Akademia Nauk Stosowanych
Wydział Nauk Inżynieryjnych
Kierunek: Informatyka
studia I stopnia, semestr 2

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU

Systemy operacyjne

WYKŁAD 6

dr inż. Stanisława Plichta

splichta@ans-ns.edu.pl

Procesy

- Najważniejszą częścią jądra systemu jest podsystem zarządzania procesami.
- Proces to ciąg czynności wykonywanych za pośrednictwem ciągu rozkazów, których wynikiem jest wykonanie pewnych zadań.
- Pod pojęciem procesu możemy rozumieć program (pasywny), który się aktualnie wykonuje (aktywny).
- Proces potrzebuje pewnych zasobów, do których należą:
  - pamięć,
  - procesor,
  - wszelkiego typu urządzenia zewnętrzne.

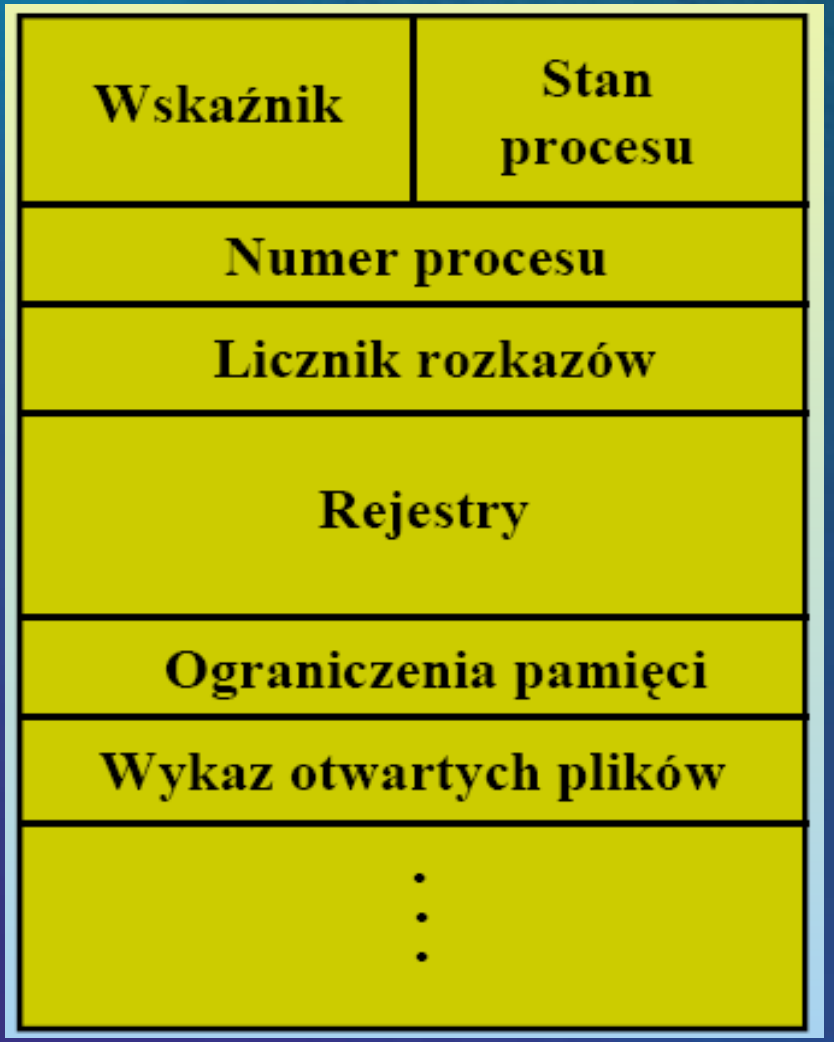
Stany procesu

- Każdy proces otrzymuje jednoznacznie go identyfikujący numer tzw. PID (proces ID).
- Wykonujący się proces zmienia swój stan.
- Stan procesu jest po części określony przez bieżącą czynność procesu.

