Wykład 2
Zarządzanie procesami

Jarosław Koźlak

Procesy

Terminologia - niejednoznaczności historyczne

- **zadania (ang, jobs) = procesy**
- **programy użytkownika (ang. user programs) = prace (ang. tasks)**

- **Program** - obiekt pasywny **(**zawartość pliku na dysku)
- **Proces** - obiekt aktywny (zawiera licznik rozkazów określający następny rozkaz do wykonania i zbiór przydzielonych zasobów)
- **Proces --** wykonujący się program

- **Proces składa się z:**
  - kodu programu (sekcji tekstu, ang. text section)
  - odpowiedniego ustawienia licznika rozkazów (program counter)
  - zawartości rejestrów procesora
  - stosu procesu (ang. process stack) z danymi tymczasowymi (parametrami procedur, adresami powrotnymi, zmiennymi tymczasowymi)
  - sekcji danych (ang. data section) ze zmiennymi globalnymi

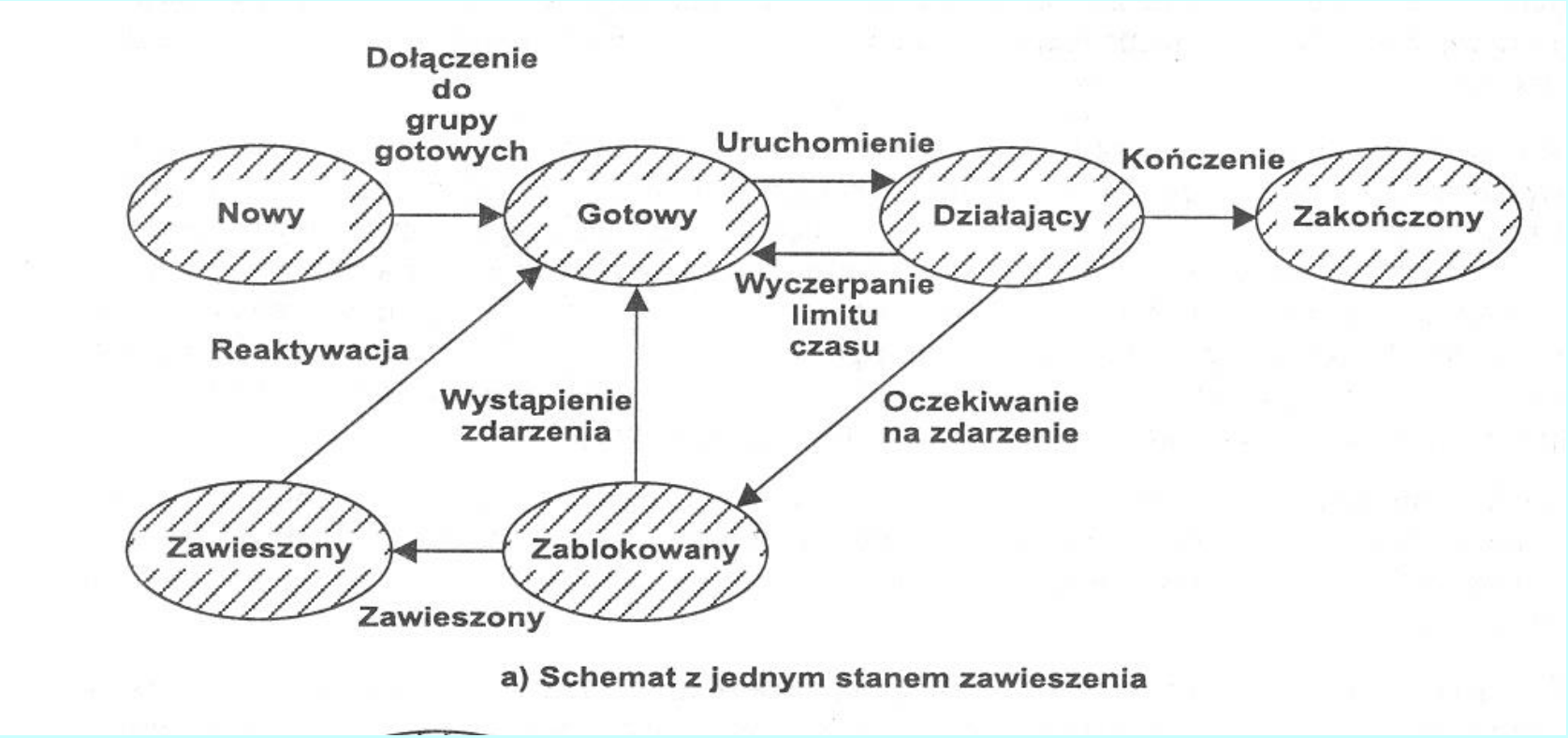
- **Brak odpowiedniości 1 program - 1 proces:**
  - wiele procesów może być uruchomionych w oparciu o jedną kopię programu
  - wykonywany proces może tworzyć procesy potomne

Stany procesu

Stany procesu 2

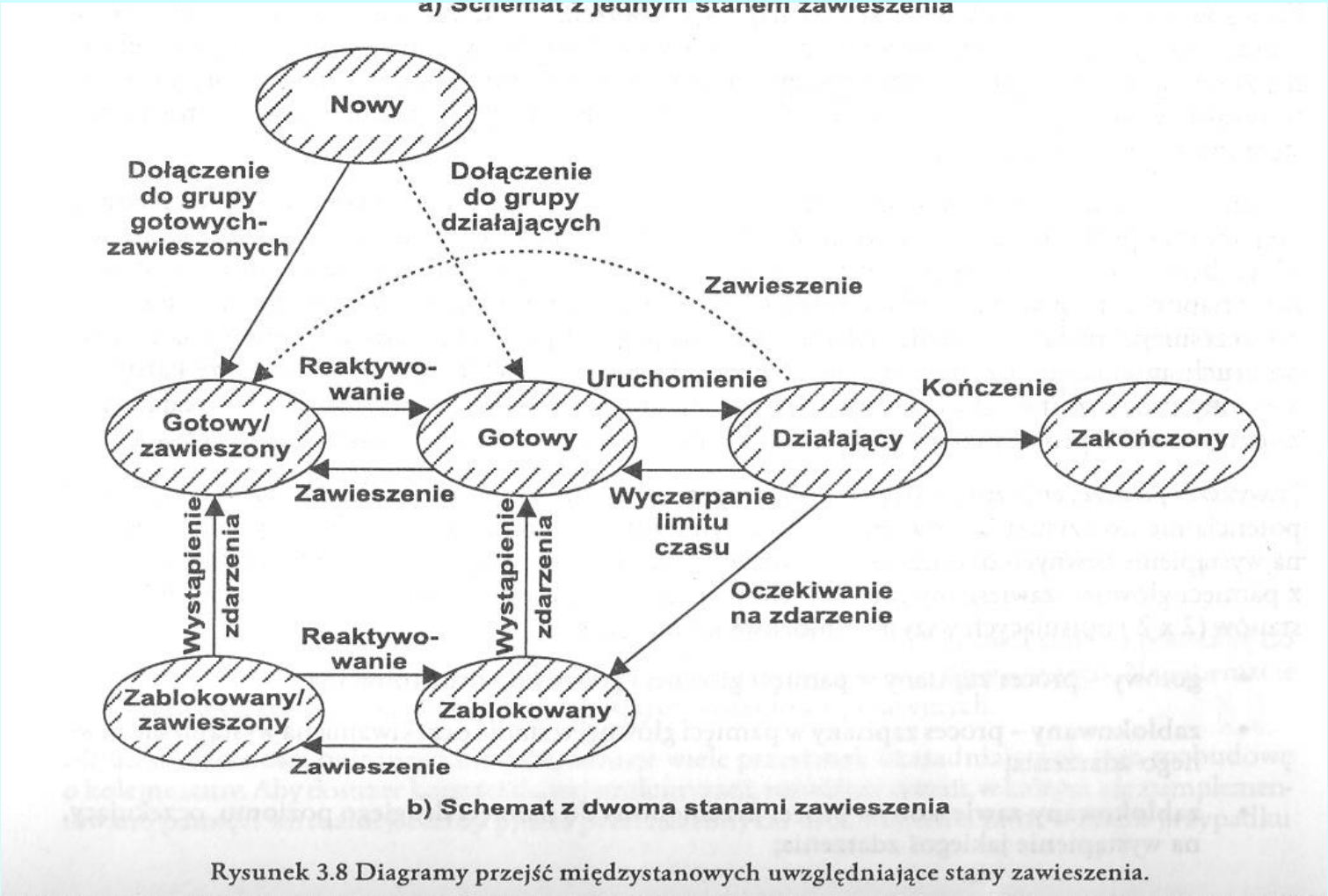
- **nowy** - proces został utworzony
- **aktywny** - są wykonywane instrukcje
- **czekający** - proces czeka na zakończenie jakiegoś zdarzenia (np. zakończenie operacji we/wy)
- **gotowy** - proces czeka na przydział procesora
- **zakończony** - proces zakończył działanie

