Systemy Operacyjne
Procesy i wątki

dr inż. Tomasz Jordan Kruk

T.Kruk@ia.pw.edu.pl

Instytut Automatyki i Informatyki Stosowanej
Politechnika Warszawska

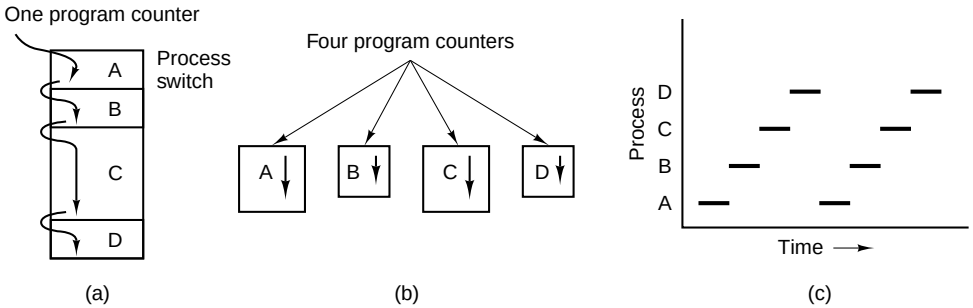
Proces w systemie operacyjnym

Procesem nazywamy wykonujący się program wraz z jego środowiskiem obliczeniowym. Proces stanowi podstawowy obiekt dynamiczny w systemie operacyjnym.

Wymagania odnośnie systemu operacyjnego odnośnie zarządzania procesami:

- √ umożliwienie przeplatania się wykonywania procesów,
- √ akceptowalnie krótki czas odpowiedzi systemu,
- √ zarządzanie przydziałem zasobów poszczególnym procesom,
- √ udostępnianie mechanizmów do komunikacji międzyprocesowej,
- √ udostępnianie mechanizmów do tworzenia procesów.

Wieloprogramowanie



- √ należy unikać programowania procesów uwarunkowanych czasowo,
- √ zależność między programem a procesem,
- √ pojęcia procesów **współbieżnych**, **równoległych** i **rozproszonych**,

Tworzenie i kończenie procesów

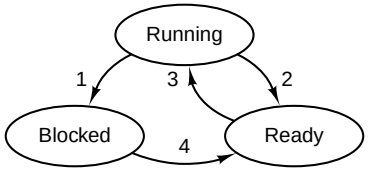
Utworzenie procesu może być rezultatem:

- √ inicjalizacji systemu,
- √ wywołania przez już uruchomiony proces funkcji systemowej do tworzenia procesu,
- √ zlecenia użytkownika utworzenia nowego procesu,
- √ uruchomienia zadania wsadowego.

Zakończenie działania procesu może być rezultatem:

- √ zakończenia algorytmu procesu,
- √ celowego zakończenia w wyniku wystąpienia błędu,
- √ wykonania niedozwolonej operacji (zakończenie wymuszone),
- √ otrzymania sygnału od innego procesu (zakończenie wymuszone).

Stany procesów



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Podstawowe stany procesów:

- √ **uruchomiony** - (ang. *running*), w danej chwili wykonuje się na procesorze,
- √ **gotowy** - (ang. *ready*), gotowy do wykonania, ale wstrzymany w oczekiwaniu na przydział czasu procesora,
- √ **wstrzymany** - (ang. *blocked*), nie może kontynuować pracy do momentu wystąpienia pewnego zewnętrznego zdarzenia.

Stany procesów w systemie Unix

- √ **uruchomiony w trybie użytkownika** - (ang. *user running*),
- √ **uruchomiony w trybie jądra** - (ang. *kernel running*),
- √ **gotowy, w pamięci** - (ang. *ready to run, in memory*),
- √ **wstrzymany, w pamięci** - (ang. *asleep in memory*),
- √ **gotowy wymieciony** - (ang. *ready to run, swapped*),
- √ **wstrzymany, wymieciony** - (ang. *sleeping, swapped*),
- √ **wywłaszczony** - (ang. *preempted*),
- √ **utworzony** - (ang. *created*),
- √ **zombie** - (ang. *zombie*).

