraz przyjmując wielkość zbioru roboczego równą 3 a następnie 4.

Algorytm optymalny

Zastąp stronę, która najdłużej nie będzie używana.

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	2	2	2	0	0	0	0	0	0	0	7	0	0
0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1	1	1	1

Liczba błędów stron = 9

Uwaga: Algorytm optymalny jest trudny do implementacji.

Algorytm LRU (ang. Least Recently Used)

Zastąp stronę, która nie była używana od najdłuższego czasu.

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	4	4	4	4	0	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	0	0	0
1	1	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	7	0	0

Liczba błędów stron = 12

Realizacja algorytmu LRU

Wykorzystanie liczników

W tablicy stron dodanie do każdej pozycji rejestru czasu użycia strony.

Dodanie do procesora zegara logicznego lub licznika.

Każde użycie strony powoduje skopiowanie czasu zegara (licznika) do rejestru tablicy stron.

Zastępuje się stronę z najmniejszym czasem (licznikiem).

Zastosowanie stosu numerów stron

Numery stron umieszczone są w stosie.

Przy każdym odwołaniu do strony - jej numer wyjmuje się z wnętrza stosu i umieszcza na szczytce.

Najdawniej używana strona jest na spodzie stosu.

Uwaga

Implementacja metody LRU wymaga pomocy dodatkowego sprzętu.

Algorytm buforowania stron

Systemy często przechowują pulę wolnych ramek.

Po wystąpieniu błędu strony wybiera się stronę - ramkę ofiarę zgodnie z algorytmem np. LRU.

Przed usunięciem "ofiary" czyta się potrzebną stronę wolnej ramki z puli wolnych ramek.

**Potem dopiero usuwa się stronę „ofiarenę” z pamięci operacyjnej.
Zwolniona ramkałączana jest do puli wolnych ramek.**

Szybkie wpisanie potrzebnej strony do pamięci pozwala szybko wznowić proces.

Przydzielanie ramek procesom

Dana jest liczba wolnych ramek pamięci operacyjnej dla procesów użytkowych.
Jak duże powinny być ich zbiory robocze.

Minimalna liczba ramek

Zmniejszenie liczby ramek przydzielonych procesowi powoduje zwiększenie częstotliwości błędów strony.

Wykonanie rozkazu w określonej architekturze komputera wymaga odwołań do określonej liczby adresów pamięci.

Liczba tych odwołań określa minimalną liczbę stron procesu.

Brak strony przed dokonaniem rozkazu, wymaga jego powtórzenia.

Minimalna liczba stron procesu zdefiniowana jest przez architekturę komputera.

Algorytmy przydziału

Przydział równy

Przydział proporcjonalny

Uwzględnienie priorytetów

Zastępowanie stron globalne i lokalne

Szamotanie (ang. Thrashing) ,

Bardzo duża aktywność stronicowania powoduje spadek wydajności systemu.

Uwaga: w mechanizmie stronicowania na żądanie wykorzystywane są struktury danych, takie jak: tablica stron, deskryptory bloków dyskowych, tablica ramek stron oraz tablica wymiany.

Ustalanie wielkości obszaru wymiany:

Brak formalnych reguł. Przydatne polecenia:

`/usr/sbin/swapinfo`

(może je wydawać tylko użytkownik uprzywilejowany (UID = 0), root)

`/usr/bin/size [-v] nazwa_programu`

na przykład:

`/usr /bin/si ze /usr /bin/vi`

wyświetla: 195552 + 27064 + 775984 = 998600

gdzie kolejne liczby oznaczają odpowiednio rozmiar segmentu kodu, segmentu danych oraz segmentu danych nieinicjalizowanych (bss);

`/usr/bin/top`

podaje między innymi (kolumna SIZE) całkowity rozmiar obrazu procesu (włączając w to segmenty kodu, danych i stosu) w kilobajtach;

Polecenie top, przykład wyświetlanych informacji

```
12:08pm up 54 min, 2 users, load average: 0.00, 0.00, 0.00
53 processes: 52 sleeping, 1 running, 0 zombie, 0 stopped
CPU states: 0.0% user, 0.5% system, 0.0% nice, 99.4% idle
Mem: 128008K av, 28160K used, 99848K free, 28336K shrd, 2252K buff
Swap: 160608K av, 0K used, 160608K free 15556K cached
```