Stany procesu: 1 stan zawieszenia



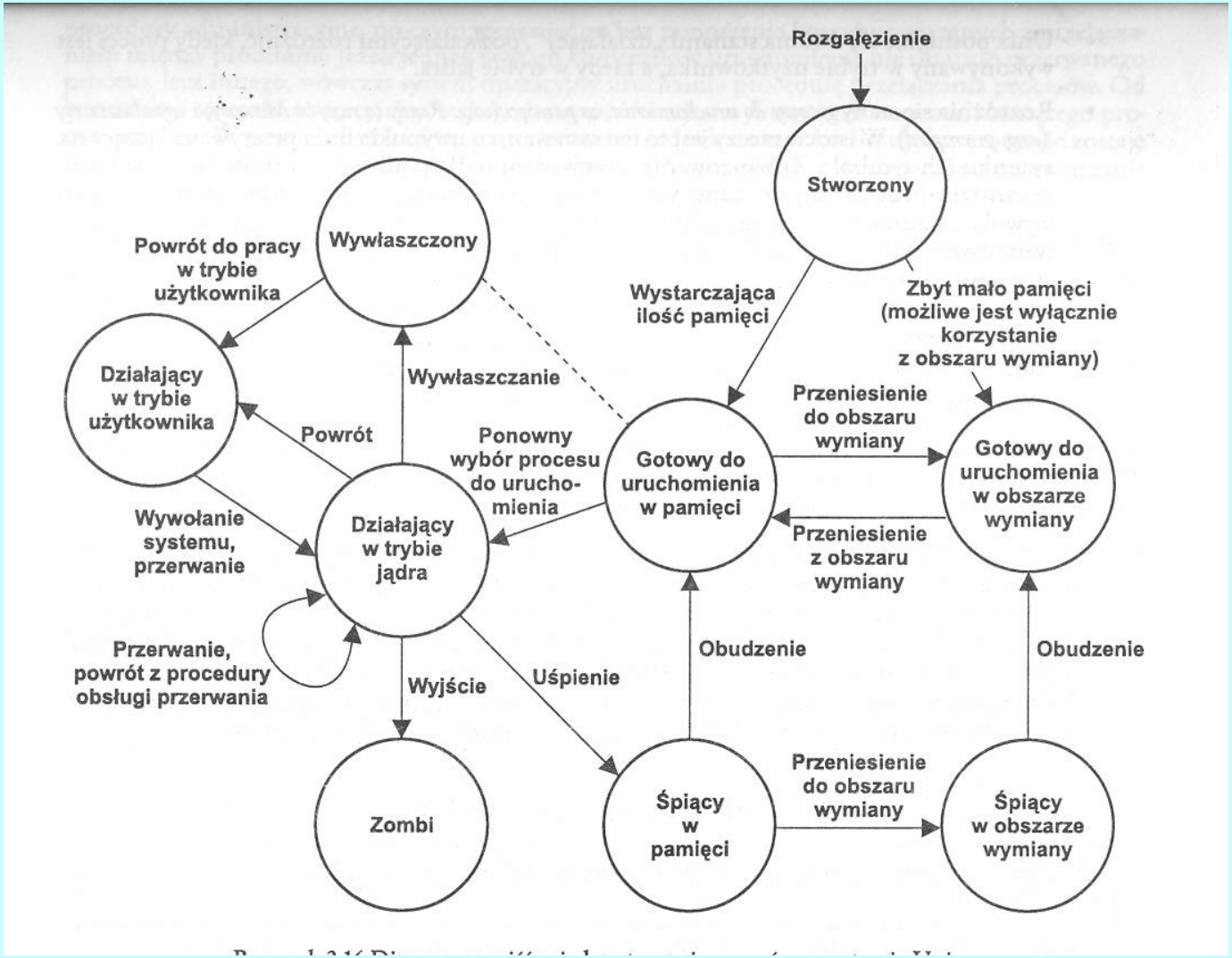
a) Schemat z jednym stanem zawieszenia

Stany procesu: 2 stany zawieszenia



Rysunek 3.8 Diagramy przejść międzystanowych uwzględniające stany zawieszenia.

Stany procesu: System Unix



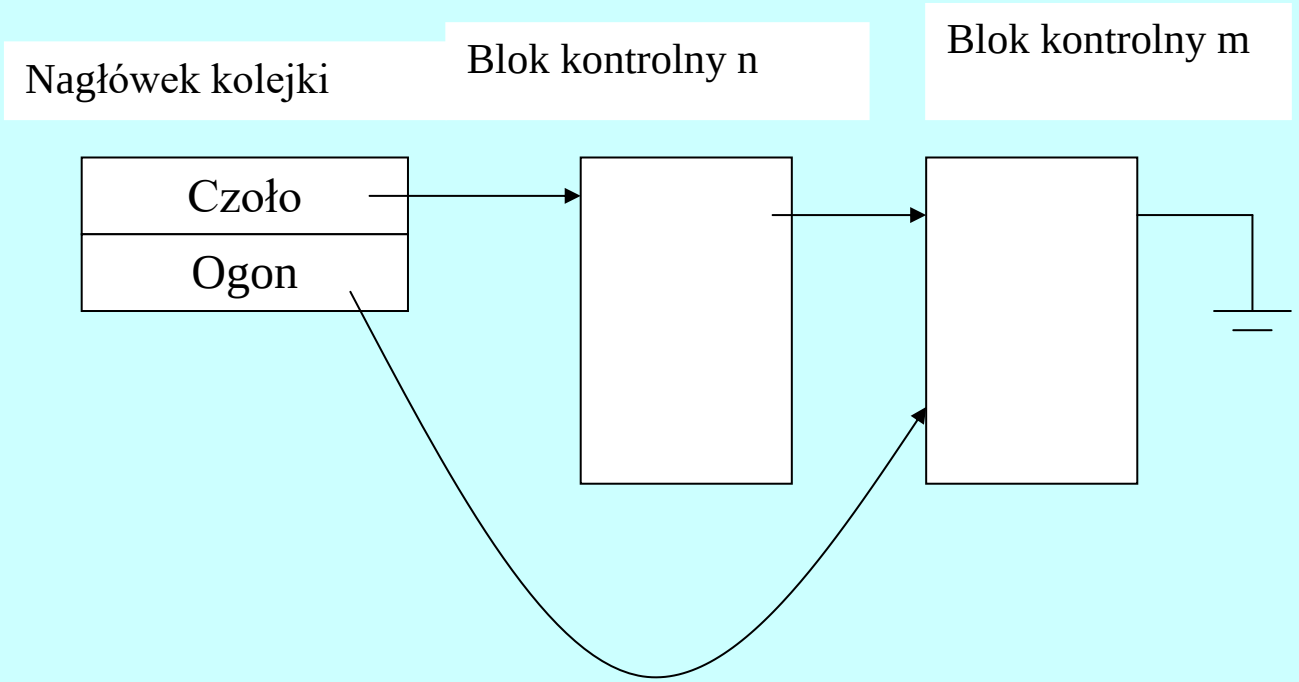
Blok kontrolny procesu

- Każdy proces jest reprezentowany w systemie operacyjnym przez blok kontrolny procesu (ang. process control block - PCB) .
- Inna nazwa bloku kontrolnego procesu - blok kontrolny zadania
- W skład bloku kontrolnego procesu wchodzą:
- **Stan procesu** : nowy, gotowy, aktywny, czekający
- **Licznik rozkazów:** z adresem następnego rozkazu do wykonania w procesie
- **Rejestry procesora:** zależne od architektury komputera np. akumulatory, rejestry indeksowe, wskaźniki stosu, rejestry ogólnego przeznaczenia, rejestry warunków
- **Informacje o planowaniu przydziału procesora:** priorytet procesu, wskaźniki do kolejek porządkujących zamówienia i inne
- **Informacje o zarządzaniu pamięcią:** zawartości rejestrów granicznych, tablice stron lub tablice segmentów
- **Informacje do rozliczeń:** ilość zużytego czasu procesora i czasu rzeczywistego, ograniczenia czasowe, numery kont, numery procesów
- **Informacje o stanie wejścia-wyjścia:** informacje o urządzeniach we/wy przydzielonych do procesu, wykaz otwartych plików itd.

