Jeden procesor, jedno zadanie

Poprawność programu sekwencyjnego

- *własność stopu* (*własność poprawności całkowitej*)
  → program na pewno się zatrzyma
- *własność częściowej poprawności*
  → jeśli program się zatrzyma, to zwróci poprawny wynik (zgodny z zapisanym algorytmem)

Wymagania – **warunki konieczne poprawności programu sekwencyjnego**

- program tworzą instrukcje, które procesor może wykonać
  → ignorowanie instrukcji niewyspecyfikowanych
  → sygnalizacja instrukcji niedozwolonych
- wszystkie zmienne są jednoznacznie identyfikowane
  → każda zmienna jest odwzorowana w pamięci operacyjnej
  → używane są dozwolone tryby adresowania
  → sygnalizacja błędów adresowania
- istnieje i jest obecna w programie *instrukcja zatrzymania* procesora
  → wznowienie po zatrzymaniu – RESET (przerwanie nieprecyzyjne)

Procesy współbieżne

Proces (zadanie) – każdy program w trakcie wykonania
(program może mieć wiele niezależnych realizacji)

Procesy współbieżne – jeden rozpoczyna się przed zakończeniem drugiego,
faktyczna współbieżność: jeden procesor, wiele procesów (zadań)

Poprawność procesu współbieżnego

- *własność żywotności*
  → każde oczekiwane zdarzenie (działanie) nastąpi
    - wykluczone zablokowanie wykonania procesu

  → każdy proces ma realną szansę wykonania
  (*zasada sprawiedliwości* lub *uczciwości*)

- *własność bezpieczeństwa*
  → program jest zawsze w stanie pożądanym
    - istnieje pełna kontrola stanu wszystkich procesów
      i sposób rozwiązywania problemów

Warunki poprawności procesu współbieżnego

Realizacja wymagań poprawności programu współbieżnego:

- *żywotność* – przydział czasu procesora
  → *na* czas wykonania procesu – możliwe *zagłodzenie* innych procesów
  → okresowo na ustalony czas – podział czasu (ang. *time-sharing)*

- *bezpieczeństwo* – ochrona procesu
  → prywatność
    - osobne przestrzenie adresowe → przestrzeń wirtualna
    - ograniczenie dostępu: tryb: (r-w-x) – zezwolenie/zakaz
    - kontrolowana komunikacja
  → ochrona zasobów – kontekst w chwili wywłaszczenia procesu
    - kontekst procesora – stan procesora
    - kontekst pamięci – stan pamięci

Ochrona procesu

Zasady ochrony zasobów:

- zapobieganie (ang. *prevention*) naruszeniu spójności systemu
- reagowanie na ingerencję w mechanizm ochrony
  - wykrywanie (ang. *detection*) błędów i ataków
  - rozpoznawanie i neutralizacja skutków ingerencji
  - unieważnianie (ang. *nulifying*) działań ingerujących w mechanizm ochrony

Wspomaganie ochrony zasobów na poziomie architektury rzeczywistej:

- *w przestrzeni kodów* – uniemożliwienie wykonania instrukcji uprzywilejowanych (ang. *privileged*) w procesie użytkownika
- *w przestrzeni operandów* – wykluczenie wykonania w trybie użytkownika operacji używających zastrzeżonych operandów
- *w przestrzeni danych* – kontrolowany dostęp do danych, (przypisanie danej znacznika (ang. *tag*) jest sprzeczne z koncepcją pamięci)

- *furtki* (ang. *gate*) – kontrolowany dostęp do procesów uprzywilejowanych

Pamięć wielu procesów – przestrzeń wirtualna

W programie, stanowiącym opis procesu
  każdy obiekt jest identyfikowany w logicznej przestrzeni adresowej
Ten opis jest treścią pliku wykonalnego (ang. *executable*)

Każdy program jest opisany w tej samej logicznej przestrzeni adresowej

Podczas współbieżnego wykonania wielu procesów
  każdy obiekt musi być jednoznacznie identyfikowany,
  ale wszystkie są zdefiniowane w tej samej logicznej przestrzeni adresowej

Rozwiązanie:
identyfikator obiektu = (identyfikator procesu, adres logiczny)

Ta przestrzeń adresowa jest wirtualna (istnieje tylko jako koncepcja)