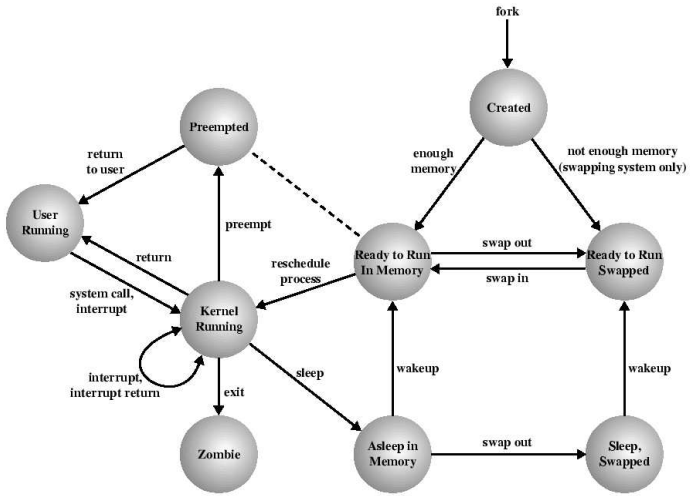
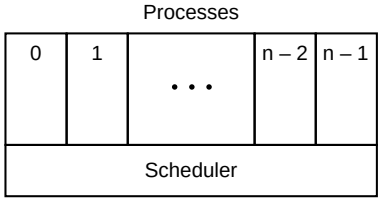


Figure 3.16 UNIX Process State Transition Diagram

Planista



Przydziałem procesora poszczególnym procesom zajmuje się część jądra systemu operacyjnego określana mianem **planisty** (ang. *scheduler*).

Implementacja procesu:

- √ system posiada strukturę danych określaną mianem **tablicy procesów**,
- √ element tablicy procesów nazywany jest **deskryptorem procesu** albo **blokiem kontrolnym procesu** (ang. *PCB, process control block*)

Typowe pola elementu tablicy procesów

Zarządzanie procesem	Zarządzanie pamięcią
Rejestry	Wskaźnik do segmentu kodu
Licznik rozkazów (PC)	Wskaźnik do segmentu danych
Słowo stanu procesora (PSW)	Wskaźnik do segmentu stosu
Wskaźnik stosu	
Stan procesu	
Priorytet	
Parametry szeregowania	**Zarządzanie plikami**
ID procesu	Katalog *root*
Proces rodzic	Katalog bieżący
Grupa procesu	Deskryptory plików
Sygnały	ID użytkownika
Czas startu procesu	ID grupy
Użyty czas CPU	
Czas następnego alarmu	

Obsługa przerwań

Obsługa przerwań przez system operacyjny:

- √ odbieranie procesora na rzecz planisty z wykorzystaniem wsparcia sprzętowego (np. przerwania zegarowego),
- √ z każdym urządzeniem I/O skojarzona jest lokalizacja w pamięci z informacją, skąd ma nastąpić kontynuacja wykonywania podprogramu w przypadku wystąpienia przerwania, tzw. **wektor przerwania** (ang. *interrupt vector*),
- √ wektor przerwania zawiera adres procedury dostarczonej przez system operacyjny (ang. *interrupt handler*),
- √ planista odgrywa rolę pośrednika, uzyskuje sterowanie po każdym wystąpieniu przerwania zegarowego,
- √ proces nie może przekazać sterowania innemu procesowi bez pośrednictwa planisty.

Obsługa przerwania - szkielet

1. Sprzętowo zapamiętane na stosie licznik rozkazów, itp.
2. Sprzętowo ładowana nowa wartość licznika rozkazów z wektora przerwań
3. Procedura w języku asemblera zapamiętuje wartości rejestrów
4. Procedura w języku asemblera ustala nowy stos
5. Obsługa przerwania w języku C (często odczyt i buforowanie wejścia)
6. Planista wybiera następny proces do wznowienia
7. Procedura w języku asemblera przygotowuje i wznawia nowy bieżący proces

Wątki wykonania

Gdy potrzeba współbieżnych wykonań sterowania i rozwiązanie jako grupa procesów, to przy rozłącznych chronionych przestrzeniach adresowych:

- √ z punktu widzenia ochrony: zaleta, ale: chronimy się przed samym sobą,
- √ z punktu widzenia komunikacji: wada,
- √ z punktu widzenia łatwości współużytkowania zasobów: wada,
- √ z punktu widzenia wydajności: wada, chyba że równolegle.