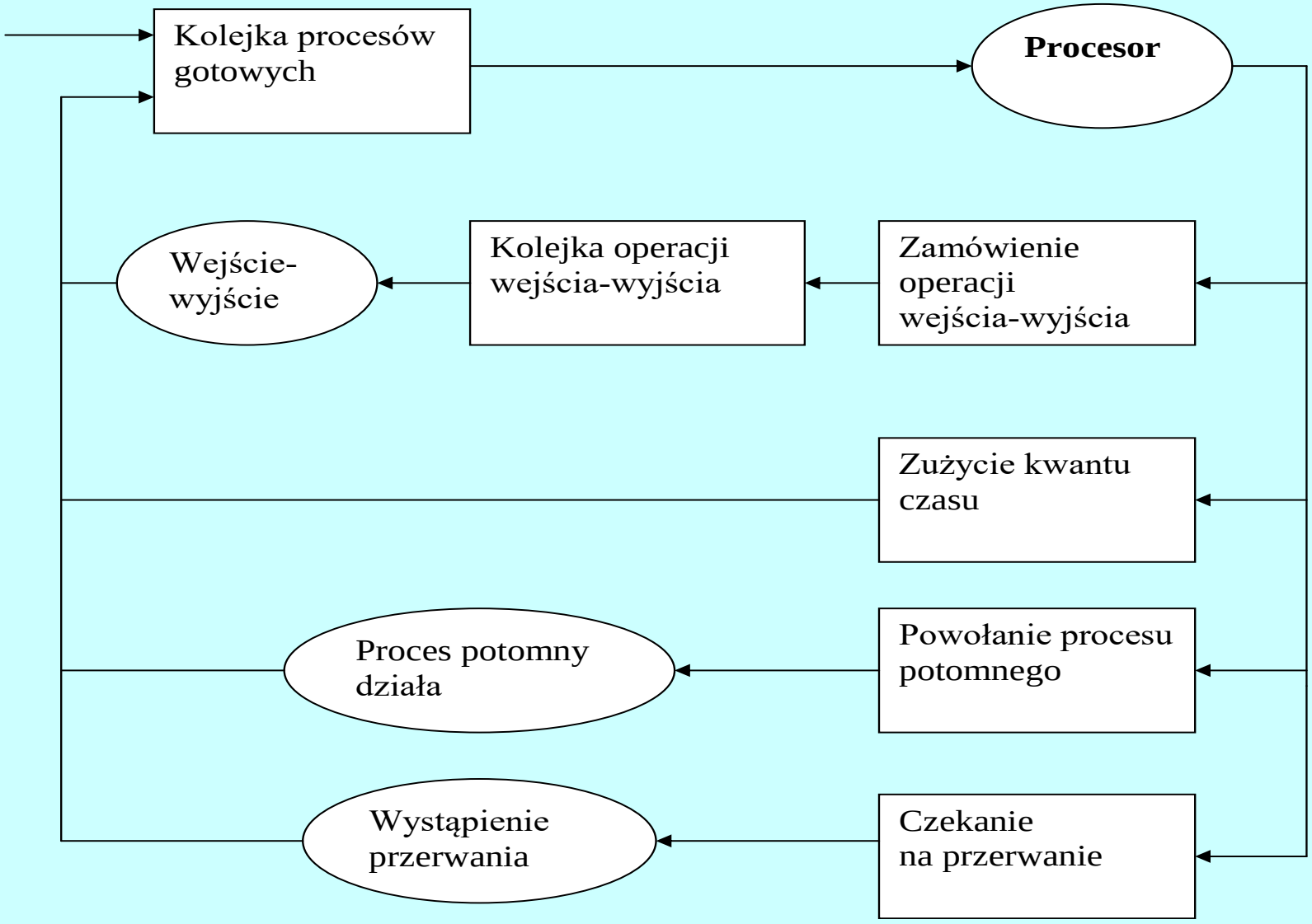
Planowanie procesów

- procesy w systemie są ulokowane w kolejkach zadań(ang. job queue)
- procesy oczekujące w pamięci głównej gotowe do działania są w kolejce procesów gotowych (ang. ready queue)
- procesy czekające na przydział konkretnego urządzenia -- w kolejkach do urządzeń (ang. device queue), każde urządzenie ma własną kolejkę
- Reprezentacja kolejek:
  - listy powiązane
  - nagłówek kolejki ma wskaźniki do pierwszego i ostatniego bloku kontrolnego procesu
  - każdy blok kontrolny ma pole wskazujący następną pozycję w kolejce procesów gotowych

Kolejka procesów



Kolejki w planowaniu procesów



Planiści

- **Planista** (program szeregujący, ang. scheduler) - proces systemowy, który odpowiada za wybór procesów z kolejek
- **Planista długoterminowy** (ang. long-term scheduler); inna nazwa - planista zadań (ang. job scheduler) - wybiera procesy z pamięci masowej i ładuje je do pamięci w celu wykonania; czas wykonywania rzędu minut (np. gdy proces opuszcza system)
- **Planista krótkoterminowy** (ang. short-time scheduler); planista przydziału procesora (ang. CPU scheduler) - wybiera jeden proces spośród procesów gotowych do wykonania i przydziela mu procesor; czas uruchamiania rzędu milisekund
- **Dwa rodzaje procesów:**
- procesy ograniczone przez we/wy - spędzają większość czasu na operacjach we/wy
- procesy ograniczone przez procesor - spędzają większość czasu na obliczeniach zajmując procesor
- **Zadanie planisty długoterminowego:** dobranie dobrej mieszanki procesów (ang. process mix) zawierającą procesy obydwu rodzajów
- **Może wystąpić dodatkowy planista średnioterminowy (ang. medium-term scheduler) -** odpowiada za usuwanie procesów z pamięci w celu zmniejszenia stopnia wieloprogramowości,
- takie postępowanie - wymiana (ang. swapping)

Przełączanie kontekstu

- przełączanie procesora dla innego procesu (koszt czasowy rzędu 1-1000us, wymaga:
- przechowania stanu starego procesu
- załadowania przechowanego stanu nowego procesu

Tworzenie procesów:

- użycie wywołania systemowego "utwórz proces"
- Proces tworzący nowy proces: proces macierzysty (ang. parent process)
- Proces tworzony przez proces macierzysty: proces potomny, potomek (ang. children)

- Powstający podproces może otrzymywać niezbędne zasoby:
  - od systemu operacyjnego
  - uzyskuje podzbiór zasobów posiadanych przez proces macierzysty

- Możliwe relacje procesu macierzystego i procesu potomka:
  - proces macierzysty kontynuuje działanie współbieżnie ze swoimi potomkami
  - proces macierzysty oczekuje na zakończenie działania niektórych lub wszystkich swoich procesów potomnych

- Przydział przestrzeni adresowej:
  - proces potomny staje się kopią procesu macierzystego
  - proces potomny otrzymuje nowy program

Tworzenie procesów. System Unix

Stworzenie procesu

- każdy proces ma jednoznaczny identyfikator procesu
- proces jest tworzony funkcją systemową **fork**
- nowy proces otrzymuje kopię przestrzeni adresowej procesu macierzystego
- oba procesy kontynuują działanie od instrukcji następującej po wywołaniu funkcji fork
- fork zwraca 0 nowemu procesowi, a identyfikator potomka - procesowi macierzystemu

Inne podstawowe funkcje związane z uruchomieniem procesu:

- funkcja **exec** - zastąpienie pamięci procesu przez inny program
- funkcja **wait** umożliwia proc. macierz. czekanie na zakończenie działania potomka

Zakończenie procesu. Unix

- Proces kończy się, gdy wykona swą ostatnią instrukcję i wywołując funkcję **exit** poprosi SO, by go usunął
- Podczas zakończenia proces może przekazać dane do procesu macierzystego (poprzez funkcję **wait** w procesie macierzystym)

- Inne metody zakończenia procesu:
  - poprzez wywołanie odpowiedniej funkcji systemowej można zakończyć działanie innego procesu (zazwyczaj przodek lub proces użytkownika o odpowiednich uprawnieniach)

Wątki

Wątki

wątek (ang. *thread*)

Wątek - podstawowa jednostka wykorzystania procesora, obejmuje:

- licznik rozkazów
- zbiór rejestrów
- obszar stosu

Wiele wątków może współdzielić:

- sekcję kodu
- sekcję danych
- zasoby SO - pliki, sygnały itd.