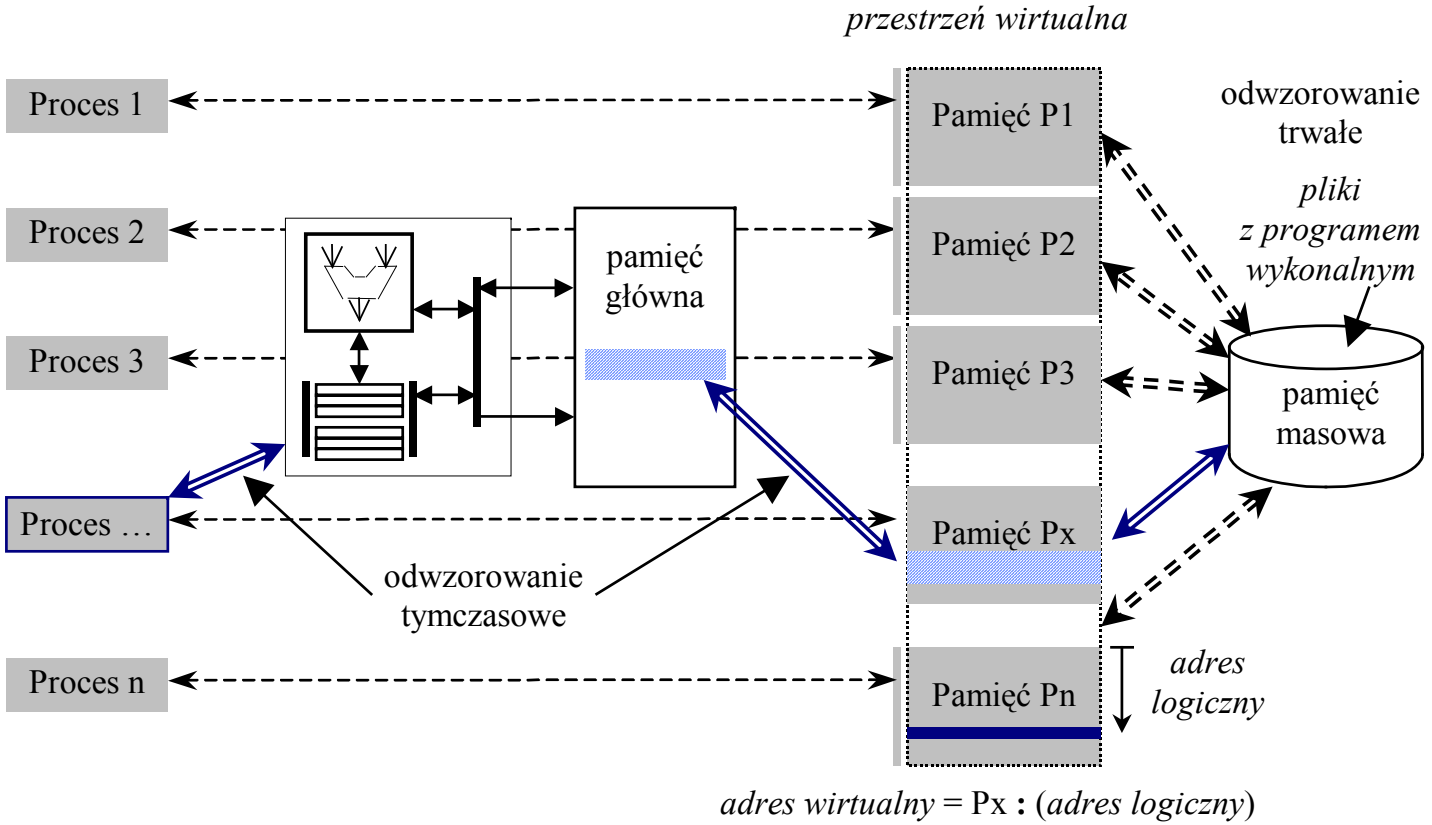
Odwzorowanie pamięci procesu w przestrzeni operacyjnej

Zawarty w pliku opis programu i jego danych można jednoznacznie odwzorować w wirtualnej przestrzeni adresowej

W danej chwili niezbędne jest odwzorowanie w przestrzeni (pamięci) operacyjnej tylko procesu aktualnie wykonywanego lub jego części

Wniosek:
Aby wykonać program w przestrzeni operacyjnej
wystarczy określić sposób odwzorowania
wirtualnej przestrzeni adresowej w przestrzeń operacyjną.

