PRZYKŁADY SYSTEMÓW OPERACYJNYCH

WYKŁAD Z PRZEDMIOTU SYSTEMY OPERACYJNE

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

HISTORIA LINUXA

W 1969 roku Ken Thompson, Dennis Ritchie i inni rozpoczęli pracę nad tym, co miało stać się UNIX-em na PDP-7 w AT&T Bell Labs.

Przez dziesięć lat rozwijano UNIX-u w AT&T w numerowanych wersjach.

V4 (1974) został napisany na nowo w języku C - jest to kamień milowy dla przenośności systemu operacyjnego pomiędzy różnymi platformami.

V6 (1975) był pierwszą wersją dostępną poza Bell Labs - stała się podstawą pierwszej wersji UNIX opracowanej na Uniwersytecie Kalifornijskim w Berkeley.

Firma Bell Labs kontynuowała prace nad UNIX w latach 80-tych, których punktem kulminacyjnym było wydanie Systemu V w 1983 roku oraz Systemu V, Release 4 (w skrócie SVR4) w 1989 roku.

Berkeley Standard Distribution (BSD) stał się drugim ważnym wariantem "UNIX". Został on szeroko wdrożony zarówno w środowiskach uniwersyteckich, jak i korporacyjnych, poczynając od wydania BSD 4.2 w 1984 roku. Niektóre z jego funkcji zostało włączonych do SVR4.

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

HISTORIA LINUX'A CD.

- Firma AT&T sprzedała swoją działalność w obszarze UNIX firmie Novell w 1993 r., a Novell sprzedał ją firmie Santa Cruz Operation dwa lata później. W międzyczasie znak towarowy UNIX został przekazany konsorcjum X/Open, które to ostatecznie połączyły się, tworząc The Open Group.
- Tradycyjnie, aby system BSD działał, potrzebna była licencja na kod źródłowy z AT&T. Jednak na początku lat 90-tych, hakerzy Berkeley wykonali tak wiele pracy nad BSD, że większość oryginalnego kodu źródłowego AT&T zniknęła.
- Kolejni programiści, począwszy od Williama i Lynne Jolitz, rozpoczęli pracę nad dystrybucją Net BSD, prowadząc do wydania 386BSD w wersji 0.1 w 1992. Ten oryginalny "free source" BSD został podzielony na trzy główne dystrybucje, z których każda posiada dedykowaną wersję: NetBSD, FreeBSD i OpenBSD, wszystkie oparte na BSD 4.4.2.
- BSD nie była pierwszą próbą "uwolnienia" UNIX-a. W 1984 roku programista Richard Stallman rozpoczął pracę nad wolnym klonem UNIX znanym jako GNU (GNU's Not UNIX). W początku latach dziewięćdziesiątych projekt GNU osiągnął kilka kamieni milowych w programowaniu, w tym wydanie biblioteki GNU C i Bourne Again SHell (bash). Cały system został zasadniczo ukończony, z wyjątkiem jednego krytycznego elementu: działającego jądra.

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

Richard Stallman

Linus Torvalds

POWSTANIE LINUXA

- Linusa Torvaldsa, student Uniwersytetu Helsińskiego w Finlandii znalazł system UNIX o nazwie Minix i zdecydował, że może go ulepszy. Jesienią 1991 roku wydał kod źródłowy wolnego oprogramowania jądra zwanego "Linux" - połączenie jego pierwszego imienia i Miniksa czytany jako lynn-nucks.
- Do 1994 roku Linus i zaangażowany zespół hakerów jądra był w stanie wydać wersję 1.0 Linux. Linus i przyjaciele mieli wolne jądro systemu; Stallman i jego zespół mieli resztę wolnego systemu klonów UNIX: Ludzie mogli wtedy połączyć jądro Linux z GNU, by stworzyć kompletny wolny system. System ten znany jest jako "Linux", chociaż Stallman preferuje nazwę "system GNU/Linux ".
- Istnieje kilka różnych dystrybucji GNU/Linux: niektóre są dostępne do zastosowania komercyjnego takie jak: CentOS, Caldera Systems i S.U.S.E.; inne, takie jak Debian GNU/Linux, są ściślej powiązane z oryginalną koncepcją wolnego oprogramowania.

HISTORIA LINUX CD.

- Rozprzestrzenianie się Linux, teraz aż do wersji jądra 2.2, było zaskakującym zjawiskiem.
- Linux działa na różnych architekturach układów scalonych i został zaadoptowany lub wspierany w różnym stopniu przez starych dostawców UNIX-a, takich jak Hewlett-Packard, Silicon Graphics i Sun Microsystems, przez dostawców komputerów PC, takich jak Compaq i Dell, oraz przez głównych producentów oprogramowania, takich jak Oracle i IBM.
- Być może najbardziej ironiczną była reakcja Microsoftu, który wyczuwał zagrożenie konkurencyjne wszechobecnego wolnego oprogramowania, ale wydawał się niechętny lub niezdolny do reagowania uwalniania własnego oprogramowania.
- Microsoft uderzył jednak systemem Windows NT (Windows 2000). Pod koniec lat 90-tych, sprzedawcy porzucili platformę UNIX na rzecz systemu Windows NT lub zrezygnowali z ich wsparcia. Silicon Graphics Inc., na przykład, obwieścił, że sprzęt firmy Intel i NT jest platformą graficzną przyszłości.

CZYM JEST UNIX I DLACZEGO GO UŻYWAĆ?

Jako podstawowe oprogramowanie Internetu, technologia UNIX jest jednym z najważniejszych osiągnięć cywilizacji XX wieku.

Jednym z głównych powodów używania Unix'a jest zdolność do pracy w sieci. Funkcjonalność ta była i jest po prostu częścią systemu operacyjnego. Unix jest idealny dla serwerów poczty, aplikacji, baz danych i połączenia z Internetem.

Unix powstał w oparciu o filozofię, którą można by nazwać "małe jest dobre". Ideą jest to, że każdy program jest zaprojektowany tak, aby dobrze wykonywać jedną pracę. Ponieważ Unix został stworzony przez różnych ludzi o różnych potrzebach, rozwinął się w system operacyjny, który jest zarówno elastyczny, jak i łatwy do zaadaptowania do konkretnych potrzeb.

Unix został napisany w języku maszynowo niezależnym. Tak więc mogą działać na różnych rodzajach sprzętu. Systemy te są dostępne z wielu różnych źródeł, niektóre z nich są bezpłatne. Z powodu tej różnorodności i możliwości wykorzystania tego samego "interfejsu użytkownika" w wielu różnych systemach, Unix jest uważany za system otwarty.

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

Zaprojektowane przez programistów, dla programistów.

Zaprojektowany tak, aby był:

Prosty	Elegancki	Spójny	Potężny	Elastyczny

CELE SYSTEMÓW OPARTYCH NA UNIX

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

PODSTAWOWE CECHY

Przenośny - Przenośność oznacza, że oprogramowanie może działać na różnych typach sprzętu w ten sam sposób. Jądro Linuksa i programy aplikacyjne wspierają ich instalację na dowolnej platformie sprzętowej.

Open Source - kod źródłowy Linuksa jest dostępny bezpłatnie i jest to projekt rozwojowy oparty na społeczności. Wiele zespołów pracuje nad zwiększeniem możliwości systemu operacyjnego Linux i stale się rozwija.

Multi-User - Linux jest systemem wielo-użytkownikowym, co oznacza, że wielu użytkowników ma dostęp do zasobów systemowych, takich jak pamięć/ram/aplikacje w tym samym czasie.

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

PODSTAWOWE CECHY CD

- Programowanie wieloprogramowe - Linux jest systemem wieloprogramowym, co oznacza, że wiele aplikacji może działać jednocześnie.
- Hierarchiczny system plików - Linux zapewnia standardową strukturę plików, w której pliki systemowe/pliki użytkowników są ułożone.
- Shell - Linux zapewnia specjalny program interpretujący, który może być używany do wykonywania poleceń systemu operacyjnego. Może być wykorzystywany do wykonywania różnego rodzaju operacji, wywoływania programów użytkowych itp.
- Bezpieczeństwo - Linux zapewnia bezpieczeństwo użytkownika za pomocą funkcji uwierzytelniania, takich jak ochrona hasłem/kontrolowany dostęp do określonych plików/szyfrowanie danych.

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

REPOZYTORIA PAKIETÓW

Każda dystrybucja Linuksa może połączyć się z jednym lub kilkoma repozytoriami pakietów.

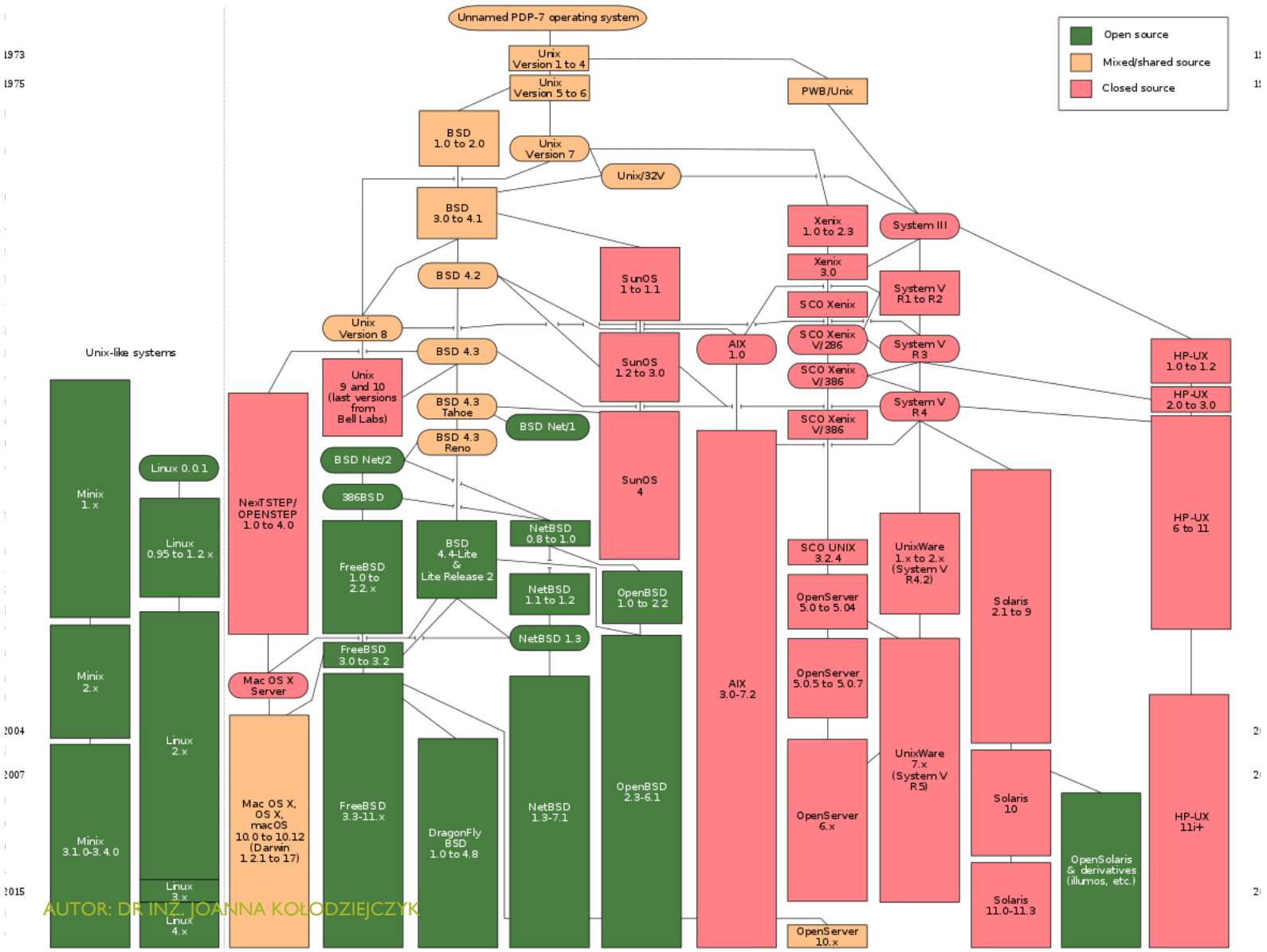
Ułatwiają wyszukiwanie/instalację/deinstalację określonych aplikacji.

Package manager (yum, apt)

"Znajdź mi wszystkie aplikacje do analizy sekwencji i zainstaluj je".

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

DRZEWO DYSTRYBUCJI

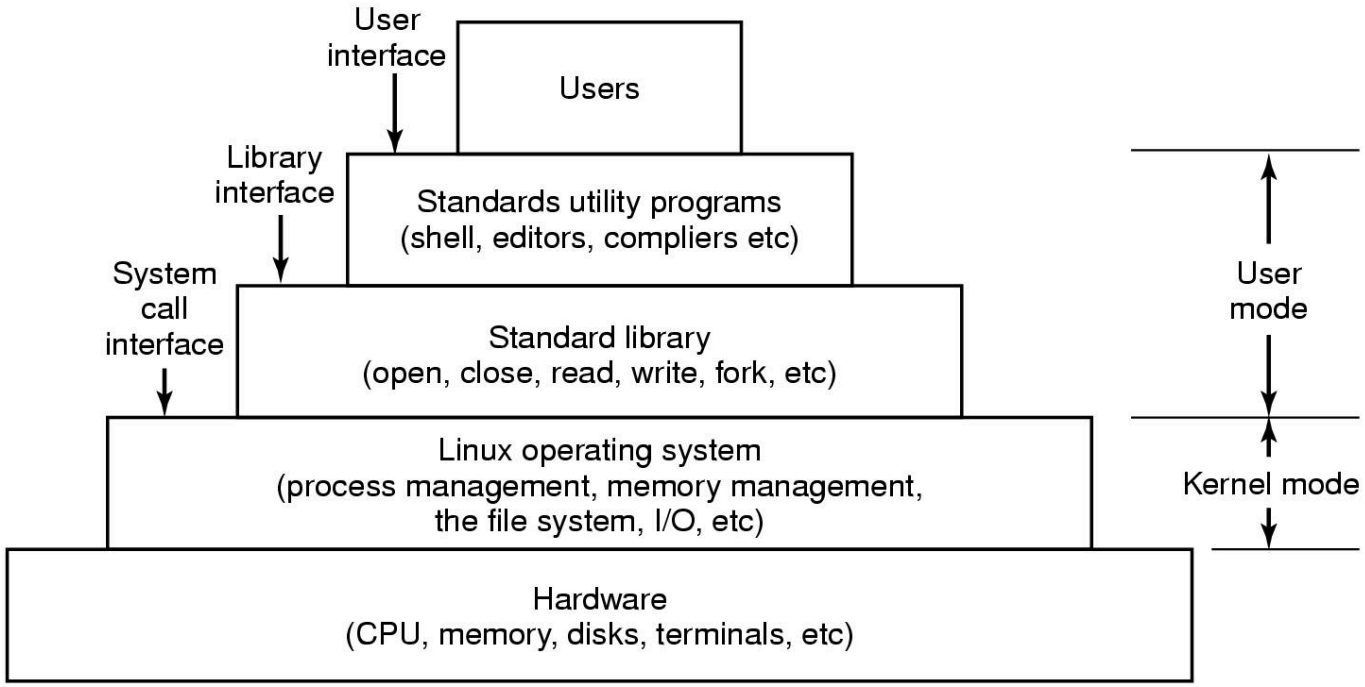


AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

JAK WYBRAĆ DYSTRYBUCJE LINUKSA?

