

Systemy operacyjne Wykład 03

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

1

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).

2

Elementy procesu

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

3

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.

4

Blok kontrolny procesu

| |
|------------------------|
| Numer procesu |
| Stan procesu |
| Licznik rozkazów |
| Rejestry |
| Adresy pamięci |
| Wykaz otwartych plików |
| ... |

5

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).

6

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.).

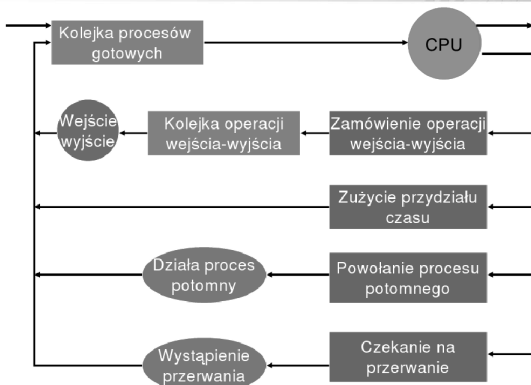
7

Kolejki planowania procesów

- Kolejka zadań (job queue) - tworzą ją procesy wchodzące do systemu.
- Kolejka procesów gotowych (ready queue) - procesy gotowe do działania, umieszczone w pamięci,
- Kolejki do urządzeń (device queue) - procesy czekające na konkretne urządzenie.

8

Diagram kolejek



9

Planiści

- Planista długoterminowy (planista zadań) - wybiera procesy **do kolejki procesów gotowych**, do pamięci.
 - Jest on wywoływany stosunkowo rzadko (sekundy) i nie musi być szybki.
- Planista krótkoterminowy (planista przydziału procesora) - wybiera proces z **puli procesów gotowych i przydziela mu procesor**.
 - Jest on wywoływany b. często (co ms) i musi być b. szybki.

10

Planiści

- Procesy możemy podzielić na:
 - Ograniczone przez wejście-wyjście (więcej czasu zajmują operacje we-wy niż korzystanie z procesora),
 - Ograniczone przez procesor (potrzebują znacznie więcej czasu procesora niż dla operacji we-wy).
- Zadaniem planisty długoterminowego jest dobór optymalnej mieszanki zadań ograniczonych przez procesor i przez I/O.

11

Planista średnioterminowy

- Występuje w niektórych systemach z podziałem czasu. Jego zadaniem jest, w koniecznych przypadkach, zmniejszanie stopnia wieloprogramowości poprzez wysyłanie części zadań chwilowo na dysk (swapping). Pomaga to w doborze lepszego zestawu procesów w danej chwili, lub dla zwolnienia obszaru pamięci.

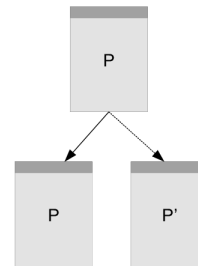
12

- Podczas przejścia procesora z wykonywania jednego procesu do drugiego należy przechować stan starego procesu i załadować przechowany stan nowego.
- Z punktu widzenia systemu są to działania nieproduktywne, tak jak przygotowanie czy sprząatanie stanowiska pracy, ale są niezbędne przy wieloprogramowości.
- Mechanizm wątków pozwala na redukcję czasu przełączania kontekstu.

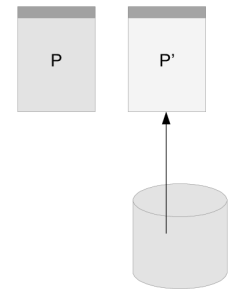
- Proces macierzysty tworzy potomne **za pomocą funkcji systemowej**.
- Nowy proces też może tworzyć potomne - powstaje wtedy drzewo procesów.
- Proces macierzysty i potomek mogą dzielić w całości, w części, lub wcale nie dzielić ze sobą zasobów.
- Proces macierzysty i potomek działają równolegle, lub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

- 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.

fork



execve



- 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.
- Wątki tworzone są funkcją **CreateThread**.
- Każda nowa funkcja tworząca procesy/wątki pochodzi od tych podstawowych (zgodność API!)

- 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).

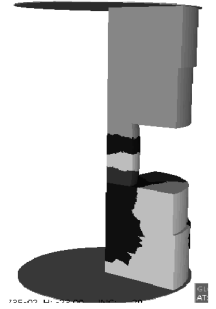
Kończenie procesu

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

19

Procesy współpracujące

- Procesy są współpracujące, jeżeli jeden proces może wpływać na inne procesy, a inne procesy mogą wpływać na niego.



