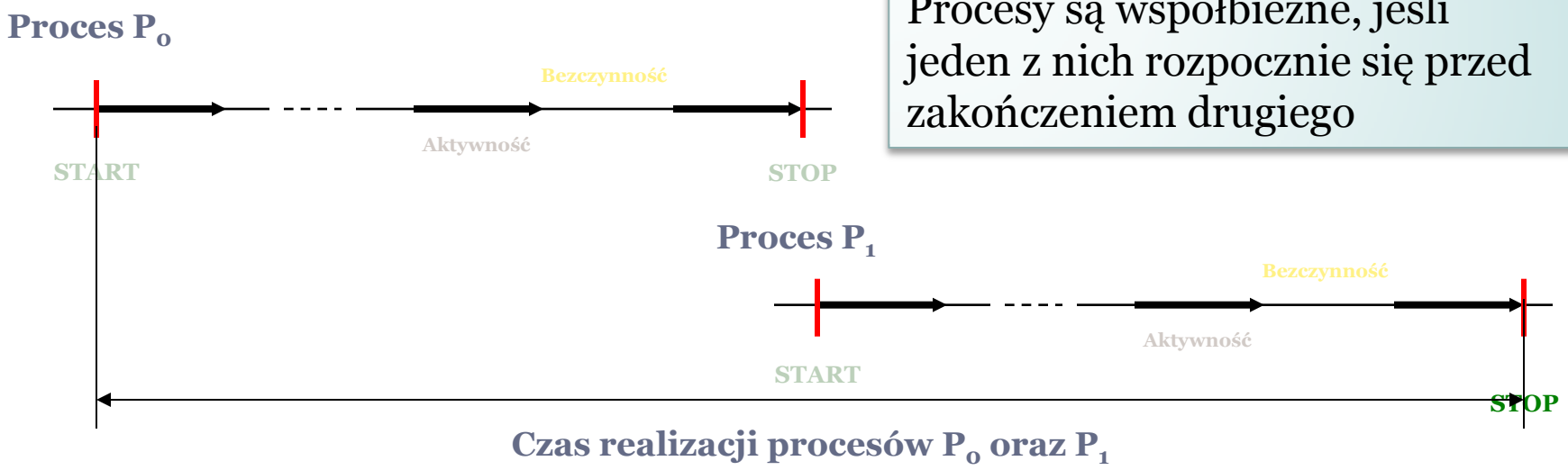
Podstawy systemów operacyjnych

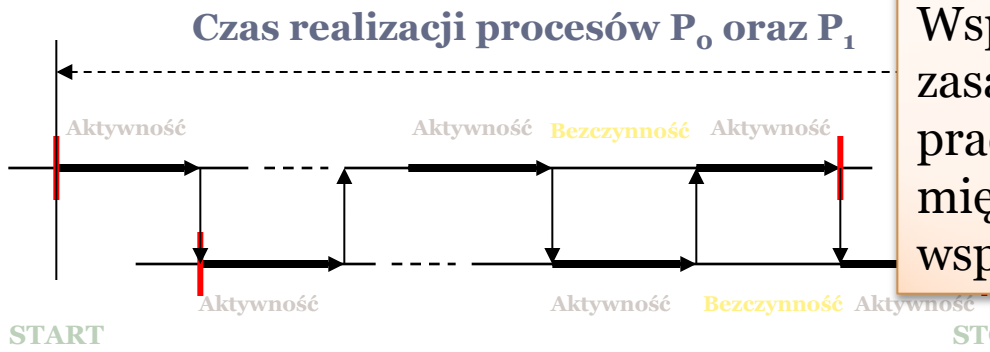
Wykład 2

Praca wieloprogramowa w systemach jednoprocesorowych

Procesy są współbieżne, jeśli jeden z nich rozpocznie się przed zakończeniem drugiego



Współbieżność uzyskuje się dzięki zasadzie podziału czasu – czas pracy procesora jest dzielony między wszystkie wykonywane współbieżnie procesy.