- Tradycyjny proces - ciężki proces (ang. *heavyweight process*): równoważny zadaniu z jednym wątkiem
- Przełączanie wątków jest znacznie mniej kosztowne niż przełączanie kontekstu procesów ciężkich

Wątki 2

- **Współpracujące wątki**
  - operują na wspólnych danych
  - są w obrębie jednego procesu
  - nie ma mechanizmów ochrony, należą do tego samego użytkownika

Ograniczenia procesów

- są aplikacje, które mają do wykonania wiele niezależnych zadań, które mogą być realizowane współbieżnie, ale muszą współdzielić przestrzeń adresową i inne zasoby:
  - zarządcy baz danych pełniący rolę serwera w modelu klient-serwer
  - monitory przetwarzające transakcje
  - programy obsługi protokołów sieciowych ze środkowych i górnych warstw sieciowych

- tradycyjne procesy mogą w danej chwili używać tylko jednego procesora, dlatego nie mogą wykorzystać zalet architektur wieloprocesorowych

Rodzaje wątków

Pojęcia

- **proces** jest złożoną całością, którą można podzielić na dwie części: zbiór wątków i zestaw zasobów
- **wątek** jest obiektem dynamicznym, reprezentującym punkt sterowania wewnątrz procesu i wykonującym pewien ciąg rozkazów

- **zasoby procesu:** przestrzeń adresowa, otwarte pliki, informacje identyfikujące użytkownika, miejsce dostępne na dysku itd.
- **zasoby są współdzielone przez wszystkie wątki w procesie**
- **każdy wątek ma obiekty związane tylko z nim, jak:** licznik rozkazów, stos, kontekst rejestrów
- **właściciel zasobów** - proces

Poziomy realizacji wątków

- Wątki poziomu użytkownika
- Procesy lekkie
- Wątki poziomu jądra

Wątki poziomu użytkownika

- abstrakcja wątków jest udostępniana na poziomie użytkownika, jądro nie wie o ich istnieniu
- realizacja za pomocą pakietów bibliotecznych jak pthreads (zgodny ze standardem POSIX)
- synchronizacja, szeregowanie i zarządzanie takimi wątkami odbywa się bez udziału jądra – jest bardzo szybkie
- kontekst wątku z poziomu użytkownika jest zapamiętywany i odtwarzany bez udziału jądra
- wątek użytkownika ma własny stos, przestrzeń do zapisania kontekstu rejestrów z poziomu użytkownika a także informacje o stanie wątku jak maska sygnałów

Szeregowanie:

- jądro szereguje procesy lub procesy lekkie, w obrębie których wykonują się wątki użytkownika
- proces do szeregowania swoich wątków stosuje funkcje biblioteczne
- jeśli jest wywłaszczany proces, lub proces lekki, są wywłaszczane także jego wątki
- jeśli watek użytkownika wykona blokującą funkcję systemowa, jest wstrzymywany proces lekki, w którym wykonuje się wątek - jeśli w obrębie procesu jest tylko jeden proces lekki, to wszystkie jego wątki są wstrzymane

Wątki poziomu użytkownika: Zalety

- umożliwiają bardziej naturalny sposób zapisu wielu programów, takich jak systemy okienkowe
- wydajność - nie zużywają zasobów jądra, jeśli nie są związane z żadnym procesem lekkim
- wydajność wynika z zaimplementowania na poziomie użytkownika i z nie stosowania funkcji systemowych

Wątki użytkownika: Ograniczenia

- ograniczenia wynikają głównie z braku przepływu informacji między jądrem i biblioteką wątków
  - jądro nie ma informacji o wątkach użytkownika, nie może używać swoich mechanizmów ochrony do ich ochrony przed niedozwolonym dostępem ze strony innych wątków
  - szeregowanie przez jądro i bibliotekę, przy czym żadne z nich nie posiada wiedzy o czynnościach drugiego
- każdy proces działa we własnej przestrzeni adresowej, wątki użytkownika nie mają takiej ochrony – biblioteka wątków musi zapewnić mechanizmy synchronizacji
- niemożność jednoczesnego wykonywania wątków, w przypadku maszyny wieloprocesorowej

Wątki jądra

- wątek jądra nie musi być związany z procesem użytkownika
- jądro tworzy go i usuwa wewnętrznie w miarę potrzeb
- wątek odpowiada za wykonanie określonej czynności, współdzieli tekst jądra i jego dane globalne, jest niezależnie szeregowany i wykorzystuje standardowe mechanizmy jądra, takie jak *sleep()* czy *wakeup()*
- wątki potrzebują jedynie następujących zasobów:
  - własnego stosu,
  - przestrzeni do przechowywania kontekstu rejestrów w czasie, gdy wątek nie jest wykonywany
- tworzenie i stosowanie wątków jądra nie jest kosztowne
  - przełączanie kontekstu między wątkami jądra jest szybkie, ponieważ nie trzeba zmieniać odwzorowań pamięci
- zastosowania wątków jądra
  - wątki jądra są przydatne do wykonywania pewnych operacji takich, jak asynchroniczne wejście-wyjście
    - zamiast udostępniać osobne operacje asynchronicznego we/wy, jadro może realizować je, tworząc odrębny wątek do wykonania każdego zlecenia –
  - wątki poziomu jądra można wykorzystywać też do obsługi przerwań
- wątki poziomu jądra nie są nowym pomysłem: procesy systemowe jak demon stronicujący (pagedaemon) pełnią w tradycyjnych jądrach uniksowych te funkcje co wątki jądra

Procesy lekkie

- proces lekki - wspierany przez jądro wątek poziomu użytkownika
- wysokopoziomowe pojęcie abstrakcyjne oparte na wątkach jądra - system udostępniający procesy lekkie, musi także udostępniać wątki jądra
- każdy proces lekki jest związany z wątkiem jądra, ale niektóre wątki jądra są przeznaczone do realizacji pewnych zadań systemowych, wtedy nie przypisuje się im żadnych lekkich procesów
- w każdym procesie może być kilka procesów lekkich, każdy wspierany przez oddzielny wątek jądra, współdzielą one przestrzeń adresową i inne zasoby procesu
- procesy lekkie są niezależnie szeregowane przez systemowego planistę,
- procesy lekkie mogą wywoływać funkcje systemowe, które powodują wstrzymanie w oczekiwaniu na we/wy lub zasób
- proces działający w systemie wieloprocesorowym może uzyskać rzeczywistą równoległość wykonania, każdy proces lekki może być uruchamiany na oddzielnym procesorze
- wielowątkowe procesy są przydatne wtedy, gdy każdy wątek jest w miarę niezależny i rzadko porozumiewa się z innymi wątkami

Procesy lekkie: Ograniczenia (1)

Ograniczenia procesów lekkich:

- większość operacji na nich (tworzenie, usuwanie, synchronizacja) wymaga użycia funkcji systemowych
- funkcje systemowe są kosztowne - każde wywołanie wymaga dwóch przełączeń trybu (przy wywołaniu i zakończeniu)
- każda zmiana trybu powoduje przejście przez granice ochrony - jądro musi przekopiować argumenty funkcji systemowej z przestrzeniu użytkownika do przestrzeni jądra i sprawdzić ich poprawność
- po zakończeniu wykonania funkcji systemowej jądro musi skopiować dane z powrotem do przestrzeni użytkownika

Procesy lekkie: Ograniczenia (2)

Ograniczenia procesów lekkich 2:

- jeśli proces lekki często odwołuje się do współdzielonych danych, narzut związany z synchronizacją może niweczyć wszelkie zyski wydajnościowe
- większość systemów wieloprocesorowych udostępnia mechanizmy blokad, które można nakładać na poziomie użytkownika na niezablokowany zasób
- jeśli wątek chce uzyskać dostęp do zasobu, który jest w danej chwili niedostępny, to może:
  - aktywnie czekać (jeśli zasób jest zajęty na krótko) - co się odbywa bez udziału jądra
  - wstrzymać swoje działanie (w innych przypadkach) – wymaga zaangażowania jądra i jest operacją kosztowną czasowo
- każdy proces lekki zużywa znaczące zasoby jądra, w tym pamięć fizyczną potrzebną na stos jądra
- system nie może wspierać dużej liczby procesów lekkich
- procesy lekkie nie nadają się do programów wymagających dużej ilości wątków, które są często tworzone i niszczone
- użytkownik może zmonopolizować procesor, tworząc dużo procesów lekkich

Modele wielowątkowości

Szeregowanie wątków

Mogą istnieć dwaj planiści*:*

- jeden dla procesów/wątków systemowych - KS (ang. *kernel scheduler*) ,
- drugi dla wątków użytkownika - UTS (ang. *user threads scheduler*).