- Spróbuj więcej niż jeden, aby mieć wyczucie.
- Z czego korzystają Twoi koledzy/członkowie zespołu?
- Czy repozytoria pakietów mają aplikacje, z których chcesz korzystać?
- Jak długo autorzy dystrybucji będą ją wspierać?
- Czy masz mniej popularny laptop/sprzęt, który może mieć problemy ze zgodnością sprzętową z tą dystrybucją? (rzadkie, ale zdarza się)

INTERFEJSY SO LINUX



AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

WARSTWY

Warstwa sprzętowa - Sprzęt składa się ze wszystkich urządzeń peryferyjnych (RAM/dysk twardy/ CPU itp.).

Kernel - Jest to główny komponent systemu operacyjnego, wchodzi w bezpośrednią interakcję ze sprzętem, świadczy usługi niskopoziomowe dla komponentów warstwy górnej.

Shell - Interfejs do jądra, ukrywający złożoność funkcji jądra przed użytkownikami. Powłoka pobiera polecenia od użytkownika i wykonuje funkcje jądra.

Narzędzia - programy narzędziowe, które zapewniają użytkownikowi większość funkcji systemów operacyjnych.

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

GŁÓWNE KOMPONENTY

- **Kernel -** Kernel jest główną częścią Linuksa. Jest odpowiedzialny za wszystkie główne działania tego systemu operacyjnego. Składa się on z różnych modułów i wchodzi w bezpośrednią interakcję z podstawowym sprzętem. Kernel zapewnia wymaganą abstrakcję do ukrywania niskopoziomowych szczegółów sprzętowych do systemu lub programów aplikacyjnych.

- **Biblioteka systemowa -** Biblioteki systemowe są funkcjami specjalnymi lub programami, za pomocą których programy aplikacyjne lub narzędzia systemowe uzyskują dostęp do funkcji jądra. Biblioteki te implementują większość funkcji systemu operacyjnego i nie wymagają uprawnień modułu jądra do dostępu do kodu.

- **System Utility -** Programy System Utility są odpowiedzialne za wykonywanie specjalistycznych, indywidualnych zadań.

TRYB JĄDRA A TRYB UŻYTKOWNIKA

Kod jądra wykonuje się w specjalnym trybie uprzywilejowanym zwanym **trybem jądra** z pełnym dostępem do wszystkich zasobów komputera. Kod ten reprezentuje pojedynczy proces, wykonuje się w pojedynczej przestrzeni adresowej i nie wymaga żadnego przełącznika kontekstowego, dzięki czemu jest bardzo wydajny i szybki. Kernel uruchamia każdy proces i świadczy usługi systemowe dla procesów, zapewnia chroniony dostęp do sprzętu do procesów.

Support code, który nie jest wymagany do uruchomienia w trybie jądra, znajduje się w Bibliotece Systemowej. Programy użytkownika i inne programy systemowe działają **w trybie użytkownika**, który nie ma dostępu do sprzętu systemowego i kodu jądra. Programy/instrumenty użytkownika korzystają z bibliotek systemowych, aby uzyskać dostęp do funkcji jądra w celu wykonywania zadań na niskim poziomie systemu.

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

PROGRAMY NARZĘDZIOWE LINUX (WARSTWA 2)

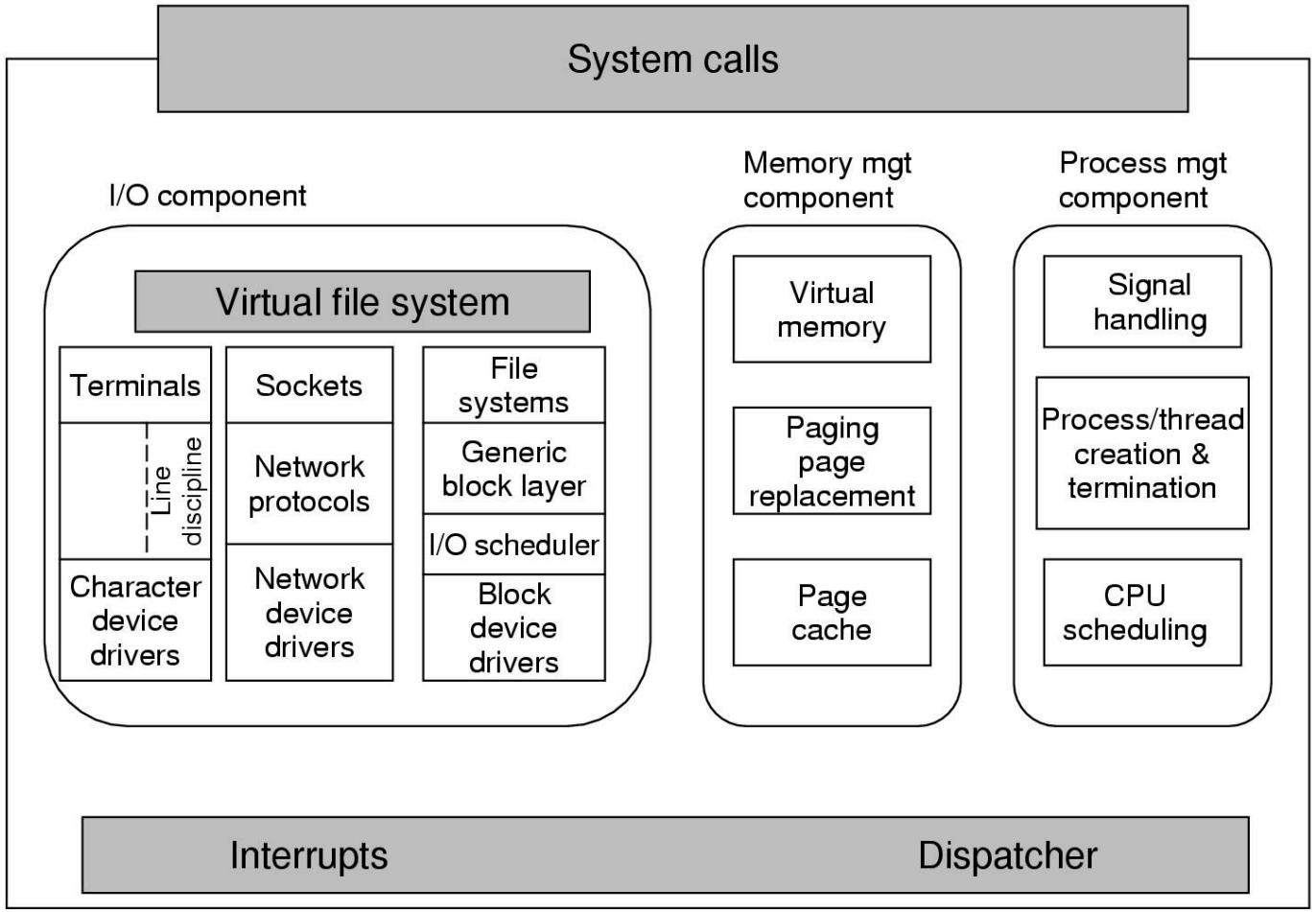
- Kategorie programów użytkowych:
  - Polecenia manipulacji plikami i katalogami.
  - Filtry.
  - Narzędzia służące do tworzenia programów, takie jak edytory i kompilatory.
  - Przetwarzanie tekstu.
  - Administracja systemem.
  - Różne.

PROGRAMY NARZĘDZIOWE CD.

Program	Typical use
cat	Concatenate multiple files to standard output
chmod	Change file protection mode
cp	Copy one or more files
cut	Cut columns of text from a file
grep	Search a file for some pattern
head	Extract the first lines of a file
ls	List directory
make	Compile files to build a binary
mkdir	Make a directory
od	Octal dump a file
paste	Paste columns of text into a file
pr	Format a file for printing
ps	List running processes
rm	Remove one or more files
rmdir	Remove a directory
sort	Sort a file of lines alphabetically
tail	Extract the last lines of a file
tr	Translate between character sets

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

STRUKTURA KERNELA



AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

KERNEL

- Zawiera manipulatory przerwań, które są podstawowym sposobem interakcji z urządzeniami, oraz mechanizm dyspozytorski (dispatch) niskiego poziomu.
- Dyspozycja ma miejsce, gdy następuje przerwanie.
- Niskopoziomowy kod zatrzymuje uruchomiony proces, zapisuje jego stan w strukturach procesu jądra i uruchamia odpowiedni sterownik.
- Wysyłanie (disparching) procesów ma miejsce również wtedy, gdy jądro zakończy niektóre operacje i nadszedł czas, aby ponownie uruchomić proces użytkownika.
- Kod dispatchera jest kdoem w assemblerze i jest dość odmienny od planowania.

TRZY GŁÓWNE KOMPONENTY JĄDRA 1- I/O

- Komponent I/O zawiera wszystkie elementy jądra odpowiedzialne za interakcję z urządzeniami oraz wykonującymi operacje sieciowe i operacje pamięci masowej we/wy.
- Na najwyższym poziomie wszystkie operacje we/wy są zintegrowane w ramach wirtualnego systemu plików VFS (Virtual File System). Oznacza to, że na najwyższym poziomie, wykonanie operacji odczytu pliku, niezależnie od tego, czy znajduje się on w pamięci, czy na dysku, jest takie samo jak wykonanie operacji odczytu w celu odzyskania znaku z wejścia terminala.
- Na najniższym poziomie wszystkie operacje we/wy przechodzą przez ten sam sterownik urządzenia.
- Wszystkie sterowniki dla Linuksa są klasyfikowane albo jako sterowniki urządzeń znakowych, albo jako sterowniki urządzeń blokowych, przy czym główną różnicą jest to, że urządzenia blokowe pozwalają na swobodny a urządzenia znakowe nie.
- Urządzenia sieciowe są naprawdę urządzeniami znakowymi, ale są traktowane nieco inaczej.

TRZY GŁÓWNE KOMPONENTY JĄDRA 1- ZARZĄDZANIE PAMIĘCIĄ

- Zadania związane z zarządzaniem pamięcią obejmują
  - utrzymanie mapowania wirtualnego do pamięci fizycznej,
  - utrzymywanie pamięci podręcznej ostatnio przeglądanych stron
  - wdrażanie dobrej polityki wymiany stron
  - wprowadzanie na żądanie nowych stron potrzebnych kodów i danych do pamięci.

TRZY GŁÓWNE KOMPONENTY JĄDRA 1- ZARZĄDZANIE PROCESAMI

- Zadaniem komponentu zarządzania procesami jest
  - tworzenie i zakańczanie procesów
  - harmonogram procesu, wybiera, który proces, a raczej wątek do uruchomienia
  - Jądro Linuksa traktuje zarówno procesy jak i wątki po prostu jako wykonywalne jednostki i będzie je planować w oparciu o globalną politykę planowania.
  - kod obsługi sygnału (signal processing).

ZARZĄDZANIE PROCESAMI – GŁÓWNE ZAŁOŻENIA

Każdy proces uruchamia jeden program i początkowo ma jeden wątek sterowania.

Innymi słowy, ma jeden licznik, który wykorzystuje do śledzenia następnej instrukcji do wykonania.

Linux pozwala na tworzenie dodatkowych wątków po rozpoczęciu procesu.

Wiele niezależnych procesów może działać w tym samym czasie.

Każdy użytkownik może mieć jednocześnie kilka aktywnych procesów, więc na dużym systemie mogą działać setki, a nawet tysiące procesów.

Nawet gdy użytkownik jest nieobecny, działają dziesiątki procesów tła, zwanych demonami.

DEAMON PROCESS

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

- Tryb pracy demona z crona:
  - Budzi się raz na minutę.
  - sprawdza, czy jest jakakolwiek praca do wykonania.
  - Jeśli tak, to wykonuje swoją pracę.
  - Następnie wraca do snu.

- Wykorzystywane do uruchamiania działań okresowych, takich jak codzienne tworzenie tworzenie kopii zapasowych dysków.
- Obsługują przychodzącą i wychodzącą pocztę elektroniczną, zarządzają kolejką drukarek itp.., itd.
- Demony są łatwe do implementowania, ponieważ każdy z nich jest odrębnym procesem, niezależnym od innych.

TWORZENIE PROCESU W LINUX

```
pid = fork( );                  /* if the fork succeeds, pid > 0 in the parent */
if (pid < 0) {
        handle_error( );        /* fork failed (e.g., memory or some table is full) */
} else if (pid > 0) {
                                /* parent code goes here. /*/
} else {
                                /* child code goes here. /*/
}
```

POTOKI

- Procesy w Linuksie mogą komunikować się między sobą używając formy przekazywania wiadomości.
- Możliwe jest stworzenie kanału pomiędzy dwoma procesami, w którym jeden proces może zapisać strumień bajtów do odczytania przez drugi proces.
- Kanały te są nazywane potokami (pipes)
- Synchronizacja jest możliwa. Gdy proces próbuje odczytać z pustego potoku, jest blokowany do czasu, gdy dane nie staną się dostępne.
- Np. polecenie powłoki:
  `sort <f | head`
  – potok jest tworzony tak, że standardowe wyjście procesu sort jest połączone ze standardowym wejściem procesu head.

KOMUNIKACJA PROCESÓW PRZEZ PRZERWANIA

- Proces może wysyłać sygnał do innego procesu.
- Procesy mogą powiedzieć systemowi, co mają zamiar zrobić, gdy nadejdzie sygnał przychodzący.
- Dostępne opcje to
  - zignorowanie sygnału
  - przechwycenie sygnału
  - pozwolenie, aby sygnał zabił proces.

POSIX PRZEGLĄD

- Portable Operating System Interface (POSIX) jest standardem IEEE, który wspomaga kompatybilność i przenośność pomiędzy systemami operacyjnymi.
- Standard POSIX został opracowany w latach 80-tych.
- Standard został zdefiniowany w oparciu o System V i BSD Unix.
- POSIX nie definiuje systemu operacyjnego, a jedynie definiuje interfejs pomiędzy aplikacją a systemem operacyjnym.
- Teoretycznie, kod źródłowy zgodny z POSIX-em powinien być bezproblemowo przenośny.
- W świecie rzeczywistym, przejście na aplikacje często prowadzi do specyficznych problemów systemowych.
- Jednak zgodność z POSIX-em ułatwia korzystanie z aplikacji komunikujących się przez porty, co może przynieść oszczędności czasu. Dlatego programiści powinni zapoznać się z podstawami tego szeroko stosowanego standardu.

