AGH

Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Systemy operacyjne
Wykład 01N

Wersja 2024

dr inż. Marek Wilkus http://home.agh.edu.pl/~mwilkus
Wydział Inżynierii Metali i Informatyki Przemysłowej
AGH Kraków

Na podstawie programu opracowanego przez dr inż. Krzysztofa Wilka

AGH

Warunki zaliczenia

- Zaliczenie z ćwiczeń laboratoryjnych (wg warunków prowadzącego) – obecność!
- Pisemny egzamin z wykładów.

AGH

Tematyka zajęć

- Ewolucja systemów operacyjnych.
- Rodzaje, właściwości i zadania systemów.
- Jądro systemu – budowa, funkcje.
- Procesy, współpraca między procesami, współistnienie.
- Zarządzanie (organizacja, adresowanie) pamięcią.
- System plików.
- Urządzenia wejścia-wyjścia.
- Systemy rozproszone.
- Systemy czasu rzeczywistego.
- Przykłady systemów i ich budowy.

AGH

Laboratoria

- System UNIX:
  - Działanie w systemie,
  - Poruszanie się w systemie plików,
  - Wykorzystanie narzędzi,
- Powłoka systemu:
  - Pisanie skryptów powłoki,
  - Automatyzacja zadań,
- Funkcje systemowe (opcjonalnie)
- Język PERL (opcjonalnie)

AGH

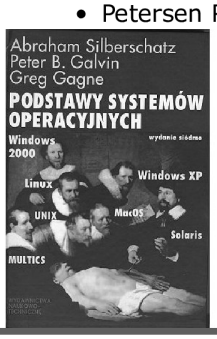
Literatura

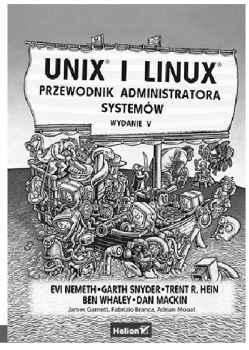
- Silberschatz A. Galvin P.B. - Podstawy systemów Operacyjnych
- Nemeth E., Snyder G. Hein T. Whaley B. - Unix i Linux: Przewodnik administratora systemów, Helion 2011
- Lister A. M. Eager R. D. - "Wprowadzenie do systemów operacyjnych", WNT 1994
- Stevens R.W. - Programowanie w środowisku systemu UNIX, WNT 2002

AGH

Literatura – c.d.

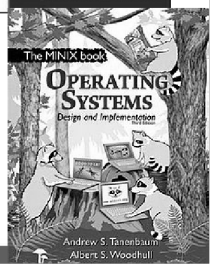
- Bach M.J. - Budowa systemu operacyjnego UNIX, WNT 1995
- Petersen R. - Arkana Linux.





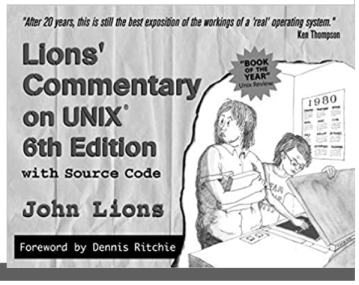
Literatura – dodatkowa

- Johnson E.F., Welch J.C., Anderson M. "Skrypty powłoki od podstaw", wyd. Helion, 2006
  - Praktyczny materiał do ćwiczeń z wykorzystania powłoki systemu, programowania Bash, Perl, sed, awk.
- Tanenbaum A.S., Woodhull A. - "Operating Systems: Design and implementation", 2006
  - Pozycja typu "Zrób to sam: uniksopodobny system operacyjny".



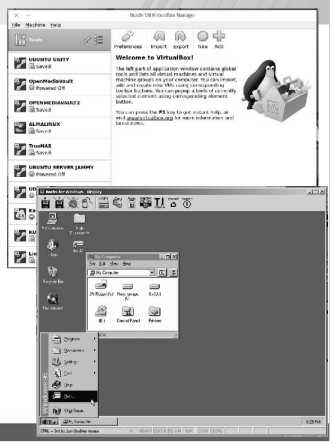
Literatura – dodatkowa c.d.

- "Podręcznik Lionsa", "Kod Lionsa" (https://warsus.github.io/lions-/)
  - Kompletny opis systemu Unix V6 wraz z porządnie komentowanym kodem źródłowym.



Chcę tylko poeksperymentować...

- VirtualBox – Program typu „Komputer w komputerze” – do testowania systemów operacyjnych o większych wymaganiach sprzętowych.
  - Dobry start jeżeli chcemy przetestować lub nauczyć się konfigurowania Linuksa.
  - Za mocne dla testowania starszych systemów.
- BOCHS – Emulatory PC do testowania starszych systemów.
  - PCem – ten emulator skupia się na wierności odtwarzanego sprzętu, głównie starszego.

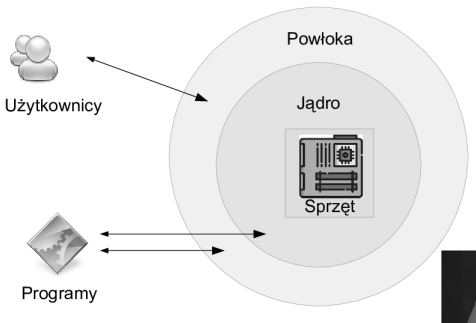


System operacyjny

- Definicja (wg A. S. i P. B. G.):

  System operacyjny jest programem, który działa jako pośrednik pomiędzy użytkownikiem komputera a sprzętem komputerowym. Zadaniem systemu operacyjnego jest tworzenie środowiska w którym użytkownik może wykonywać programy.