Modele wielowątkowości - 1:N

- KS widzi 1 byt/proces
- UTS widzi *N* wątków.
- kernel nie ma pojęcia o istnieniu wątków w aplikacji, natomiast zarządzanie wątkami opiera się o biblioteki działające na poziomie użytkownika
- *KS i UTS* nie muszą się ze sobą komunikować

Modele wielowątkowości - 1 : 1

- wszystkie wątki z punktu widzenia KS są bytami podlegającymi szeregowaniu;
- UTS nie jest potrzebny, gdyż cała praca jest wykonywana przez KS.
- konieczna jest organizacja obsługi wątków za pomocą odpowiedniej biblioteki (tworzenie, synchronizacja, itp.), ale planowanie zadań jest wykonywane tylko i wyłącznie na poziomie jądra

Modele wielowątkowości M : N

- dla jednej aplikacji wielowątkowej KS widzi *M* bytów do zarządzania,
- na poziomie użytkownika (i UTS) istnieje *N* wątków (*M* < *N*)
- koncepcja teoretycznie najbardziej wszechstronna i daje największe możliwości
- najtrudniejsza implementacyjnie, wymagana jest komunikacji pomiędzy UTS a KS, aby przemapować M wątków użytkownika na N wątków jądra

Planowanie przydziału procesora. Algorytmy planowania

Planowanie wywłaszczające

Decyzje o przydziale procesora zapadają w następujących 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 czekania, do stanu gotowości (np. po zakończeniu operacji we/wy)
4. Proces kończy działanie

Planowanie niewywłaszczające i wywłaszczające

- **Planowanie niewywłaszczające** (ang, non preemptive) - planowanie uwzględniające tylko warunki 1 i 4.
  - proces, który dostał procesor, nie odda go aż do swego zakończenia lub przejścia w stan czekania
  - stosowane np. w Microsoft Windows (16 bitowych)
  - nie wymaga dodatkowego wsparcia sprzętowego (np. zegara)

- **Planowanie wywłaszczające** (ang. preemptive) - uwzględnia poza 1 i 4 także 2 i 3
  - kosztowniejsze - wymaga mechanizmów koordynacji

Ekspedytor (ang. dispatcher)

- **Ekspedytor -** przekazuje procesor do dyspozycji procesu wybranego przez planistę krótkoterminowego.

- Zadania ekspedytora:
  - przełączanie kontekstu
  - przełączanie do trybu użytkownika

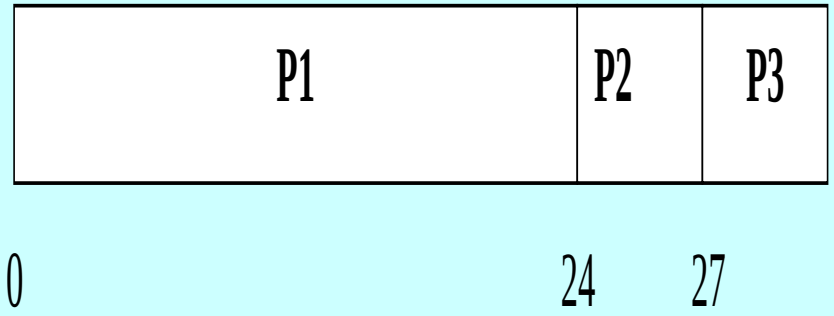
- wykonanie skoku do odpowiedniej komórki w programie użytkownika, w celu wznowienia działania programu
- opóźnienie ekspedycji (ang. *dispatch latency*) - czas, jaki zajmuje ekspedytorowi wstrzymanie jednego procesu i uaktywnienie innego

Kryteria planowania

- **Wykorzystanie procesora -** procesor powinien być zajęty pracą
- **Przepustowość (ang.** *throughput***) -** liczba procesów kończonych w jednostce czasu
- **Czas cyklu przetwarzania (ang.** *turnaround time***) -** czas upływający między nadejściem procesu do systemu i zakończeniem przetwarzania procesu;
  suma czasów czekania na wejście do pamięci, czekania w kolejce procesów gotowych do wykonania, wykonania procesu przez procesor i wykonywania operacji we/wy
- **Czas oczekiwania -** suma okresów oczekiwania przez proces w kolejce procesów gotowych do wykonania (algorytm planowania ma wpływ na tę wielkość)
- **Czas odpowiedzi** (ang. *response time*) - czas upływający między wysłaniem żądania i pierwszą odpowiedzią

Algorytmy planowania. Pierwszy zgłoszony-pierwszy obsłużony (ang. *first come- first served, FCFS*) (1)

- proces, który pierwszy zamawia procesor, pierwszy go otrzymuje
- wada: średni czas oczekiwania bywa b. długi
- Przykład:
- Proces Czas trwania fazy
- P1 24
- P2 3
- P3 3



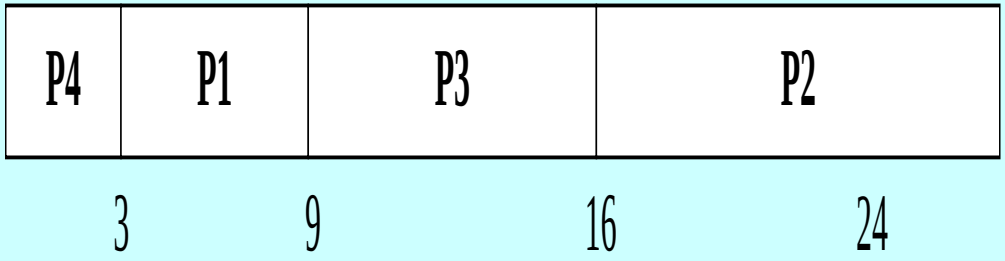
Algorytmy planowania. Pierwszy zgłoszony-pierwszy obsłużony (ang. *first come- first served, FCFS*) (2)

- "nieszczęśliwa konfiguracja" - krótkie procesy muszą czekać na wykonanie długiego:
- Średni czas oczekiwania = (0+24+27) /3 = 17ms
- Dla "szczęśliwej konfiguracji" (odwrotna kolejność)
- Średni czas oczekiwania = (6+3+0) /3 = 3ms
- **Efekt konwoju (ang. convoy effect) -** wszystkie procesy czekają na zwolnienie procesora przez jeden wielki proces
- FCFS jest niewywłaszczający

Algorytmy planowania. ***"Najpierw najkrótsze zadanie"*** **(ang.** *shortest job first* **, *SJF*)**

- **z każdym procesem jest związana długość jego najbliższej fazy procesora**
- **wolny procesor zostaje przydzielony procesowi mającemu najkrótszą wstępną fazę**
- **jeżeli 2 procesy mają następne fazy procesora równej długości, wtedy stosuje się algorytm FCFS**
- Proces Czas trwania fazy
- P1 6
- P2 8
- P3 7
- P4 3
- **Średni czas oczekiwania = (3+16+ 9+0)/4 = 7ms**
- **Średni czas oczekiwania dla FCFS - 10.25ms**

Diagram Gantta:



Algorytmy planowania. "Najpierw najkrótsze zadanie" (ang. *shortest job first* , ***SJF***) (2)

- **Algorytm SJF jest optymalny -** daje minimalny średni czas oczekiwania dla danego zbioru procesów
- **Trudność w algorytmie:** określenie długości następnego zamówienia na przydział procesora
- w planowaniu długoterminowym może być to limit czasu procesu
- można szacować długość zamówienia
- Algorytm SJF może być **wywłaszczający** (gdy w kolejce pojawia się proces o krótszej następnej fazie procesora od tej, która została aktualnemu procesowi wykonywanemu), lub **niewywłaszczający**

Algorytmy planowania. "Najpierw najkrótsze zadanie"

- realizacja przybliżona

- **Szacowanie długości następnej fazy procesora:**
- można przyjąć, że jej długość jest podobna do długości poprzednich faz procesora danego procesu:
- **Następna faza procesora:** średnia wykładnicza pomiarów długości poprzednich faz procesora

$\tau_{n+1} = \alpha t_n + (1-\alpha)\ \tau_n$ (*)

t_n - długość n-tej fazy procesora

τ_{n+1} -przewidywana długość następnej fazy procesora

τ_n - zawiera dane o minionej historii

α - względna waga między niedawną i wcześniejszą historią

$0 \leq \alpha \leq 1$,

$\alpha = 1$ - uwzględnia się tylko ostatnie notowania wartości fazy (starsza historia jest pomijana),

$\alpha = 0$ - niedawna historia nie ma wpływu na wynik (ostatnie wyniki miały charakter przejściowy),

najczęściej przyjmuje się $\alpha = 0.5$ (jednakowa waga obu członów)

wartość początkowa $\tau 0$ - stała lub średnia opisująca cały system

Rozwinięcie wzoru **(*)**

$\tau_{n+1} = \alpha t_n + (1-\alpha)\ \alpha\ t_{n-1} + \ldots + (1-\alpha)^j\ \alpha\ t_{n-j} + \ldots + (1-\alpha)^n + 1\ \tau_0$

Planowanie priorytetowe

- każdemu procesowi przypisuje się priorytet
- procesor przydziela się temu procesowi, którego priorytet jest najwyższy
- w razie równych priorytetów - alg. FCFS

Proces	Czas trwania fazy	Priorytet
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

- Kolejność:
  P2 P5 P1 P3 P4
- Średni czas oczekiwania: 8.2 ms

Planowanie priorytetowe (2)

- Sposób definicji priorytetu:
  - wewnętrzny- używa się mierzalnej właściwości procesu (np. limity czasu, wielkość obszaru wymaganej pamięci, liczba otwartych plików, stosunek średniej fazy we/wy do średniej fazy procesora)
  - zewnętrzny- kryteria zewnętrzne wobec SO (np. ważność procesu, rodzaj i kwota opłat użytkownika, instytucja sponsorująca pracę itd.)