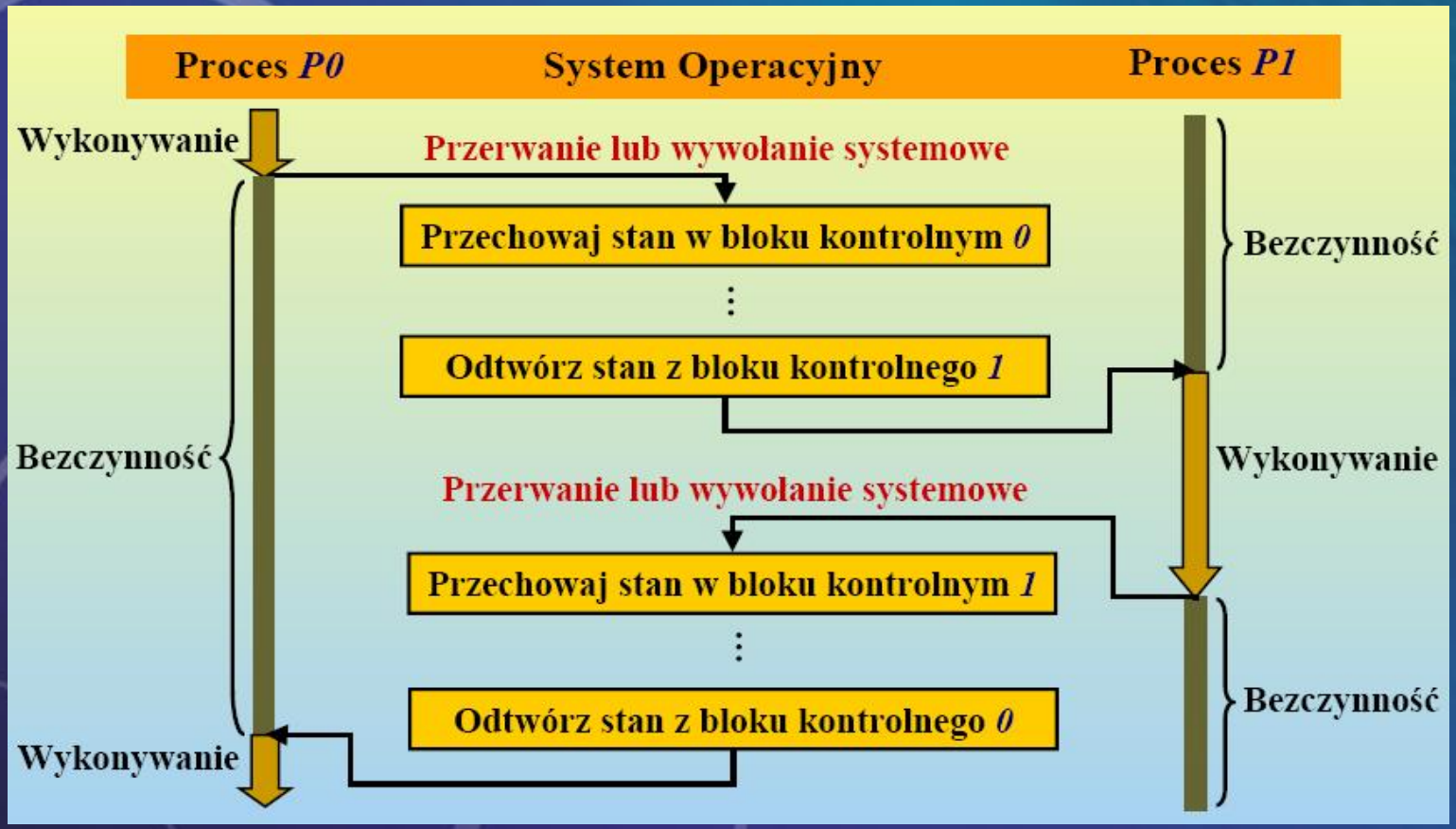
- **AKTYWNY** - są wykonywane instrukcje
- **CZEKAJĄCY** - proces czeka na wystąpienie jakiegoś zdarzenia
- **GOTOWY** - proces czeka na przydział procesora

Blok kontrolny procesu

- SO aby zaspokoić wszystkie potrzeby procesów musi posiadać zbiór informacji o każdym procesie.
- Każdy proces jest reprezentowany w systemie operacyjnym przez swój *blok kontrolny procesu*.