PID	USER	PRI	NI	SIZE	RSS	SHARE	STAT	LIB	%CPU	%MEM	TIME	COMMAND
1021	root	10	0	1004	1004	824	R	0	0.5	0.7	0:01	top
1	root	0	0	472	472	408	S	0	0.0	0.3	0:04	init
2	root	0	0	0	0	0	SW	0	0.0	0.0	0:00	kflushd
3	root	0	0	0	0	0	SW	0	0.0	0.0	0:00	kupdate
4	root	0	0	0	0	0	SW	0	0.0	0.0	0:00	kpiod
5	root	0	0	0	0	0	SW	0	0.0	0.0	0:00	kswapd
103	root	0	0	460	460	400	S	0	0.0	0.3	0:00	apmd
350	bin	0	0	376	376	308	S	0	0.0	0.2	0:00	portmap
373	root	0	0	728	728	592	S	0	0.0	0.5	0:00	syslogd
384	root	0	0	740	740	388	S	0	0.0	0.5	0:00	klogd
398	daemon	0	0	472	472	400	S	0	0.0	0.3	0:00	atd
412	root	0	0	592	592	504	S	0	0.0	0.4	0:00	crond
426	root	0	0	592	592	504	S	0	0.0	0.4	0:00	inetd
440	root	0	0	1396	1396	904	S	0	0.0	1.0	0:00	snmpd
472	root	0	0	528	528	452	S	0	0.0	0.4	0:00	rpc.statd
483	root	0	0	376	376	312	S	0	0.0	0.2	0:00	rpc.rquotad
494	root	0	0	732	732	600	S	0	0.0	0.5	0:00	rpc.mountd

Polecenie

/usr/bin/ipcs –a

podaje między innymi w części Shared Memory (w kolumnie SEGSZ) rozmiary segmentów współdzielonych przez procesy.

Definiowanie obszaru wymiany

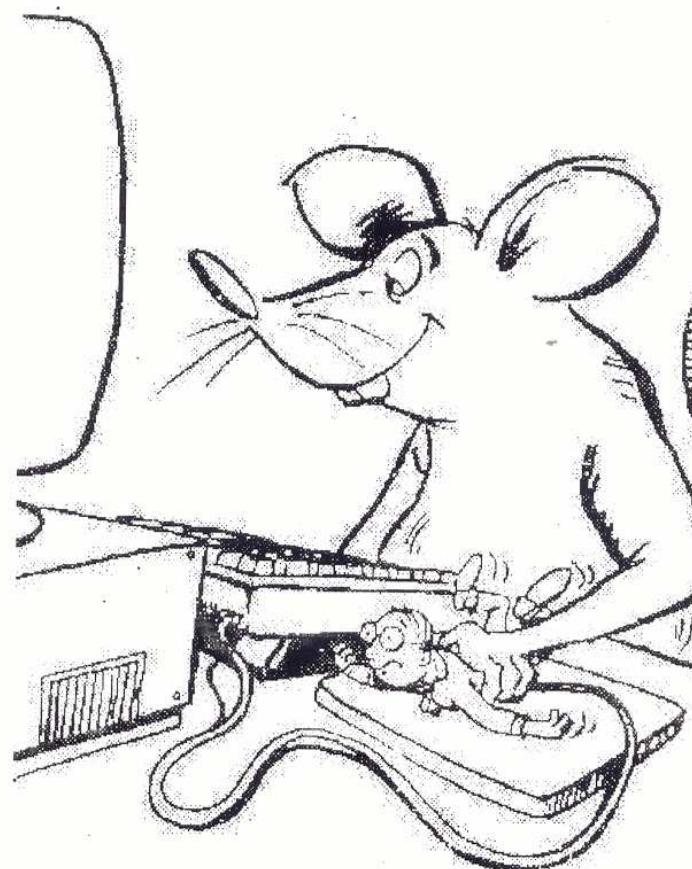
/usr/sbin/swapon urządzenie_wymiany

Urządzeniem wymiany może być cały dysk lub ustalona partycja:

/usr/sbin/swapon /dev/dsk/c1t4d0

Uwaga: Tylko użytkownik uprzywilejowany może konfigurować obszar wymiany.

PODSYSTEM WEJŚCIA - WYJŚCIA



(źródło: Internet)

Podsystem wejścia-wyjścia

(M. Bach, Budowa systemu operacyjnego UNIX, rozdz. 10)

Wiele rodzajów urządzeń:

- Urządzenia zewnętrzne fizyczne

urządzenia pamięci: napędy dysków, taśmy, cd, dvd, ...

przesyłania danych: karty sieciowe, modemy, ...

interfejs komunikacji z człowiekiem: ekran, klawiatura, mysz, drukarki, ...

urządzania specjalistyczne: do sterowania procesami np. chemicznymi, czujniki, ...

