Systemy operacyjne

Zarządzanie procesami

Dr inż. Dariusz Caban
mailto:dariusz.caban@pwr.edu.pl
tel.: (071)320-2823

Wrocław University of Technology

1

Procesy i zasoby

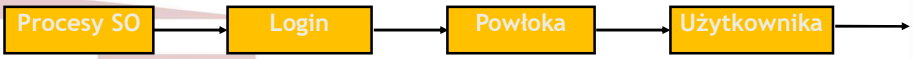
- Proces a program
  - Proces jest to wykonywanie się programu (dynamiczne)
  - Niejednoznaczność nazwy programu w systemie wieloprogramowym
  - Programy wykonywane przez wiele procesów
  - Procesy systemu operacyjnego
- Zasoby systemu
  - Zasoby fizyczne (pamięć, urządzenia zewnętrzne)
  - Zasoby logiczne (dane, pliki, bufory, usługi SO)
- Zasoby przydzielane są procesom
  - Przydział dzielony i wykluczający

Wrocław University of Technology

2

Tworzenie procesów, hierarchia

- Nowy proces tworzony jest funkcją systemową
  - W Unix'ie tylko jedna funkcja: *fork*
  - Proces rodzic
- Hierarchia procesów
  - Budowana relacją proces rodzic – proces potomny
  - Problem: co z procesem potomnym, gdy rodzic się kończy?



Wrocław University of Technology

3

Procesy i wątki

- Procesy i wątki są wykonywane sekwencyjnie
  - Procesy mają własne zasoby
    - Zbędne spowolnienie, gdy wykonują wspólnie program
    - Trudna komunikacja międzyprocesowa
  - Wątki mają wspólne zasoby w ramach procesu
    - Procesy wielowątkowe (nie są sekwencyjne)
    - Wspólna przestrzeń adresowa
- Biblioteki wielowątkowe (wątki Posix – pthreads)
- Systemy operacyjne wielowątkowe
  - Mach, OS2, NT

Wrocław University of Technology

4

Kontekst procesu

- Wszystkie informacje konieczne do uruchomienia procesu od punktu przerwania
- Konteksty
  - rejestrowy
    - licznik rozkazów (instruction pointer)
    - wskaźnik stosu
    - rejestry ogólne procesora (oraz pewnych sterowników)
  - użytkownika (pamięć dostępna dla procesu)
  - systemowy (pozycja tablicy procesów, u-area, stos systemowy)

Wrocław University of Technology

5

Przełączanie kontekstu

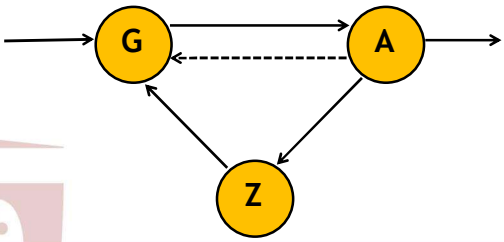
- Składowanie kontekstu
  - Rejestrowego w przestrzeni pamięci procesu lub SO
  - Pamięć rozdzielona lub składowana na dysku (swapping)
  - Kontekst systemowy podzielony między
    - tablicę procesów
    - obszar pamięci procesu, niedostępny dla użytkownika (u-area)
- Odtworzenie kontekstu
  - Dualizm licznika rozkazów (rejestr procesora, wskaźnik odkąd kontynuować proces)

Wrocław University of Technology

6

Diagram stanów procesu

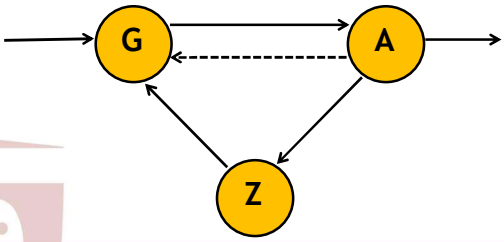
- Stany
  - Proces aktywny (A)
  - Proces gotowy (G)
  - Proces uśpiony (Z)
- Wywłaszczanie procesora



7

Planowanie (scheduling)

- Wybór procesu do obsługi
  - planowanie krótkoterminowe - uaktywnienie procesu gotowego
  - planowanie długoterminowe - przyjęcie procesu do wykonywania
  - planowanie średnioterminowe - wybór procesu z pamięci pomocniczej (swapped out) do pamięci głównej