Przełączanie procesów



Planowanie procesów

- Celem planowania procesów jest jak najlepsze wykorzystanie procesora – szczególnie ważne w systemach wieloprogramowych z **podziałem czasu**

- **Kolejki planowania** – zbiory procesów czekających na jakieś zdarzenia:
  - **Kolejka zadań** *(job queues)*
  - **Kolejka procesów gotowych** *(ready queue)*
  - **Kolejka do urządzenia** *(device queue)*

Procesy wędrują między różnymi kolejkami

Kolejki procesów

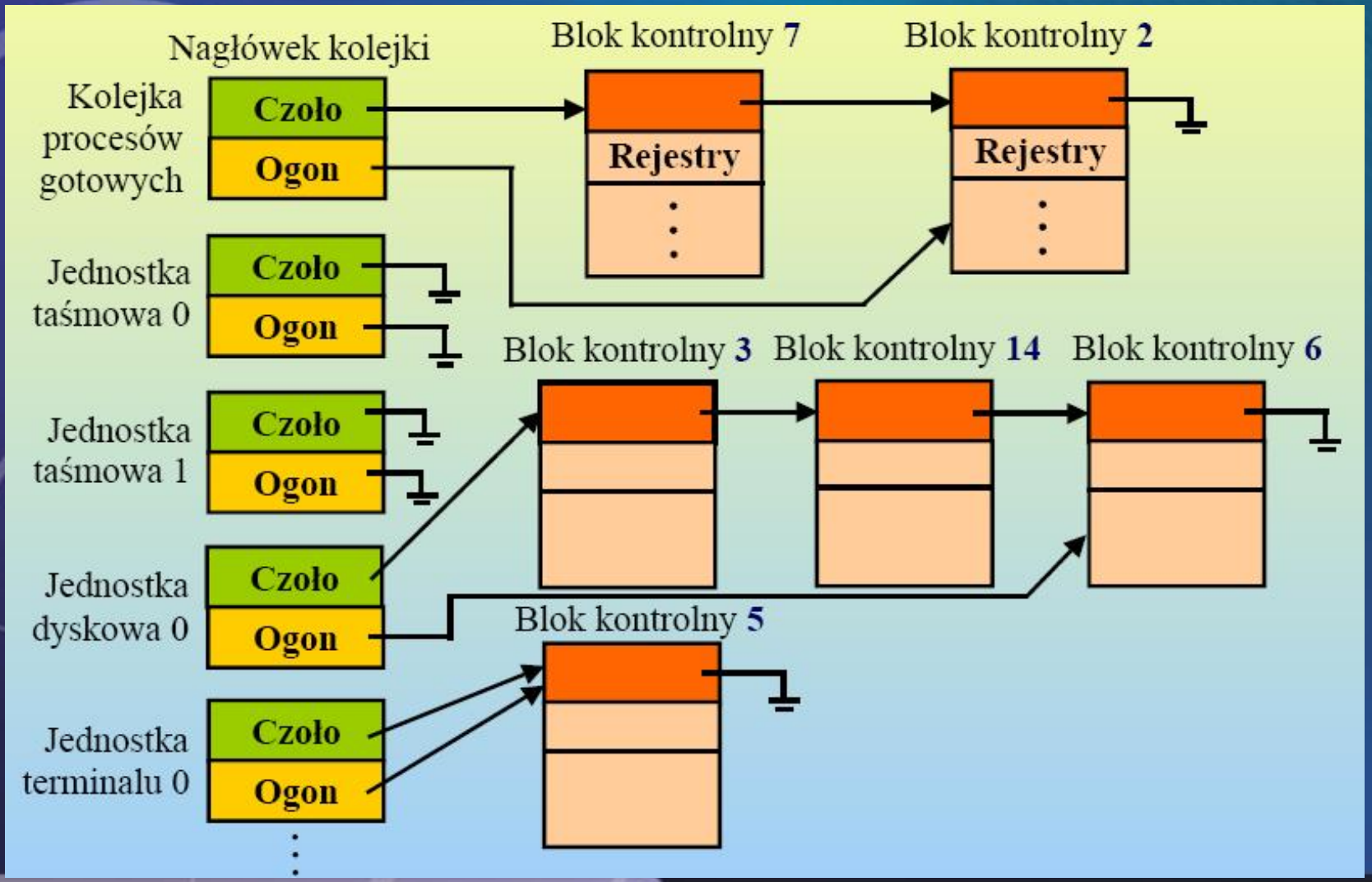
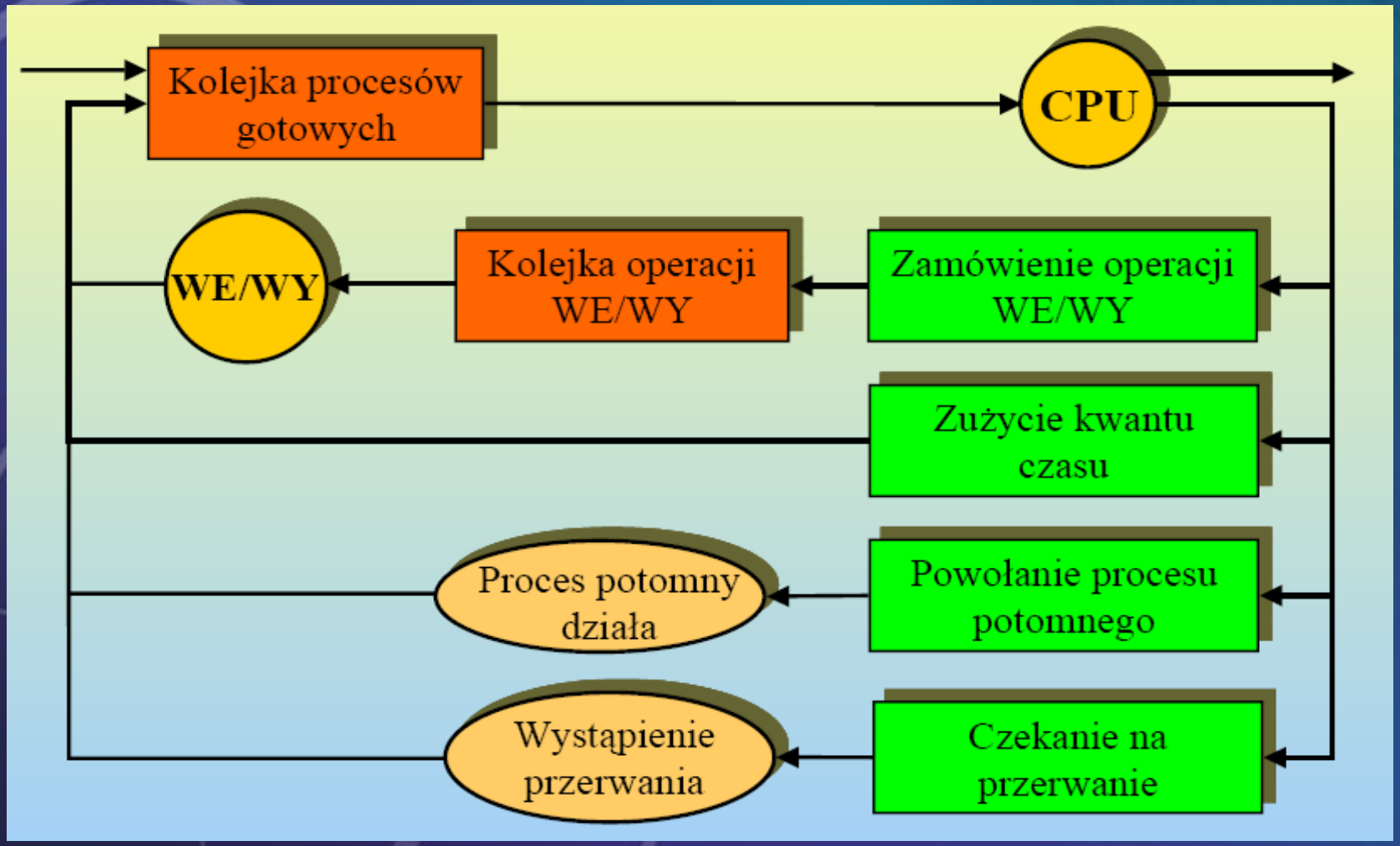