Kolejki procesów

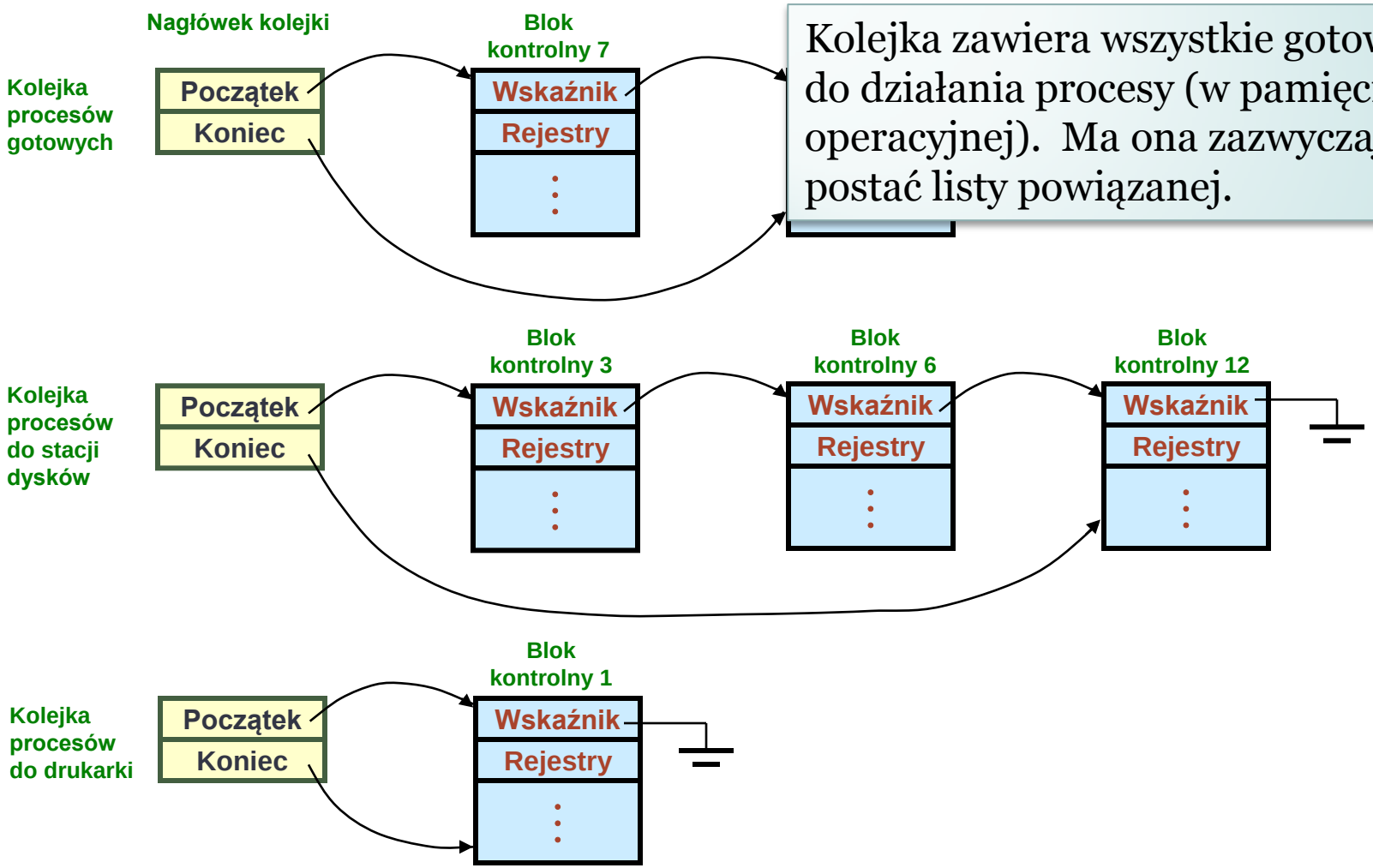
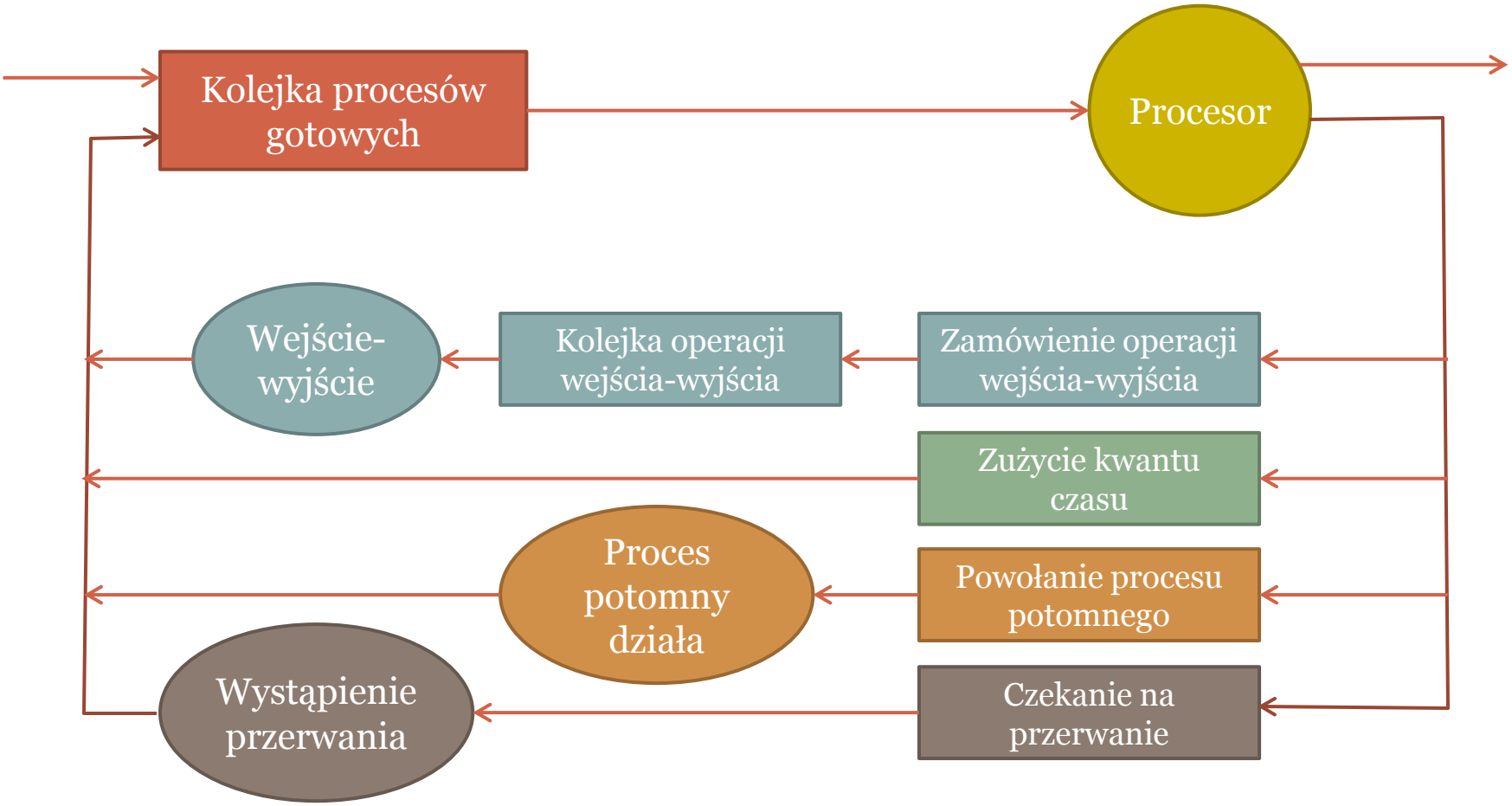


