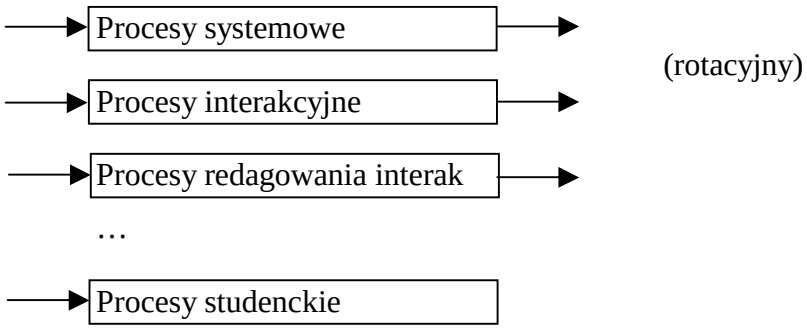
Koniec poprzedniego wykładu

5. Wielopoziomowe planowanie kolejek rozdziela kolejkę procesów gotowych na osobne kolejki. Procesy zostają na stałe przypisane do jednej z tych kolejek. Każda kolejka ma własny algorytm planujący najwyższy priorytet.

Najwyższy priorytet



Najniższy priorytet

Między kolejkami stosuje się na ogół planowanie stałopriotytetowe wywłaszczające. Inna możliwość to operowanie przedziałami czasu między kolejkami (np. kolejka procesów pierwszoplanowych (rotacyjna) – 80% czasu, kolejka procesów tła – 20% (FCFS)).

6. Wielopoziomowe kolejki ze sprzężeniem zwrotnym.

Możliwość przemieszczania procesów między kolejkami. Rozdzielanie procesów między kolejki – np. na podstawie czasu faz procesora. Jeśli proces zużywa za dużo czasu procesora, to zostanie przeniesiony do kolejki o wyższym priorytecie. Proces czekający zbyt długo w kolejce o niskim priorytecie zostanie (na skutek mechanizmu „postarzania") przeniesiony do kolejki o wyższym priorytecie.

Planista wielopoziomowych kolejek ze sprzężeniem zwrotnym może być określony przy pomocy następujących parametrów:

- liczba kolejek,
- algorytm planowania dla każdej kolejki,
- metoda użyta do decydowania o awansie (do kolejki o wyższym priorytecie),
- metoda użyta do decydowania o dymisji (do kolejki o niższym priorytecie),
- metoda określająca kolejkę, do której trafia proces potrzebujący obsługi.

F) Ocena algorytmów

Analityczna – używany jest algorytm i robocze „załadowanie" w celu wytworzenia wzoru lub wartości oszacowujących zachowanie algorytm dla danego załadowania. Jedną z odmian jest modelowanie deterministyczne, czyli z góry określone „załadowanie" robocze, określa się konkretne parametry (czas przetwarzania, czas oczekiwania, przepustowość).

Modele obsługi kolejek – na podstawie rozkładów statystycznych dla faz procesora i wejścia-wyjścia określa się prawdopodobieństwa wystąpienia faz (rozkład wykładniczy). Potrzebny jest też rozkład czasów przybywania procesów do systemu. Na podstawie tych rozkładów można obliczać średnią przepustowość, czas oczekiwania, czas cyklu przetwarzania itp.

SYMULACJA-wymaga określenia i zaprogramowania modelu systemu komputerowego. Symulator ma zmienna reprezentującą zegar .W miarę przyrostu tej zmiennej symulator zmienia stan systemu odzwierciedlając pracę urządzeń i planisty.

Dane do symulacji można generować lub wykorzystywać dane z „taśmy śladów" rzeczywistego działającego systemu.

WYKŁAD 5

A)Koordynowanie procesów

Aby procesy mogły być wykonywane w sposób uporządkowany ,w systemie muszą znajdować się mechanizmy do synchronizowania i komunikowania się procesów.
W systemach operacyjnych powszechnie spotykany jest układ producent – konsument .