Pamięć wielu procesów – przestrzeń wirtualna



- tylko jeden proces jest tymczasowym użytkownikiem procesora
- w pamięci głównej potrzebne odwzorowanie pamięci procesu aktywnego

Ochrona pamięci procesu

Separacja obszarów pamięci – prawo dostępu (ang. *access right*)
– zakaz/zezwolenie + tryb użycia: odczyt-zapis-wykonanie (r-w-x)

jednolite reguły dostępu:

- pamięć jednozakresowa (ang. *single domain*) – proces otrzymuje wyłącznie przydział własnej pamięci, niedostępnej dla innych procesów, co wyklucza użycie wspólnych danych i komunikację przez pamięć

alternatywne prawa dostępu do zasobów pamięci

- pamięć dwuzakresowa (ang. *two domain*) – jeden z obszarów jest współdzielony i dostępny dla wszystkich procesów, drugi jest obszarem własnym procesu, niedostępnym dla innych procesów, wszystkie obszary są separowane, komunikacja tylko przez obszar współdzielony

selektywne prawa dostępu do zasobów pamięci

- pamięć wielozakresowa (ang. *multi–domain*) – cała pamięć jest podzielona na N rozłącznych obszarów, każdy proces uzyskuje prawo dostępu do pewnego podzbioru tych obszarów

Model realizacji procesu

Mechanizmy ochrony muszą być jednolite dla wszystkich procesów

Fazy realizacji procesu:

- uaktywnienie – przydział czasu procesora (uwłaszczenie procesu)
- start – odtworzenie kontekstu
- wykonanie – synchronizacja (przerwania, wyjątki, punkty komunikacji)
- wstrzymanie – przechowanie kontekstu
    - zwolnienie procesora (wywłaszczenie zadania/procesu)

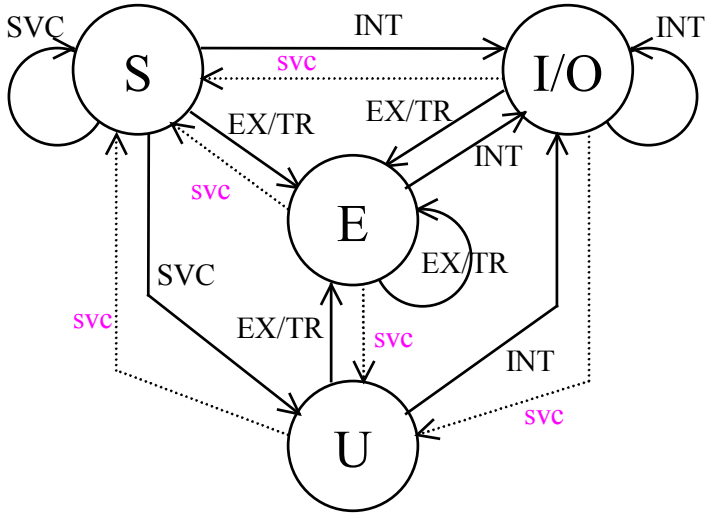
Kolejność wykonania procesów – *szeregowanie* zadań

- kolejka krótkoterminowa – procesy aktywne, często wykonywane
- kolejka średnioterminowa – procesy aktywne, rzadko wykonywane
- kolejka długoterminowa – procesy uśpione

Przerwanie – sygnalizacja zdarzenia w środowisku procesów (żądanie komunikacji, wystąpienie błędu, upływ czasu, wymagającego obsługi programowej

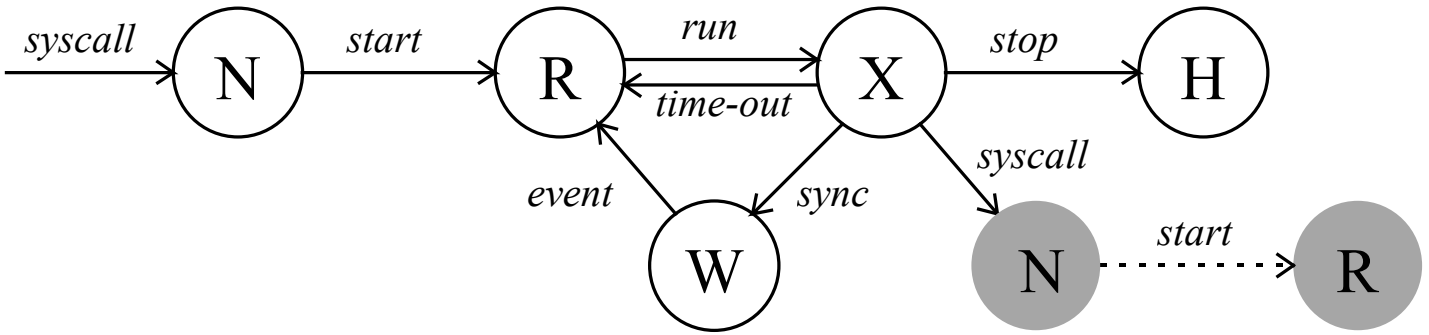
Poziomy uprzywilejowania procesów – kolejność wykonania