Diagram kolejek w planowaniu procesów

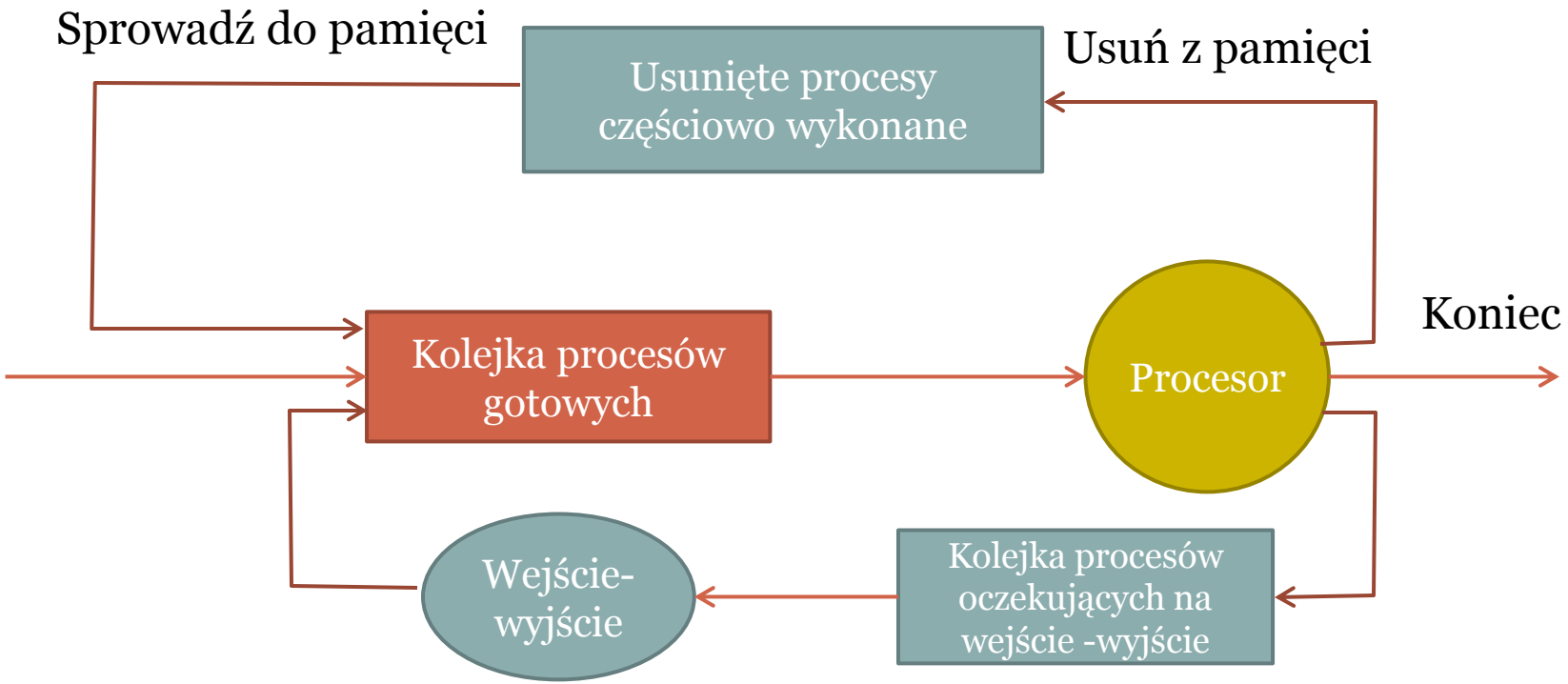


Planowanie zadań

Wyboru procesów z kolejek do przetwarzania dokonuje specjalny proces systemu operacyjnego nazywany planistą (Scheduler).

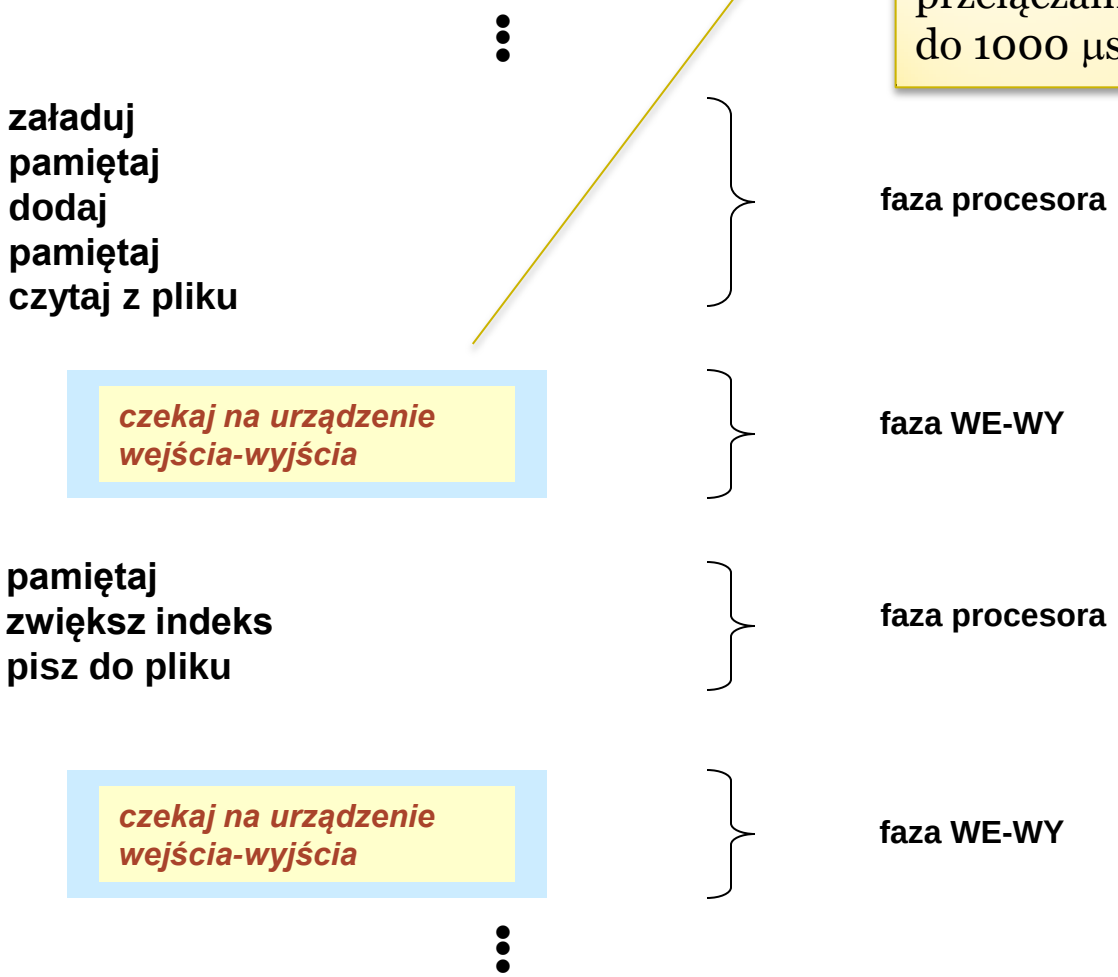
- Planista długoterminowy – wybiera procesy z pamięci masowej (np. z dysków) i umieszcza je w pamięci operacyjnej. Kolejny proces jest wybierany z pamięci masowej, wtedy gdy jakiś inny proces opuszcza system.
- Planista krótkoterminowy (planista przydziału procesora) – wybiera procesy z listy procesów gotowych i przydziela im procesor.
- Niekiedy (np. w systemach z podziałem czasu) wprowadza się planistę średnioterminowego (uwalnianie pamięci).