Producent wytwarza informację (np. program drukujący wytwarza znaki)którą zużywa konsument (znaki są pobierane przez sterownik drukarki).

Koordynacja procesów producenta i konsumenta dokonywana jest za pomocą puli buforów .Producent odsyła wyniki do jednego bufora ,podczas gdy konsument pobiera dane z innego bufora .

Możliwe są dwie sytuacje :
-bufor jest nieograniczony (Może występować sytuacja ,że konsument czeka ,ale producent nigdy)
-bufor ograniczony (Konsument czeka jeśli wszystkie bufory są puste ,producent gdy wszystkie są pełne)

Przykład: Zagadnienie koordynacji z ograniczonym buforem)
Zmienne współdzielone(przez proces producenta i konsumenta)

```
var n;
type jednostka =...;
var bufor:array[0...n-1] of jednostka ;(ta tablica będzie cykliczna z dwoma wskaźnikami
                                        we, wy)
we ,wy : 0..n-1;
```

we, wy mają początkową wartość ∅
we-następny pusty bufor;
wy-pierwszy zajęty bufor;
Pula jest pusta gdy we = wy i jest pełna gdy (we + 1) mod n = wy

Kod producenta

```
repeat
  ....
  produkuj jednostkę w następ
  ....
  while (we=1) mod n = wy do nic;
  bufor[we]:=następ;
  we:=(we+1) mod n;
until false;
```

Kod konsumenta

```
repeat
  while we = wy do nic;

  nastk:=bufor[wy]
  wy:=(wy+1) mod n;
  ....
  konsumuj jednostkę z nastk;
  ....
until false;
```

nastp- zmienna lokalna producenta do przechowywania nowo wyprodukowanej jednostki
nastk- zmienna lokalna konsumenta do przechowywania jednostki do skonsumowania.

To działa pod warunkiem, że co najmniej n-1 buforów jest zapełnionych w tym samym czasie.
Chcąc usunąć tę wadę dodajemy współdzieloną zmienną licznik z wartością początkową 0.

Zmienna ta będzie zwiększana przy każdym dołączeniu do puli pełnego bufora i zmniejszana ilekroć z puli będzie zabierany któryś z pełnych buforów.

Producent	Konsument
repeat	repeat
...	while licznik=0 do nic;
while licznik=n do nic;	nastk:=bufor[wy];
bufor[we]:=nastp;	wy:=(wy+1) mod n;
we:=(we+1) mod n;	...
licznik:=licznik+1;	konsumowanie jednostki z nastk;
until false;	...
	until false;

Powyższe procedury są poprawne jeśli rozpatrujemy je osobno, lecz mogą nie działać właściwie gdy będą wykonane współbieżnie.

Uzasadnienie: niech licznik=5 i niech producent i konsument wykonują współbieżnie instrukcje:
licznik:=licznik+1 i licznik:=licznik-1;

Po wykonaniu tych instrukcji wartość zmiennej licznik może wynieść 4, 5 albo 6.
(Poprawny jest tylko 5!)

Instrukcja licznik:=licznik+1 jest realizowana w typowym procesorze jako:

		zaś licznik:=licznik-1 jako:	
	rej1:=licznik;		rej2:=licznik;
(*)	rej1:=rej1+1;	(**)	rej2:=rej2-1;
	licznik:=rej1;		licznik:=rej2;

gdzie: rej1 i rej2 są rejestrami procesora

Współbieżne wykonywanie instrukcji licznik:=licznik+1 oraz licznik:=licznik-1 jest równoważne sekwencyjnemu wykonywaniu rozkazów (*) i (**) z przeplatanką w dowolnym porządku, np.:

		licznik=5
T_0: producent	rej1:=licznik	{rej1=5}
T_1: producent	rej1:=rej1+1	{rej1=6}
T_2: konsument	rej2:=licznik	{rej2=5}
T_3: konsument	rej2:=rej2-1	{rej2=4}
T_4: producent	licznik:=rej1	{licznik=6}
T_5: konsument	licznik:=rej2	{licznik=4}⟵ !