Warstwowa budowa systemu operacyjnego



The 3D depiction of this model in „The UNIX operating system - AT&T Archives”, 1982

Składniki systemu operacyjnego

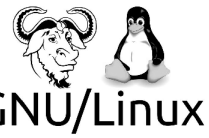
- Jądro - komunikuje się z komputerem przez sterowniki urządzeń i wykonuje kolejkowanie zadań, obsługę pamięci.
- Powłoka - stanowi interpreter poleceń systemu (komunikacja z użytkownikiem).
- Programy - polecenia systemowe nie zawarte w jądrze, programy narzędziowe, programy użytkowe.

System operacyjny

- Głównym celem systemu operacyjnego jest to, aby był system komputerowy był **wygodny w użyciu**.
- Drugim celem jest **wydajna eksploatacja** sprzętu komputerowego.

Kilka pojęć na początek

- UNIX - Wielozadaniowy, wielodostępny system operacyjny, pierwotnie opracowany jako jądro + zestaw programów, rozwijanych głównie na uniwersytetach. Dostępny komercyjnie w różnych dystrybucjach sprzedawany przez różne firmy (Solaris, IBM AIX, Tru64 UNIX, HP-UX).
- GNU - wolna implementacja systemu UNIX-owego oparta w zamierzeniu na jądrze Hurd.
- Linux - Jądro systemu na wolnej licencji. Jeżeli działa na nim ekosystem GNU, to pełna nazwa (której raczej nikt nie używa) to GNU/Linux.



- BSD (Berkeley Software Distribution) - Kolejna wolna implementacja systemu uniksowego, oparta na własnym jądrze systemu.

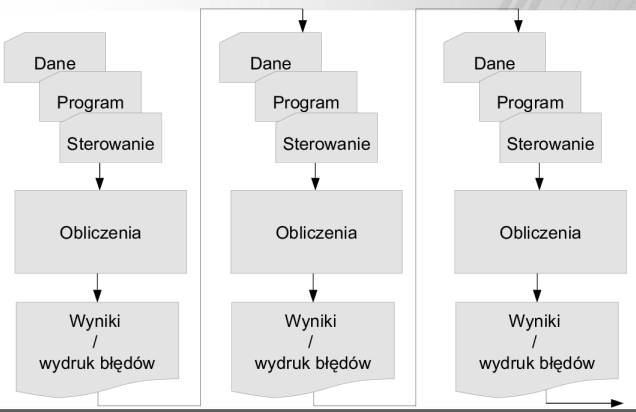
Dystrybucje

- Dystrybucje to zestawy jądro + system bazowy + programy. Takimi dystrybucjami GNU/Linuksa będzie np. Debian, Arch, CentOS, Ubuntu, a systemu BSD FreeBSD, NetBSD, PC-BSD.
- Czy można zrealizować np. GNU/BSD, czyli ekosystem GNU na jądrze BSD? Projekt Debian przeprowadził taki eksperyment, lecz korzyści wydajnościowe nie były zbyt duże.
  - Projekt Nexenta: GNU na jądrze OpenSolarisa.
  - Cygwin - ekosystem GNU działa na Windowsach.

ubuntu debian archlinux gentoo linux CentOS SUSE

Ewolucja systemów operacyjnych

Systemy wsadowe



Systemy wsadowe

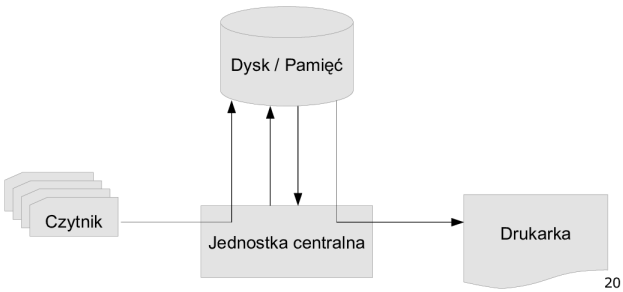
- Zadania wykonywane są kolejno.
- Każde zadanie składa się z wczytywania programu i danych, obliczeń oraz zapisu (lub wydruku) wyników.
- Następne zadanie wykonywane jest po zakończeniu poprzedniego.
- Kolejność zadań ustalana przez operatora.
- Zadania o podobnych wymaganiach grupowane są w tzw. wsad (batch).

Systemy wsadowe

- Zalety:
  - Bardzo prosty system, tylko automatyczne przekazywanie sterowania od jednego zadania do drugiego.
- Wady:
  - Brak nadzoru użytkownika podczas wykonywania zadania (niemożność podjęcia nawet prostych decyzji, interakcji).
  - Tylko jedno zadanie w danym czasie.
  - Duża bezczynność jednostki centralnej podczas wczytywania i wydruku danych (obliczenia: tysiące operacji / s, wczytywanie: kilka/kilkanaście kart / s).