Planowanie średnioterminowe w diagramie kolejek



Przełączanie kontekstu

Przełączanie kontekstu – zmiana bloku kontrolnego procesu. Wartość czasu przełączania waha się od 1 do 1000 μs



Planista przydziału procesora

Planista przydziału procesora rozpoczyna pracę w momencie, gdy procesor przechodzi w stan bezczynności.

Planowanie krótkoterminowe:

1. proces przeszedł od stanu wykonywania do stanu czekania (np. z powodu oczekiwania na WE-WY);
2. proces przeszedł od stanu wykonywania do stanu gotowości (np. wskutek przerwania);
3. proces przeszedł ze stanu gotowości do stanu czekania (np. po zakończeniu operacji WE-WY);
4. proces zakończył działanie.

Algorytmy planowania

Istnieje wiele różnych algorytmów planowania przydziału procesora. Kryteria oceny algorytmów powinny uwzględniać następujące czynniki:

1. wykorzystanie procesora;
2. przepustowość, czyli liczbę procesów kończonych w jednostce czasu;
3. czas cyklu przetwarzania, czyli czas potrzebny na realizację procesu;
4. czas oczekiwania w kolejce procesów gotowych;
5. czas odpowiedzi, w systemach interakcyjnych lub systemach czasu rzeczywistego.

Algorytmy planowania

1. Algorytm FCFS **(First-Come First-Serve)** – modelowany kolejką FIFO. Nie zawsze efektywny (może dawać duże średnie czasy oczekiwania).
2. Algorytm SJF (**Shortest-Job-First**) – algorytm ten daje minimalny średni czas oczekiwania. Trudno oszacować czas realizacji procesu. Modyfikacja tego algorytmu to: algorytm *najpierw najkrótszy pozostały czas*.
3. Algorytm priorytetowy. Wymaga nadawania procesom priorytetów z góry (może to być robione wewnętrznie przez system operacyjny lub zewnętrznie na podstawie kryteriów ważnych np. z punktu widzenia Centrum Obliczeniowego).

Algorytmy planowania

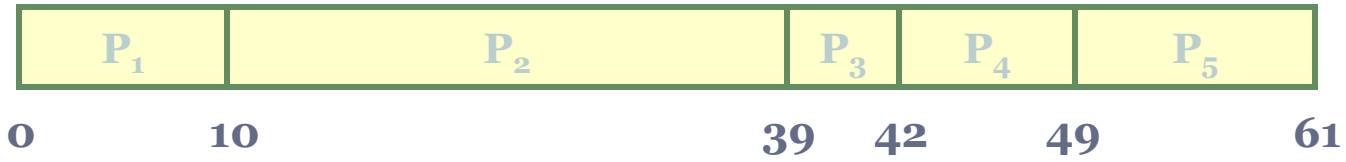
4. Algorytm rotacyjny (**Round-Robin**) – zaprojektowany dla systemów z podziałem czasu (*Time Shared*). Kolejka procesów jest cykliczna i każdemu procesowi przydziela się stały kwant czasu pracy procesora. Na parametry systemu istotny wpływ ma wielkość kwantu czasu.
5. Algorytm planowania z *kolejkami wielopoziomowymi*. Istnieje szereg kolejek dla różnych procesów, które mogą być obsługiwane różnymi algorytmami planowania.
6. Algorytm planowania z *kolejkami wielopoziomowymi oraz sprzężeniem zwrotnym*. Podobne jak 5, ale istnieje możliwość zmiany kolejki.

Średni czas oczekiwania

Niech będą dane procesy z czasami obliczeń równymi T_i:

P_1 – T_1 = 10 ms;
P_2 – T_2 = 29 ms;
P_3 – T_3 = 3 ms;
P_4 – T_4 = 7 ms;
P_5 – T_5 = 12 ms.

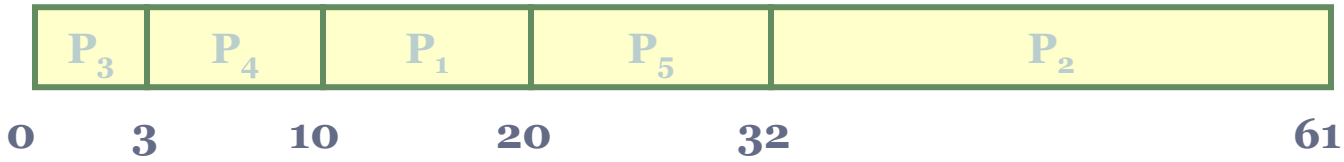
FCFS



Średni czas oczekiwania = (0 + 10 + 39 + 42 + 49) / 5 = 28 ms.

Średni czas oczekiwania

SJF



Średni czas oczekiwania = (0 + 3 + 10 + 20 + 32) / 5 = 13 ms.