Diagram kolejek



Planowanie procesów

- **Planista długoterminowy – planista zadań**
  (long term scheduler
- **Planista krótkoterminowy**
  (short term scheduler
- **Planista średnioterminowy**
  (*medium term scheduler)*

Planowanie procesów

- Wykonanie procesu - naprzemiennie występujące cykle działań procesora i oczekiwań na op. we/wy
- częstość występowania fazy = f(czas trwania fazy)

  - krzywa wykładnicza (wiele krótkich faz, mało długich)

procesy ograniczone przez we/wy

- wiele krótkich faz

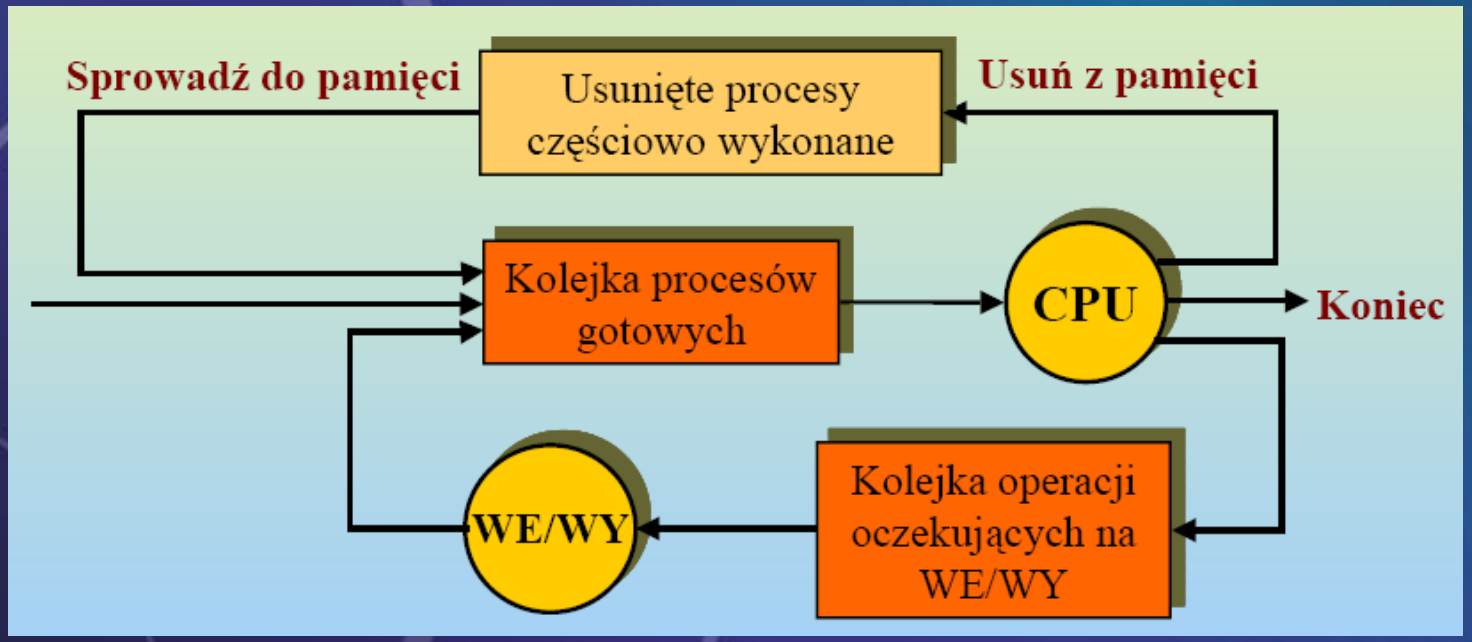
procesy ograniczone przez procesor

- długie fazy procesora

Zadanie planisty długoterminowego - dobranie dobrej mieszanki procesów (*process mix*) zawierającej procesy obydwu rodzajów

Planowanie średnioterminowe

- **Planista średnioterminowy** odpowiedzialny jest za wymianę *(swapping)* procesów między pamięcią operacyjną a dyskiem – stosowany często w systemach z podziałem czasu.