STANDARDY POSIX

- POSIX został stworzony w celu ułatwienia przenoszenia aplikacji. Nie jest to więc standard tylko dla systemów UNIX. Systemy nie-UNIX mogą być również zgodne z POSIX-em.
- Standard nie dyktuje rozwoju aplikacji czy systemu operacyjnego. Tylko definiuje umowę pomiędzy nimi.
- Kod źródłowy aplikacji zgodny z POSIX-em powinien móc działać w wielu systemach, ponieważ standard jest zdefiniowany na poziomie kodu źródłowego. Standard nie gwarantuje jednak żadnej przenośności na poziomie obiektu lub kodu binarnego. Tak więc binarny plik wykonywalny nie może działać nawet na podobnych maszynach z identycznym sprzętem i systemami operacyjnymi. Tylko przenośność kodu źródłowego jest uwzględniona w standardzie.
- POSIX jest napisany w języku Standard C. Ale programiści mogą go zaimplementować w dowolnym języku.
- Standard zajmuje się tylko tymi aspektami systemu operacyjnego, które współdziałają z aplikacjami.
- Standard jest zwięzły pod względem długości i szeroki pod względem zakresu, aby objąć szeroki wachlarz systemów.

SYGNAŁY WYMAGANE PRZEZ STANDARD POSIX

Signal	Cause
SIGABRT	Sent to abort a process and force a core dump
SIGALRM	The alarm clock has gone off
SIGFPE	A floating-point error has occurred (e.g., division by 0)
SIGHUP	The phone line the process was using has been hung up
SIGILL	The user has hit the DEL key to interrupt the process
SIGQUIT	The user has hit the key requesting a core dump
SIGKILL	Sent to kill a process (cannot be caught or ignored)
SIGPIPE	The process has written to a pipe which has no readers
SIGSEGV	The process has referenced an invalid memory address
SIGTERM	Used to request that a process terminate gracefully
SIGUSR1	Available for application-defined purposes
SIGUSR2	Available for application-defined purposes

WYBRANE PROCEDURY SYSTEMOWE ODNOSZĄCE SIĘ DO PROCESÓW

System call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, opts)	Wait for a child to terminate
s = execve(name, argv, envp)	Replace a process' core image
exit(status)	Terminate process execution and return status
s = sigaction(sig, &act, &oldact)	Define action to take on signals
s = sigreturn(&context)	Return from a signal
s = sigprocmask(how, &set, &old)	Examine or change the signal mask
s = sigpending(set)	Get the set of blocked signals
s = sigsuspend(sigmask)	Replace the signal mask and suspend the process
s = kill(pid, sig)	Send a signal to a process
residual = alarm(seconds)	Set the alarm clock
s = pause( )	Suspend the caller until the next signal

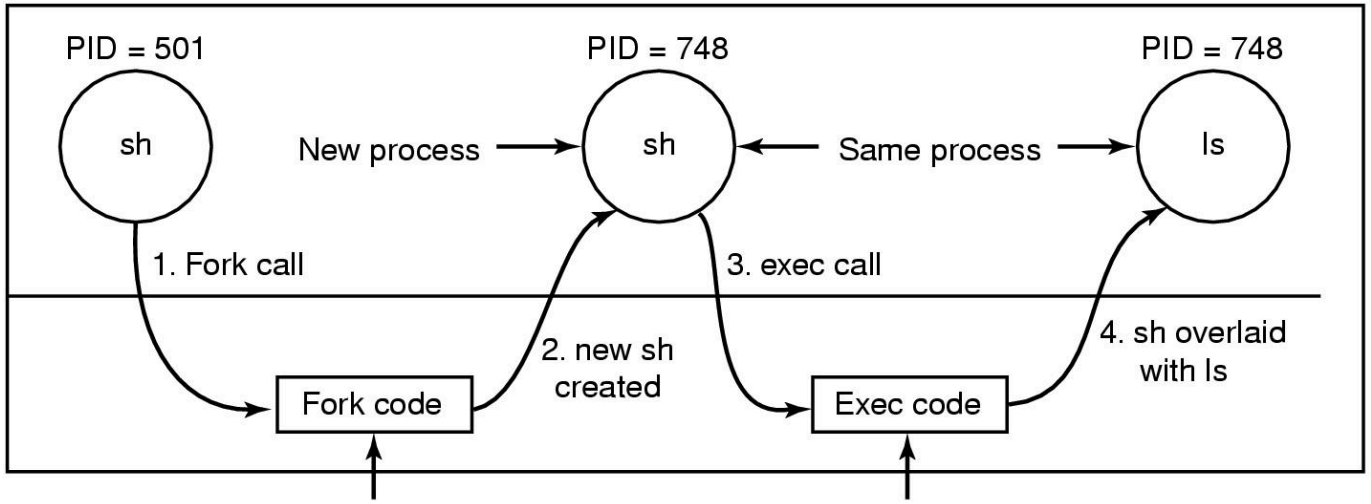
BARDZO UPROSZCZONY SHELL

```
while (TRUE) {                              /* repeat forever /*/
        type_prompt( );                     /* display prompt on the screen */
        read_command(command, params);      /* read input line from keyboard */

        pid = fork( );                      /* fork off a child process */
        if (pid < 0) {
                printf("Unable to fork0);   /* error condition */
                continue;                   /* repeat the loop */
        }

        if (pid != 0) {
                waitpid (–1, &status, 0);   /* parent waits for child */
        } else {
                execve(command, params, 0); /* child does the work */
        }
}
```

IMPLEMENTACJA PROCESÓW I WĄTKÓW – DESKRYPTOR PROCESU

- Kategorie informacji w deskryptorze procesu:
  - Parametry harmonogramowania: (priorytety procesów, ilość wykorzystanego czasu procesora), itp., itd
  - Obraz pamięci (wskaźniki segmentu tekstu, danych i stosu, tablice stron. gdy proces w pamięci przechować informacje jak znaleźć składowe na dysku)
  - Sygnały (maski określające jakie sygnały należy ignorować, przechowywać, itp.)
  - Rejestry maszynowe (zapis rejestrów maszynowych w chwili wykonani rozkazu pułapki)
  - Stan wywołania systemowego
  - Tabela deskryptorów plików (jeżeli wywołanie systemowe wymaga deskryptora pliku, to deskryptor jest indeksem w tablicy deskryptorów umożliwiając znalezienie właściwego i-węzła)
  - Rozliczenia (accounting) (wskaźnik do tabeli z czasem użytkownika i procesora - wykorzystanie przez dany proces)
  - Stos jądra
  - Różne (deskryptor procesu: stan bieżący, informacje o oczekiwaniu na zdarzenia).

KROKI W WYKONANIU POLECENIA LS



Allocate child's task structure
Fill child's task structure from parent
Allocate child's stack and user area
Fill child's user area from parent
Allocate PID for child
Set up child to share parent's text
Copy page tables for data and stack
Set up sharing of open files
Copy parent's registers to child

Find the executable program
Verify the execute permission
Read and verify the header
Copy arguments, environ to kernel
Free the old address space
Allocate new address space
Copy arguments, environ to stack
Reset signals
Initialize registers

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

PRZYDZIAŁ PAMIĘCI DLA NOWEGO PROCESU

- Kopiowanie pamięci jest kosztowne, zatem dokonuje się obejść
- Proces macierzysty i potomny nie mogą współdzielić przestrzeni pamięci poza tekstem
- Dla procesu potomnego tworzy się odrębne tablice stron, ale wypełniają się wskaźnikami do procesów macierzystych oznaczając bloki jak ‚tylko do odczytu'.
- Jeżeli proces próbuje się odwołać do danej strony i zmienić jej zawartość zgłaszany jest błąd i wydzielany zostaje nowy blok na dane procesu potomnego jako możliwy do odczytu i zapisu. Nazywa się to kopiowaniem przy zapisie.

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

WĄTKI W SO LINUX

- W 2000 roku w Linux wprowadzono pid = **clone(function, stack_ptr, sharing_flags, arg)**
- Wywołanie tworzy nowy wątek albo w ramach bieżącego procesu lub w ramach nowego procesu (w zależności od tego jak ustawione są flagi w sharing_flags)
- Jeżeli nowy wątek należy do bieżącego procesu to współdzieli przestrzeń adresową z istniejącymi wątkami, a każdy kolejny zapis bajtu w przestrzeni adresowej przez dowolny wątek jest natychmiast widoczny dla wszystkich pozostałych wątków w procesie.
- Jeśli przestrzeń adresowa nie jest współdzielona, to nowy wątek dostaje dokładną kopię przestrzeni adresowej, ale kolejne zapisy przez nowy wątek nie są widoczne dla innych. To podejście jest takie same jak w POSIX.
- W obu przypadkach, nowy wątek rozpoczyna wykonywanie funkcji, która jest wywoływana z **arg** jako jedynym parametrem. W obu przypadkach, nowy wątek dostaje swój własny prywatny stos, ze wskaźnikiem stosu inicjowanym **stack_ptr**.

ARGUMENT SHARING_FLAGS (USTAWIENIA BITÓW)

Flag	Meaning when set	Meaning when cleared
CLONE_VM	Create a new thread	Create a new process
CLONE_FS	Share umask, root, and working dirs	Do not share them
CLONE_FILES	Share the file descriptors	Copy the file descriptors
CLONE_SIGHAND	Share the signal handler table	Copy the table
CLONE_PID	New thread gets old PID	New thread gets own PID
CLONE_PARENT	New thread has same parent as caller	New thread's parent is caller

OPIS ARGUMENTU SHARING_FLAGS

- Bit CLONE_VM określa, czy pamięć wirtualna (tzn. adres space) jest współdzielona ze starymi wątkami lub kopiowana. Jeśli jest ustawiony, nowy wątek po prostu włącza się do istniejących, więc klonowe wywołanie skutecznie tworzy nowy wątek w istniejącym procesie. Jeśli bit jest wyczyszczony, nowy wątek otrzymuje swój własny adres prywatny.
- Bit CLONE_FS kontroluje współdzielenie katalogów głównych i roboczych oraz flagi **umask**. Nawet jeśli nowy wątek ma własną przestrzeń adresową, jeśli ten bit jest ustawiony, stare i nowe wątki dzielą katalogi robocze. Oznacza to, że wywołanie **chdir** przez jeden wątek zmienia katalog roboczy drugiego wątku, nawet jeśli drugi wątek może mieć własną przestrzeń adresową. W UNIX, wywołanie **chdir** przez wątek zawsze zmienia katalog roboczy dla innych wątków w swoim procesie, ale nigdy dla w innym procesie. Tak więc ten bit umożliwia rodzaj współdzielenia, które nie jest możliwe w tradycyjne wersje UNIX.

OPIS ARGUMENTU SHARING_FLAGS CD

Bit CLONE_FILES jest analogiczny do bitu CLONE_FS. Jeśli jest ustawiony, nowy wątek dzieli swoje deskryptory plików z innymi wątkami, jak to ma miejsce w przypadku wątków w obrębie tego samego ale nie dla wątków w różnych procesach.

Podobnie, CLONE_SIGHAND włącza lub wyłącza współdzielenie tabeli procedur obsługi sygnałów pomiędzy stary i nowy wątek. Jeśli tabela jest współdzielona, nawet pomiędzy wątkami w różnych przestrzeniach adresowych, wówczas zmiana handlera w jednym wątku wpływa na procedury w innych.

Bit CLONE_PARENT kontroluje, kto jest rodzicem nowego wątku. Może być taki sam jak wątek wywołujący (w którym to przypadku nowy wątek jest rodzeństwem) lub może być samym wątkiem wywołującym, w którym to przypadku nowy wątek jest dzieckiem. Istnieje kilka innych bitów, które kontrolują inne elementy, ale są one mniej ważne.

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

HARMONOGARMOWANIE/SZERFOWANIE W LINUX

- Wątki Linuksa są wątkami jądra, więc planowanie opiera się na wątkach, a nie na procesach.
- Linux rozróżnia trzy klasy wątków do celów planowania:
  1. FIFO w czasie rzeczywistym - Wątki FIFO w czasie rzeczywistym mają najwyższy priorytet i nie można ich wyłączyć, z wyjątkiem nowo odczytanego w czasie rzeczywistym wątku FIFO o jeszcze wyższym priorytecie.
  2. Round-robin w czasie rzeczywistym - są takie same jak wątki FIFO w czasie rzeczywistym, z wyjątkiem tego, że mają związane z nimi kwanty czasowe i są wyłączane przez zegar. Jeśli wiele wątków jest gotowych, każdy z nich jest uruchamiany na swój kwant, po czym przechodzi na koniec listy wątków. Wątki w czasie rzeczywistym są wewnętrznie reprezentowane z poziomami priorytetu od 0 do 99, przy czym 0 jest najwyższym, a 99 najniższym.
  3. Współdzielenie czasu.

ALGORYTM O(1)

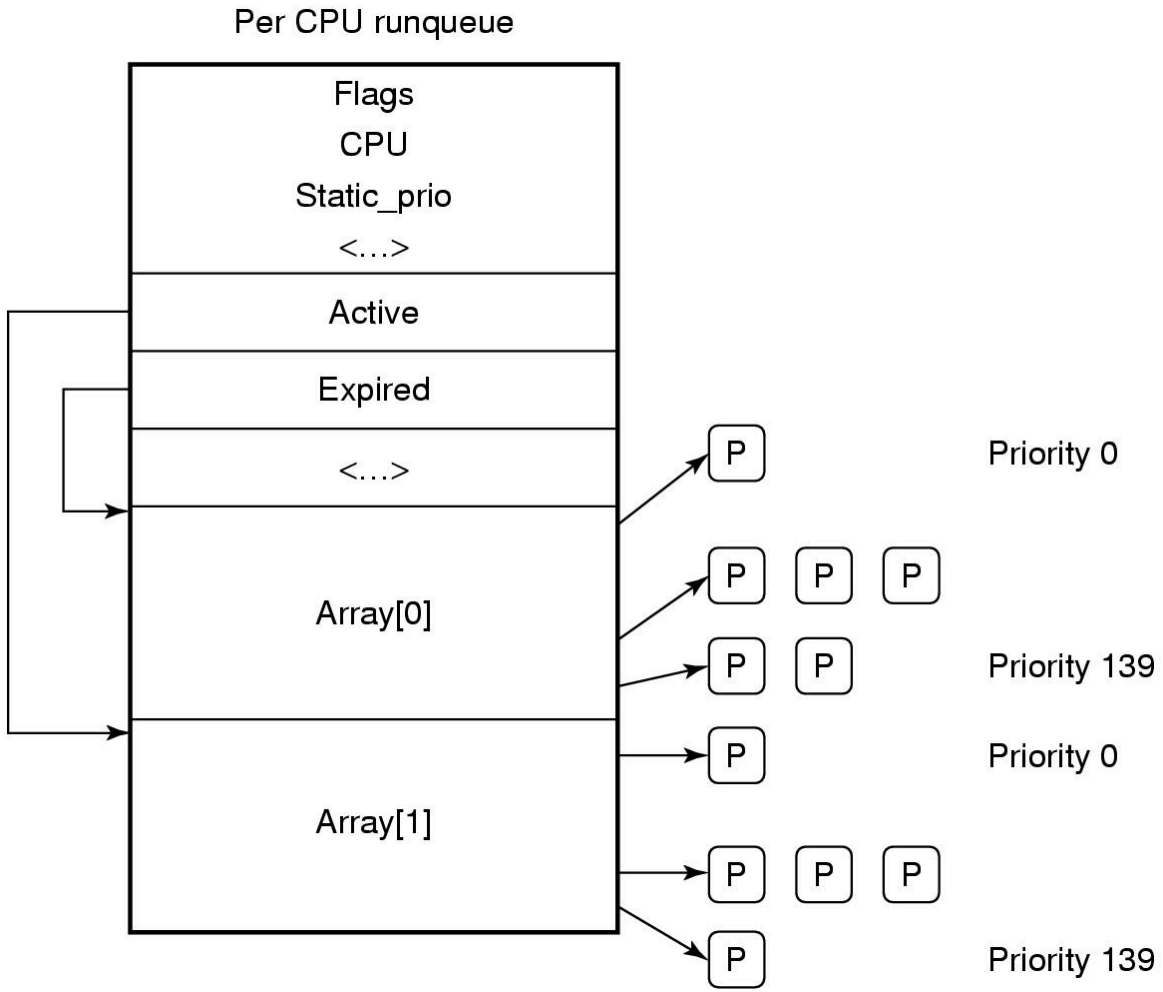
- W schedulerze O(1) runqueue jest zorganizowana w dwie tablice,: aktywna i wygasła.
- Każda z nich jest tablicą 140 nagłówków list, z których każda odpowiada innemu priorytetowi. Każdy nagłówek listy wskazuje na podwójnie powiązaną listę procesów o danym priorytecie. Podstawowe działanie schedulera można opisać w następujący sposób:
  - Harmonogram wybiera zadanie z listy najwyższych priorytetów w aktywnej tablicy.
  - Jeżeli w zadaniu timeslice (quantum) expires, jest on przenoszony na listę wygasłych (potencjalnie na innym poziomie priorytetu).
  - Gdy nie ma już zadań w aktywnej tablicy tablica, planista po prostu wymienia wskaźniki, więc tablice, które wygasły, stają się teraz aktywny i odwrotnie.
- Metoda ta zapewnia, że zadania o niskim priorytecie nie będą ‚głodować' (z wyjątkiem sytuacji, gdy wątki FIFO w czasie rzeczywistym całkowicie zatrzymają procesor, co jest mało prawdopodobne).