SPOOLing

- Simultaneous Peripheral Operation On-Line



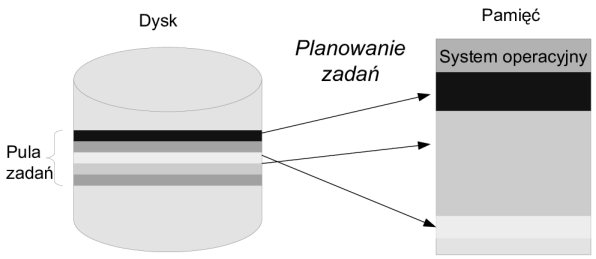
Spooling

- W tym samym czasie wykonywane są obliczenia jednego zadania, i operacje I/O innego.
- Kosztem zajęcia niewielkiej części pamięci stało się możliwe efektywniejsze wykorzystanie zasobów CPU i urządzeń peryferyjnych.
- Niektóre maszyny miały wsparcie sprzętowe na np. wypełnienie nieużywanego banku pamięci niezależnie od aktualnie wykonywanego programu.

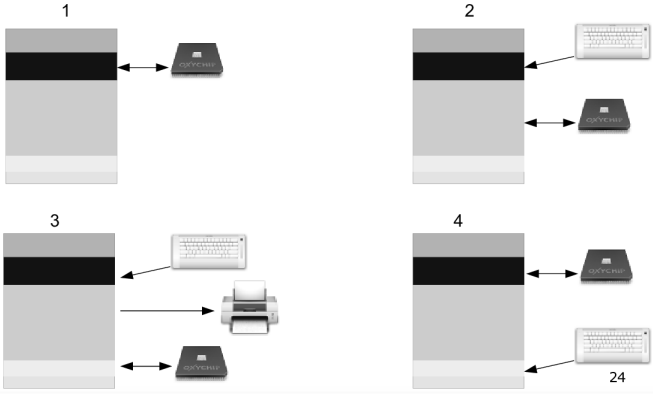
Spooling

- Zalety:
  - Lepsze wykorzystanie jednostki centralnej.
  - Minimalizacja "przestojów" urządzeń wejścia/wyjścia.
  - Możliwość pracy programu i urządzeń I/O w tym samym czasie.
- Wady:
  - Konieczność wykorzystania pamięci.

Wieloprogramowy system wsadowy



Zadania do pamięci zaczytywane są „całe" i w pamięci pozostają do ich końca.



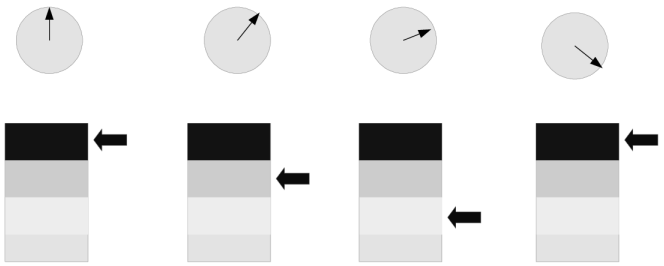
Wieloprogramowy system wsadowy

- W tym samym czasie system operacyjny przechowuje w pamięci kilka zadań.
- Gdy aktualnie wykonywane zadanie oczekuje na usługę lub zakończenie operacji (np. I/O), wykonywane jest następne zadanie.
- Gdy zakończyło się oczekiwanie, a następne zadania są zajęte, system powraca do wykonywania poprzedniego zadania.

Wieloprogramowy system wsadowy

- Zalety:
  - Lepsze wykorzystanie procesora i urządzeń I/O.
  - Szybsze wykonywanie puli programów.
- Wady:
  - Wymaga skomplikowanego systemu operacyjnego (planowanie zadań, przydział procesora).
  - Zadanie obliczeniowe, bez operacji I/O, może na długo zablokować resztę zadań.

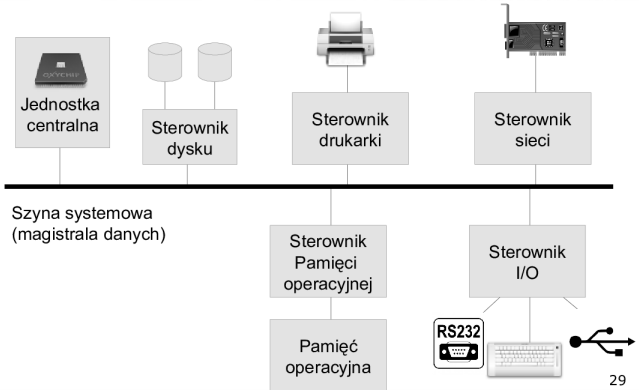
System z podziałem czasu



System z podziałem czasu

- Procesor wykonuje wiele różnych zadań na przemian.
- Przełączanie następuje tak często, że użytkownicy mogą współdziałać z każdym programem podczas jego wykonywania.
- Każdy z użytkowników odnosi wrażenie, ze dysponuje własnym komputerem, choć jest on w rzeczywistości współdzielony.

Budowa systemu komputerowego



Przerwania

- **Przerwanie** jest sygnałem pochodzącym od sprzętu lub oprogramowania, sygnalizującym wystąpienie **zdarzenia**.
- Sygnał przerwania sprzętowego pochodzi z zewnętrznych układów obsługujących przerwania sprzętowe. Służą one do komunikacji z urządzeniami zewnętrznymi (np. napędy dyskowe, sieć, porty).
- Sygnały przerwań od sprzętu wysyłane są do procesora najczęściej za pośrednictwem szyny systemowej.