Planowanie procesów

Decyzję o przydziale procesora mogą zapadać w czterech sytuacjach:

1. Proces przeszedł od stanu aktywności do stanu czekania (np. z powodu zamówienia na we/wy lub rozpoczęcia czekania na zakończenie któregoś z procesów potomnych).
2. Proces przeszedł od stanu aktywności do stanu gotowości (np. wskutek wystąpienia przerwania).
3. Proces przeszedł od stanu aktywności do stanu gotowości (po zakończeniu operacji we/wy).
4. Proces kończy działanie

planowanie niewywłaszczeniowe (1, 4)
planowanie wywłaszczeniowe (2, 3)

Kryteria planowania

- **Planowanie niewywłaszczeniowe (*non-preemptive*)**
  - Proces, który dostał procesor, nie odda go aż do swego zakończenia lub przejścia w stan czekania.
  - Nie wymaga wsparcia sprzętowego (np. zegara).
- **Planowanie wywłaszczeniowe (*preemptive*)**
  - Kosztowne - wymaga mechanizmów koordynacji

Ekspedytor

- Ekspedytor - moduł, który faktycznie przekazuje procesor do dyspozycji procesu wybranego przez planistę krótkoterminowego.
  - przełączanie kontekstu
  - przełączanie do trybu użytkownika
  - wznowienia działania programu

Kryteria planowania

- **Wykorzystanie procesora -** (40%-90%)
- **Przepustowość** - liczba procesów kończonych w jednostce czasu (10 proc/s - 1proc/1h)
- **Czas cyklu przetwarzania -** nadejście procesu - zakończenie procesu
- **Czas oczekiwania -** suma okresów czekania w kolejce procesów gotowych do wykonania
- **Czas odpowiedzi -** wysłanie żądania – początek pierwszej odpowiedzi

Algorytmy planowania przydziału procesora

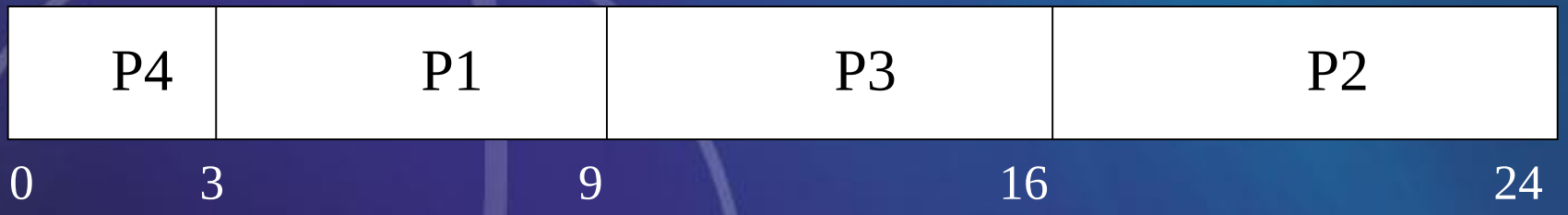
Planowanie metodą FCFS (first come first served)

- Proces, który pierwszy zamówił procesor pierwszy go otrzyma.
- Implementacja - za pomocą kolejki FIFO.
- Blok kontrolny procesu wchodzącego do kolejki procesów gotowych jest dołączany do końca.
- Wolny procesor przydziela się procesowi z czoła kolejki procesów gotowych.
- Średni czas oczekiwania bywa bardzo długi.