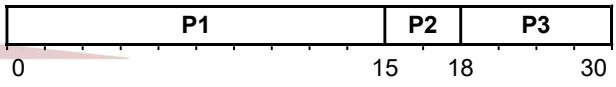
8

Kryteria planowania

- Wykorzystanie procesora
  - procentowy udział czasu zajętości procesora
- Przepustowość
  - liczba procesów wykonywanych w jednostce czasu
- Czas cyklu przetwarzania
  - czas od nadejścia procesu do jego zakończenia
- Czas oczekiwania
  - czas przebywania procesu w stanie gotowości
- Czas odpowiedzi (inaczej czas reakcji)
  - czas od operacji wejścia do widocznej odpowiedzi (wyjścia)
- Efektywność

Algorytmy planowania bez wywłaszczania

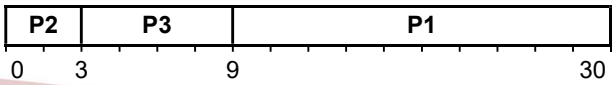
- Algorytm FCFS (First Come First Served)
  - Wybierany proces, który najdłużej czeka gotowy
  - Zaleta: prostota
    Wada: stosunkowo długi czas oczekiwania
- Przykład
  P1 - 15, P2 - 3, P3 – 6



$T_{śr} = (0 + 15 + 18)/3 = 11$

Algorytm najkrótszego zadania

– Zawsze wybierany proces o najkrótszym czasie do zablokowania
– Zaleta: optymalny, tzn. najkrótszy czas oczekiwania
  Wada: trudno przewidzieć czas przetwarzania



$T_{śr} = (0 + 3 + 9)/3 = 4$

Algorytm priorytetowy

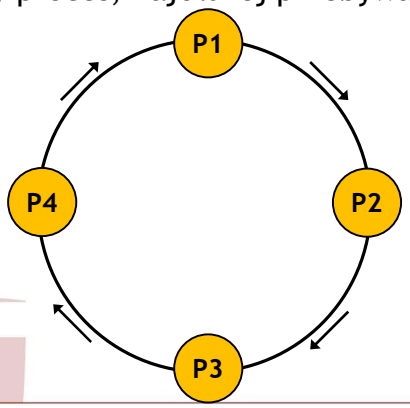
- Wybierany proces o najwyższym priorytecie
  – Priorytet zewnętrzny
    - Zadany przez użytkownika
  – Priorytet wewnętrzny
    - Ustalany przez SO na podstawie parametrów procesu
  – Priorytet dynamiczny
    - Modyfikowany w trakcie wykonywania
- Zaleta: elastyczność
  Wada: niebezpieczeństwo nieskończonego czekania

Algorytmy planowania z wywłaszczaniem

- Proces aktywowany na krótki kwant czasu, po którym jest wywłaszczany, tzn. przesuwany do stanu gotowych
- Wymaga sprzętowych mechanizmów „czasomierza”
  - przerwanie zegarowe
- Zalety:
  - ograniczenie czasu reakcji
  - uprzywilejowanie procesów interaktywnych
  - algorytmy zapewniają w miarę krótki czas oczekiwania
- Wada:
  - straty na przełączanie procesów
  - problem ustalenia kwantu czasu
    - decyduje o czasie reakcji
    - korzystne, gdy procesy blokują się, a nie są wywłaszczane