- Planowanie priorytetowe może być wywłaszczające lub niewywłaszczające
- Problem nieskończonego blokowania (ang. *indefinite blocking*), głodzenia (ang, *starvation*) - procesy niskopriorytetowe czekają w nieskończoność na procesor
- Rozwiązanie: postarzanie procesów niskopriorytetowych (ang. *aging*) - podwyższanie priorytetów długo oczekujących procesów

Planowanie rotacyjne (ang. round robin, RR)

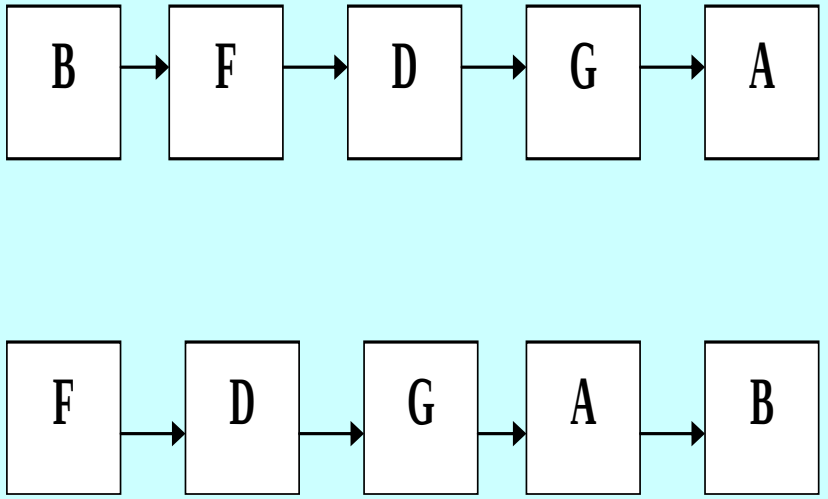
- podobny do FCFS, dodano wywłaszczanie
- ustala się małą jednostkę czasu - kwant czasu (ang. *time quantum*) ; 10-100 ms

- kolejka procesów gotowych do wykonania - kolejka cykliczna
- każdy proces dostaje procesor na odcinek czasu, nie dłuższy od kwantu czasu, kiedy jest generowane przerwanie od zegara i proces jest usuwany na koniec kolejki procesów gotowych
- planista pobiera procesy w kolejności przyjścia FCFS

- Średni czas oczekiwania: dość wysoki

Planowanie rotacyjne 2

- Wydajność algorytmu mocno zależy od rozmiaru kwantu czasu
- Wpływ przełączania kontekstów na szybkość -> wskazane, by kwant czasu był długi w porównaniu z czasem przełączania kontekstu
- Jeśli kwant jest zbyt długi, to algorytm działa jak FCFS, co może prowadzić do nieakceptowalnie długiego oczekiwania na reakcję systemu
- Można przyjąć, że 80% faz procesora powinno być krótszych niż 1 kwant czasu



Planowanie gwarantowane (*Guaranteed Scheduling*) (1)

- realne obietnice szybkości przetwarzania dla użytkowników:
  - jeśli jest zalogowanych n użytkowników, to każdy użytkownik otrzymuje 1/n czasu procesora

- System musi dysponować następującymi informacjami:
  - jak wiele czasu CPU użytkownik otrzymał dla wszystkich swoich procesów od czasu zalogowania się
  - jak długo każdy użytkownik jest zalogowany
  - czas procesora przysługujący użytkownikowi jest ilorazem:
  - czas zalogowania użytkownika/ilość zalogowanych użytkowników

Planowanie gwarantowane (2)

- Określenie czasu przypadającego na proces/użytkownika
  - wyliczenie stosunku dotąd przyznanego czasu CPU do czasu przysługującego
    - współczynnik 0.5 oznacza, że użytkownik wykorzystał tylko połowę przysługującego mu czasu
    - współczynnik 2.0 oznacza, że użytkownik otrzymał dwa razy więcej czasu, niż mu przysługuje
    - proces jest uruchamiany z najniższym priorytetem, gdy jego współczynnik jest wyższy od współczynników innych procesów
- Systemy czasu rzeczywistego:
  - może być zastosowana podobna idea
  - procesy których `deadline` się zbliża otrzymują priorytety gwarantujące im pierwszeństwo wykonania

Planowanie loteryjne (ang. Lottery Scheduling)

- daje niezłe wyniki, przy znacznie prostszej od algorytmu **Guaranteed Scheduling** realizacji
- każdy proces dostaje los do rożnych rodzajów zasobów (np. właśnie CPU)
- podczas procesu planowania wybierany jest (metoda losową) los (bilet loteryjny ang. lottery tickets) i proces, który jest jego właścicielem dostaje zasób
- bardziej istotne procesy mogą mieć przyznaną większą ilość losów, co zwiększa ich szanse zwycięstwa w losowaniu

Planowanie loteryjne (2)

- Właściwości planowania loteryjnego:
- planowanie loteryjne jest algorytmem bardzo czułym - jeżeli procesowi zwiększy się ilość losów, to jego szansa zwycięstwa już w następnym losowaniu zasobów wzrośnie
- współpracujące procesy mogą wymieniać losy między sobą np.
  - klient wysyła komunikat do procesu serwera i przechodzi w stan oczekiwania na wynik
  - może następnie przekazać swoje losy serwerowi co zwiększa szansę, że serwer zostanie szybko wykonany
  - po obsłużeniu zlecenia przez serwer, zwraca on losy klientowi,
  - klient zostaje uruchomiony
  - w przypadku braku klientów serwer nie potrzebuje losów

- Planowanie loteryjne - może być używane do rozwiązywania problemów, które trudno rozwiązać innymi metodami

Wielopoziomowe planowanie kolejek

- procesy są zaliczane do kilku grup np. procesy pierwszoplanowe (ang. foreground)- interakcyjne, oraz drugoplanowe (ang. background) - wsadowe.
- kolejka procesów gotowych jest rozdzielona na osobne kolejki
- w zależności od swych cech, procesy zostają na stałe przypisane do poszczególnych kolejek
- każda kolejka ma własny algorytm planujący (np. procesy pierwszoplanowe - alg. planowania rotacyjnego, procesy drugoplanowe - algorytm FCFS)
- konieczny jest algorytm planowania między kolejkami (np. stałopriorytetowe planowanie z wywłaszczaniem)
- inna możliwość - przyznanie każdej kolejce pewnego % czasu procesor

Planowanie wielopoziomowych kolejek ze sprzężeniem zwrotnym

- umożliwia przemieszczanie się procesów między kolejkami
- idea: rozdzielenie procesów o różnych rodzajach faz procesora
- proces, który zużywa za dużo czasu procesora, zostaje przeniesiony do kolejki o niższym priorytecie
- np., 3 kolejki, im zadanie ~w kolejce o niższym priorytecie, tym na dłużej może uzyskać procesor:
  - 1 - algorytm z kwantem 8
  - 2 - algorytm z kwantem 16
  - 3 - algorytm FCFS
- Parametry określające planistę wielopoziomowych kolejek ze sprzężeniem zwrotnym:
  - liczba kolejek
  - algorytm planowania dla każdej kolejki
  - metoda użyta do decydowania o awansie procesu do kolejki o wyższym priorytecie
  - metoda użyta do zdymisjonowania procesu do kolejki o niższym priorytecie
  - metoda wyznaczania kolejki, do której trafia proces

Planowanie: Polityka a mechanizmy

- generalnie przyjmuje się, że wszystkie procesy w systemie należą do różnych użytkowników i dlatego konkurują w dostępie do CPU

- w rzeczywistości zdarza się jednak, że jeden użytkownik uruchamia wiele procesów, które mogą współpracować ze sobą np.
  - proces uruchamia wielu potomków, którzy działają na jego rzecz np. system zarządzania bazą danych
  - każdy potomek pracuje nad innym zleceniem
  - proces macierzysty posiada wiedzę, na temat wagi zadań potomków, jednak generalnie algorytmy planowania nie wykorzystują tych związków między procesami
  - w konsekwencji planista rzadko podejmuje optymalne decyzje

- **rozwiązanie: separacja mechanizmów planowania i polityki planowania**
  - algorytmy planowania mogą być parametryzowane na różne sposoby, a parametry te są dobierane przez procesy użytkownika np.
  - przodek dysponuje funkcją systemową, która może zmieniać priorytety swoich potomków

Planowanie wieloprocesorowe

- Rozpatrywany przypadek: procesory identyczne (homogeniczne) pod względem wykonywanych funkcji