- obsługa wyjątków (ang. *exception handling*) – utrzymanie integralności systemu
- obsługa we/wy (ang. *I/O handling*) – funkcje krytyczne względem czasu
- zadania nadzoru (ang. *supervisor functions*) – zarządzanie procesami i pamięcią,
- zadania użytkowników (ang. *user jobs*)



- zdarzenia asynchroniczne – przerwanie zewnętrzne (INT)
- zdarzenia synchroniczne – wywołanie systemowe (SVC), pułapka (TRAP), wyjątek (EX) (błąd (ERR), naruszenie ochrony (PF/SF))

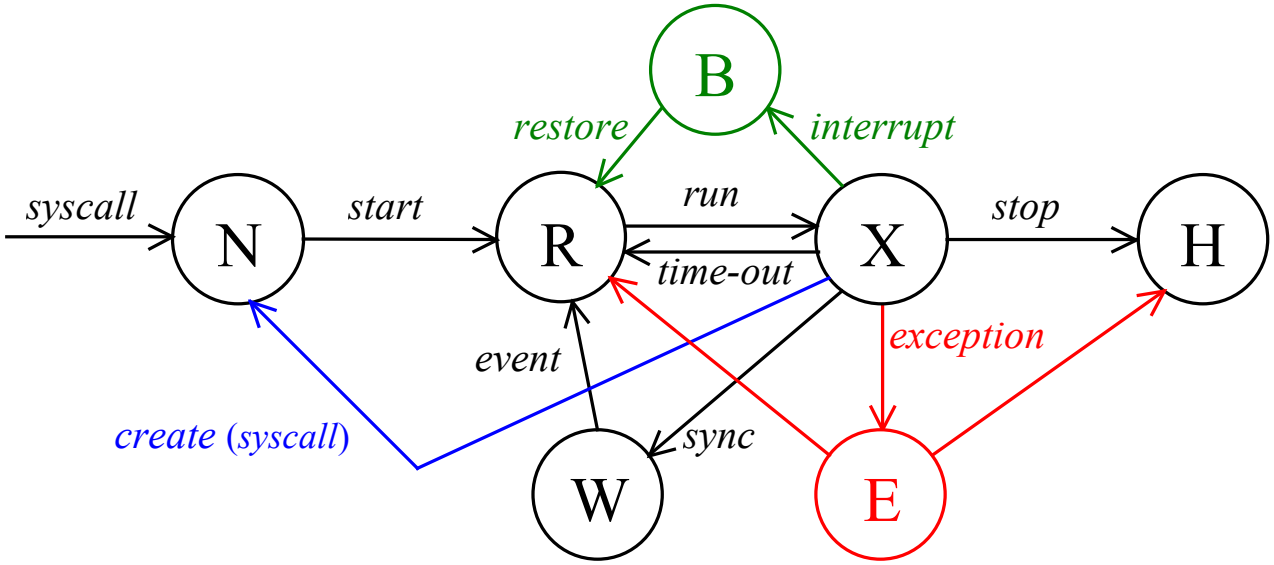
Model procesowy - schemat wykonywania procesu



syscall	*start*	*run*
• nadanie identyfikatora	• wpis do kolejki procesów	• odtworzenie kontekstu
• nadanie priorytetu	•	• przekazanie sterowania
• definicja środowiska	•	• usunięcie z kolejki

time-out	*sync*	*event*
• wstrzymanie	• wstrzymanie	• wybudzenie procesu
• przechowanie kontekstu	• przechowanie kontekstu	• powrót do kolejki
• powrót do kolejki	• uśpienie procesu	*stop*
		• likwidacja środowiska