- Programowe urządzenia zewnętrzne

pamięć operacyjna, pseudoterminale.

Podstawowe pojęcia dotyczące sprzętu wejścia – wyjścia

szyna

sterownik

port we – wy

uzgadnianie działania między procesorem głównym a sterownikiem urządzenia

odpytywanie

przerwania

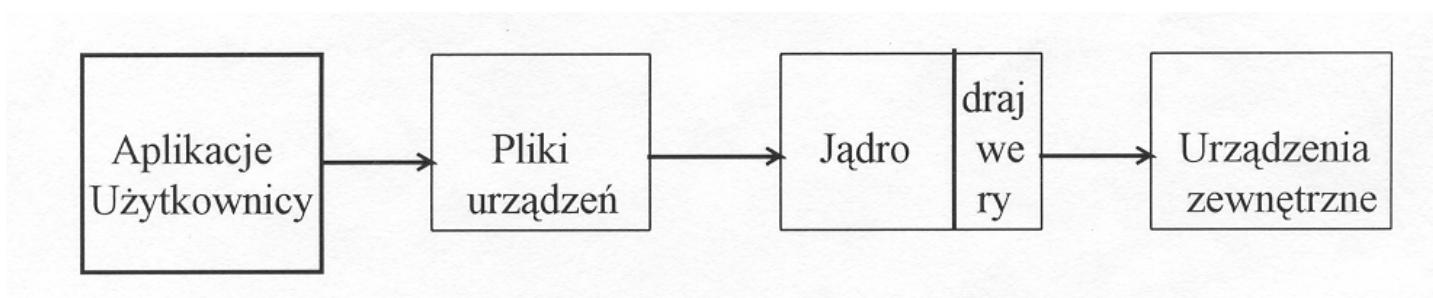
sterownik DMA

Dwa rodzaje urządzeń zewnętrznych:

urządzenia blokowe,
urządzenia znakowe

Podprogramy obsługi urządzeń (drajwery) - moduły jądra przeznaczone do sterowania pracą urządzeń zewnętrznych.

Pliki urządzeń lub **pliki specjalne** (ang. special files, device files) zapewniają aplikacjom i użytkownikom współpracę z urządzeniami zewnętrznymi.



Aby odczytać dane z urządzenia peryferyjnego lub je tam zapisać wykorzystuje się mechanizmy przeadresowania do lub z pliku specjalnego, np.:

banner Testujemy terminal > /dev/ttyp7

cpio -icv plik1 < /dev/rmt/0m

Charakterystyka plików urządzeń:

- typ pliku specjalnego: blokowy lub znakowy,
- położenie w strukturze katalogów: katalog /dev,
- określone zasady nazywania, np. /dev/dsk/clt4d0
- liczba główna, np. 17,
- liczba pomocnicza, np. 0x014000.

Przykład:

```
ls -l /dev/rdsk/clt4d0
crw-r 1 root root 187 0x014000 Aug 29 1996 clt4d0
```

```
ls -l /dev/dsk/clt4d0
brw-r 1 root root 187 0x014000 Aug 29 1996 clt4d0
```

Tablica rozdzielcza urządzeń blokowych

	<u>open</u>	<u>close</u>	<u>strategia</u>
0	open_dev0	close_dev0	strategia_dev0
1	open_dev1	close_dev1	strategia-devel
2	open_dev2	close_dev2	strategia_dev2
...
255	open_dev255	close_dev255	strategia_dev255

Tablica rozdzielcza urządzeń znakowych

	<u>open</u>	<u>close</u>	<u>read</u>	<u>write</u>	<u>ioctl</u>
0	open_rdev0	close_rdev0	read_rdev0	write_rdev0	ioctl_rdev0
1	open_rdev1	close_rdev1	read_rdev1	write_rdev1	ioctl_rdev1
2	open_rdev2	close_rdev2	read_rdev2	write_rdev2	ioctl_rdev2
...
255	open_rdev255	close_rdev255	read_rdev255	write_rdev255	ioctl_rdev255

Schemat przedstawiający rolę tablic rozdzielczych w operacjach wejścia - wyjścia