Symulacja numeryczna na 4 procesory. Jeden kolor - jeden proces.

20

Procesy współpracujące

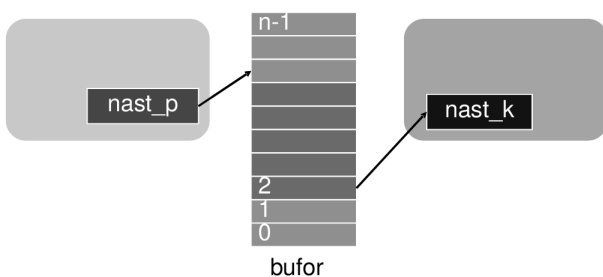
- Zalety:
 - Dzielenie informacji – kilka procesów może korzystać z np. jednego pliku.
 - Przyspieszenie obliczeń w systemach wieloprotocowych, gdy można podzielić zadanie na wykonywane równolegle mniejsze.
 - Modularność rozwiązań,
 - Wygoda – jeden użytkownik może wykonywać w tym samym czasie kilka zadań, np. edycję, kompilację, podgląd, drukowanie.

21

Komunikacja międzyprocesowa: Pamięć współdzielona

22

Współpraca – problem producenta i konsumenta



23

Bufor ograniczony – wspólne zmienne

```
var n;  
type jednostka = ...;  
jednostka bufor[n-1];  
jednostka* we,wy;
```

Następne wolne miejsce

Pierwsze zajęte miejsce

24


```
while true {
  ...
  produkuj jednostka do nast_p;
  ...

  while ((we+1)%n==wy) {} //nie produkuj
  bufor[we]=nast_p;
  we=(we+1)%n;
}
```

25

```
while true {
  while (we==wy) {} //pusto
  nast_k = bufor[wy];
  wy=(wy+1)%n;
  ...
  przetwarzaj (nast_k);
  ...
  • }
```

26

Komunikacja międzyprocesowa: System komunikatów

27

Komunikacja międzyprocesowa

- Dwie operacje:
 - Nadaj komunikat,
 - Odbierz komunikat.
- W celu ich realizacji procesy:
 - Ustalają łącze komunikacyjne
 - Nadają i odbierają (również kodują i dekodują) komunikaty.
- Komunikacja odbywa się za pomocą pamięci dzielonej lub szyny systemowej.

28

Komunikacja międzyprocesowa – podstawowe kwestie:

- Jak ustanawia się połączenie?
- Czy jedno łącze może obsłużyć >2 procesy?
- Ile łączy pomiędzy parą procesów?
- Jaka przepustowość łącza? Obszar buforowy?
- Komunikaty: stałej czy zmiennej długości?
- Łącze jedno czy dwukierunkowe?
- Komunikacja: Bezpośrednia czy pośrednia?

29

Komunikacja bezpośrednia

- Dwie operacje elementarne:
 - nadaj(PID1, komunikat),
 - odbierz(PID2, komunikat).

(PID1, PID2 – identyfikatory procesów)

30

- Właściwości łącza:
 - Ustanawiane automatycznie pomiędzy parą procesów, wystarczy, aby procesy znalazły swoje identyfikatory.
 - Dokładnie dla 2 procesów. Między parą dokładnie jedno łącze.
 - Zazwyczaj dwukierunkowe, dopuszczalne jednokierunkowe.

produkuj jednostka do nast_p;
nadaj(konsument, nast_p);

odbierz(producent,nast_k);
 przetwarzaj(nast_k);

- Komunikaty nadawane i odbierane są za pomocą portów ("skrzynek pocztowych"). Procesy mogą się komunikować jedynie mając wspólny port.
- Łącze jest ustanawiane gdy procesy dzielą port.
- Łącze może być związane z więcej niż dwoma procesami.
- Każda para procesów może mieć kilka łączy, przez różne porty.
- Łącze może być jedno i dwukierunkowe.

- Trzy procesy P1, P2, P3 dzielą wspólny port. Proces P1 wysyła komunikat, procesy P2 i P3 próbują go odebrać – powstaje konflikt.
- Rozwiązania:
 - Zezwać jedynie na łącza pomiędzy dwoma procesami,
 - Zezwać co najwyżej jednemu procesowi na wykonanie odbioru w danym momencie,
 - Dopuszczać, by system wybrał proces docelowy. System powinien poinformować nadawcę o wyborze.