Model procesowy rozszerzony



interrupt	*restore*	*exception*
• zdarzenie środowiskowe	• zakończenie obsługi	• błąd – próba naprawy
• wstrzymanie procesu	• wznowienie	• wznowienie *albo*
• obsługa zdarzenia		• usunięcie z kolejki

Kontekst procesu

Kontekst procesu – ***kompletna informacja o stanie procesu***, która obejmuje:

- zawartość *wszystkich rejestrów procesora* (kontekst procesora)
- wartości wszystkich zmiennych (kontekst pamięci)

Kontekst procesu jest **statyczną strukturą danych!**

Tablica procesów (ang. *process table, PT*) – statyczna struktura danych menadżera (SO)

- obiekt w tablicy *PT* – *blok sterujący procesu* (ang. *process control block, PCB*)

Blok sterujący procesu

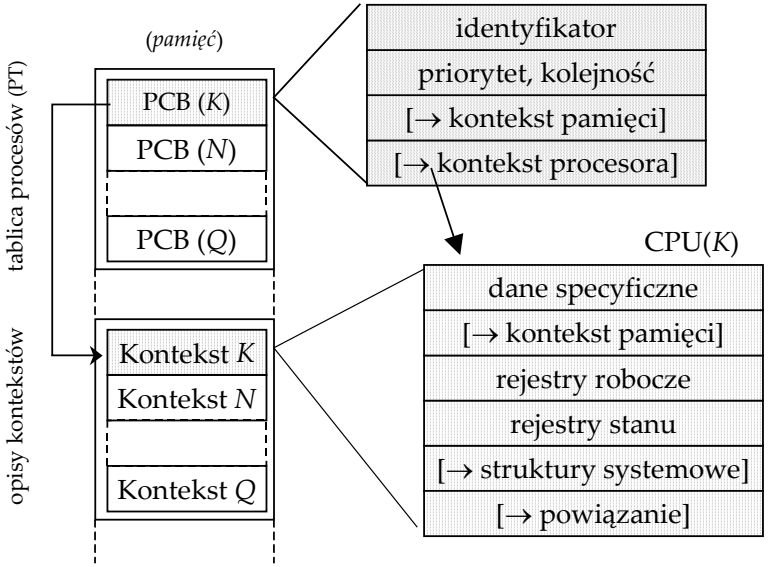
- *kontekst minimalny procesu*:
  - **minimalny kontekst procesora** (rejestr stanu i licznik programu)
  - minimalne **zapotrzebowanie na pamięć** (rozmiar zbioru roboczego)
- *wskaźnik pełnego kontekstu procesora*
- *wskaźnik pełnego kontekstu pamięci*
- *priorytet i parametry harmonogramowania.*

Minimalny kontekst procesora – zmienne stanu, które
mogą być automatycznie zmienione podczas wykonania kolejnej instrukcji

Kontekst procesora

Kontekst procesora (ang. *CPU context*) – opis bieżącego *stanu procesora*:

- zawartość rejestrów roboczych, rejestrów stanu i rejestrów wyjątków
  - [wskaźnik kontekstu pamięci]
- wskaźniki dostępnych systemowych struktur danych (stosy programowe)
- wskaźnik powiązania z procesem wywołującym (nr procesu wywołującego)
- dane specyficzne