OCENA ALGORYTMU O(1)

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

- Algorytm planowania O(1) stał się popularny w wczesnych wersjach jądra 2.6.
- Wcześniejsze algorytmy wykazywały niską wydajność w systemach wieloprocesorowych i nie wykazywały się wystarczającą skalowalnością, przy zwiększonej liczbie zadań.
- Decyzja dotycząca wyboru procesu może zostać podjęta w drodze dostępu do odpowiedniej aktywnej listy, można to zrobić w stałym czasie O(1), niezależnie od liczba procesów w systemie.
- Jednakże, pomimo pożądanej własności w zakresie pracy w trybie ciągłym, algorytm O(1) miał istotne braki. W szczególności, heurystyki wykorzystywane do określenia interaktywności zadania, a zatem ich priorytet. Dlatego był nie najlepszym rozwiązaniem dla zadań interaktywnych.

RUNQUEUE I TABLICA PRIORYTETÓW ALGORYTM O(1)

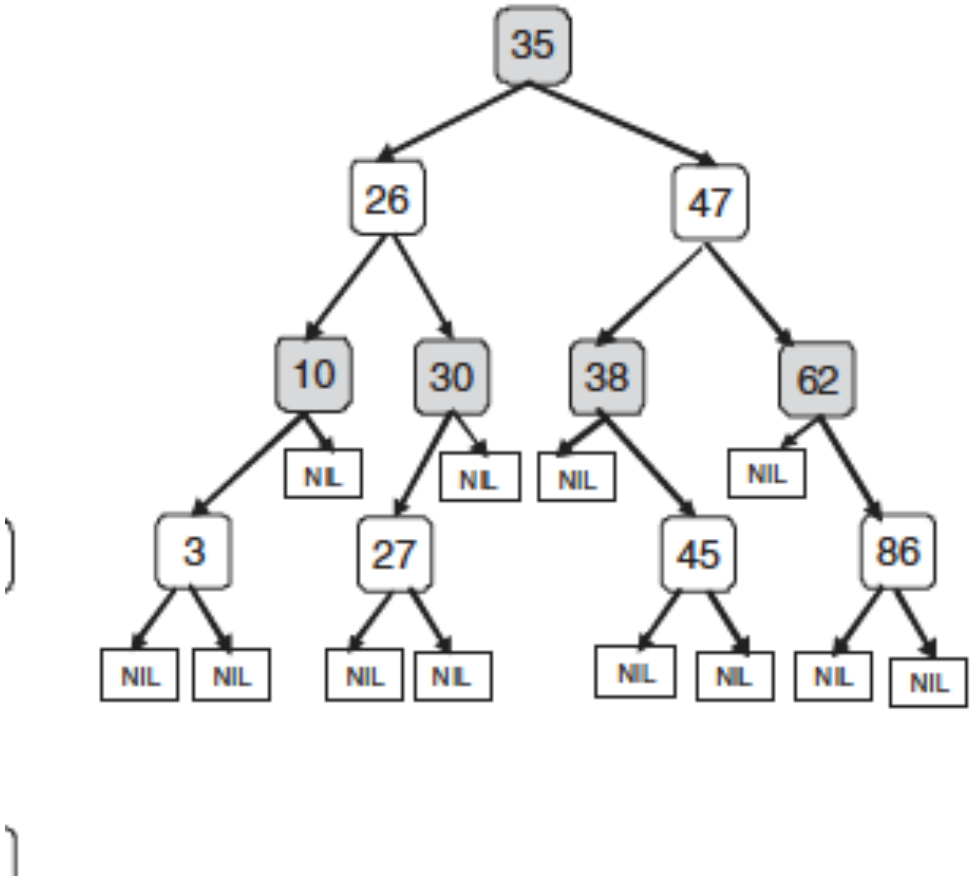


ALGORYTM CFS

- Jest domyślnym algorytmem harmonogramowania dla zadań w czasie rzeczywistym od wersji jądra 2.6.23.
- Głównym założeniem CFS jest wykorzystanie ‚czerwono-czarnego' drzewa jako struktury danych runqueue.
- Zadania są uporządkowane w drzewie w oparciu o ilość czasu spędzonego na procesorze, zwanym vruntime.
- CFS rozlicza czas pracy zadań z dokładnością do nanosekundy.
- Każdy wewnętrzny węzeł w drzewie odpowiada zadaniu. Dzieci po lewej stronie odpowiadają zadaniom, które miały mniej czasu na procesorze i dlatego będą zaplanowane wcześniej, a dzieci po prawej stronie węzła to te, które do tej pory zużywały więcej czasu na procesor. Węzły liście w drzewie nie odgrywają żadnej roli.

OCENA ALGORYTMU CFS

- CFS zawsze planuje zadanie, które miało najmniej czasu na procesorze, zazwyczaj jest to najbardziej lewy węzeł w drzewie. Okresowo CFS zwiększa wartość vruntime zadania w oparciu o czas, który już upłynął, i porównuje ją z bieżącym lewym węzłem w drzewie.
- W przypadku zadań o niższym priorytecie, czas mija szybciej, ich wartość vruntime będzie rosła szybciej, a w w stosunku do innych zadań, stracą one CPU i zostaną ponownie włożone do drzewa wcześniej, niż gdyby miał wyższy priorytet. W w ten sposób CFS unika stosowania oddzielnych struktur runqueue dla różnych priorytetów.
- wybranie węzła do uruchomienia można wykonać w stałym czasie, natomiast wstawienie zadania do runqueue odbywa się w czasie O(log(N)), gdzie N to liczba zadań w systemie.



ALGORYTM CSF

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

URUCHAMIANIE SYSTEMU LINUX

1. Po uruchomieniu komputera, system BIOS wykonuje funkcję Power-On-Self-Test (POST) oraz początkowe wykrywanie i inicjalizację urządzeń, ponieważ proces uruchamiania systemu operacyjnego może opierać się na dostępie do dysków, ekranów, klawiatur itp.
2. Pierwszy sektor dysku startowego, MBR (Master Boot Record), jest odczytywany do stałej lokalizacji pamięci i wykonywany. Sektor ten zawiera mały (512-bajtowy) program który ładuje samodzielny program zwany boot z urządzenia rozruchowego, np. dysku.
3. Program startowy najpierw kopiuje się do stałej pamięci o wysokiej wartości. aby uwolnić niską pamięć dla systemu operacyjnego.

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

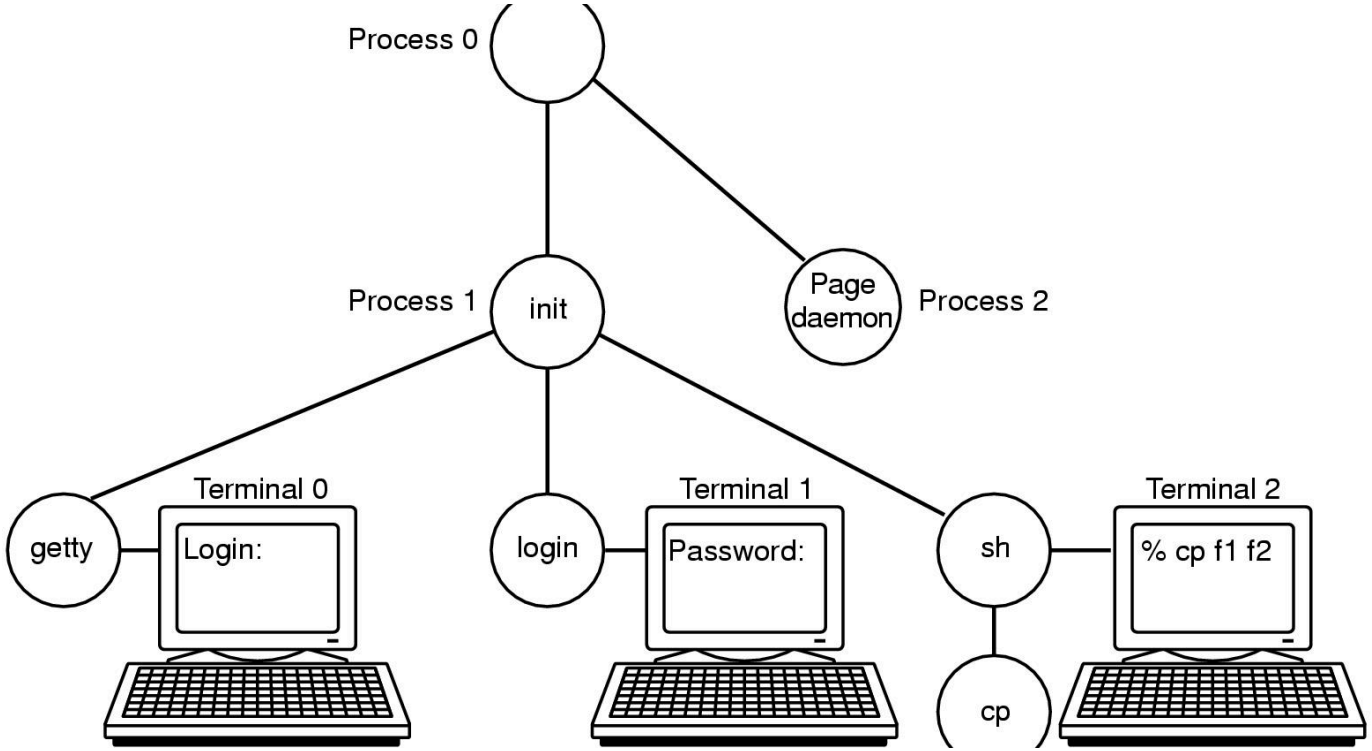
URUCHAMIANIE SYSTEMU LINUX CD.

4. Po przeniesieniu, boot odczytuje katalog główny urządzenia rozruchowego. Aby to zrobić, musi rozumieć system plików i format katalogu, co ma miejsce w przypadku niektórych bootloaderów, takich jak GRUB (GRand Unified Bootloader). Inne popularne bootloadery, takie jak Intel's LILO, nie opierają się na żadnym konkretnym systemie plików. Zamiast tego, potrzebują mapy bloków i adresów niskiego poziomu, które opisują sektory fizyczne, głowice i cylindry, aby znaleźć odpowiednie sektory do załadowania.
5. Następnie boot czyta jądro systemu operacyjnego i przeskakuje do niego. W tym momencie zakończył on swoją pracę.
6. Kod rozruchu jądra jest napisany w asemblerze i jest zależny od procesora. Typowe zadania obejmują
   - ustawienie stosu jądra,
   - identyfikację typu procesora,
   - obliczenie ilości pamięci RAM,
   - wyłączenie przerwań,
   - włączenie MMU,
   - a wreszcie wywołanie głównej procedury języka C w celu uruchomienia głównej części systemu operacyjnego.

URUCHAMIANIE SYSTEMU LINUX CD.

7. Następnie przydzielane są struktury danych jądra. Większość z nich ma stały rozmiar, ale kilka, takich jak pamięć podręczna strony i pewne struktury tablicy stron, zależy od ilości dostępnej pamięci RAM.
8. Następnie system rozpoczyna automatyczną konfigurację. Korzystając z plików konfiguracyjnych informujących o tym, jakie urządzenia I/O mogą być obecne, system zaczyna badać urządzenia, aby zobaczyć, które z nich są rzeczywiście obecne. Jeśli urządzenie zareaguje na sygnał, to jest dodane do tabeli dołączonych urządzeń. Jeśli nie odpowiada, przyjmuje się, że jest nieobecne i od tej pory ignorowanie. W przeciwieństwie do tradycyjnych wersji UNIX, w Linux sterowniki nie muszą być połączone statycznie i mogą być ładowane dynamicznie (tak jak we wszystkich wersjach MS-DOS i Windows).
9. Po skonfigurowaniu całego sprzętu, następne jest wykonanie procesu 0, ustawienie jego stosu i uruchomienie go. Proces 0 kontynuuje inicjalizację, robiąc takie rzeczy jak
   - programowanie zegara czasu rzeczywistego,
   - montowanie systemu plików root oraz
   - tworzenie init (proces 1) i
   - demona stron (proces 2).