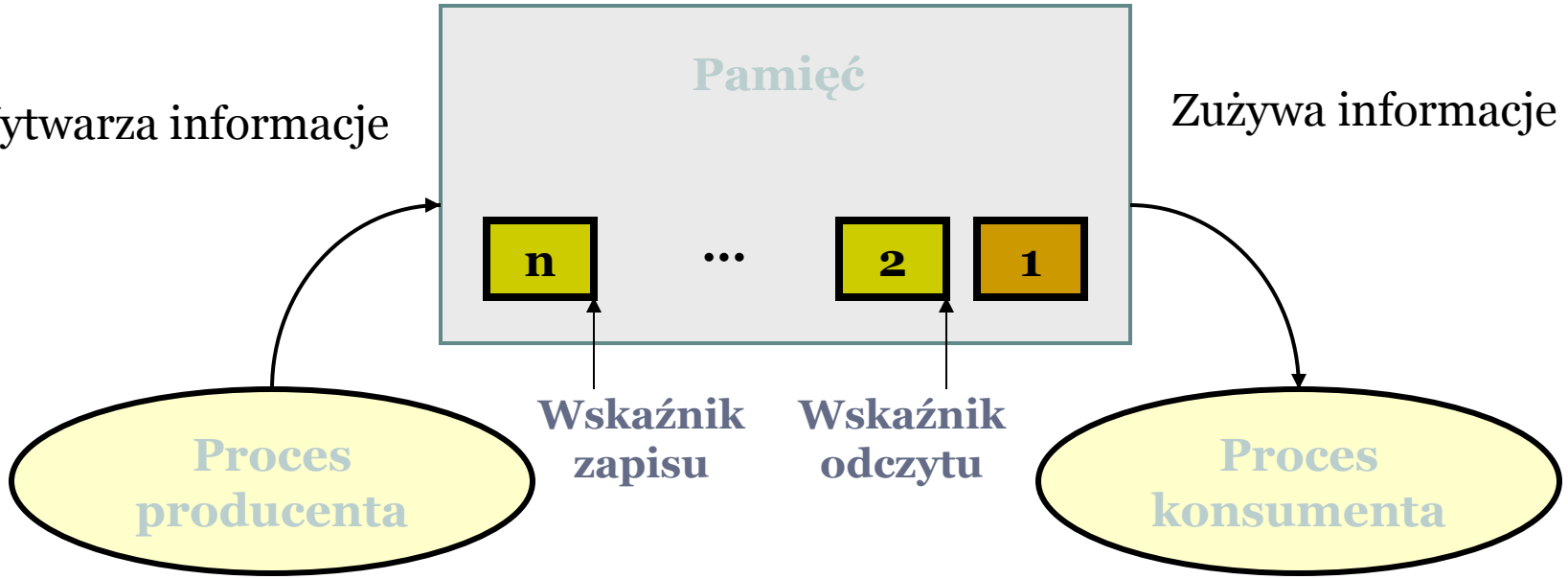
RR – z kwantem 10 ms

Średni czas oczekiwania = zależy od kwantu, ilości procesów i długości procesu =(0+10+20+23+30)/5 = 16.6ms.

Koordynacja procesów

Problem producenta-konsumenta z *nieograniczonym* buforem – może się zdarzać, że konsument musi czekać na nowe jednostki a producent je bezustannie produkuje .

Problem producenta-konsumenta z *ograniczonym* buforem – konsument musi czekać jeśli bufor jest pusty a producent musi czekać jeśli bufor jest pełny.



Bufor pamięci do przekazywania informacji jest buforem cyklicznym, który mogą modyfikować zarówno proces producenta jak i konsumenta.

Koordynacja procesów

Proces producenta

```
while (1)
{
    while (licznik == R_BUF);
                    /*nic nie rób*/
    bufor [we] = nastProd;
    we = (we + 1) % R_BUF;
    licznik ++;
}
```

Proces konsumenta

```
while (1)
{
    while (licznik == 0);
                    /*nic nie rób*/
    nastKons=bufor [wy];
    wy = (wy + 1) % R_BUF;
    licznik --;
    /* konsumuj jednostkę z
    nastKonsum*/
}
```

Sekcja krytyczna

- *Sekcja krytyczna* procesu jest segmentem kodu, w którym dokonuje się modyfikacji wspólnych zmiennych. Gdy jeden proces wykonuje kod sekcji krytycznej, wówczas żaden inny proces nie może realizować swojej sekcji krytycznej. Wykonywanie sekcji krytycznej przez procesy podlega wzajemnemu wyłączeniu w czasie.
- Każdy proces przed wejściem do sekcji krytycznej musi uzyskać zezwolenie na wejście do sekcji krytycznej (realizuje to *sekcja wejściowa*), zaś po wyjściu z sekcji krytycznej informuje o zwolnieniu zasobów (realizuje to *sekcja wyjściowa*).
- Pozostały kod programu nazywa się *resztą*.

Sekcja krytyczna

do{

sekcja wejściowa

sekcja krytyczna

sekcja wyjściowa

reszta

} while(1);

Problem sekcji krytycznej – polega na skonstruowaniu protokołu , który mógłby posłużyć do organizowania współpracy procesów.

Sekcja krytyczna

Rozwiązanie problemu sekcji krytycznej musi spełniać trzy warunki:

- wzajemne wykluczanie – jeśli proces P_i realizuje kod sekcji krytycznej, to żaden inny proces nie działa w sekcji krytycznej;
- postęp – jeśli żaden proces nie działa w sekcji krytycznej, to do swoich sekcji krytycznych mogą wejść tylko te procesy, które nie wykonują swoich reszt i wybór nie może być odwlekany w nieskończoność;
- ograniczone czekanie – musi istnieć wartość graniczna liczby wejść innych procesów do swoich sekcji krytycznych, po tym jak dany proces zgłosił chęć wejścia do sekcji krytycznej, co oznacza, że proces może oczekiwać na wejście do sekcji krytycznej skończony odcinek czasu.

Sekcja krytyczna – przykład

Wieloprogramowy system operacyjny, z jednym użytkownikiem, jednym procesorem i uruchomionymi 2 programami do wprowadzania tekstu:

```
void echo()
{
    chin = getchar();
    chout = chin;
    putchar(chout);
}
```

Sekcja krytyczna – dwa procesy

Proces P1

```
•
•
chin = getchar();
•
chout = chin;
putchar(chout);
•
•
```

Proces P2

```
•
•
•
chin = getchar();
•
chout = chin;
putchar(chout);
•
•
```

Semafory

- W ogólnym przypadku rozwiązanie problemu sekcji krytycznej z wykorzystaniem dla więcej niż dwóch procesów jest niezwykle złożone.
- Dlatego zaproponowano wykorzystanie narzędzia zwanego semaforem (Dijkstra 1968).
- Semafor S jest zmienną całkowitą, która jest dostępna jedynie za pomocą dwu, standardowych, niepodzielnych operacji:
  - operacji czekaj (Wait lub z holenderskiego Proberen) oznaczonej przez P;
  - operacji sygnalizuj (Signal lub z holenderskiego Verhogen) oznaczonej przez V.