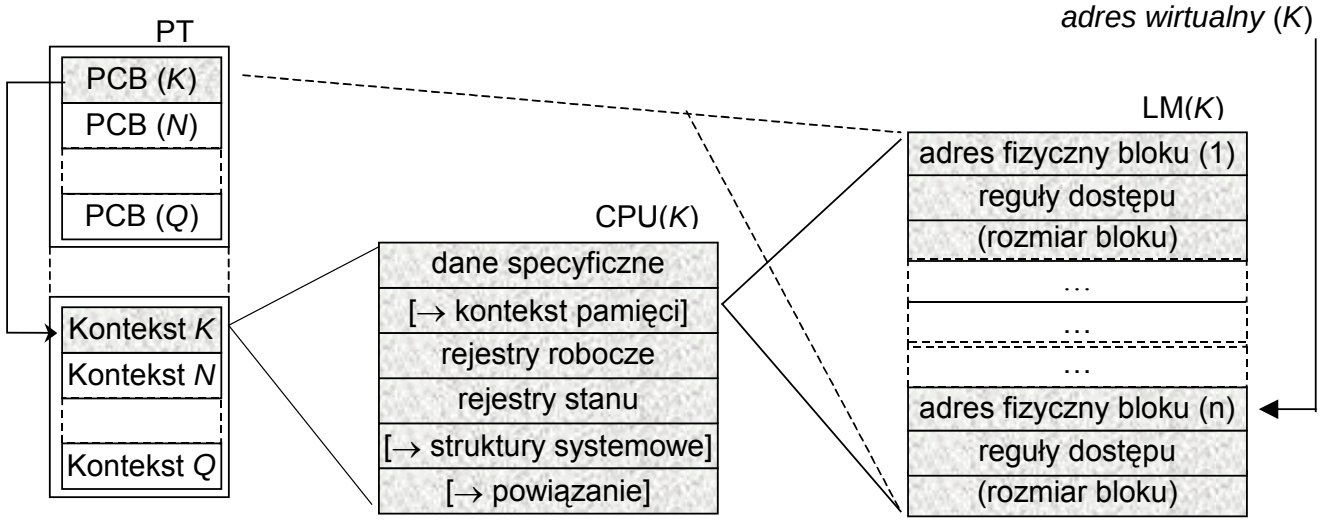
Kontekst pamięci

Kontekst pamięci – lokalny (LM) i globalny (GM)

Aktualizacja wskaźników kontekstu pamięci tylko w trybie nadzoru

Kontekst pamięci (ang. *memory context*) – w *tablicy opisów* (*deskryptorów*) *pamięci*

- adresy i rozmiary bloków pamięci procesu
- obecność bloków pamięci procesu w pamięci głównej
- reguły dostępu do bloków pamięci procesu



Kontekst - przykłady (IA-32)

Kontekst procesora – TSS (ang. *Task State Segment*):

- rejestry procesora
- wskaźniki stosu
- rejestry systemowe
- wskaźniki kontekstu pamięci (LDTR, GDTR, CR3)

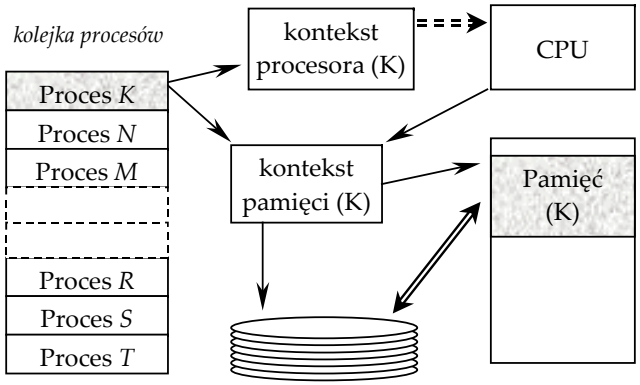
Kontekst pamięci:

- LDT (ang. Local Descriptor Table) – deskryptory segmentów
- GDT (ang. Global Descriptor Table) – deskryptory segmentów
- PT (ang. Page Table) – – deskryptory stron

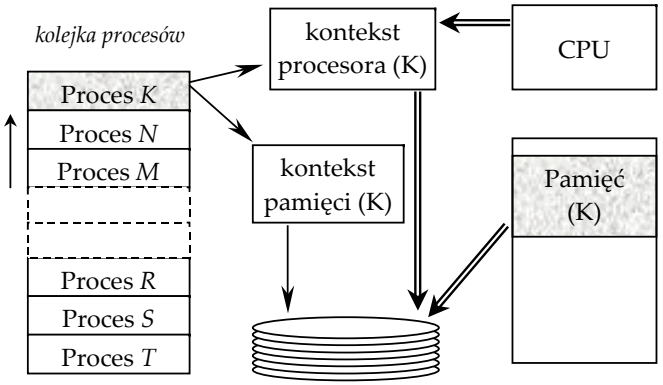
Deskryptor segmentu (opis dostępu do segmentu) – element LDT lub GDT:

<table>
<tr><td colspan="8">Adres bazowy segmentu ($A_{31...24}$)</td><td>G</td><td>D/B</td><td>0</td><td>AVL</td><td>Rozmiar seg. ($L_{19...16}$)</td></tr>
<tr><td>P</td><td>DPL</td><td>DPL</td><td>S</td><td>t</td><td>t</td><td>t</td><td>t/a</td><td colspan="5">Adres bazowy segmentu ($A_{23...16}$)</td></tr>
<tr><td colspan="13">Adres bazowy segmentu ($A_{15...0}$)</td></tr>
<tr><td colspan="13">Rozmiar segmentu ($L_{15...0}$)</td></tr>
</table>