Przerwania

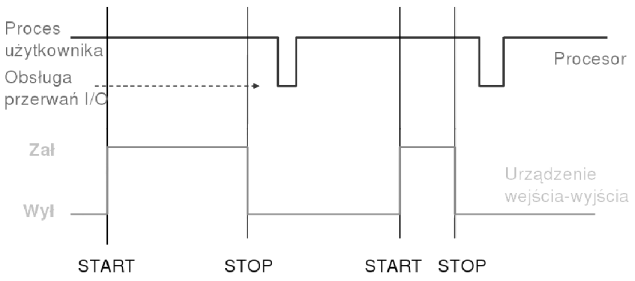
- Urządzenie wymagające obsługi procesora (np. kontroler I/O, moduł DMA) posiada specjalne fizyczne połączenie do procesora. (tzw. linia przerwania). W momencie, gdy urządzenie sygnalizuje przerwanie, wystawia odpowiedni sygnał na linii przerwania (z reguły stan wysoki lub zbocze z 0 na 1).

Zdarzenia powodujące przerwanie

- Zakończenie operacji wejścia/wyjścia,
- Niektóre błędy (np. dzielenie przez zero),
- Niedozwolony dostęp do pamięci,
- Zapotrzebowanie na usługę systemu,
- Dostęp przez interfejs zarządzania.
- itd...

Każdemu przerwaniu odpowiada procedura jego obsługi.

Przerwania procesu wykonującego operację I/O



Wektor przerwań

- Aby przyspieszyć operację obsługi przerwań stosuje się tablicę wskaźników do procedur obsługujących poszczególne przerwania.
- Indeksy tej tablicy odpowiadają numerom urządzeń „generujących" przerwania, a elementami tablicy są adresy procedur obsługujących przerwania (ISR – Interrupt Service Routine).

Wejście w ISR

- Przy przejściu do obsługi przerwania należy zapamiętać adres przerwanego rozkazu, a także np. zawartości rejestrów, jeżeli obsługa przerwania zmienia je.
- W nowych systemach adres powrotny przechowywany jest na stosie systemowym.
- Podczas obsługi jednego przerwania inne są wyłączone, lub ustalone są **priorytety przerwań** (przerwania maskowane).
- Po powrocie z procedury przerwania często nie mamy żadnej informacji że cokolwiek takiego się odbyło.

SMI i NMI

- **NMI** jest niemaskowalne (zawsze zostanie wykonane) i często wykorzystywane jest w obsłudze poważnych problemów (np.. zatrzymanie systemu zasygnalizowane przez watchdog timer).

- **SMI** - System Management Interrupt – Jądro systemu zostaje przerwane i nie wiemy co się stało. System coś robił. Uznane przez ITL za zagrożenie bezpieczeństwa, bo wymyka się wszelkim analizatorom. Musi się je wykrywać specjalnymi timerami.

Przerwania w systemie Linux

- Przerwań jest dużo, pracy jest dużo, a nie możemy „zakorkować” systemu procedurami obsługi.
- **SoftIRQ** umożliwiają kolejkowanie prac wyzwolonych przez sprzętowe przerwanie (najwyższy priorytet). Inne procesory mogą zająć się wtedy przetwarzaniem zadań, a sam kod przerwania sprzętowego służy wyłącznie "zamówieniu" zadań w softirq wyzwalanym jako zadanie o dużym priorytecie.
- **Tasklets** są niepodzielnymi zadaniami bez własnego stosu/kontekstu, które umożliwiają wykonanie większych prac związanych z soft IRQ, są one z reguły przywiązane do konkretnego rdzenia.

Wyjątki

- Przerwania wewnętrzne, nazywane wyjątkami (ang. exceptions), zgłaszane są przez procesor dla sygnalizowania sytuacji wyjątkowych (np. błąd dzielenia przez 0, nieprawidłowy dostęp do pamięci, naruszenie zasobów współdzielonych).

Podział wyjątków

- **Faults** (niepowodzenia) – sytuacje, w których aktualnie wykonywana instrukcja powoduje błąd. Powrót do wykonywania przerwanego kodu powoduje ponowne wykonanie tej samej instrukcji.
  - Błędy arytmetyczne,
  - Niewłaściwy kod instrukcji (Intel),
  - Operacja na FPU bez FPU,
  - Niewłaściwy dostęp do pamięci,
  - Odwołanie do ramki, której nie ma (trzeba ją sprowadzić).

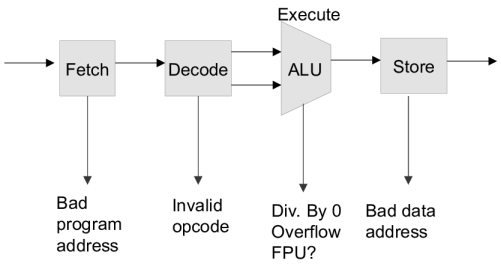
Podział wyjątków (c.d.)

- **Traps** (pułapki) – sytuacja ma na celu intencjonalne wykonanie określonego kodu. Wykorzystywana przede wszystkim w debuggerach gdy niezbędne jest zatrzymanie programu w danym punkcie. Powrót do wykonywania przerwanego kodu powoduje wykonanie **następnej** instrukcji.
  - Niewłaściwy kod instrukcji (Motorola),
  - Specyficzna instrukcja programu,
  - Breakpoint.

Podział wyjątków (c.d.)