Semafory

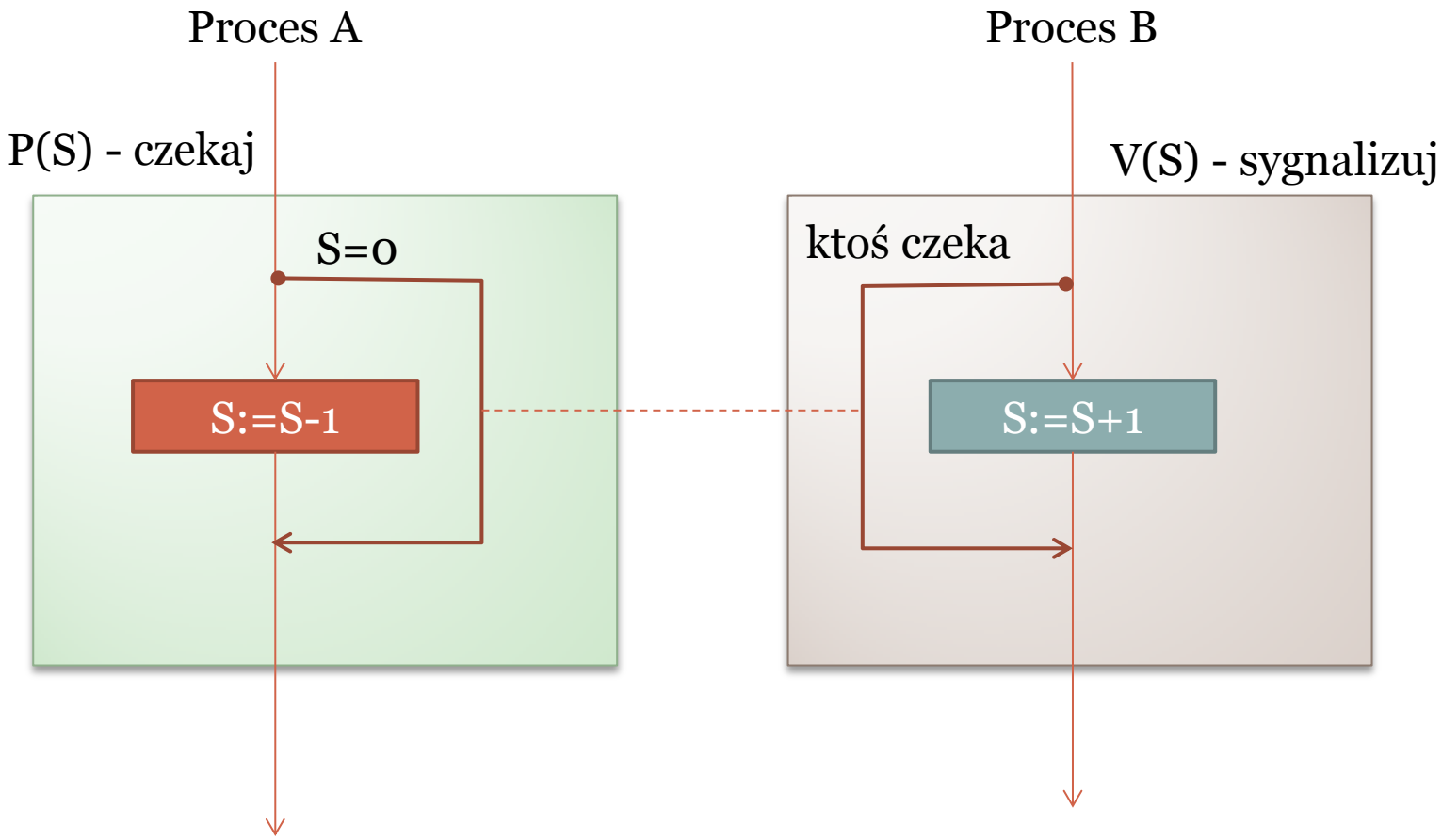
- Klasyczne definicje operacji czekaj (opuszczenie) i sygnalizuj (podniesienie) dla semaforów są następujące:

czekaj(S): while S ≤ O do nic;
S := S – 1;

sygnalizuj(S): S := S + 1;

- **Niepodzielność** semafora oznacza, że gdy jeden proces modyfikuje zmienną semafora, wówczas żaden inny proces nie ma prawa zmieniać wartości tej zmiennej.
- Ponadto, dla instrukcji czekaj **nie może wystąpić przerwanie**, zarówno w czasie sprawdzania zmiennej S jak i jej modyfikacji.

Zasada działania semafora



Zgodnie z definicją, podniesienie semafora w chwili gdy czekają na to inne procesy, powoduje, że któryś z nich będzie wznowiony.

Semafor binarny

- W odróżnieniu od semafora ogólnego, semafor binarny jest zmienną logiczną (true, false).
- Podniesienia semafora binarnego, to wykonanie instrukcji:
  - S:=1
- a jego opuszczenie, to wykonanie instrukcji:
  - czekaj aż S=1; S:=0

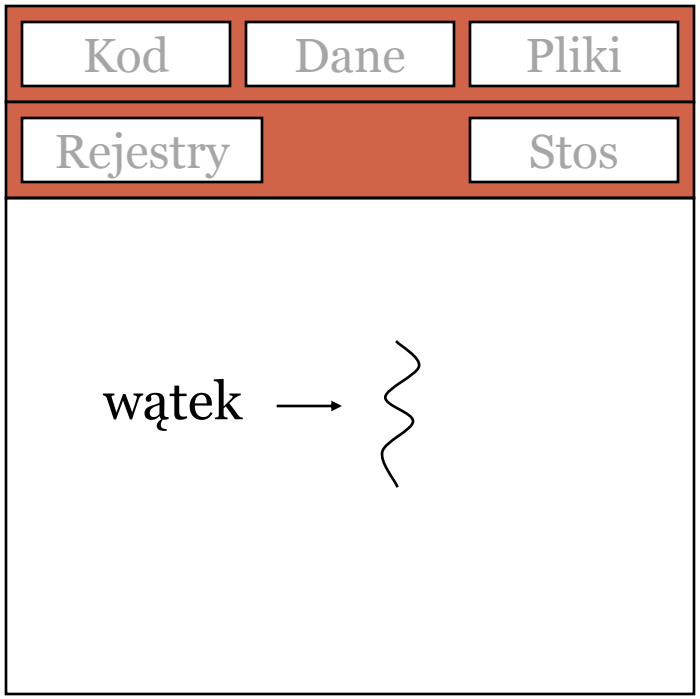
Wątki

- Wątek – proces lekki – jest podstawową jednostką wykorzystania procesora
- Skład wątku:
  - identyfikator;
  - licznik rozkazów;
  - zbiór rejestrów;
  - stos;
- Wraz z innymi wątkami tego samego procesu, wątek współużytkuje zasoby.
- Proces tradycyjny – ciężki – ma jeden wątek sterowania.