Jeśli odwrócimy to licznik=6 (też źle!)

Przyczyną jest zezwolenie na to aby oba procesy manipulowały zmianą licznika współbieżnie.
Trzeba zagwarantować aby tylko jeden proces mógł to robić.
Wymaga to jakiejś formy synchronizacji obu procesów.

Podane zostaną teraz opisy sposobów rozwiązywania takich jak wyżej przedstawiamy problemów.

B) Problem sekcji krytycznej (SK)
Mamy system złożony z u procesów {P0, P1, .. , Pu}. Każdy proces ma segment kodu zwany sekcją krytyczną, w którym może zmieniać wspólne zmienne, aktualizować tabele, pisać do pliku, itd.
Gdy jeden proces wykonuje kod swojej sekcji krytycznej, wówczas żaden inny proces nie może wykonywać swojej sekcji krytycznej.
Zatem wykonywanie sesji krytycznych podlega wzajemnemu wyłączaniu w czasie. Problem SK polega na skonstruowaniu protokołu (zbioru zasad) do organizowania współpracy procesów.

Każdy proces musi prosić o pozwolenie na wejście do swojej SK. Fragment kodu realizującego taką prośbę nazywamy się *sekcją wejściową*. Po SK może występować *sekcja wyjściowa*. Pozostały kod zwie się *resztą*.
Aby skutecznie rozwiązać problem sekcji krytycznej muszą być spełnione 3 warunki:

a) wzajemne wyłączanie – jeśli P1 działa w SK to żaden inny proces nie działa w SK)
b) postęp – jeśli żaden proces nie działa w SK oraz istnieją procesy, które chcą wejść do SK, to tylko procesy nie wykonujące swoich reszt mogą kandydować jako następne do wejścia do SK; wybór ten nie może być odwlekany w nieskończoność
c) ograniczone czekanie – musi istnieć wartość graniczna liczby wejść innych procesów do sekcji krytycznych po tym, gdy dany proces zgłosił chęć wejścia do SK i zanim uzyskał na to pozwolenie.

Przykład: (dla dwóch procesów) P0 i P1. Oznaczamy Pi i Pj (j-1-i)
Algorytm 1 (wspólna zmienna numer) (na początku 0) (Dla procesu Pi)

repeat
  while numer ≠ i do nic
  sekcja krytyczna
  numer:=j
  reszta
until false
	To gwarantuje, że tylko 1 proces wykonuje sekcję krytyczną, ale nie zapewnia spełnienia warunku postępu (dlaczego?)
	Algorytm nie ma wystarczających informacji o procesach; pamięta tylko, który proces może wejść do SK

Algorytm 2
var flaga:array[0..1] of boolean; (początkowo false)
dla procesu Pi
(tu też nie jest spełniony warunek postępu

```
repeat
    flaga[i]:=true
    while flaga[j] do nic;
    sekcja krytyczna;
    flaga[i]:=false;
    reszta;
until false;
```

Algorytm 3 (flaga i numer – flaga-false, numer-bez znaczenia)

Dla procesu Pi

```
    repeat
       flaga [i]: = true;
          numer: = j;
       while (flaga [j] and numer = j) do nic;
            sekcja krytyczna
       flaga [i]: = false;
            reszta
    until false;
```

To rozwiązanie spełnia trzy warunki:

- wzajemnego wyłączania,
- postępu,
- ograniczonego czekania.