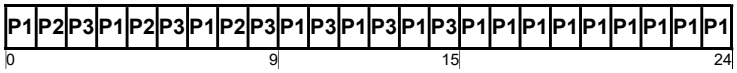
Algorytm karuzelowy

- Wybierany proces, najdłużej przebywający w stanie gotowości



Algorytm karuzelowy - przykład

P1 - 15, P2 - 3, P3 – 6; kwant czasu – 1



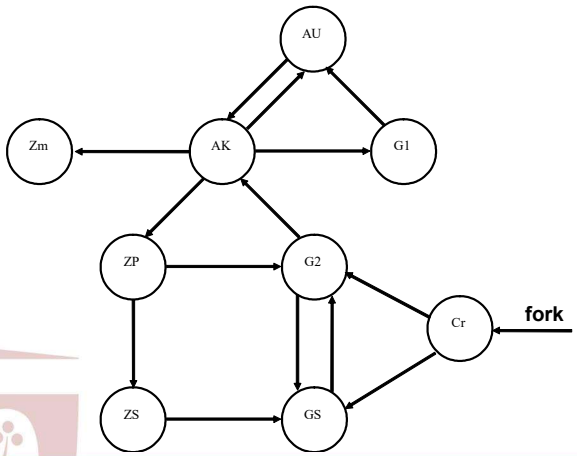
$T_{śr\ reakcji} < 2$

$T_{śr\ ocz} = [(0 + 3*2 + 3*1) + (1 + 2*2) + (3*2 + 3*1)]/3$
$= (9 + 5 + 9)/3$
$= 7,66$

Wrocław University of Technology

15

Diagram stanów w systemie Unix



Wrocław University of Technology

16

Przełączanie procesów w Unixie

- Algorytm wielokolejkowy ze sprzężeniami
- Algorytm z wywłaszczaniem procesów użytkownika
- Priorytet procesu
  - Procesy zablokowane: stały, zależny od przyczyny blokady
  - Prirytet obniżany po odblokowaniu
  - W przerwaniu zegarowym:
    - dla procesu aktywnego CPU = CPU +1
    - dla procesów gotowych CPU = decay (CPU)
    - co 1 sek. modyfikowane priorytety i wymuszone planowanie

    priorytet = <wartość progowa> + CPU/2 + <nice>

Wrocław University of Technology

17

Tworzenie nowego procesu

int fork(void)

- Tworzy proces potomny o identycznym obrazie pamięci (czyli robiący dokładnie to samo)
- Powstaje hierarchia procesów (relacja rodzic-potomek)
- Funkcja fork zwraca inną wartość w procesie potomnym (zawsze 0)
- W procesie rodzicu identyfikator (pid) potomka
- Wartość ujemną, gdy błąd wykonania funkcji

Wrocław University of Technology

18

Przykład tworzenia nowego procesu

```
int pid;
........
   if ((pid=fork( )) < 0)
        { sygnalizacja błędu; }
   else if (pid = = 0)
        { instrukcje do wykonania w procesie potomnym; }
   else
        { instrukcje do wykonania w procesie rodzicu; }
  .........
```


Koniec procesu

exit (int status);

- Kończy proces z zadanym statusem wykonania (exit code)
- System operacyjny nie narzuca interpretacji statusu wykonania
- Konwencja programowania wymaga aby 0 oznaczało poprawne zakończenie
- Dalsze instrukcje po exit nie będą wykonane
- W języku C: } i return w programie głównym niejawnie wywołują funkcję exit(0)

Czekanie na zakończenie potomka

int wait (*int status);

- Proces rodzic zawiesza się do czasu zakończenie procesu potomka
  - Dowolnego
  - Proces potomny może zakończyć się przed wykonaniem wait
- Zakończony proces potomny przebywa w stanie „zombie” do czasu, aż wykluczone jest wykonanie wait przez rodzica
- Funkcja zwraca identyfikator (pid) zakończonego procesu potomnego
- Pod adres w argumencie zapisuje status wykonania z funkcji exit potomka

Wrocław University of Technology

21

Wymiana kontekstu procesu

exec??(char *nazwa, char *argv[ ], char *envp[ ]);

- Powoduje załadowanie programu <nazwa> w miejsce obrazu pamięci procesu wywołującego
- Wykonuje się jako ten sam proces
- Niemożliwy jest powrót z funkcji exec i wykonanie instrukcji występujących po niej!
- Trzeba przygotować argumenty wywołania i środowisko wykonania programu i podać przy wywołaniu exec
- Funkcje w rodzinie różnią się postacią argumentów (execve)

Wrocław University of Technology

22

Wykonanie programu „z powrotem”

- W języku C funkcja system
- Często przydatne:
  - wywołanie programu z rodzica
  - poczekanie na jego wykonanie, a następnie
  - kontynuowanie wykonania rodzica
- Realizacja:

```
if ((pid = fork( ))<0)  { sygnalizacja błędu; }
else if (pid = = 0)
        execve(„program”, argumenty, otoczenie);
else
        while (wait(&status) != pid);
```

Wrocław University of Technology

23