Algorytmy planowania przydziału procesora

Najpierw najkrótsze zadanie (SJF shortest job first)

proces	czas trwania fazy
P_1	6
P_2	8
P_3	7
P_4	3



średni czas oczekiwania dla **SFJ** = (0+3+9+16)/4=**7 ms**
dla algorytmu **FCFS** (0+6+14+21)/4=**10,25 ms**

Planowanie priorytetowe

- SJF (PRI=1/dł. fazy) - algorytm SJF jest szczególnym przypadkiem ogólnego algorytmu planowania priorytetowego.
- Każdemu procesowi przypisuje się priorytet, po czym przydziela się procesor temu procesowi, którego priorytet jest najwyższy.
- Procesy o równych priorytetach planuje się w porządku FCFS.
- Priorytet definiowany wewnętrznie

  (limity czasu, wielkość pamięci, liczba otwartych plików)
- Priorytet definiowany zewnętrznie

  (ważność procesu, opłaty, polityka)
- Wywłaszczające lub nie wywłaszczające
- Problem - nieskończone blokowanie, czyli głodzenie niskopriorytetowych procesów

  - postarzanie (*aging*) np. co 15 min PRI+=1

Algorytm planowania rotacyjnego (round-robin)

- Przeznaczony dla systemów z podziałem czasu.
- Kolejka procesów gotowych to kolejka cykliczna.
- Każdy proces otrzymuje odcinek czasu nie dłuższy od jednego kwantu czasu.
- Jeśli faza procesora w danym procesie przekracza 1 kwant czasu, to proces będzie wywłaszczony i wycofany do kolejki procesów gotowych.
- Implementacja - kolejka FIFO.
- Długi średni czas oczekiwania.
- Wywłaszczający.

Wielopoziomowe planowanie kolejek ze sprzężeniem zwrotnym

Planista wielopoziomowych kolejek ze sprzężeniem zwrotnym jest określony za pomocą następujących parametrów:

- liczba kolejek,
- algorytm planowania dla każdej kolejki,
- metoda wykorzystana do decydowania o awansowaniu procesu do kolejki o wyższym priorytecie,
- metoda wykorzystana do decydowania o dymisjonowaniu procesu do kolejki o niższym priorytecie,
- metoda określająca kolejkę, do której trafia proces potrzebujący obsługi.