- Rozwiązania
  - aby uniknąć bezczynności procesorów przyjmuje się, że istnieje jedna wspólna kolejka procesów gotowych dla wszystkich procesorów
  - lub – aby zapobiec powstaniu wąskiego gardła (szczególnie istotne dla dużej liczby procesorów) każdy procesor ma własną kolejkę

Planowanie wieloprocesorowe 2

Dwie metody planowania:

- **każdy procesor sam planuje swoje działanie**
  - każdy procesor przegląda kolejkę procesów gotowych i wybiera proces do wykonania
  - konieczność właściwej realizacji obsługi kolejki procesów gotowych (np. nie dopuścić, by dwa procesory wybrały jednocześnie ten sam proces)
- **asymetryczne wieloprzetwarzanie (ang. asymmetric multiprocessing) wyróżniony jeden procesor - serwer główny**
  - serwer główny podejmuje wszystkie decyzje planistyczne, wykonuje operacje we/wy i czynności systemowe
  - pozostałe procesory wykonują tylko kod użytkowy

Planowanie wieloprocesorowe – Planowanie procesów

- Dobór konkretnego algorytmu planowania (RR, FCFS,...) traci na znaczeniu wraz ze wzrostem ilości procesorów

- W przypadku systemu z dużą ilością procesorów użycie FCFS lub priorytetowego (ze statycznymi priorytetami) może być wystarczające

Planowanie wieloprocesorowe – Planowanie wątków

- Różnica w porównaniu z planowaniem procesów:
  - Dla realizacji jednoprocesorowych: wątki ułatwiają strukturalizację programu oraz umożliwiają wykonywanie procesu nawet wtedy, gdy jakiś jego wątek czeka na operację we/wy
  - Dla realizacji wieloprocesorowych: Wątki wchodzące w skład jednego procesu zwykle blisko współpracują ze sobą, a zatem ich równoległe wykonywanie może znacząco zwiększać szybkość
    - Wątki nie muszą być blokowane w oczekiwaniu na wykonanie akcji przez inne, co powoduje wykonanie kosztowych funkcji systemowych oraz zmianę kontekstu wykonania na procesorach
    - Dla aplikacji wymagających znacznych interakcji między wątkami, niewielkie zmiany w zarządzaniu i szeregowaniu wątków mogą znacząco zmienić wydajność aplikacji

Planowanie wieloprocesorowe – Planowanie wątków (2)

- Podstawowe podejścia stosowane przy wieloprocesorowym planowaniu wątków:
  - Współdzielenie obciążenia (**Load Sharing**) – stosowana jest globalna kolejka gotowych do wykonania wątków, procesor gdy jest wolny wybiera wątek z kolejki
  - Szeregowanie grupowe (zespołowe) (**Gang scheduling**) – zbiór wątków tego samego procesu jest jednocześnie wykonywany na wielu procesorach
  - Rezerwacja procesorów (**Dedicated processor assignment**) – przydział wątków do wykonania na konkretnych procesorach
  - Dynamiczne szeregowanie (**Dynamic scheduling**) – ilość wątków w procesach może być zmieniana w trakcie wykonania

Planowanie wieloprocesorowe – Współdzielenie obciążenia

- Proste podejście, inspirowane w dużym stopniu środowiskami jednoprocesorowymi
- Zalety:
  - Obciążenie rozproszone równomiernie między procesory, gwarantuje, że żaden procesor nie pozostaje wolny, gdy jest praca do wykonania
  - Nie jest wymagany centralny planista, gdy procesor jest dostępny, procedura szeregująca systemu operacyjnego jest na nim wykonywana i dokonuje wyboru wątku
  - Globalna kolejka może być obsługiwana przy użyciu dowolnego algorytmu przedstawionego dla podejść jednoprocesorowych
- Wersje współdzielenia obciążenia
  - FCFS – kiedy zadanie przybywa, każdy z jego wątków jest kolejno umieszczany na końcu współdzielonej kolejki. Kiedy procesor jest bezczynny, wybiera on kolejny wątek, który wykonuje do chwili zakończenia lub zablokowania
  - Najpierw najmniejsza liczba wątków – kolejka gotowych wątków jest organizowana na zasadzie priorytetowej, najwyższy priorytet otrzymują wątki zadań o najmniejszej liczbie gotowych do wykonania wątków, zadania o jednakowym priorytecie są szeregowane według kolejności przybycia, przydzielony wątek jest wykonywany do chwili zakończenia lub zablokowania
  - Najpierw najmniejsza liczba wątków z wywłaszczeniem -- jak dla poprzedniego przypadku, ale przybywający proces z mniejszą ilością gotowych do wykonania wątków, niż aktualnie wykonywany, wywłaszcza wykonujące się wątki
    W [Stallings] stwierdzono, że przeprowadzone badania symulacyjne wykazały, że FCFS jest lepszy od pozostałych polityk, ale generalnie gorszy od szeregowania grupowego

Planowanie wieloprocesorowe – Współdzielenie obciążenia (2)

- Wady współdzielenia obciążenia:
  - Dostęp do centralnej kolejki należy uzyskiwać stosując wzajemne wykluczenie. Dla wielu procesorów (dziesiątek, setek) kolejka może się stać wąskim gardłem.
  - Wywłaszczane wątki rzadko wznawiają wykonanie na tym samym procesorze. Jeżeli każdy procesor jest wyposażony w lokalną pamięć podręczną, buforowanie staje się mniej efektywne
  - Jeżeli wszystkie wątki traktuje się jak wspólną pulę, mała jest szansa, że wszystkie wątki danego procesu uzyskają jednocześnie dostęp do procesorów, co przy silnych związkach koordynacyjnych między nimi może znacząco obniżyć wydajność
- Pomimo potencjalnych wad, współdzielenie obciążenia jest jedną z najczęściej stosowanych metod planowania we współczesnych systemach wieloprocesorowych

Planowanie wieloprocesorowe – Szeregowanie grupowe

- Jednoczesne szeregowanie wątków tworzących jeden proces
- Bardzo korzystne w przypadku aplikacji równoległych, których wydajność znacznie spada, gdy część aplikacji nie działa razem z innymi
- Przydatne także w przypadku aplikacji mniej wrażliwych na kwestie wydajnościowe
- Szeregowanie grupowe minimalizuje przełączenia procesów (wątki nie muszą oddawać procesora w oczekiwaniu na działania np. zwolnienia muteksów, wysłanie danych ze strony innych wątków)
- Sposoby przydziału procesorów:
  - Zakładamy, że mamy N procesorów i M aplikacji, z których każda składa się z co najwyżej N wątków
  - (1) Każdej aplikacji można przydzielić 1/M czasu na N procesorach
    Taka strategia może być nieefektywna, jeśli mamy różne ilości wątków w procesach, wtedy dobrze by było uruchomić na wolnych procesorach aplikacje o mniejszych ilościach wątków (np. jednowątkowe)
  - (2) Można dzielić czas używając szeregowania ważonego według liczby wątków np. proces i z M_i wątków może uzyskać $M_i / \sum_{j=1...M} M_j$

Planowanie wieloprocesorowe – Rezerwacja procesorów

- Skrajna forma szeregowania grupowego polegająca na zarezerwowaniu grupy procesorów na wyłączny użytek aplikacji przez czas jej trwania
- Gdy aplikacja zostaje skierowana do wykonania, każdemu z jej wątków przydziela się procesor, który pozostaje dla niego zarezerwowany do chwili ukończenia aplikacji
- Ocena
  - Wada:
    - Podejście wydaje się być bardzo rozrzutnym z punktu widzenia użytkowania procesorów – nie ma wieloprogramowania procesorów, wątek zablokowany na we/wy nadal zajmuje procesor
  - Argumenty „za”:
    - (1) W systemie z wysokim stopniem paralelizmu z dziesiątkami lub setkami procesów, z których każdy stanowi niewielki koszt systemu, wykorzystanie procesora nie jest tak kluczowe z punktu widzenia efektywności czy wydajności
    - (2) Całkowity brak przełączania procesów podczas życia danego procesu powinien doprowadzić do jego znacznie szybszego wykonania
      Przeprowadzone eksperymenty dowiodły słuszność stwierdzenia (2) → efektywną strategią jest ograniczenie liczby aktywnych wątków do liczby procesorów w systemie
- Analogie przydziału procesorów do przydziału stron. Określenie rozmiaru „zbioru roboczego aktywności” (activity working set) na oznaczenie minimalnej liczby wątków, których potrzebuje aplikacja, by czynić postępy

Planowanie wieloprocesorowe –

Dynamiczne szeregowanie (1)

- Często ilość wątków w procesie może się zmieniać
- System operacyjny mógłby regulować ilość wątków, a w konsekwencji obciążenie systemu, aby zoptymalizować stopień wykorzystania procesora
- Propozycja: zarówno system operacyjny jak i aplikacja uczestniczą w podejmowaniu decyzji o szeregowaniu
  - System operacyjny odpowiada za rozdział procesorów między procesy
  - Każdy proces używa przydzielonych mu procesorów, aby wykonać pewien zbiór czynności mapując je na wątki (decyzje, które wątki zawiesić, a które wykonać pozostają w gestii zależą od aplikacji, wspomaganej przez zbiory procedur bibliotecznych)