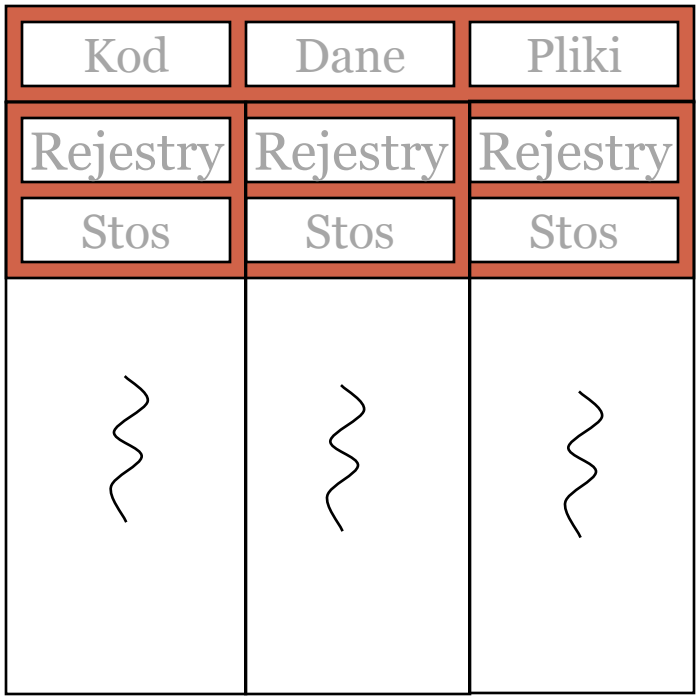
Wątki

- Wątki, czasami nazywane procesami lekkimi, są podstawową jednostką wykorzystania procesora.
- Wraz z innymi wątkami należącymi do tego samego procesu, wątek współużytkuje:
  - sekcję kodu,
  - sekcję danych,
  - oraz inne zasoby systemu operacyjnego
- Wątek posiada własny stan rejestrów oraz w większości przypadków własny stos.
- Przełączanie procesora pomiędzy wątkami jest rozwiązaniem znacznie tańszym niż przełączanie procesora pomiędzy klasycznymi procesami.

Procesy jedno i wielowątkowe



Proces jednowątkowy



Proces wielowątkowy

Korzyści wielowątkowości

Korzyści płynące z programowania wielowątkowego można podzielić na:

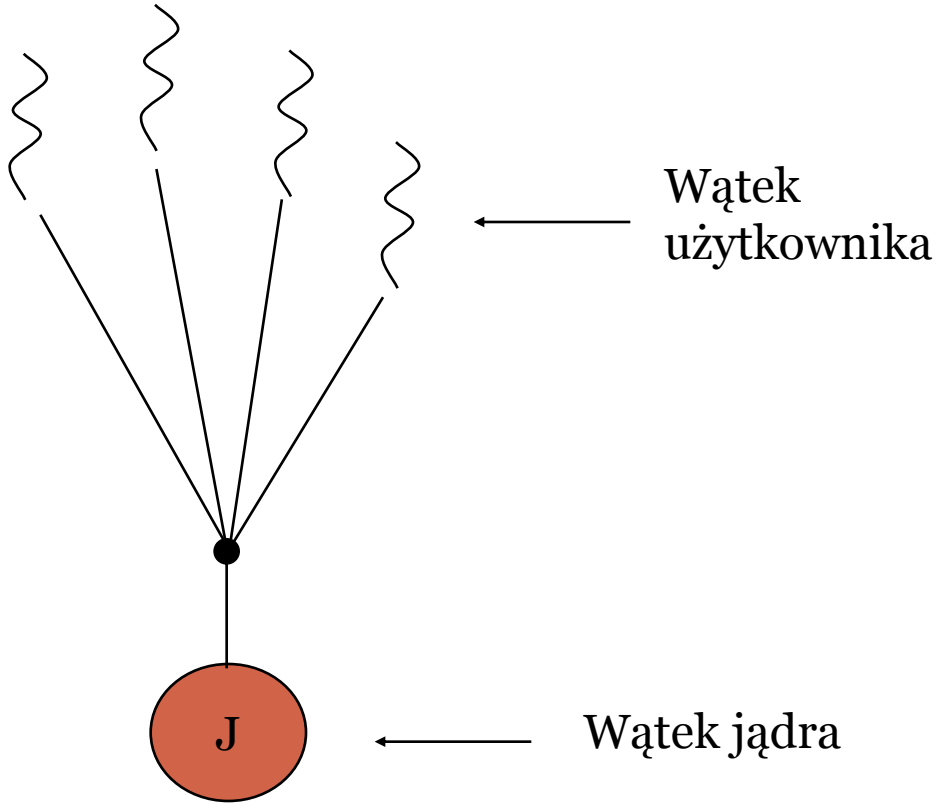
1. Zdolność do reagowania.
2. Dzielenie zasobów – wspólna pamięć i zasoby procesu do którego należą.
3. Ekonomika – bardziej opłacalne przełączanie kontekstu niż przydzielanie osobnych zasobów dla procesów.
4. Wykorzystanie architektury wieloprocesorowej – zwiększenie wykorzystania zalet pracy wielowątkowej.

Różne typy wątków

- Wątki użytkownika – realizowane na poziomie użytkowym. Jądro nie posiada informacji na temat wątków tworzonych przez użytkownika.
- Wątki jądra – udostępniane bezpośrednio przez system operacyjny. Jądro zajmuje się tworzeniem i planowaniem wątków oraz administruje je we własnej przestrzeni.