- Kolejka komunikatów:
 - pojemność zerowa - łącze nie dopuszcza aby czekał w nim jakikolwiek komunikat - nadawca czeka aż odbiorca odbierze,
 - pojemność ograniczona - w kolejce może pozostawać tyle komunikatów, na ile zaprojektowano kolejkę. W przypadku kolejki pełnej nadawca musi czekać.
 - Pojemność nieograniczona - kolejka ma potencjalnie nieskończoną długość. Nadawca nigdy nie czeka.

- Zakończenie procesu - system musi rozwiązać problemy:
 - Gdy proces czeka na komunikaty z zakończonego,
 - Gdy nadaje komunikaty do zakończonego,
- Utrata komunikatów
 - System wykrywa to i ponownie nadaje komunikat,
 - Proces nadawczy wykrywa i ew. powtarza komunikat,
 - System wykrywa i powiadamia proces nadawczy.
- Zniekształcenie komunikatów - kontrola poprawności przez sumy kontrolne, sprawdzanie parzystości itd.



Komunikacja: Jądro Mach

- Większość wywołań i przepływ informacji odbywa się za pomocą komunikatów
- Przy tworzeniu zadania powstają dwa porty („skrzynki pocztowe”):
 - Jądra - komunikacja jądra z zadaniem,
 - Zawiadomień - wysyłanie informacji o zdarzeniach.
- Funkcje systemowe:
 - msg_send - wysyłanie komunikatu
 - msg_receive - odbieranie komunikatu
 - msg_rpc - wysyłanie blokujące do momentu odpowiedzi,
 - port_allocate - tworzenie nowego portu,

37



Gdy proces nie odpowiada...

- Skrzynka jest pełna. Proces nadający może:
 - Czekać na zwolnienie miejsca w skrzynce,
 - Czekać z limitem czasu,
 - Zignorować fakt i nie czekać,
 - Przekazać komunikat do systemu do późniejszego przesłania (może tak czekać tylko jeden komunikat).
- Podobny interfejs zastosowano w jądrze GNU/Hurd.

38



Implementacja

- Solraix: Mechanizmy pamięci dzielonej, regionów, proste potoki.
 - Linux: Pamięć dzielona, potoki (pipe), z możliwością przekierowania do/z plików i urządzeń
 - Dla użytkownika i jego skryptów: /dev/shm – pamięć dzielona. RAM-dysk uniwersalny.
- + Mechanizmy synchronizacji

39



Komunikaty w Windows linii NT

- LPC - Local Procedure Call - zmodyfikowany mechanizm RPC do przesyłania komunikatów. Są dwa typy portów:
 - Łączące
 - Komunikacyjne.
- Komunikacja:
 - Klient uzyskuje uchwyt do obiektu portu,
 - Klient wysyła prośbę o połączenie (funkcja systemowa),
 - System/serwer tworzy dwa prywatne porty i przekazuje klientowi uchwyt do jednego z nich,
 - Klient i serwer wykorzystują uchwyty do przesyłu komunikatów.

40



Komunikacja w Windows linii NT (c.d.)

- Trzy sposoby przekazywania komunikatów:
 - Dla <256B - kolejka komunikatów jako pamięć tymczasowa, kopiowanie komunikatów z jednego procesu do drugiego.
 - Dla większych - pamięć dzielona (obiekt sekcji), by uniknąć kopiowania.
 - Dla np. funkcji graficznych - szybkie wywoływanie procedur lokalnych. Obiekt sekcji do przekazywania komunikatów, obiekt pary zdarzeń do synchronizacji.

41



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)

42

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

43

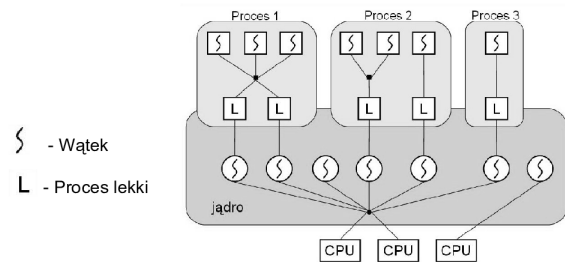
- Działanie wątków przypomina działanie procesów. Mogą być w stanach: gotowości, zablokowania, aktywności, kończenia.
- Wątek może tworzyć wątki potomne, może się zablokować do czasu wykonania wywołania systemowego.
- Jeśli jeden wątek jest zablokowany, może działać inny wątek.
- Wątki jednego zadania są od 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.

44

- Obsługiwane przez jądro (Mach, Windows, OS/2, Linux)
- Wykonywane na poziomie użytkownika,
- Mieszane (np. Solaris 2.x).
- Wątki poziomu użytkownika są najszybsze w przelączeniu, 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.

45

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



46

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

47

Dziękuję za uwagę

48