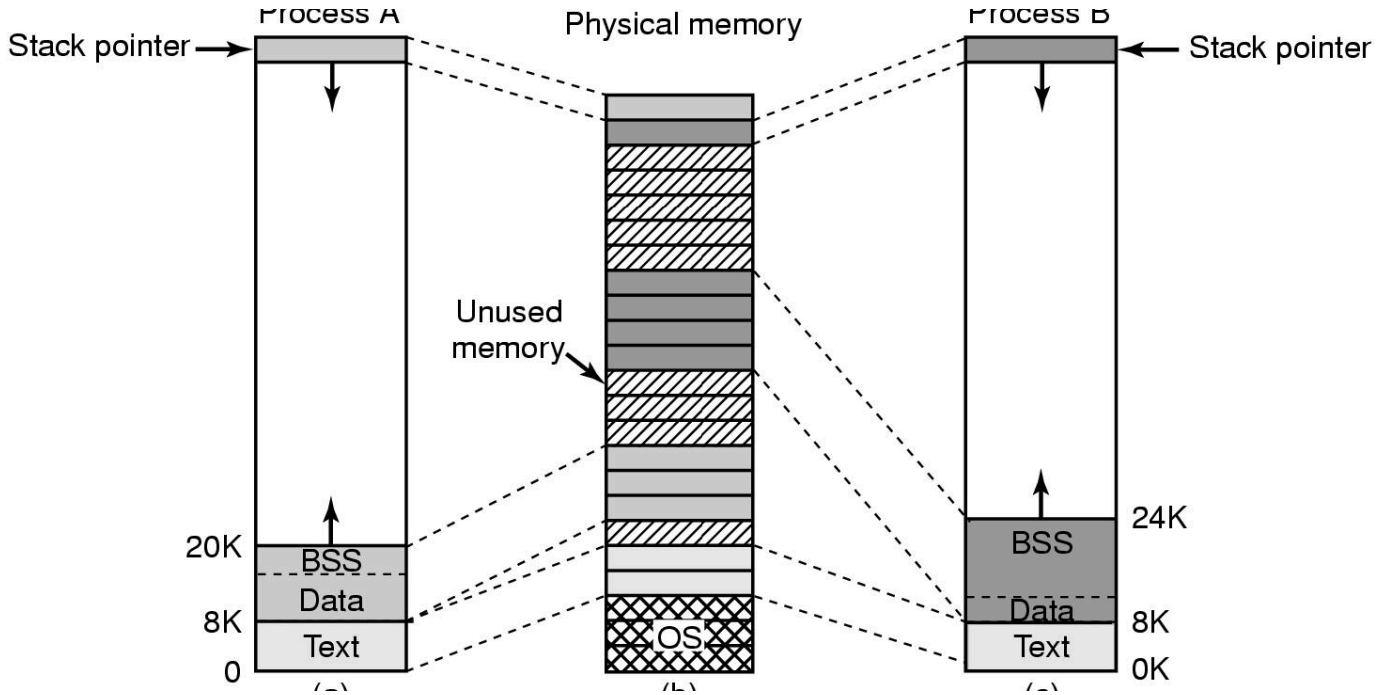
BOOTING LINUX



AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

ZARZĄDZANIE PAMIĘCIĄ PODSTAWOWE POJĘCIA:

- Każdy proces Linuksa ma przestrzeń adresową, która składa się z trzech segmentów: tekstu, danych i stosu.
- Segment tekstowy zawiera instrukcje maszynowe tworzące kod wykonywalny programu. Jest on wytwarzany przez kompilator i asembler, tłumacząc program C, C++ lub inny program na kod maszynowy. Segment tekstowy jest zazwyczaj tylko do odczytu. Tekst nie rośnie ani nie kurczy się, ani nie zmienia się w żaden inny sposób.
- Segment danych zawiera dane dla wszystkich zmiennych programu, łańcuchów, tablic i innych danych. Składa się on z dwóch części, danych inicjowanych i danych nieinicjalizowanych. Z przyczyn historycznych ten ostatni znany jest jako BSS (historycznie nazywany Block Started by Symbol). Zainicjalizowana część segmentu danych zawiera zmienne i stałe kompilatora, które potrzebują wartości początkowej w momencie uruchomienia programu. Wszystkie zmienne w części BSS są inicjalizowane do zera po załadowaniu.
- Segment stosu najczęściej zaczyna się od góry wirtualnej przestrzeni adresowej i rośnie w dół w kierunku 0. Na przykład, na 32-bitowe platformy x86, stos zaczyna się od adresu 0xC000000000, z limitem adresu 3-GB. Jeśli stos rośnie poniżej dolnej część segmentu stosu, pojawia się przerwanie i system operacyjny obniża dolną część segmentu stosu o jedną stronę. Programy nie zarządzają wielkością stosu.



ZARZĄDZANIE PAMIĘCIĄ W SYSTEMIE LINUX (A I B ADRESY WIRTUALNE, PAMIĘĆ FIZYCZNA)

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

PLIK ODWZOROWANY

Procesy w systemie Linux mogą uzyskać dostęp do pliku poprzez pliki z mapami pamięci.

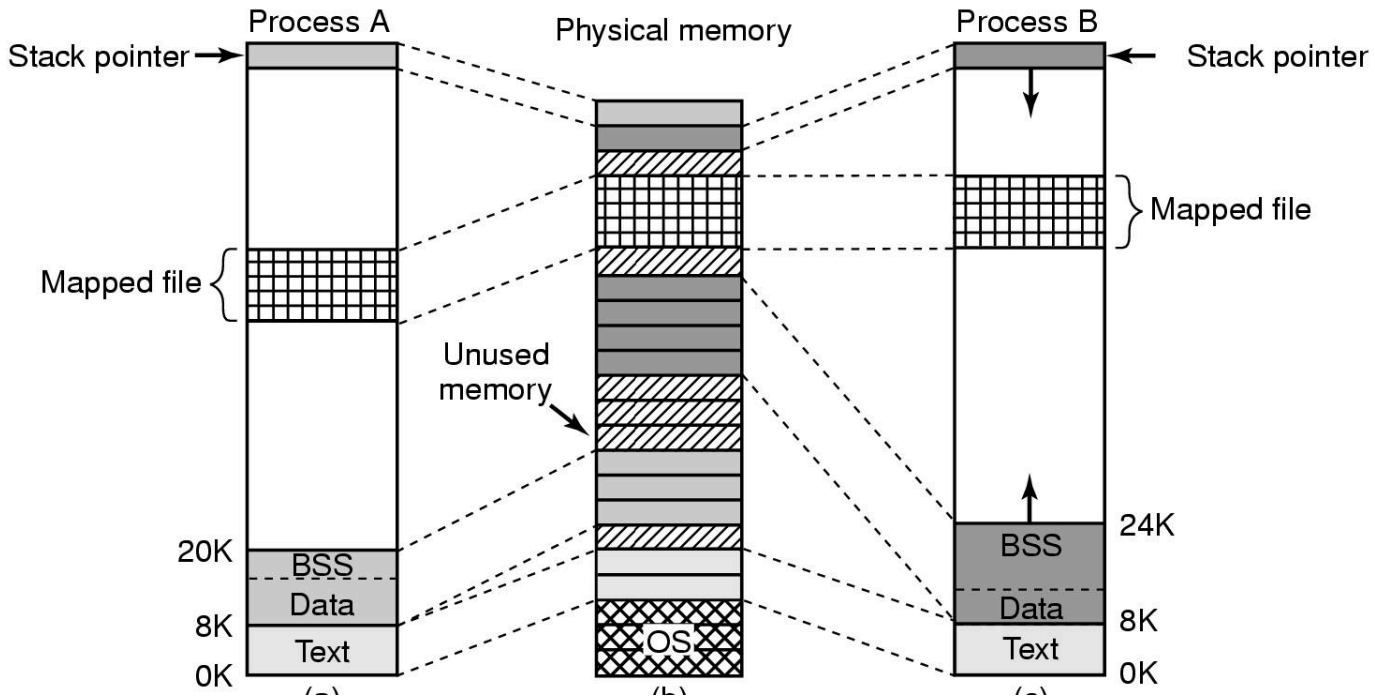
Funkcja ta umożliwia mapowanie pliku na część przestrzeni adresowej procesu, tak aby plik mógł być odczytany i zapisany tak, jakby był tablicą bajtową w pamięci.

Mapowanie pliku w pamięci sprawia, że losowy dostęp do niego jest znacznie łatwiejszy niż korzystanie z wywołań systemowych I/O, takich jak odczyt i zapis.

Współdzielone biblioteki są dostępne poprzez mapowanie ich tym samym mechanizmem .

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

WSPÓŁDZIELENIE MOGĄ WSPÓŁDZIELIĆ ZMAPOWANE PLIKI



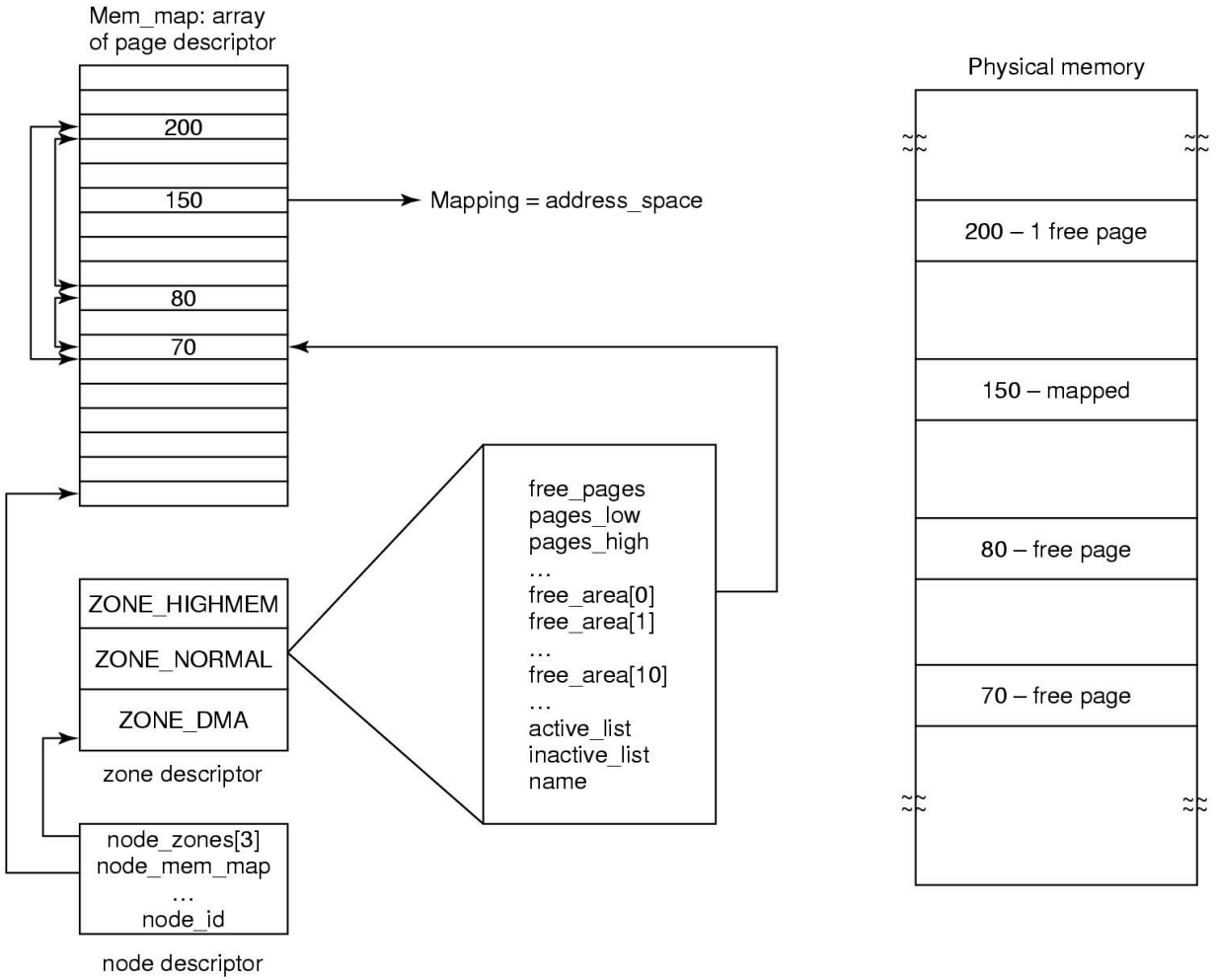
WYWOŁANIA SYSTEMOWE - ZARZĄDZANIE PAMIĘCIĄ

System call	Description
s = brk(addr)	Change data segment size
a = mmap(addr, len, prot, flags, fd, offset)	Map a file in
s = unmap(addr, len)	Unmap a file

ZARZĄDZANIE PAMIĘCIĄ POZIOM FIZYCZNY

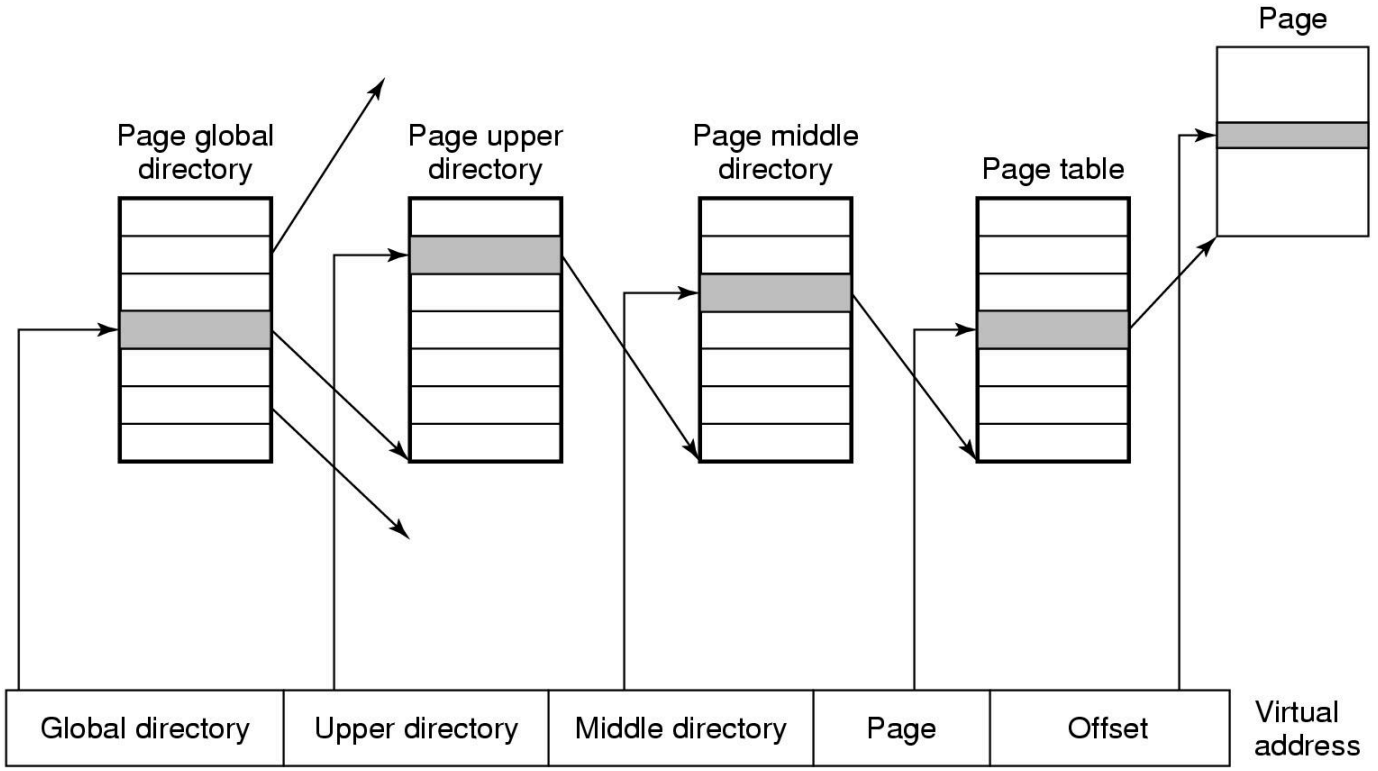
- Pamięć jądra zazwyczaj znajduje się w niskiej pamięci fizycznej. Na obecnych 64-bitowych maszynach x86 do adresowania używa się tylko do 48 bitów, co oznacza teoretyczny limit 256 TB dla adresów. Linux rozdziela tę pamięć pomiędzy jądro i przestrzeń użytkownika, co daje maksymalnie 128 TB wirtualnej przestrzeni adresowej na proces. Przestrzeń adresowa jest tworzona gdy proces jesr tworzony.
- Linux rozróżnia trzy strefy pamięci:
  - ZONE_DMA - strony, które mogą być używane w operacjach DMA.
  - ZONE_NORMAL - normalne, regularnie mapowane strony.
  - ZONE_HIGHMEM - strony z wysokimi adresami pamięci, które nie podlegają ustawicznemu odwzorowaniu.