Więc może dwa sterowania spróbować zawrzeć tak jakby w jednym procesie, wtedy:

- √ z punktu widzenia ochrony: wada, ale jesteśmy autorami kodu,
- √ z punktu widzenia komunikacji: zaleta,
- √ z punktu widzenia łatwości współużytkowania zasobów: zaleta.

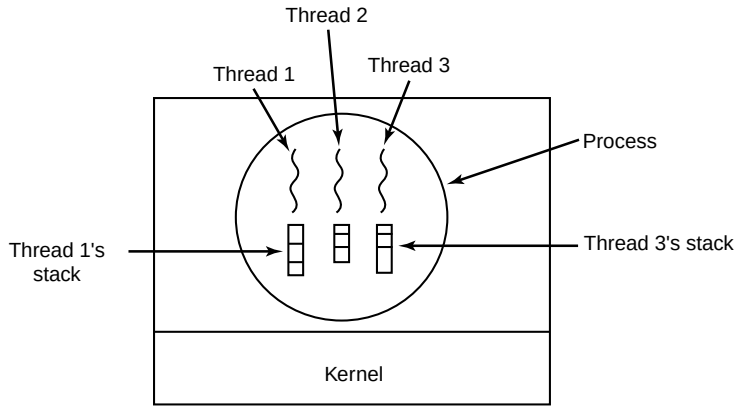
Atrybuty procesów i wątków

Zarządzanie procesem	Zarządzanie pamięcią
Przestrzeń adresowa	Licznik rozkazów
Wartości zmiennych globalnych	Wartości rejestrów
Otwarte pliki	Stos
Procesy potomne	Stan
Alarmy	
Sygnały i obsługa sygnałów	
Informacje rozliczeniowe	

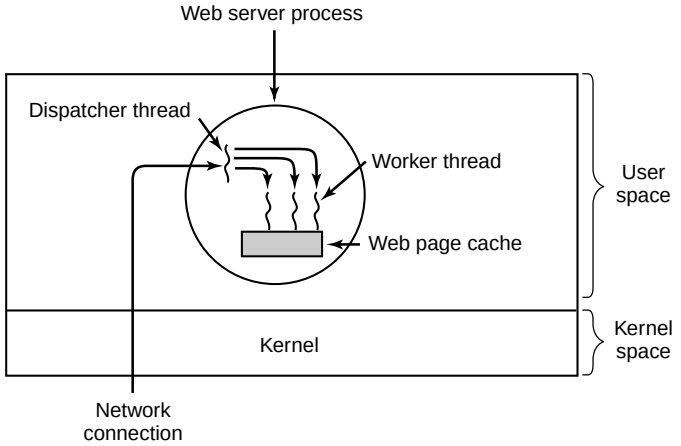
Wątki tego samego procesu mogą wymieniać informacje przez zwienne globalne procesu.

- √ co zrobić z wątkami w przypadku zlecenia stworzenia podprocesu,
- √ na jakim poziomie obsługiwać sygnały.

Stos wątków



Serwer wielowątkowy



Zarys algorytmów serwera wielowątkowego

```
while (TRUE) {
    get_next_request(&buf);
    handoff_work(&buf);
}
```

(a)

```
while (TRUE) {
    wait_for_work(&buf)
    look_for_page_in_cache(&buf, &page);
    if (page_not_in_cache(&page))
        read_page_from_disk(&buf, &page);
    return_page(&page);
}
```