- **Aborts** (problemy nienaprawialne) – Wywołane przez błędy, których nie można naprawić. Często nie można określić z którego miejsca w pamięci nastąpiło wywołanie wyjątku ani ponownie wykonać problematycznej instrukcji. Mogą być spowodowane przez błędy sprzętowe, niespójności w systemowych częściach pamięci, błędy niskopoziomowe, "bałagan" wprowadzony w system management mode.
  - Błąd CPU wykryty przez sam procesor,
  - Wyjątek podczas procedury obsługi wyjątku (double fault),
  - Wyjątek podczas obsługi double fault.

Przykłady generowanych wyjątków

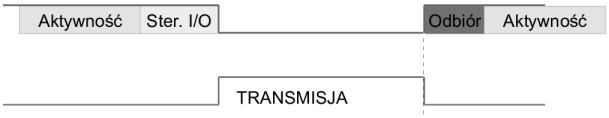


Przerwania programowe

- Z kodu programu wywoływana jest procedura obsługi przerwania;
- Najczęściej wykorzystywane do komunikacji z systemem operacyjnym, który w procedurze obsługi przerwania (np. w DOS 21h, Windows 2Fh, Linux x86 przerwanie 80h) umieszcza kod wywołujący odpowiednie funkcje systemowe w zależności od zawartości rejestrów ustawionych przez program wywołujący, lub do komunikacji z oprogramowaniem wbudowanym (procedury BIOS, firmware).
- Operacje Wejścia-wyjścia z reguły nie są dostępne wprost dla programu użytkownika – należy używać funkcji systemowej.

Wejście-wyjście

- Obsługa synchroniczna:



- Obsługa asynchroniczna:

Asynchroniczne I/O

- W czasie wykonywania wejścia-wyjścia jednostka centralna może zostać użyta do obliczeń lub do rozpoczęcia operacji.
- Ponieważ operacje I/O są powolne w porównaniu z szybkością procesora, procesor może w międzyczasie obsłużyć kilka innych zadań.

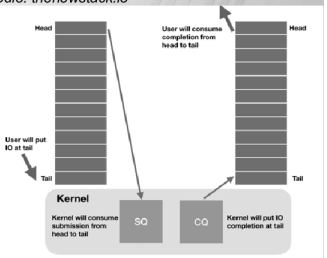
I/O Uring w Linuksie

- Obecne dopiero od Linuksa serii 5.x (2020)
- PROBLEM: Operacje I/O (read, write) są blokujące - program musi czekać na ich zakończenie.
- Wyobraźmy sobie więc serwer bazodanowy oczekujący ze wszystkimi swoimi użytkownikami, aż jeden z nich w końcu uzyska z bazy zamówiony wielki, binarny „blob", który czyta się z dysku już dłuższy czas…

- Dotychczasowe rozwiązanie: Wątki
  - Problem: Duży nakład na ich tworzenie i zamykanie,
  - Problem: Użycie pamięci,
  - Problem: Brak zapewnienia systemowego buforowania/cache'owania.
  - Narzut na otwieranie i zamykanie deksryptorów plików, nieraz w każdym wątku.

io_uring

Źródło: thenewstack.io



- Program „zamawia" operację I/O dodając ją do końca kolejki SQ (Submission Queue, kolejka zgłoszeń) i kontynuuje dalej pracę.
- Program okresowo sprawdza wyniki dla siebie na początku kolejki CQ (Completion Queue, zakończonych operacji), i przetwarza je odpowiednio.
- Przy operacji na plikach cała kolejka korzysta ze **wspólnego mapowania** deskryptorów (uchwytów) i **wspólnych buforów** jak i możliwe jest **grupowanie** operacji co znacznie zwiększa wydajność.

Bezpośredni dostęp do pamięci (DMA)

- Gdy urządzenie I/O jest powolne, nie angażuje specjalnie procesora przy przesyłaniu danych bajt po bajcie.
- Dla szybszych urządzeń wygodniej przesyłać cały blok danych (sieć, dyski). Umożliwia to mechanizm Direct Memory Access (DMA) realizowany sprzętowo.

- DMA równocześnie "kradnie" cykle pamięci (procesy wolniej uzyskują dostęp do pamięci!)

Tryby pracy procesora

- Tryb użytkownika (z ograniczeniami),
- Tryb monitora / nadzorcy / systemu – wykonuje potencjalne niebezpieczne dla spójności pamięci operacje. Są to tzw. **operacje uprzywilejowane**.

  O ile w trybie użytkownika istnieją pewne, często powiązane z programami, bloki pamięci, w trybie monitora mamy kontrolę nad cała pamięcią, niezależnie od tego co w niej działa.

Tryby pracy procesora

- **Operacje wejścia/wyjścia** nie są bezpośrednio dostępne dla użytkownika (musi o nie prosić system operacyjny).
- Użytkownik ma dostęp do pamięci przydzielonej **tylko swojemu programowi.**
- System nie może utracić kontroli nad procesorem np. przez nieskończoną pętlę w programie użytkownika.

Praca procesorów Intel

- Tryb **rzeczywisty** – Zgodność z podstawowym, 16-bitowym x86 z adresowaniem pamięci włącznie. Używany na wczesnych etapach uruchamiania komputera (UEFI go opuszcza, BIOS rzadko, Win9x – wraz z inicjalizacją VMM, WinNT i Unix – jądra).
- Tryb **chroniony** – działa ochrona pamięci, pamięć wirtualna, kontekst i działania na MMU. Większość systemów operacyjnych działa w tym trybie.
- Tryb **wirtualny** – możliwość uruchomienia wirtualnego CPU w trybie rzeczywistym. Używany w niektórych sterownikach w systemach 32-bit, w systemach 64bit w zasadzie niedostępny.