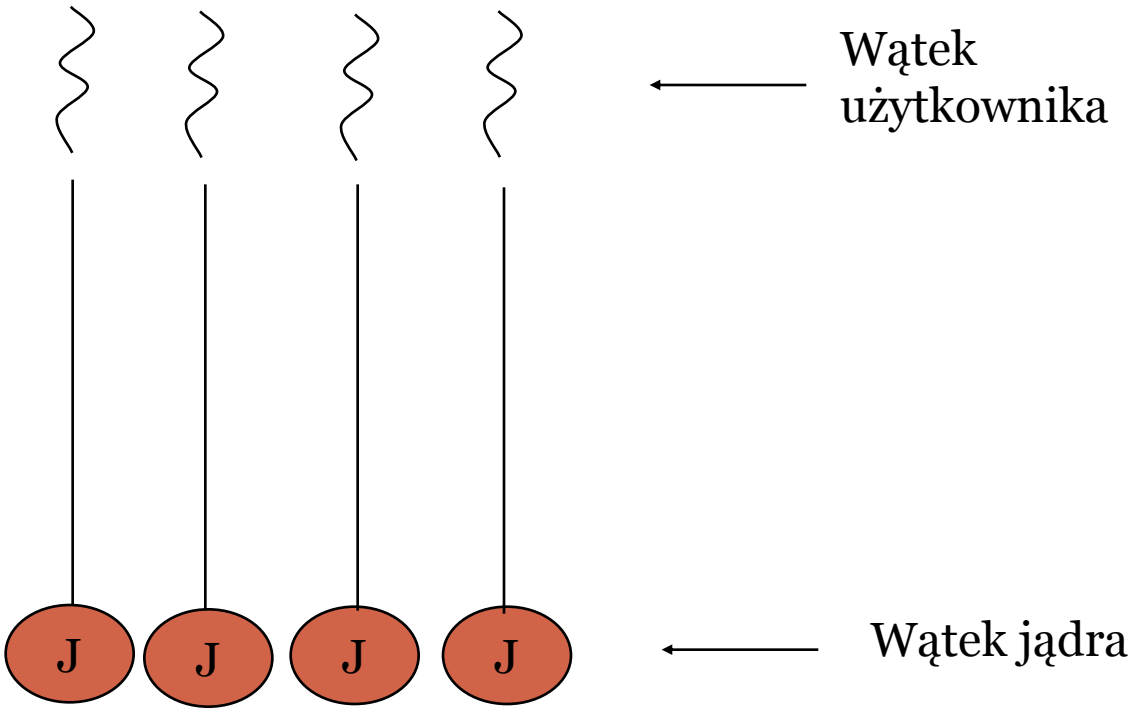
Modele wielowątkowości

- Model „wiele na jeden”



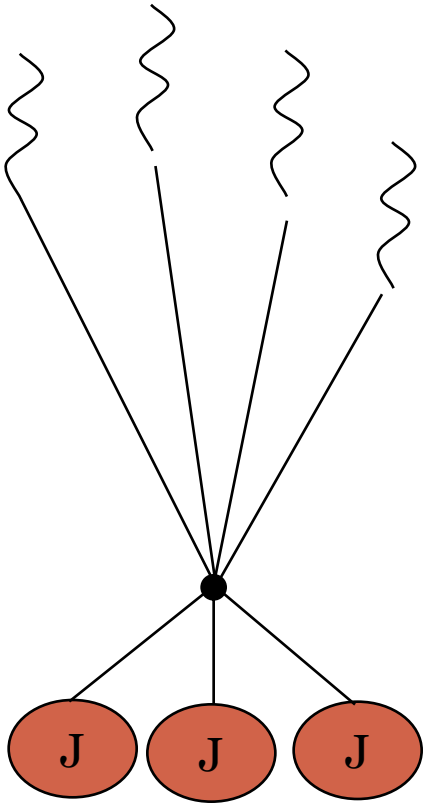
Modele wielowątkowości

- Model „jeden na jeden”



Modele wielowątkowości

- Model „wiele na wiele”



Wątek użytkownika

Wątek jądra

Kasowanie

- Kasowanie **asynchroniczne** – dany wątek kończy natychmiast wątek obrany za cel;
- Kasowanie **odroczone** – wątek obrany za cel okresowo sprawdza, czy powinien się zakończyć – co sprzyja zakończeniu procesu w sposób uporządkowany.
- Przy kasowaniu odroczonym – **punkty kasowania** umożliwiają wątkowi sprawdzenie, czy będzie go można skasować bezpiecznie.

Pula wątków

- Istotą puli wątków jest utworzenie pewnej liczby wątków w czasie rozruchu procesu i zaliczenie ich do puli, w której oczekują na zadania.
- Po zakończeniu zadania wątek wraca do puli i oczekuje na następne zadanie
- Gdy pojawi się zadanie do wykonania serwer budzi wątek z puli – i jeśli jest jakiś dostępny – przekazuje mu zamówienie do obsługi, jeśli nie ma wolnych – serwer czeka do czasu aż któryś się zwolni.

Zalety puli wątków

- Łatwiej jest obsłużyć zadanie przy pomocy istniejącego wątku niż oczekiwać na utworzenie wątku
- Pula wątków ogranicza liczbę wątków istniejących w danej chwili. Jest to ważne w systemach, w których nie jest możliwe zapewnienie dużej liczby wątków.

P-wątki

- Mianem P-wątków określa się standard POSIX, definiujący interfejs programów użytkowych API do tworzenia wątków i ich synchronizowania.
- Jest to specyfikacja a nie implementacja.
- Projektanci systemów operacyjnych mogą implementować specyfikację w dowolny dla siebie sposób.

Wątki w Windows 2000

- Realizuje interfejs programów użytkowych Win32API.
- Aplikacja Windows działa jako osobny proces, przy czym każdy proces może zawierać jeden lub więcej wątków.
- W systemie Windows zastosowano odwzorowanie „jeden na jeden” jednak zastosowanie biblioteki fiber zapewnia funkcjonalność modelu „wiele na wiele”

Wątki Linuxa

- Od wersji jądra 2.2 – dostępne są wątki z funkcjami: clone– powielającą proces i fork – do tworzenia procesu. W funkcji fork, nie jest tworzona kopia procesu wywołującego ale nowy proces.
- Dla każdego procesu w systemie istnieje niepowtarzalna struktura danych, w której przechowuje się (zamiast danych poszczególnych procesów), wskaźniki do innych struktur danych, w których są one pamiętane.

Wątki Linuxa c.d.

- Linux nie rozróżnia procesów i wątków, natomiast mówi się o zadaniach (tasks).
- Poza klonowaniem procesu Linux nie wspiera wielowątkowości, nie dostarcza odrębnych struktur danych ani procedur jądra.
- Dostępne są natomiast rozmaite implementacje P-wątków na użytkowym poziomie wielowątkowości.