C) Semafory (inna metoda synchronizacji procesów)

Semafor jest zmienną całkowitą, która – niezależnie od inicjacji – jest dostępna tylko za pomocą dwóch niepodzielnych operacji: czekaj(wait) i sygnalizuj(signal).
Definicje semaforów są nstępujące:

```
czekaj(s):      while s <= 0 do nic;
                s: = s - 1;

sygnalizuj(s):  s: = s + 1;
```

Zmiany wartości semafora są wykonywane w sposób niepodzielny, tzn. że gdy jeden proces modyfikuje wartość semafora, to żaden inny proces nie może jednocześnie zmieniać tej wartości. Dodatkowo w przypadku instrukcji **czekaj(s)** nie może wystąpić przerwanie podczas sprawdzania(**s <= 0**) i modyfikacji(**s: = s - 1**) zmiennej **s**.

Semafory służą do rozwiązywania **sekcji krytycznej** z udziałem procesów. Wspólny semafor wzajwy (od wzajemnego wyłączania) jest współdzielony przez n procesów. Na początku wartość wzajwy =1.

Każdy proces Pi jest zorganizowany następująco:

```
repeat
   czekaj(wzajwy);
      sekcja krytyczna
   sygnalizuj(wzajwy);
      reszta;
until false;
```

Cechą charakterystyczną tego rozwiązania jest, że gdy jeden proces jest w **sekcji krytycznej** pozostałe procesy usiłujące wejść do **sekcji krytycznych** muszą nieustannie wykonywać w pętli instrukcje sekcji wejściowych.

Problem posilających się filozofów.
Pięciu filozofów spędza czas na myśleniu i jedzeniu. Filozofowie dzielą wspólny okrągły stół, wokół którego ustawiono 5 krzeseł. Na środku stołu stoi miska ryżu, a naokoło leży 5 pałeczek. Jeśli filozof odczuwa głód, próbuje ująć w ręce 2 pałeczki leżące najbliżej jego miejsca (lewa, prawa). Za każdym razem filozof może podnieść tylko 1 pałeczkę. Po zjedzeniu odkłada obie pałeczki i zaczyna myślenie.

STRUKTURA i-tego FILOZOFA:

```
repeat
   czekaj(pałeczka [i] );                    {lewa}
   czekaj(pałeczka [ (i + 1) mod 5] );       {prawa}
   ...
   jedzenie
   ...
   sygnalizuj(pałeczka[i] );
   sygnalizuj(pałeczka[ (i = 1) mod 5] );
   ...
   myślenie
   ...
until false;
```

Niedoskonałe rozwiązanie (możliwość blokady)

Na początku wszystkie 1
(var pałeczka: array [0...4] of semafor;)
Może powstać blokada gdy wszyscy podniosą równocześnie lewą pałeczkę, bo wtedy wszystkie elementy tablicy staną się 0.
Jak unikać blokady:

- Przy stole <= czterech filozofów
- Pozwolić filozofowi na podnoszenie pałeczek tylko gdy obie są dostępne i czynić to w **sekcji krytycznej**
- Zastosować asymetrię. Filozof o numerze: nieparzystym (lewa, prawa)
  parzystym (prawa, lewa).

D) Komunikacja międzyprocesowa
Istnieją dwa uzupełniające się wzajemnie schematy komunikacji między procesami:
- pamięć dzielona: wymaga się, aby procesy współużytkowały pewne zmienne, za pomocą których odbywa się wymiana informacji między procesami (ograniczony bufor). Odpowiedzialność za zorganizowanie komunikacji spada na programistę. SO dostarcza tylko środków do współdzielenia pamięci.

- system komunikatów: odpowiedzialność za organizację komunikacji spada na SO (nadaj, odbierz)

W jednym SO mogą być wykorzystane obie metody. Zadaniem systemu komunikatów jest umożliwienie procesom wzajemnej komunikacji bez uciekania się do zm. dzielonych. Komunikaty mogą mieć stałą bądź zmienną długość.
Jeśli 2 procesy P i Q chcą się ze sobą komunikować, to muszą wzajemnie nadawać i odbierać komunikaty – musi więc istnieć między nimi łącze komunikacyjne.