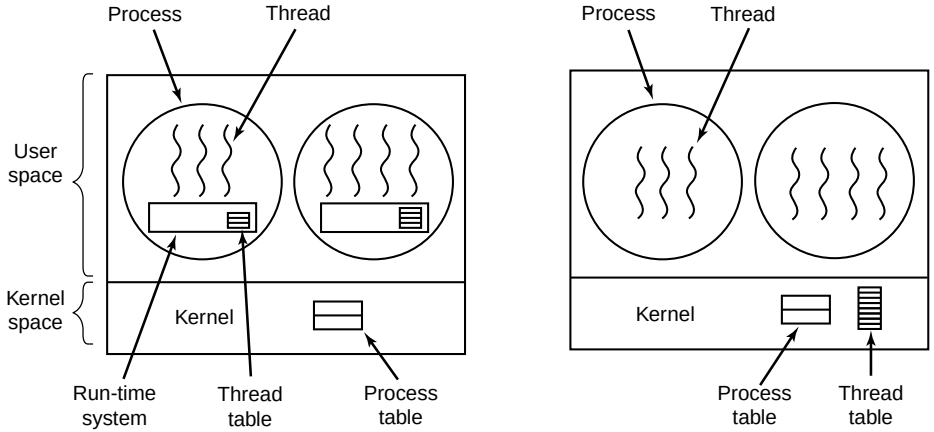
(b)

Metody konstrukcji serwerów

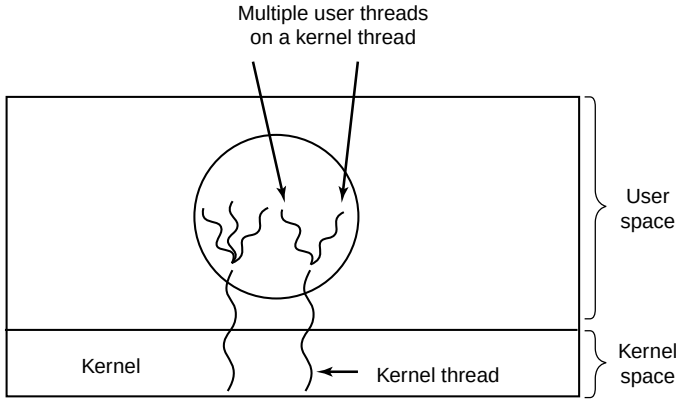
Metody organizacji serwerów usług:

- √ **serwery wielowątkowe** - współbieżność, blokujące wywołania systemowe,
- √ **procesy jednowątkowe** - brak współbieżności, blokujące wywołania systemowe,
- √ **automaty skończone** - współbieżność, nieblokujące wywołania systemowe, przerwania.

Wątki poziomu jądra a wątki poziomu użytkownika



Wątki - rozwiązania hybrydowe



Architektura wielowątkowa w systemie Solaris

Solaris wykorzystuje cztery rozłączne koncepcje:

- √ **Procesy** - standardowe procesy systemu Unix,
- √ **Wątki poziomu użytkownika** - zaimplementowane bibliotecznie, nierozróżnialne z punktu widzenia jądra, stanowią interfejs do współbieżności,
- √ **Procesy lekkie** (ang. *Lightweight processes, LWP*), LWP stanowi formę odwzorowania między wątkami poziomu użytkownika a wątkami jądra.
  - ⋆ każdy LWP obsługuje jeden bądź więcej wątków poziomu użytkownika odwzorowując w jeden wątek jądra,
  - ⋆ LWP rozróżniane i szeregowane przez jądro,
  - ⋆ LWP mogą być uruchomione równolegle w architekturze wieloprocesorowej,
- √ **Wątki jądra** podstawowe elementy szeregowane i rozmieszczane na procesorach.