GŁÓWNA REPREZENTACJA PAMIĘCI W LINUX



AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

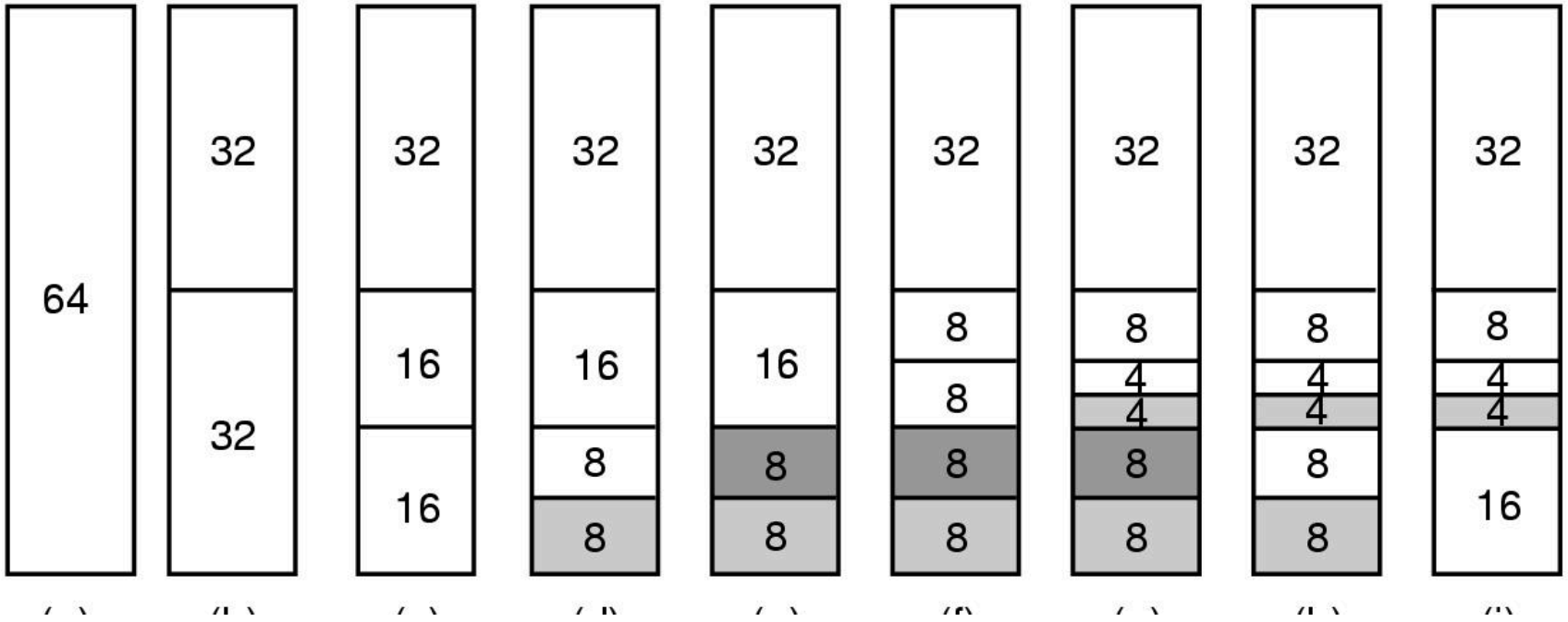
CZTERO-POZIOMOWA TABLICA STRON



MECHANIZMY ALOKACJI PAMIĘCI – BUDDY ALGORITHM

Liczba stron jest
Zaokrąglana do najbliższej
Potęgi 2

1 żądanie: 8 stron
2 żądanie 8 stron
3 żadanie 4 strony
4 zwolnien 8 stron
5 zwolnienie 8 stron



Nie łączy się obszarów należących do różnych „buddy”

STRONNICOWANIE

- Podstawowa idea stronicowania w SO Linux jest prosta: proces nie musi być całkowicie w pamięci, aby mógł zostać uruchomiony. Wymagane są struktura użytkownika i tabele stron.
- Jeśli są wprowadzone, proces jest uważany za będący "w pamięci" i może być zaplanowany do uruchomienia. Strony tekstu, danych i segmentów stosu są wprowadzane dynamicznie, jeden po drugim.
- Jeśli struktura użytkownika i tabela stron nie są w pamięci, proces nie może zostać uruchomiony, dopóki nie zostanie wprowadzony przez swapper.
- Stronicowanie jest realizowane częściowo przez jądro, a częściowo przez proces zwany ”page deamon” (proces 2).
- Strony nie są przydzielane do urządzenia lub partycji, dopóki nie są potrzebne.

TYPY STRON W SO LINUX

- Linux rozróżnia cztery różne typy stron:
  - nieodzyskiwalne (unreclaimable)- zawierają strony zastrzeżone lub zablokowane, stosy trybu jądra i tym podobne, nie mogą być wykorzystane,
  - wymienialne (swappable) - muszą zostać zapisane z powrotem do obszaru wymiany lub partycji dysku wywoławczego, zanim strona będzie mogła zostać odzyskana
  - synchronizowane (syncable) - muszą zostać zapisane z powrotem na dysk, jeśli zostały oznaczone jako brudne
  - usuwalne (discardable) - można odzyskać natychmiast.

ALGORYTM WYMIANY PFRA (PAGE FRAME RECLAIMING ALGORITHM)

PFRA najpierw próbuje odzyskać łatwe strony, a następnie przechodzi do stron trudniejszych.

Strony usuwalne i takie bez odwołań można odzyskać natychmiast, przenosząc je na listę wolnych w danej strefie.

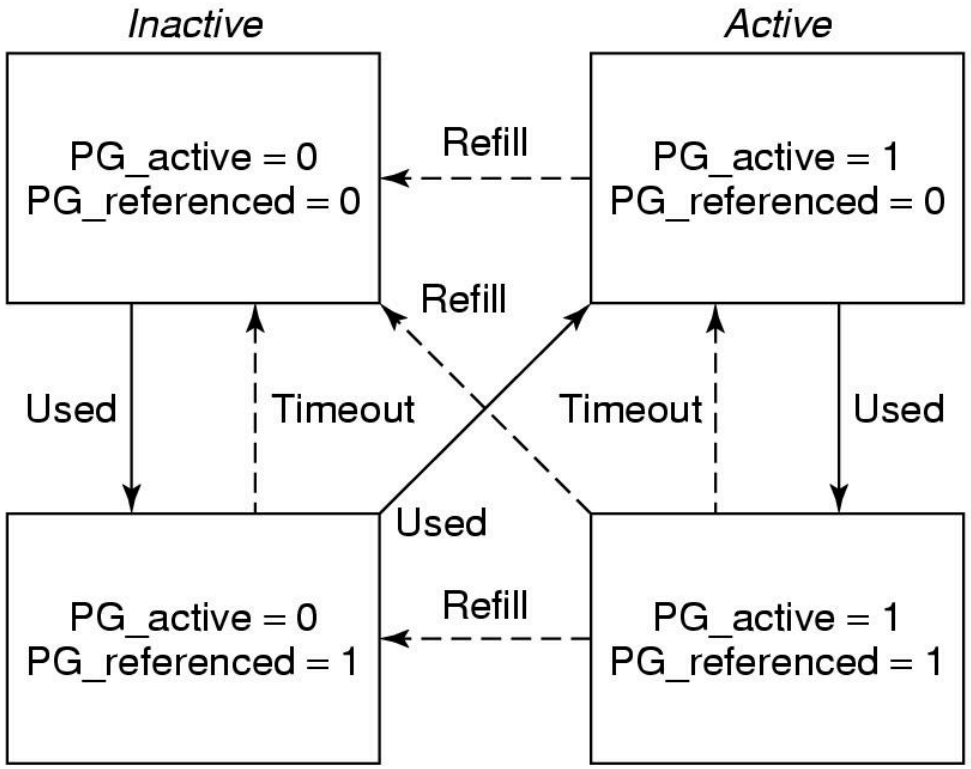
Następnie szuka stron z pamięci zapasowej, do których nie było dawno odwołań.

Strony wyjątkowo rzadko odwiedzane próbuje się usunąć.

Następnie zwykłych stron, które były używane i chce je zwolnić przenosząc do obszaru wymiany.

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

STANY STRON Z PERSPEKTYWY ALGORYTMU ZAMIANY STRON



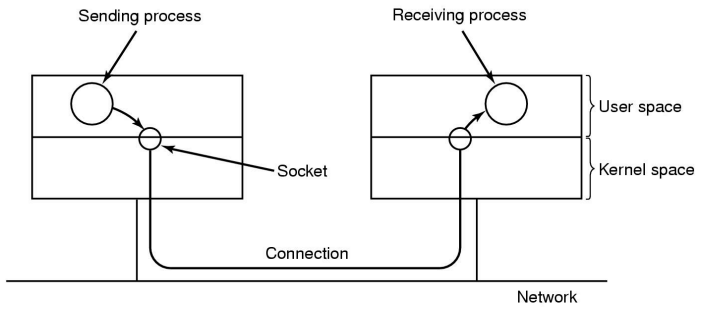
OPERACJE WEJŚCIA-WYJŚCIA W SO LINUX

- System I/O w Linuksie jest dość prosty i taki sam jak w innych UNIXach.
- Zasadniczo, wszystkie urządzenia we/wy są wykonane tak, aby wyglądały jak pliki i są dostępne jako takie z tymi samymi wywołaniami systemowymi, które są używane do uzyskiwania dostępu do wszystkich zwykłych plików.
- W niektórych przypadkach parametry urządzenia muszą być ustawione i odbywa się to za pomocą specjalnego wywołania systemowego.
- Urządzenia to pliki specjalne i mają przypisaną ścieżkę do katalogu /dev np.
  - dysk twardy /dev/hda1
  - drukarka /dev/lp
  - Interface sieciowy /dev/net

I/O C.D.

- Dostęp do urządzeń jak do innych plików poprzez funkcje systemowe open, read i write.
- Pliki blokowe: to pliki składający się z sekwencji ponumerowanych bloków. Kluczową właściwością jest to, że każdy blok może być indywidualnie adresowany i udostępniany. Program może otworzyć plik blokowy i odczytać np. blok 124 bez konieczności wcześniejszego odczytywania bloków od 0 do 123. Pliki te są zazwyczaj używane na dyskach.
- Pliki znakowe są zwykle używane dla urządzeń, które wprowadzają lub wysyłają strumień znaków. Klawiatury, drukarki, sieci, myszy, plotery i większość innych urządzeń we/wy, które akceptują lub produkują dane dla ludzi Nie ma możliwości (lub nawet znaczenia) blok 124 dla odczytu sygnału myszy.
- Do każdego pliku specjalnego przypisany jest sterownik urządzenia, który obsługuje odpowiednie urządzenie. Każdy sterownik ma numer urządzenia, który służy do zidentyfikuj go. Jeśli sterownik obsługuje wiele urządzeń, na przykład dwa dyski tego samego typu, każdy dysk ma pomniejszy numer urządzenia, które go identyfikuje.

I/O SIEĆ



- Kluczową koncepcją w rozwiązaniach z Berkeley jest socket. Socket są podobne do skrzynek pocztowych i gniazdek telefonicznych, ponieważ umożliwiają użytkownikom łączenie się z siecią, podobnie jak skrzynki pocztowe pozwalają ludziom łączyć się z systemem pocztowym, a gniazdka telefoniczne pozwalają im podłączać telefony i łączyć się z systemem telefonicznym.

WYKORZYSTANIE SOCKET W POŁĄCZENIACH SIECIOWYCH

- Każde gniazdo (socket) obsługuje określony typ komunikacji sieciowej. Najczęściej spotykane typy to

  1. Niezawodny strumień bajtów zorientowany na połączenie - pozwala na połączenie dwóch procesów na różnych maszynach w postaci potoku. Bajty są „pompowane” do tego potoku z jednej strony, a z drugiej wypływaj. System gwarantuje niezawodność.

  2. Niezawodny, zorientowany na połączenie strumień pakietów – jak pierwsza, ale z limitem wysyłanej paczki np. zamiast wysłać wszystko za jednym razem dzieli się dane na porcje i dopiero wysyła.