!!!! Pytania związane z implementacją (łącza)
- jak ustawia się łącze;
- czy łącze może być powiązane z więcej niż dwoma procesami;
- ile może być łączy między każdą parą procesów;
- jak duża jest pojemność łącza;
- czy łącze ma jakiś obszar buforowy (jak duży);
- jaki jest rozmiar komunikatów;
- czy łącza akceptują komunikaty o stałej czy o zmiennej długości;
- czy łącze jest jedno czy dwukierunkowe;

Metody logicznej implementacji łącza
- komunikacja pośrednia lub bezpośrednia;
- komunikacja symetryczna lub asymetryczna;
- buforowanie automatyczne lub jawne;
- wysyłanie na zasadzie tworzenia kopii lub odwołania (wskaźnika);
- komunikaty stałej lub zmiennej długości;

Komunikacja bezpośrednia
Proces nadający lub odbierający musi jawnie nazwać odbiorcę lub nadawcę. Mamy: nadaj (P, komunikat) odbierz (Q, komunikat).
Cechy:
łącze jest ustawiane automatycznie a procesy muszą znać swoje identyfikatory;
łącze dotyczy dokładnie dwóch procesów;
między dwoma procesami jedno łącze;
łącze jest dwukierunkowe;

Komunikacja pośrednia
Komunikaty nadawane i odbierane są za pośrednictwem skrzynek pocztowych (zwanych też postami). Skrzynka jest obiektem, w którym proces może umieszczać komunikaty oraz je odbierać.

Skrzynki mają swą identyfikację. Możliwość komunikacji miedzy procesami istnieje, gdy mają one jakąś wspólną skrzynkę.
Cechy:
łącze (dwa procesy) – gdy procesy dzielą skrzynkę;
łącze może dotyczyć więcej niż 2 procesów;
para procesów może mieć kilka łączy;
łącze może być jedno- lub dwukierunkowe;

Buforowanie
Łącze ma pewną pojemność określającą liczbę komunikatów, które mogą w innym czasowo przekazywać. Może to być widziane jako kolejka komunikatów przypisana do łącza.
Może być :

- pojemność zerowa – żaden komunikat nie może czekać
- pojemność ograniczona – kolejka ma skończoną długość
- pojemność nieograniczona

Sytuacje wyjątkowe

- może się zdarzyć, że zakończy się proces nadawcy (bądź odbiorcy) przed zakończeniem przetwarzania komunikatu, pozostaną wówczas komunikaty, których nikt nie odbierze lub jakieś procesy będą czekać na komunikaty, które nigdy nie zostaną nadane. (Zakończenie procesu)
- komunikat nadany przez proces P do procesu Q może zaginąć (na skutek awarii sprzętu lub wewnątrz komunikacyjnej), istnieją trzy metody postępowania:

1. system operacyjny jest odpowiedzialny za wykrywanie i ponowne nadanie
2. proces nadający wykrywa utratę i powtórnie nadaje
3. system operacyjny wykrywa i zawiadamia nadawcę, który może postępować różnie (utrata komunikatu)

- zniekształcenia komunikatu – sumy kontrolne (parzystość). System operacyjny wykrywa i albo wysyła albo powiadamia proces nadawcy.

Spis treści wykładu 5

a) Koordynowanie procesów
- pula buforów(ograniczony, nieograniczony)
- producent , konsument
- manipulacja zmiennymi współdzielonymi

b)Problem sekcji krytycznej Warunki rozwiązania:
- wzajemne wyłączanie
- postęp
- ograniczone czekanie

c) Semafory (u procesów)
- czekaj (s), sygnalizuj (s) – niepodzielne
- problem posilających się filozofów i próba rozwiązania

d) Komunikacja międzyprocesowa
- pamięć dzielona
- system komunikatów
    - komunikacja bezpośrednia
    - komunikacja pośrednia
    - buforowanie (kolejka do skrzynki)
        - bufor zerowy
        - bufor ograniczony
        - bufor nieograniczony

- sytuacje wyjątkowe