Wątki w systemie Solaris - przykład

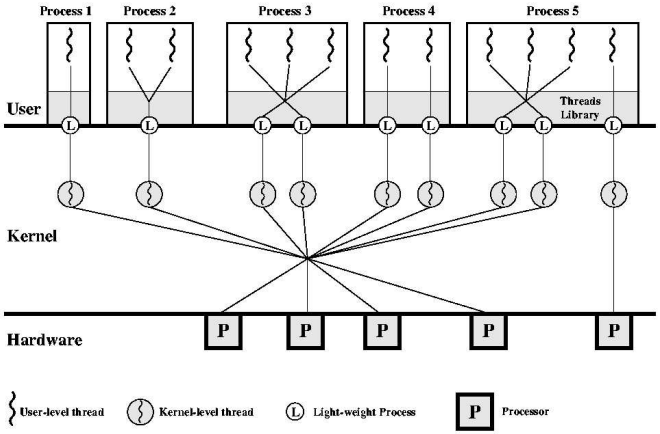
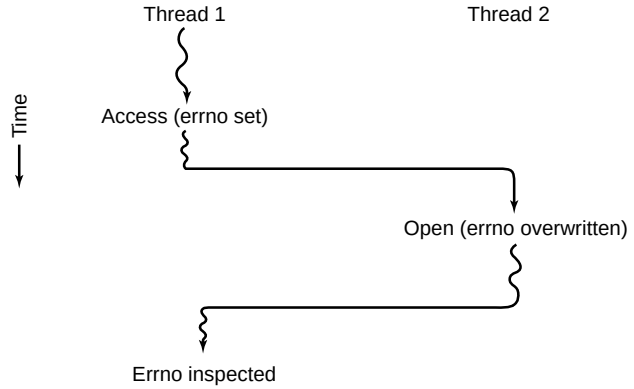
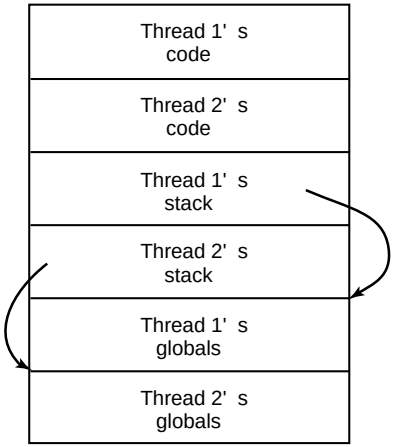


Figure 4.15 Solaris Multithreaded Architecture Example

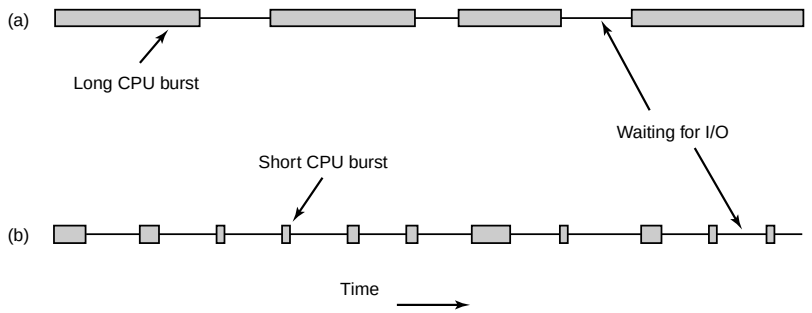
Migracja do kodu wielowątkowego



Prywatne zmienne globalne



Procesy obciążające procesor i we/ wy



Szeregowanie procesów

Wyróżniamy następujące techniki szeregowania:

- √ szeregowanie bez wywłaszczania (ang. *nonpreemptive scheduling*),
- √ szeregowanie z wywłaszczaniem (ang. *preemptive scheduling*).

Różne wymagania dla różnych środowisk: przetwarzania wsadowego, systemów interaktywnych, systemów czasu rzeczywistego.

Cechy dobrego szeregowania

Wszystkie systemy

- √ **sprawiedliwość** - każdemu równą część czasu CPU,
- √ **zgodność z polityką** - praca wedle założeń,
- √ **wyrównywanie** - zbliżona zajętość wszystkich części systemu.

Systemy wsadowe

- √ **przepustowość** - maksymalizacja liczby zadań w czasie,
- √ **czas w systemie** - min. czasu między uruchomieniem a zakończeniem,
- √ **wykorzystanie procesora** - minimalizacja przerw w pracy procesora.

Systemy interaktywne

- √ **czas odpowiedzi** - możliwie szybka odpowiedź na żądanie,
- √ **proporcjonalność** - spełnianie oczekiwań użytkownika.