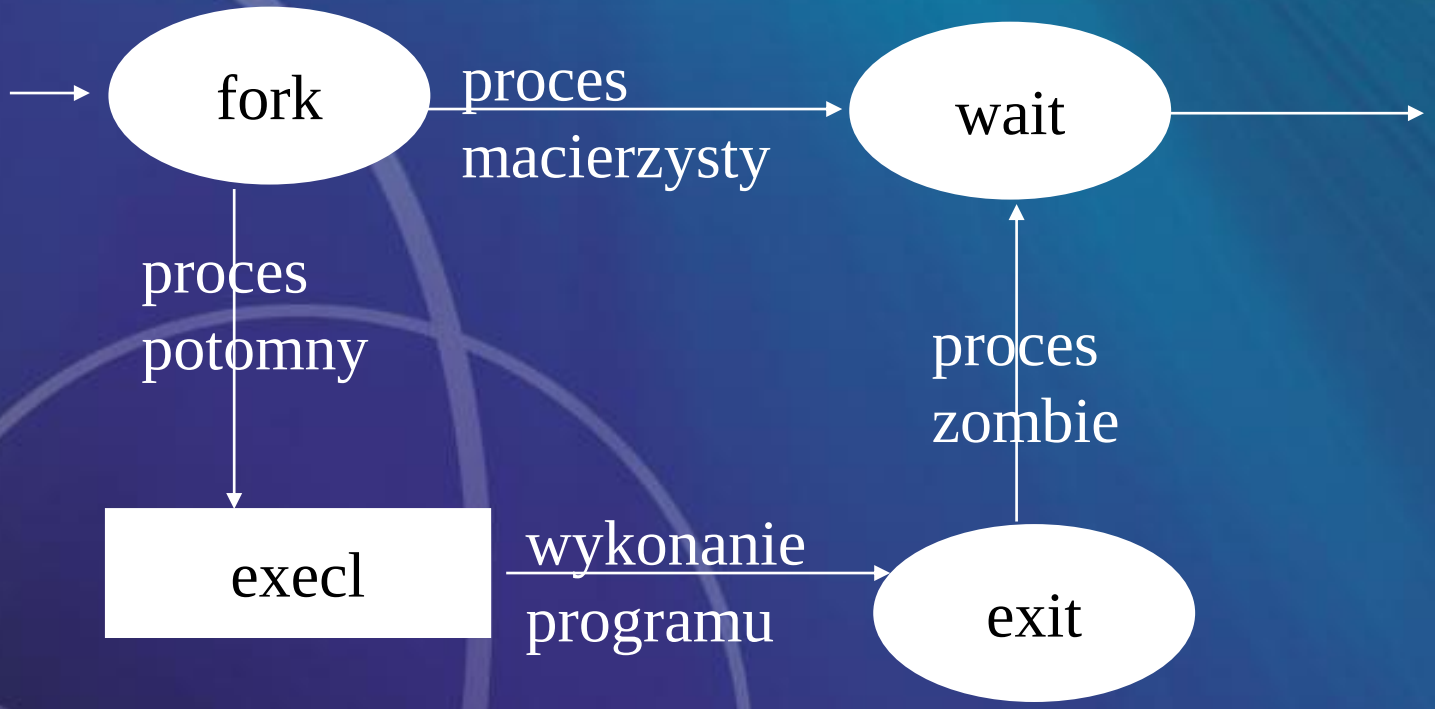
Planowanie wieloprocesorowe

Ważny czynnik - rodzaj zastosowanych procesorów

– **procesory jednakowe (system homogeniczny)**
  - metoda dzielenia obciążeń *(load sharing)*
  - wspólna kolejka procesów gotowych do wykonania
  - przydziela się im dowolny z dostępnych procesorów

– **procesory różne (system heterogeniczny)**
  - możliwości wyboru są ograniczone
  - każdy procesor ma własną kolejkę i własny algorytm planowania
  - procesy muszą być wykonywane przez konkretne procesory
  - procesy są samoistnie poklasyfikowane
  - każdy procesor zajmuje się własnym planowaniem

Zarządzanie procesami – tworzenie nowego procesu



Zarządzanie procesami – tworzenie nowego procesu

Utworzenie procesu potomnego

fork()

W procesie macierzystym funkcja zwraca identyfikator (pid) procesu potomnego (wartość większą od 0, w praktyce większą od 1), a w procesie potomnym wartość 0

Zwrócenie identyfikatora procesu

getpid()

Funkcja zwraca własny identyfikator (pid) procesu, który ją wywołał

Zarządzanie procesami – tworzenie nowego procesu

Zmiana programu wykonywanego przez proces

execl(const char* path, const char* arg, ...)

Proces rozpoczyna wykonywanie nowego programu, którego kod znajduje się w pliku wskazywanym przez path

Oczekiwanie na zakończenie potomka

wait(int *status)

Funkcja zwraca identyfikator (pid) procesu, który się zakończył. Pod adresem wskazywanym przez status umieszczany jest status zakończenia

Zarządzanie procesami w systemie Linux

Do wywołania powyższych funkcji niezbędne pliki nagłówkowe:

<sys/types.h>

<unistd.h>

Typowe wywołanie funkcji fork()

```
switch (fork())
{           case –1:
               perror(„fork error”);
               exit(1);
            case 0:
                /* akcja dla procesu potomnego */
            default:
               /* akcja dla procesu macierzystego */
}
```

Zarządzanie procesami w systemie Linux

Typowe wywołanie fork i execl

```
switch (fork())
{case –1:
              perror(„fork error”);
              exit(1);
case 0: /* proces potomny */
              execl("./nowy_program.x","nowy_program.x",NULL);
              exit(2);
default:  /* proces macierzysty */
}
```

Zarządzanie procesami w systemie LINUX