Planowanie wieloprocesorowe – Dynamiczne szeregowanie (2)

- Zadania systemu operacyjnego ograniczają się do przydziału procesorów procesom i postępowania jak powyżej:
  1. Jeżeli są bezczynne procesory – użyć ich do spełnienia zadania
  2. W przeciwnym razie, jeżeli żądanie zostało zgłoszone przez nowo przybyłe zadanie, przydzielić mu jeden procesor, zabierając go zadaniu, które obecnie dysponuje więcej niż jednym procesorem
  3. Jeśli nie można spełnić dowolnej części żądania, żądanie czeka, aż zwolni się dla niego procesor albo zadanie wycofa żądanie
  4. Po zwolnieniu jednego lub kilku procesorów (lub zakończeniu zadania) :
     - Przeskanować kolejkę niezrealizowanych żądań przydziału procesora. Przydzielić procesor każdemu zadaniu na liście, które obecnie nie ma procesora (tzn. wszystkim oczekującym, nowo przybyłym zadaniom),
     - Ponownie przeskanować listę przydzielając resztę procesorów na zasadzie – kto pierwszy ten lepszy)
- Przeprowadzone analizy sugerują zalety metody (lepsza niż szeregowanie zespołowe i rezerwacja procesorów) w przypadku aplikacji umożliwiających wykorzystanie szeregowania dynamicznego, koszty dodatkowe mogą jednak zanegować te korzyści

Planowanie w czasie rzeczywistym 1

- **Planowanie w rygorystycznych systemach czasu rzeczywistego**
  - proces jest dostarczany wraz z przepisem określającym wymaganą ilość czasu do jego zakończenia (lub wykonania operacji we/wy)
  - rezerwacja zasobów (ang. resource reservation) - planista akceptuje proces i zapewnia jego wykonanie na czas lub odrzuca zlecenie jako niewykonalne
  - planista musi mieć dokładne rozeznanie, ile czasu zużywają poszczególne funkcje systemu operacyjnego - dla każdej operacji systemowej musi być przypisany maksymalny czas jej wykonania (taka gwarancja - niemożliwa w systemach z pamięcią wirtualną lub pomocniczą)

Planowanie w czasie rzeczywistym 2

- **Planowanie w łagodnych systemach czasu rzeczywistego**
  - procesy o decydującym znaczeniu muszą mieć wyższy priorytet od innych procesów
  - dodanie do systemu takich uprzywilejowanych procesów może powodować niesprawiedliwy podział zasobów, większe opóźnienia innych procesów i ich głodzenie

- **Realizacja planisty:**
  - system musi mieć planowanie priorytetowe,
  - procesy działające w czasie rzeczywistym muszą mieć najwyższy priorytet
  - priorytety procesów czasu rzeczywistego nie mogą maleć z upływem czasu
  - opóźnienie ekspediowania procesów do procesora musi być małe - istotnym problemem jest to, że w większości systemów Unix przed przełączeniem kontekstu należy poczekać do zakończenia wykonania funkcji systemowej lub zablokowania procesu w oczekiwaniu na operację we/wy

Planowanie w czasie rzeczywistym 3

- zezwolenie na wywłaszczanie funkcji systemowych
- wstawienie do długotrwałych funkcji systemowych tzw. punktów wywłaszczeń (ang. preemption points), gdzie sprawdza się. czy nie ma wysokopriorytetowych procesów wymagających uaktywnienia
- punkty wywłaszczeń - tylko w "bezpiecznych miejscach" - gdzie nie są zmieniane struktury danych jądra
- wywłaszczanie całego jądra - struktury danych jądra muszą być chronione przez specjalne mechanizmy synchronizacyjne

- **odwracanie priorytetów** (ang. *priority inversion*)
  - wysokopriorytetowy proces, chcący odwołać się do danych jądra. które są używane przez proces niskopriorytetowy musiałby czekać na ich zwolnienie
  - stosowanie protokołu dziedziczenia priorytetów (ang. *priority-inheritance protocol)* - wszystkie procesy dziedziczą wysoki protokół do czasu zakończenia korzystania przez nie z zasobów na które czeka proces wysokopriorytetowy

Proste algorytmy szeregowania – zestawienie (1)

- FCFS – First Come First Served
- RR – Round Robin
- SPN – Shortest Process Next (SJF)
- SRT – Shortest Remaining Time (wywłaszczalna wersja SJF)
- HRRT – Highest Response Ratio Next
- Feedback - karanie procesów, które się długo wykonywały

Proste algorytmy szeregowania – zestawienie (2)

	FCFS	Round robin	SPN	SRT	HRRN	Feedback
Selection function	max[w]	constant	min[s]	min[s − e]	$\max\left(\frac{w+s}{s}\right)$	(see text)
Decision mode	Non-preemptive	Preemptive (at time quantum)	Non-preemptive	Preemptive (at arrival)	Non-preemptive	Preemptive (at time quantum)
Throughput	Not emphasized	May be low if quantum is too small	High	High	High	Not emphasized
Response time	May be high, especially if there is a large variance in process execution times	Provides good response time for short processes	Provides good response time for short processes	Provides good response time	Provides good response time	Not emphasized
Overhead	Minimum	Minimum	Can be high	Can be high	Can be high	Can be high
Effect on processes	Penalizes short processes; penalizes I/O bound processes	Fair treatment	Penalizes long processes	Penalizes long processes	Good balance	May favor I/O bound processes
Starvation	No	No	Possible	Possible	No	Possible

Funkcja wyboru procesu

- Określa, jaki proces jest wybrany do wykonania
- Jeśli jest oparta na charakterystykach wykonania, rozważane są następujące parametry:
  - ***w*** = dotychczasowy czas spędzony w systemie
  - ***e*** = dotychczasowy czas wykonania
  - ***s*** = całkowity wymagany czas wykonania, obejmujący *e*;

Ocena algorytmów planowania - wprowadzenie

- problem zdefiniowania kryterium wyboru algorytmu np.
  - maksymalizacja wykorzystania procesora przy założeniu, że maksymalny czas odpowiedzi wyniesie 1s,
  - maksymalizacja przepustowości tak, by czas cyklu przetwarzania był średnio liniowo proporcjonalny do ogólnego czasu wykonania

Ocena algorytmów planowania - modelowanie deterministyczne

- przyjmuje się konkretne obciążenie systemu i definiuje się zachowanie każdego algorytmu w warunkach tego obciążenia
- proste i szybkie
- daje wyniki pozwalające na porównanie algorytmów
- wymaga dokładnych liczb na wejściu, a odpowiedzi odnoszą się tylko do zadanych przypadków

Ocena algorytmów planowania - modele obsługi kolejek

- procesy w systemie zmieniają się
- badania dotyczą ustalenia rozkładów faz procesora i we/wy, ich mierzenia i szacowania, co w rezultacie może dać wzór opisujący prawdopodobieństwo wystąpienia poszczególnych faz procesora (zwykle - rozkład wykładniczy) (*)
- inne badania - ustalenie rozkładu czasów przybywania procesów do systemu (**)
- na podstawie (*) i (**) można obliczyć średnią przepustowość, wykorzystanie procesora, czas oczekiwania itd.
- system komputerowy: sieć usługodawców (procesorów i urządzeń we/wy), z których każdy posiada własną kolejkę czekających procesów (analiza obsługi kolejek w sieciach, ang. queuing-network analysis)

- **Wzór Little'a**

  $n = \lambda \times W$

  n - średnia długość kolejki

  λ **-** tempo przybywania nowych procesów do kolejki (ilość/s)

  W - średni czas oczekiwania w kolejce

  trudność matematycznego opracowania algorytmów

Ocena algorytmów planowania - symulacje

- symulacje w oparciu o implementacje modelu systemu komputerowego
- dane do sterowania symulacją można uzyskać poprzez:
- generator liczb losowych do tworzenia procesów, określania czasów trwania faz procesora itd.
- rozkłady można definiować matematycznie lub doświadczalnie (pomiary w badanym systemie)
- zastosowanie taśm śladów (ang. trace tapes), które powstają w wyniku nadzorowania rzeczywistego systemu i zapisywania rzeczywistej kolejności zdarzeń

Ocena algorytmów planowania - implementacja

- ostateczna weryfikacja - implementacja algorytmu, zastosowanie go w rzeczywistym systemie i analiza uzyskanych wyników

- trudność - koszt (zakodowanie algorytmu, włączenie do systemu, reakcja użytkowników), zmiana środowiska (nowe programy, zmiana zachowań użytkowników)