Proces

- Proces jest programem, który jest aktualnie wykonywany.
- Jest to jednostka pracy w systemie.
- System składa się ze **zbioru procesów**, których część to procesy systemu, pozostałe są procesami użytkowymi.

Zarządzanie procesami przez OS

- Tworzenie i usuwanie procesów użytkowych i systemowych,
- Wstrzymywanie i wznawianie procesów,
- Dostarczanie mechanizmów synchronizacji procesów,
- Dostarczanie mechanizmów komunikacji między procesami.
- Dostarczanie mechanizmów obsługi zakleszczeń.

Zarządzanie pamięcią przez OS

- Ewidencja aktualnie zajętych obszarów pamięci, informacja o użytkownikach danych obszarów,
- Decydowanie o załadowaniu procesów do zwolnionych miejsc w pamięci,
- Przydzielanie i zwalnianie pamięci stosownie do potrzeb,

Zarządzeni plikami przez OS

- Tworzenie i usuwanie plików,
- Tworzenie i usuwanie katalogów,
- Dostarczanie informacji do manipulowania plikami i katalogami, obsługa metadanych systemu plików,
- Odwzorowanie plików na obszary pamięci pomocniczej,
- Składowanie plików na nośnikach w pamięci nieulotnej.

Inne funkcje systemu operacyjnego

- Zarządzanie systemem wejścia/wyjścia (buforowanie, pamięć, spooling, programowanie interfejsów, moduły sterujące),
- Zarządzanie pomocniczą pamięcią dyskową,
- Praca sieciowa,
- System ochrony,
- Uruchomienie systemu interpretacji poleceń (powłoka).

Usługi systemu operacyjnego

- Wykonanie programu,
- Operacje wejścia-wyjścia,
- Manipulowanie systemem plików,
- Komunikacja między procesami,
- Wykrywanie błędów.
- Przydzielanie zasobów,
- Rozliczanie
- Ochrona

Funkcje (wywołania) systemowe

- Tworzą **interfejs pomiędzy wykonywanym programem a systemem operacyjnym.**
- Poprzez funkcje systemowe program użytkownika "daje zlecenia" systemowi operacyjnemu.

Funkcje systemowe – nadzorowanie procesów

- Załadowanie lub wykonanie programu,
- Zakończenie lub zaniechanie procesu,
- Utworzenie lub zakończenie procesów potomnych,
- Pobieranie lub ustawianie parametrów procesów,
- Oczekiwanie "czasowe",
- Oczekiwanie na zdarzenie lub sygnalizacja jego wystąpienia,
- Przydział i zwolnienie pamięci.

Funkcje systemowe – operacje na plikach

- Utworzenie lub usuwanie plików,
- Otwarcie lub zamknięcie plików,
- Zapis, Odczyt lub zmiana położenia w pliku,
- Pobranie lub ustawianie atrybutów i metadanych pliku.

Funkcje systemowe – operacje na urządzeniach

- Zarezerwowanie lub zwolnienie urządzenia,
- Zapis, odczyt lub zmiana położenia (o ile to możliwe),
- Pobieranie i ustawianie atrybutów i ustawień urządzenia,
- Logiczne podłączanie lub odłączanie urządzeń.

Funkcje systemowe – utrzymywanie informacji

- Pobieranie lub ustawianie daty/czasu,
- Pobieranie lub ustawianie danych systemowych, konfiguracji, ustawień NVR (o ile istnieją),
- Pobieranie atrybutów procesów, plików lub urządzeń,
- Ustawianie atrybutów procesów, plików lub urządzeń,

Funkcje systemowe - komunikacja

- Tworzenie lub usuwanie połączeń komunikacyjnych,
- Nadawanie i odbieranie komunikatów,
- Przekazywanie informacji o stanie systemu i komponentów,
- Przyłączanie i odłączanie urządzeń zdalnych.

Przykładowe funkcje systemowe

- Tworzenie i uruchamianie programów:
  - fork(2)
  - execve(2)
  - exec(5)
- Operacje na plikach:
  - creat(2)
  - open(2)
  - read(2)
  - write(2)
  - close(2)
  - chown(2)
  - chmod(2)
- Operacje na katalogach:
  - opendir(3)
  - closedir(3)
  - scandir(3)
- Środowisko:
  - getenv(3)
  - setenv(3)
  - putenv(3)

Programy systemowe

Wchodzą w skład systemu korzystając z funkcji systemowych umożliwiają wypełnianie zadań.

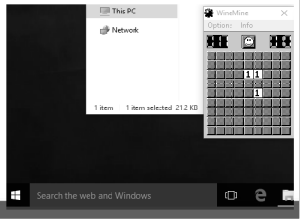
- Manipulowanie plikami,
- Informowanie o stanie systemu,
- Tworzenie i modyfikacja zawartości plików,
- Ładowanie i wykonywanie programów,
- Komunikacja.

API, ABI, zgodność oprogramowania

- Funkcje oferowane przez system dla programów użytkownika tworzą **API** - Application Programming interface.
  - W Linuksie, oferowane funkcje systemowe i GLIBC są standardowe, stosunkowo niewielkie i realizują podstawowe operacje. Zgodność na poziomie API zapewniona jest przez dłuższy czas.
  - W Windows, funkcji API przybywa z wersji na wersję, lecz zachowywane są starsze funkcje celem zgodności wstecznej. Dzięki temu można kompilować wiele programów napisanych na starsze systemy.