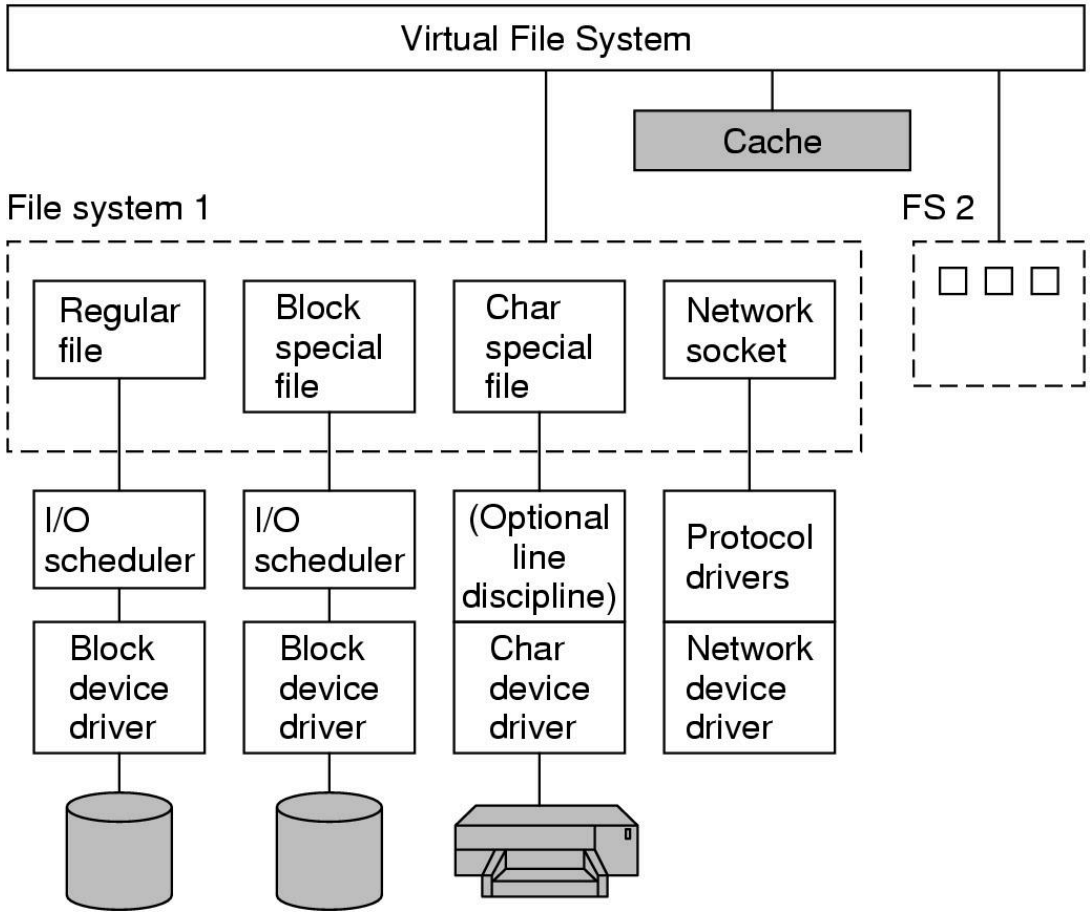
  3. Zawodna transmisja pakietów –bezpośredni dostęp do sieci, używany w systemach czasu rzeczywistego

WYWOŁANIA ZGODNE Z POSIX DO ZARZĄDZANIA TERMINALEM

Function call	Description
s = cfsetospeed(&termios, speed)	Set the output speed
s = cfsetispeed(&termios, speed)	Set the input speed
s = cfgetospeed(&termios, speed)	Get the output speed
s = cfgtetispeed(&termios, speed)	Get the input speed
s = tcsetattr(fd, opt, &termios)	Set the attributes
s = tcgetattr(fd, &termios)	Get the attributes

NIEKTÓRE OPERACJE ZWIĄZANE Z PLIKAMI DLA URZĄDZEŃ ZNAKOWYCH.

Device	Open	Close	Read	Write	Ioctl	Other
Null	null	null	null	null	null	...
Memory	null	null	mem_read	mem_write	null	...
Keyboard	k_open	k_close	k_read	error	k_ioctl	...
Tty	tty_open	tty_close	tty_read	tty_write	tty_ioctl	...
Printer	lp_open	lp_close	error	lp_write	lp_ioctl	...



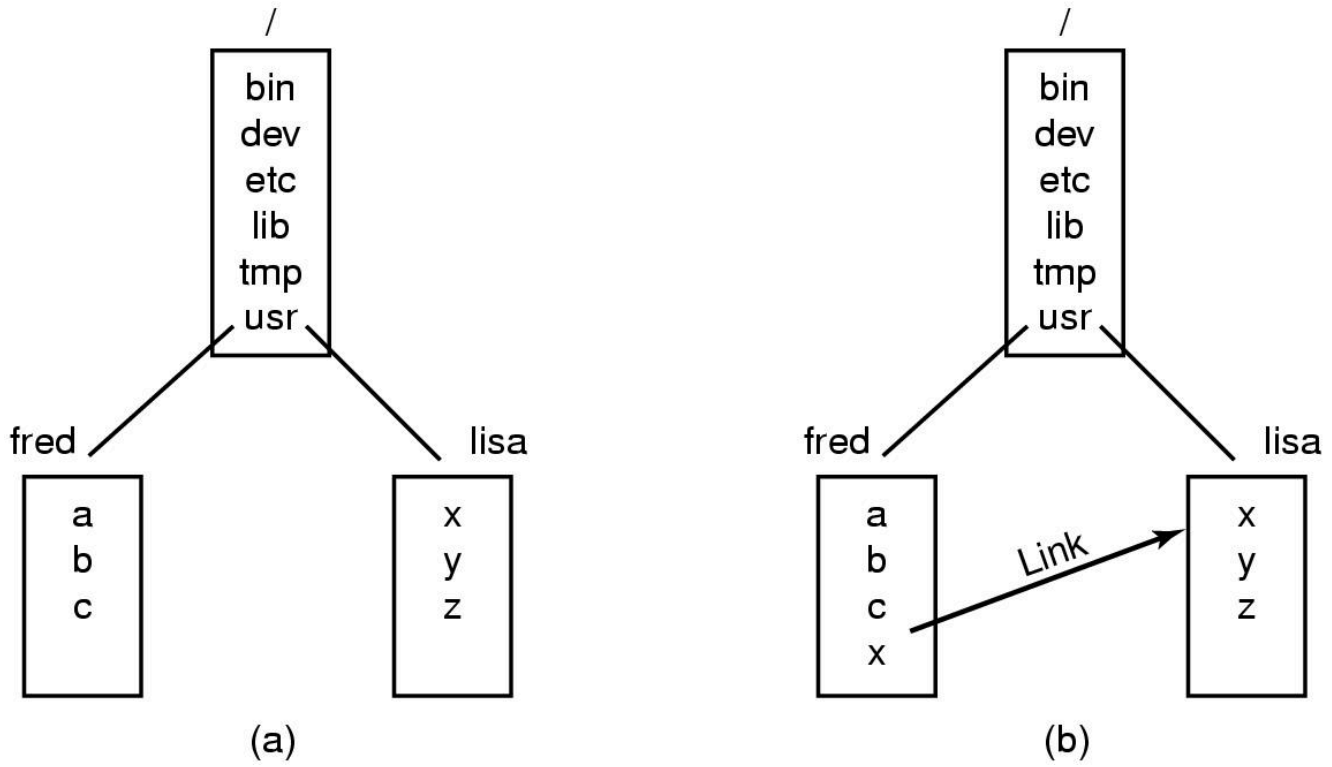
SYSTEM I/O POKAZUJĄCY SZCZEGÓŁOWO JEDEN SYSTEM PLIKÓW.

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

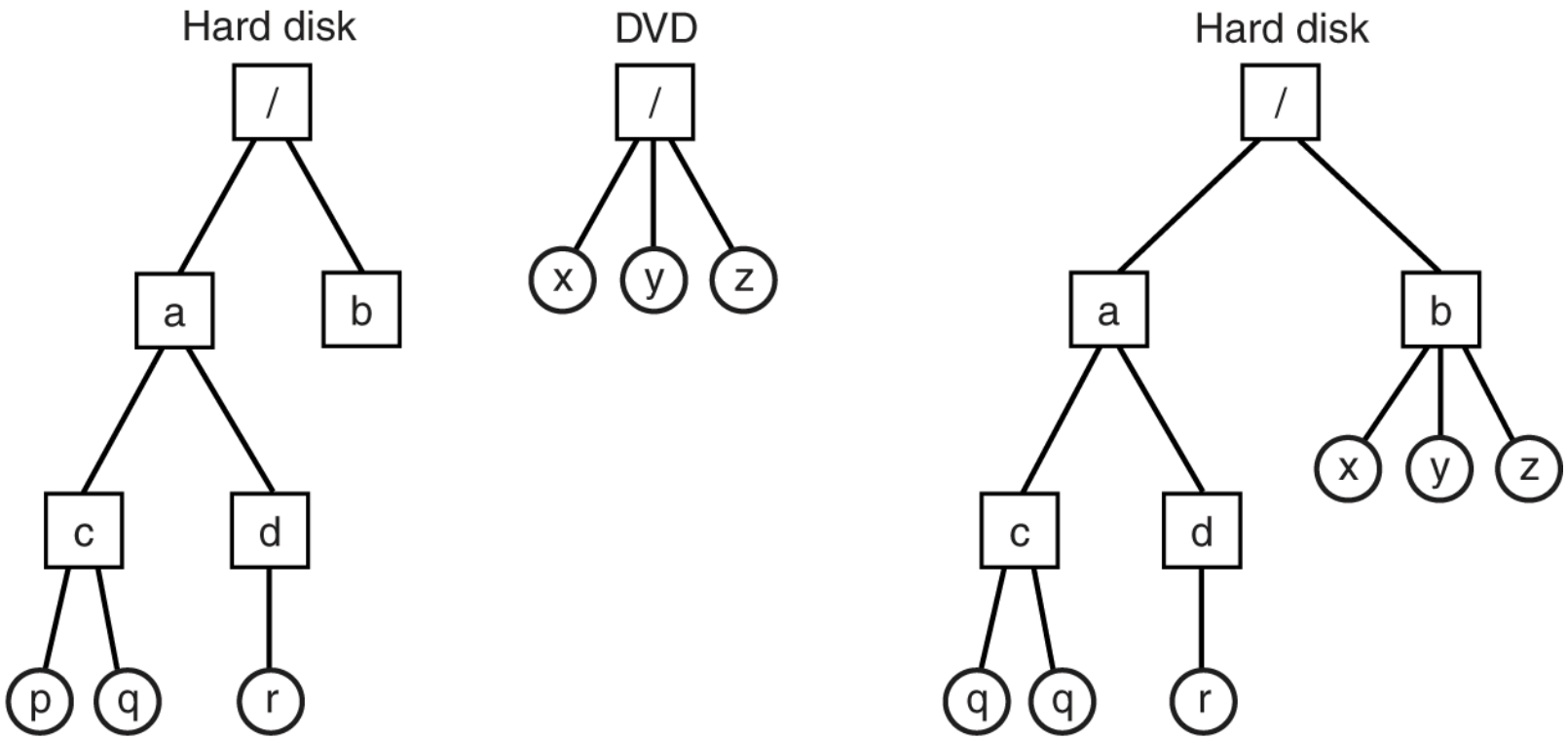
SYSTEM PLIKÓW

Directory	Contents
bin	Binary (executable) programs
dev	Special files for I/O devices
etc	Miscellaneous system files
lib	Libraries
usr	User directories

SYSTEM PLIKÓW – DOWIĄZANIA



SYSTEM PLIKÓW – MONTOWANIE URZĄDZEŃ



WYWOŁANIA SYSTEMOWE ZWIĄZANE Z SYSTEMEM PLIKÓW

System call	Description
fd = creat(name, mode)	One way to create a new file
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information
s = fstat(fd, &buf)	Get a file's status information
s = pipe(&fd[0])	Create a pipe
s = fcntl(fd, cmd, ...)	File locking and other operations

STAT – POLECENIE SYSTEMOWE

Device the file is on
I-node number (which file on the device)
File mode (includes protection information)
Number of links to the file
Identity of the file's owner
Group the file belongs to
File size (in bytes)
Creation time
Time of last access
Time of last modification

WYWOŁANIA SYSTEMOWE ZWIĄZANE Z KATALOGAMI

System call	Description
s = mkdir(path, mode)	Create a new directory
s = rmdir(path)	Remove a directory
s = link(oldpath, newpath)	Create a link to an existing file
s = unlink(path)	Unlink a file
s = chdir(path)	Change the working directory
dir = opendir(path)	Open a directory for reading
s = closedir(dir)	Close a directory
dirent = readdir(dir)	Read one directory entry
rewinddir(dir)	Rewind a directory so it can be reread

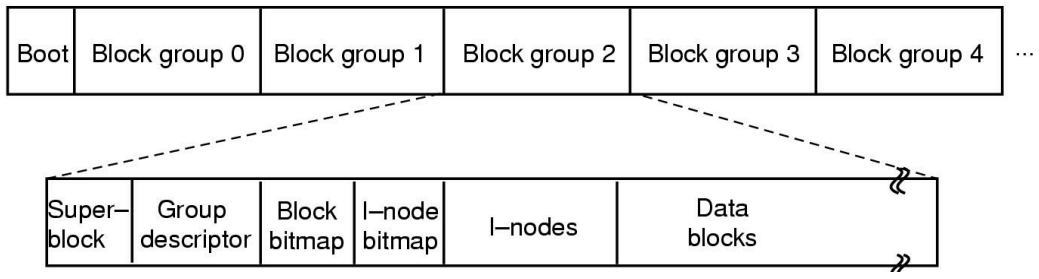
VFS – WIRTUALNY SYSTEM PLIKÓW

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

- Aby umożliwić aplikacjom interakcję z różnymi systemami plików, zaimplementowanymi na różnych typach urządzeń lokalnych lub zdalnych, Linux stosuje podejście stosowane w innych systemach UNIX: wirtualny system plików (VFS).
- VFS definiuje abstrakcje systemów plików i operacje, które są dozwolone na tych abstrakcjach.
- Cztery abstrakcje VFS

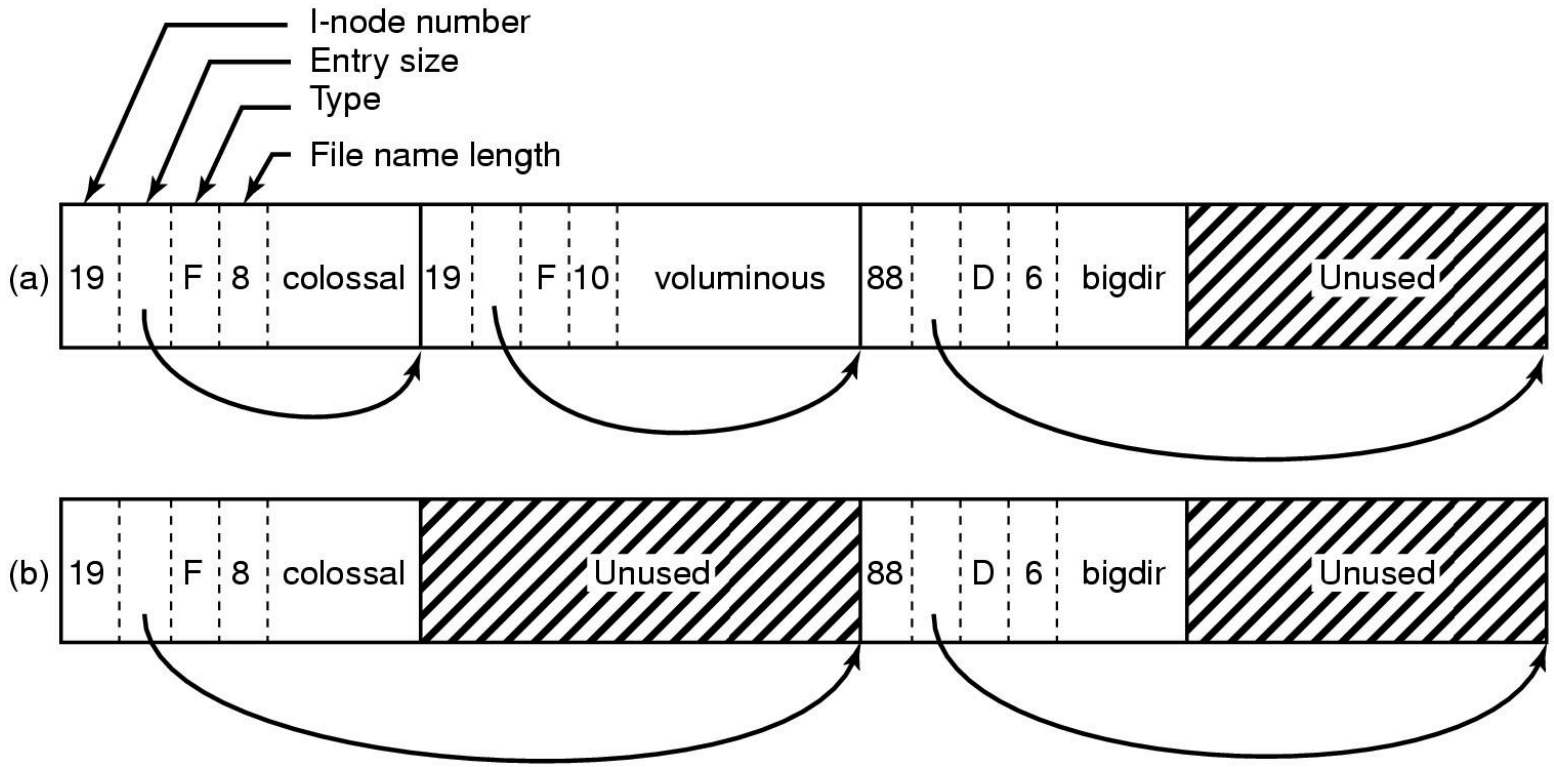
Object	Description	Operation
Superblock	specific file-system	read_inode, sync_fs
Dentry	directory entry, single component of a path	create, link
I-node	specific file	d_compare, d_delete
File	open file associated with a process	read, write