Systemy czasu rzeczywistego

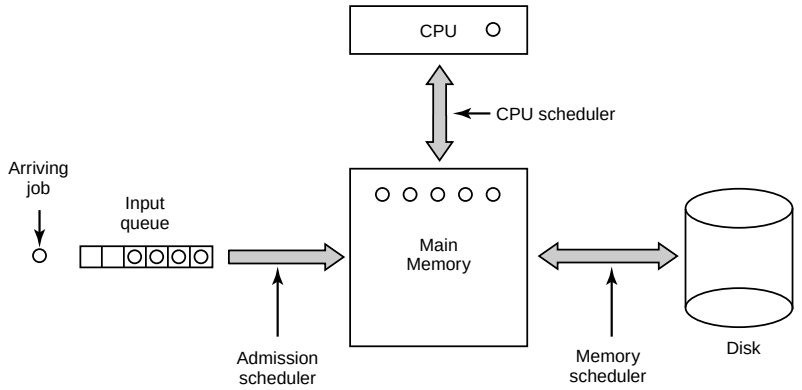
- √ **spełnianie wymagań** - spełnianie ograniczeń czasowych,
- √ **przewidywalność** - np. unikanie spadku jakości w przekazie multimediów.

Najszybciej najkrótsza praca

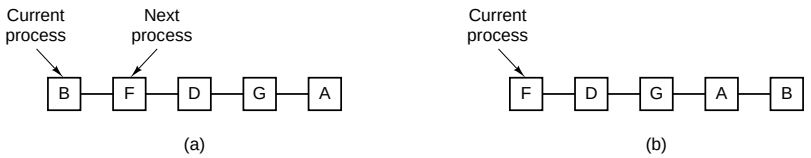
Szeregowanie w systemach wsadowych

- √ FCFS, kolejka FIFO, (ang. *First-Come First-Served*),
- √ SJF, najszybciej najkrótsza praca (ang. *Shortest Job First*),
- √ SRTN, (ang. *Shortest Remaining Time Next*),
- √ szeregowanie trzypoziomowe (ang. *Three-Level Scheduling*).

Szeregowanie trzypoziomowe



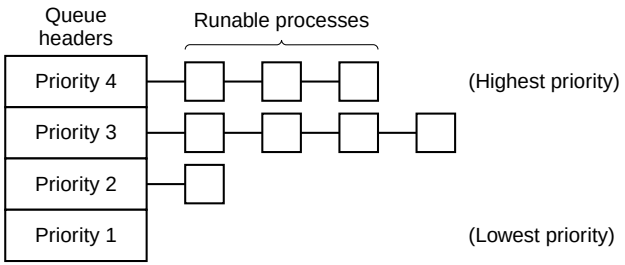
Szeregowanie Round-Robin



Szeregowanie w systemach interaktywnych

- √ algorytm Round-Robin,
- √ szeregowanie priorytetowe,
- √ najkrótszy proces jako następny (estymacja).

Szeregowanie z klasami priorytetowymi



Szeregowanie w systemach czasu rzeczywistego

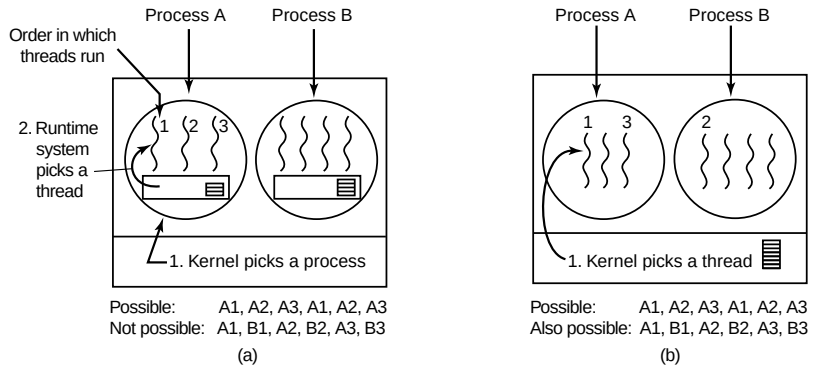
- √ system z ograniczeniami twardymi i miękkimi,
- √ specyfika obsługi zdarzeń okresowych i nieokresowych,

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \ 1$$

System rzeczywisty spełniający powyższą nierówność jest szeregowalny (ang. *schedulable*).

- √ C_i czas wykonania zadania okresowego,
- √ P_i okres wykonywania zadania okresowego.

Szeregowanie wątków



- √ a. dla wątków poziomu użytkownika,
- √ b. dla wątków poziomu jądra.