ABI

- Gdy program zostaje skompilowany, komunikuje się z systemem za pomocą interfejsu binarnego (**ABI** - Application Binary Interface), - odpowiadają API struktury pamięci i wywoływane bloki kodu.
  - W Linuksie, ABI jest stabilne, lecz nie jest to ścisłe wymaganie. Wiele programów więc w kolejnych wersjach trzeba przekompilować. Nie pomagają również stale ewoluujące biblioteki. Ze względu na to, że większość oprogramowania jest open source, nie stanowi to krytycznego problemu w typowych sytuacjach.
  - W Windows, ABI jest szczególnie stabilne. Dzięki temu programy własnościowe, których kod jest tajny, można uruchamiać a kompatybilność wsteczna jest zachowana.



Źródło: N. Lineback

ABI we współpracy

- Program współpracujący z biblioteką musi wiedzieć jak wywołać z niej funkcję. Jak przesłać do niej zmienną konkretnego typu. Jak odebrać z niej wynik. To również ABI.
- Kompilatory **nie mogą** naruszać ABI z wersji na wersję. Spowoduje to niekompatybilność binariów w zależności od wersji kompilatora, który je zbudował i konieczność ich przekompilowania.
- Niestety czasami się to dzieje (time_t i problem roku 2038!). Niektóre systemy, w celu zapewnienia zgodności z przestarzałymi programami, których kody źródłowe zaginęły, są wydawane w wersjach intencjonalnie zbudowanych w starych kompilatorach (Haiku OS).

Jądro Linuksa

- Stałe API na zewnątrz (funkcje systemowe),
- W większości przypadków stałe ABI na zewnątrz.
- Ale wewnątrz samego jądra ani API, ani ABI nie jest zachowywane. Istnieje stabilne API dla sterowników, ale ABI już zmienia się w kolejnych wersjach.

  Co z tym zmiennym ABI mają począć autorzy sterowników?

  DKMS - Dynamic Kernel Module Support
- Gdy instalowane jest nowe jądro systemu, z nim instalowane są jego nagłówki (pliki .h). W systemie jest już kompilator. Dodatkowe sterowniki (np. grafika nVidia) występują w postaci kodu źródłowego i są automatycznie dokompilowane do nowego jądra jako moduły.

Mac OS

- Toolbox ROM - stopień pośredni między funkcją firmware a funkcją systemową. Wyzwalanie przerwania "A-trap" przy intencjonalnie niewłaściwym rozkazie procesora powoduje wywołanie funkcji z pamięci stałej sprzętu. W ten sposób zaimplementowane są nawet proste operacje graficzne.
- Gdy procesory stały się odpowiednio szybkie, a duża pamięć stała zaczęła generować koszty, symulowano zachowanie Toolbox za pomocą funkcji systemu, a później porzucono to całkowicie.
- Mac OS X ma już zestaw wywołań systemowych w dużej mierze zgodnych z BSD.

Proces

- **Proces** jest wykonywanym programem.
- Wykonanie procesu musi przebiegać w sposób sekwencyjny (w dowolnej chwili na zamówienie procesu może zostać wykonany co najwyżej jeden rozkaz programu).

Elementy procesu

- Kod programu (sekcja tekstu),
- Bieżąca czynność (reprezentowana przez licznik rozkazów),
- Aktualna zawartość rejestrów procesora,
- Stos procesu,
- Sekcja danych.

Stan procesu

- Nowy – proces został utworzony.
- Aktywny – są aktualnie wykonywane instrukcje.
- Oczekujący – czeka na wystąpienie zdarzenia, np. zakończenie operacji I/O.
- Gotowy – Oczekuje jedynie na przydział procesora.
- Zakończony – zakończył działanie.

Blok kontrolny procesu

Blok kontrolny procesu

- Stan procesu (nowy, gotowy, aktywny, itp.).
- Licznik rozkazów – adres **następnego** rozkazu do wykonania.
- Rejestry procesora.
- Informacje do planowania przydziału procesora (priorytet, wskaźniki do kolejek).

Blok kontrolny procesu (c.d.)

- Informacje zarządzania pamięcią (granice pamięci, tablice stron, tablice segmentów),
- Informacje do rozliczeń (użyty czas procesora, czas całkowity, konta użytkowników i uprawnienia),
- Informacje o stanie I/O (urządzenia przydzielone do procesu, wykaz otwartych plików itd.).

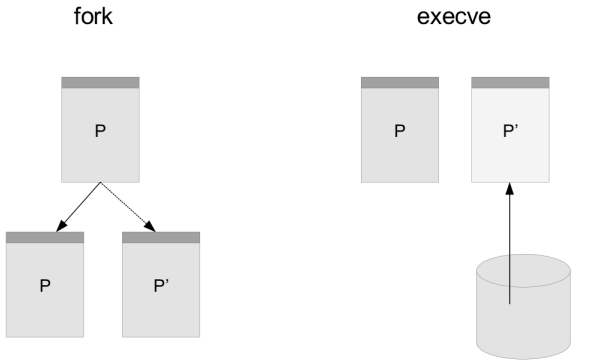
Tworzenie procesu

- Proces macierzysty tworzy potomne **za pomocą funkcji systemowej**.
- Nowy proces też może tworzyć potomne - powstaje wtedy drzewo procesów.
- Proces macierzysty i potomek mogą dzielić w całości, w części, lub wcale nie dzielić ze sobą zasobów.
- Proces macierzysty i potomek działają równolegle, lub też macierzysty czeka, aż potomek zakończy działanie.
- Proces potomny może być kopią procesu macierzystego lub otrzymać zupełnie nowy program.