SYSTEM PLIKÓW EXT 2



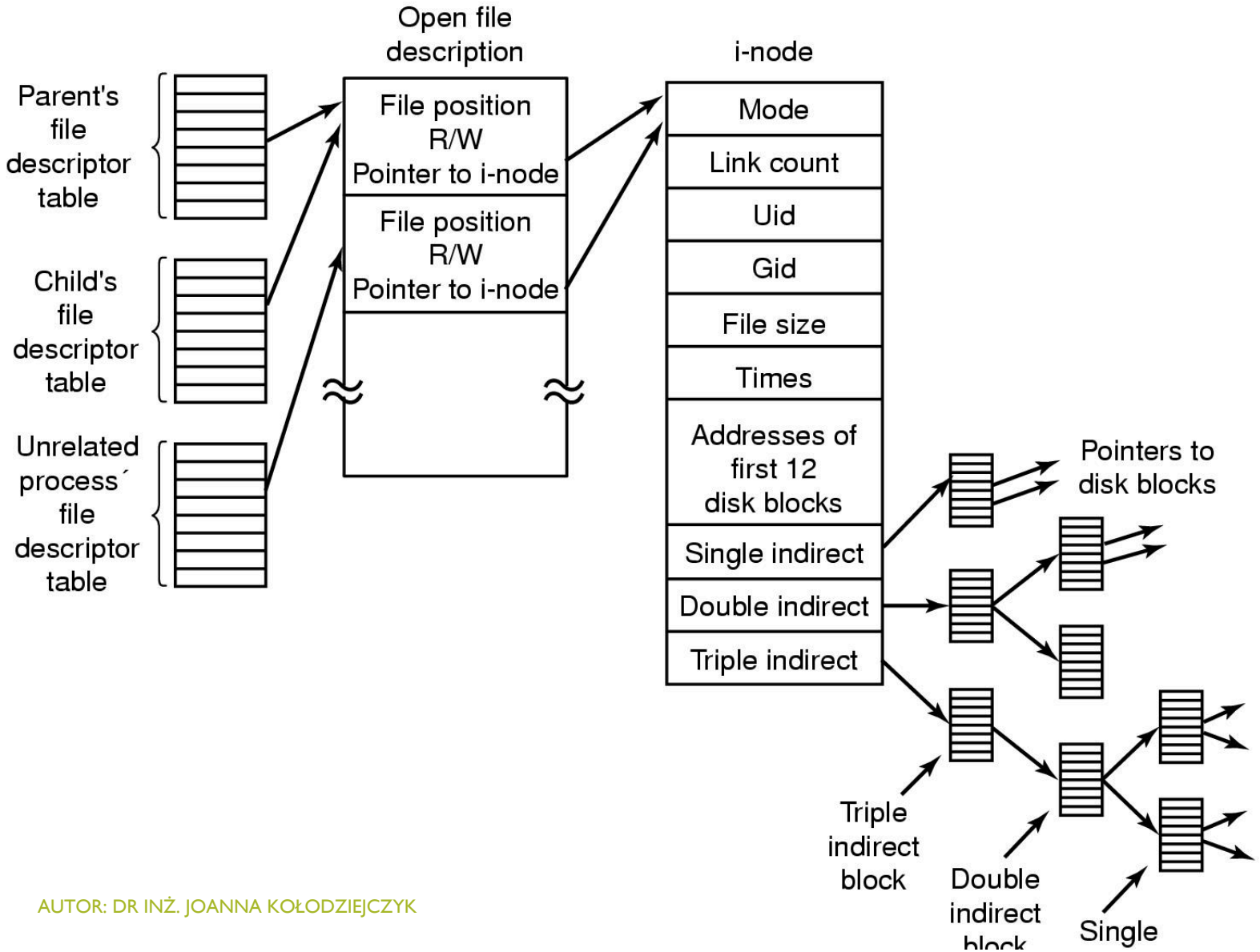
- Superblock zawiera informacje na temat układu graficznego system plików, w tym liczbę i-węzłów, liczbę bloków dyskowych, oraz początek listy wolnych bloków dyskowych (zazwyczaj kilkaset wpisów).
- Deskrytpor grup – ważny ponieważ ext2 próbuje się rozprzestrzeniać. katalogi równomiernie nad dyskiem.

A) KATALOG LINUX Z TRZEMA PLIKAMI. B) TEN SAM PO USUNIĘCIU PLIKU O DUŻEJ OBJĘTOŚCI.

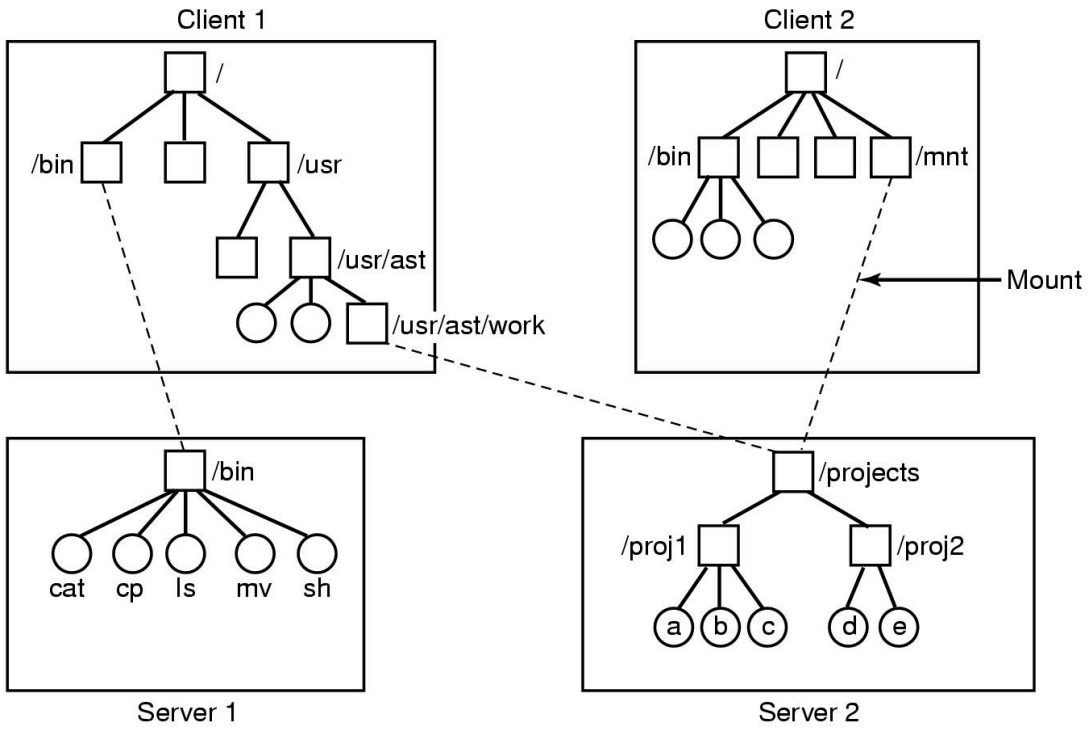


NIEKTÓRE POLA W STRUKTURZE I-NODE W SYSTEMIE LINUX

Field	Bytes	Description
Mode	2	File type, protection bits, setuid, setgid bits
Nlinks	2	Number of directory entries pointing to this i-node
Uid	2	UID of the file owner
Gid	2	GID of the file owner
Size	4	File size in bytes
Addr	60	Address of first 12 disk blocks, then 3 indirect blocks
Gen	1	Generation number (incremented every time i-node is reused)
Atime	4	Time the file was last accessed
Mtime	4	Time the file was last modified
Ctime	4	Time the i-node was last changed (except the other times)



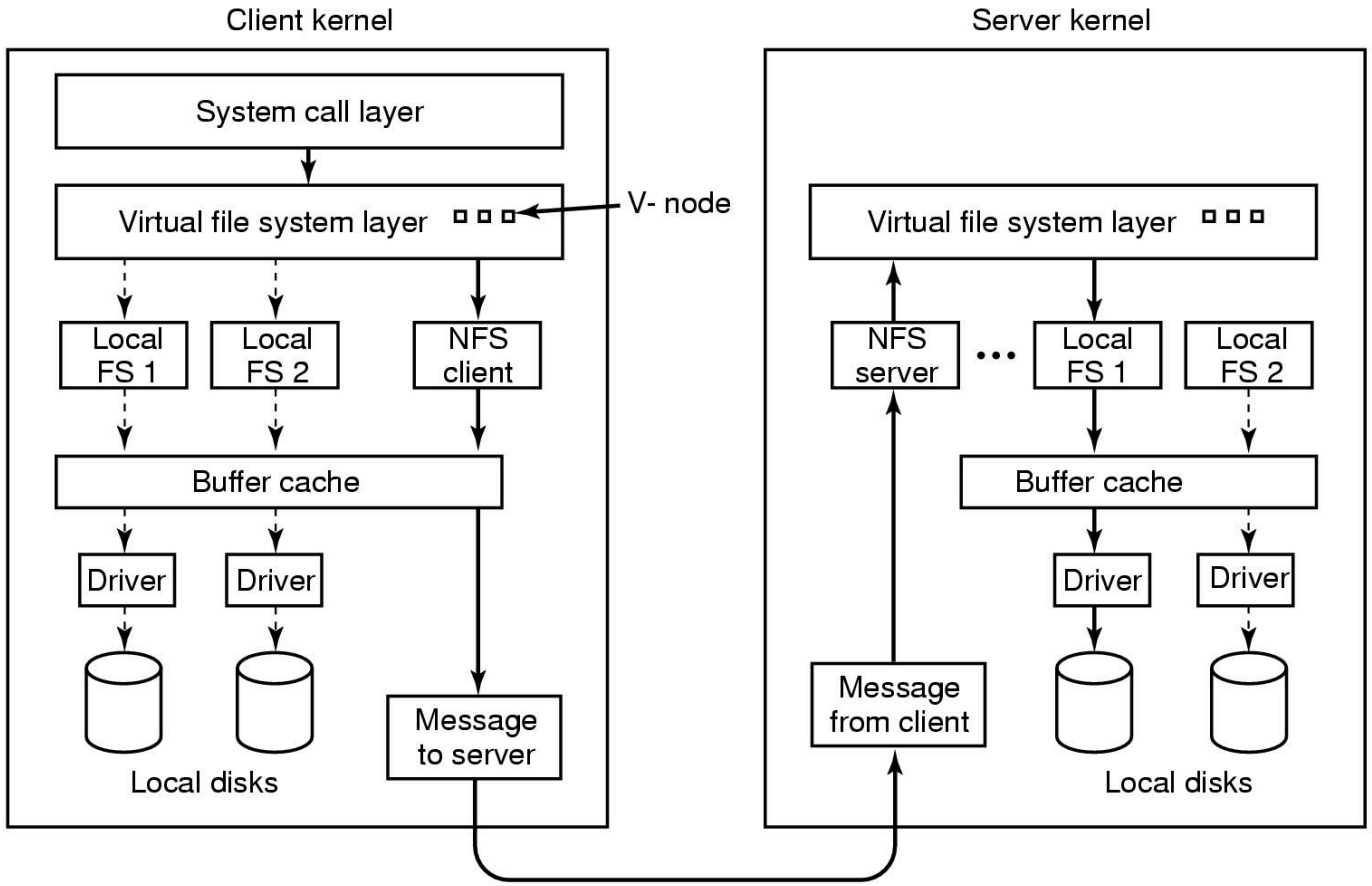
RELACJA MIĘDZY TABELĄ DESKRYPTORÓW PLIKÓW, OTWARTĄ TABELĄ OPISU PLIKÓW I TABELĄ I-NODE.



PRZYKŁADY ZDALNIE MONTOWANYCH SYSTEMÓW PLIKÓW. KATALOGI POKAZANE JAKO KWADRATY, PLIKI POKAZANE JAKO KOŁA – SYSTEM NFS

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

STRUKTURA WARSTWOWA NFS



AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

BEZPIECZEŃSTWO

Pojęcia podstawowe:

- UID – User ID
- GID – Group ID

Każdy proces ma przypisane UID i GID swojego właściciela i poziom uprawnień.

AUTOR: DR INŻ. JOANNA KOŁODZIEJCZYK

NIEKTÓRE PRZYKŁADOWE TRYBY OCHRONY PLIKÓW

Binary	Symbolic	Allowed file accesses
111000000	rwx------	Owner can read, write, and execute
111111000	rwxrwx---	Owner and group can read, write, and execute
110100000	rw-r-----	Owner can read and write; group can read
110100100	rw-r--r--	Owner can read and write; all others can read
111101101	rwxr-xr-x	Owner can do everything, rest can read and execute
000000000	---------	Nobody has any access
000000111	------rwx	Only outsiders have access (strange, but legal)

WYWOŁANIA SYSTEMOWE ZWIĄZANE Z BEZPIECZEŃSTWEM

System call	Description
s = chmod(path, mode)	Change a file's protection mode
s = access(path, mode)	Check access using the real UID and GID
uid = getuid( )	Get the real UID
uid = geteuid( )	Get the effective UID
gid = getgid( )	Get the real GID
gid = getegid( )	Get the effective GID
s = chown(path, owner, group)	Change owner and group
s = setuid(uid)	Set the UID
s = setgid(gid)	Set the GID