Tworzenie procesu w systemie UNIX

- Nowy proces tworzy się funkcją systemową **fork**.
- Potomek zawiera kopię przestrzeni adresowej przodka – daje to możliwość komunikacji pomiędzy procesami.
- Funkcja systemowa **execve** ładuje nowy program do przestrzeni adresowej procesu (niszcząc poprzednią zawartość) i rozpoczyna jego wykonywanie.
- Proces macierzysty albo tworzy nowych potomków, albo czeka na zakończenie procesu potomnego.

Tworzenie w systemie UNIX



Tworzenie procesów w systemie Windows:

- Występują obydwa mechanizmy: Możliwa jest kopia przestrzeni adresowej przodka, albo ładowany jest nowy program wykonywalny ("obraz").
- Funkcja systemowa **CreateProcess.**
- Możliwa jest symulacja zachowania fork-a.

Kończenie procesu

- Po zakończeniu ostatniej instrukcji proces prosi system o usunięcie (funkcja systemowa **exit**).
- System:
  - Przekazuje wynik działania potomka do procesu macierzystego (wykonującego funkcję systemową **wait**).
  - Zamyka procesowi potomnemu zasoby (pamięć, otwarte pliki, buforowanie).

Kończenie procesu

- Proces macierzysty może "awaryjnie" zakończyć proces potomny, np. gdy:
  - Proces potomny nadużył przydzielonych zasobów,
  - Zadania wykonywane przez proces potomny stało się zbędne,
  - Proces macierzysty kończy się, a system nie pozwala na działanie "sieroty".

Wątki

- Wątek jest podstawową jednostką wykorzystania procesora. Jest to część składowa procesu wielowątkowego.
- Wątek składa się z:
  - licznika rozkazów,
  - zbioru rejestrów,
  - obszaru stosu
- Obszary wspólne dla kilku równorzędnych wątków:
  - sekcja kodu,
  - sekcja danych,
  - zasoby systemu (otwarte pliki, sygnały)

Wątki

- Zalety:
  - Przełączanie między wątkami i tworzenie nowych nie wymaga dużej aktywności procesora,
  - Przy przełączaniu nie trzeba wykonywać prac związanych z zarządzaniem pamięcią.
- Wątki poziomu użytkownika – przełączanie ich nie wymaga wywołania systemu operacyjnego, stąd jest szybsze. Jednak wywołanie funkcji systemowej jest **blokujące dla wszystkich wątków danego zadania.**

Wątki

- Działanie wątków przypomina działanie procesów. Mogą być w stanach: gotowości, zablokowania, aktywności, kończenia.
- Wątek może tworzyć wątki potomne, może się zablokować do czasu wykonania wywołania systemowego.
- Jeśli jeden wątek jest zablokowany, może działać inny wątek.
- Wątki jednego zadania są do siebie zależne - mogą np. nadpisywać stosy innych wątków.
- Ale z drugiej strony - producent i konsument mogą być wątkami jednego zadania, a wspólny obszar danych znacznie zwiększy wydajność procesu.

Obsługa wątków

- Obsługiwane przez jądro (Mach, Windows, OS/2)
- Wykonywane na poziomie użytkownika,
- Mieszane (np. Solaris 2.x).

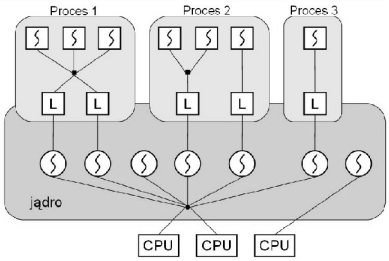
- Wątki poziomu użytkownika są najszybsze w przełączaniu, ale jądro systemu nie uwzględnia ich na poziomie przydziału procesora. Tak więc proces o jednym wątku i proces o 1000 wątków dostają taki sam kwant czasu.
- W systemie **Solaris** istnieje również **pośredni poziom** wątków, zwanych **procesami lekkimi** (lightweight processes, LWP).
- Każde zadanie ma przynajmniej jeden proces lekki do którego podłączone są wątki poziomu użytkownika.

Wątki a procesy lekkie

- Wątki są kontrolowane przez proces, nie tylko przez system operacyjny.
- Procesy lekkie działają w kontekście swojego procesu, ale są niezależnie planowane przez system.

S - Wątek

L - Proces lekki



Wątki a procesy lekkie

- Wątki poziomu jądra:
  - Posiadają małą strukturę danych i stos,
  - Podlegają planowaniu,
  - Przełączanie nie wymaga informacji o pamięci,
  - Przełączanie jest stosunkowo szybkie.
- Procesy lekkie:
  - Posiadają blok kontrolny procesu,
  - Potrzebne są informacje o pamięci,
  - Przełączanie kontekstu jest wolniejsze.
- Wątki użytkownika:
  - Posiadają stos i licznik rozkazów,
  - Szybkie przełączanie, jądro systemu nie jest angażowane.

Następny wykład

- Komunikacja między procesami
- Przydział procesora, metody przydziału.
- Synchronizacja procesów
- Mechanizmy synchronizacji: Semafory, liczniki, zamki.

Dziękuję za uwagę