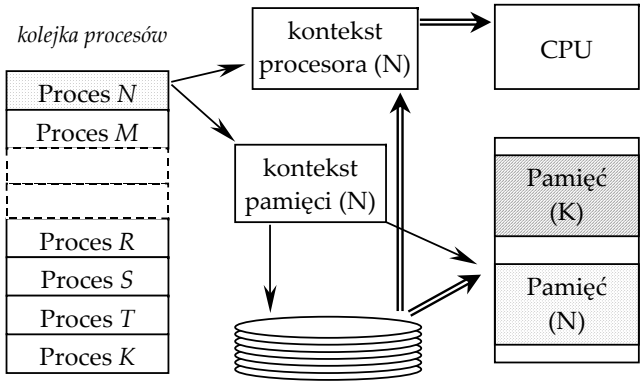
Przełączanie kontekstów



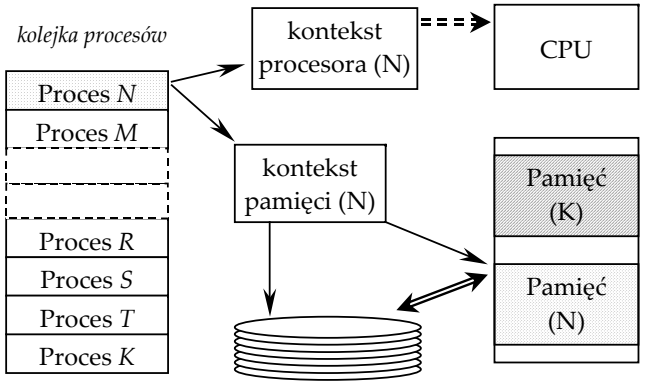
- wykonywany proces *K*



- wywłaszczenie procesu *K*



- wznowienie procesu *N*



- wykonywany proces *N*

Synchronizacja procesów

Procesy współbieżne wymagają synchronizacji

Syndromy współbieżności procesów

- współpraca → komunikacja (wymiana danych)
  - synchronizacja *wewnętrzna*, określona w programie
- współzawodnictwo → konkurowanie o unikatowy zasób
  - synchronizacja *zewnętrzna*, realizowana w trybie nadzoru

Wzajemne wykluczanie – niezbędny warunek poprawnej synchronizacji

- schemat dostępu do zasobu współdzielonego (*shared resource*)

  …własne sprawy…
  protokół wstępny – naleganie (czekanie na dostęp)
  sekcja krytyczna – realizacja dostępu do zasobu unikatowego
  protokół końcowy – zwolnienie zasobu
  …własne sprawy…
- **warunek bezpieczeństwa** – w każdej chwili w sekcji krytycznej może przebywać tylko jeden proces
- **postulat żywotności** – proces nalegający na pewno uzyska dostęp

Współpraca - model producentów i konsumentów

N-elementowy bufor cykliczny, semafor całkowity

- producent wstawia porcję do bufora, jeśli nie jest zapełniony (semafor <N), (i nie używa go inny producent) w przeciwnym razie czeka
- konsument pobiera porcję z bufora, jeśli nie jest on pusty (semafor >0) (i nie używa go inny producent) w przeciwnym razie czeka

Producent
czekaj na miejsce
wstaw
sygnalizuj pełny

Konsument
czekaj na pełny
pobierz
sygnalizuj miejsce

Problem producentów i konsumentów*)

```
const: N=?;                          {pojemność bufora}
       K=?;                          {liczba konsumentów}
       P=?;                          {liczba producentów}
var:   j: integer:= 1;               {wskaźnik końca kolejki}
       i: integer:= 1;               {wskaźnik czoła kolejki}
       bufor: array[1..N] of porcja      {bufor N-elementowy]
       miejsce: semaphore:= N;       {semafor wypełniania}
       pełne: semaphore:= 0;         {semafor opróżniania}
       ochrona_j: bin_semaphore:= 1;     {ochrona indeksu końca kolejki}
       ochrona_i: bin_semaphore:= 1;     { w sekcji krytycznej}

process producent(i=1,..P)
var p: porcja;
begin
       while true do begin
              produkuj(p);
              czekaj (miejsce);      {jeśli bufor jest pełny czekaj,
                                     {w przeciwnym razie i obniż semafor}
              czekaj_bin (ochrona_j);    {tylko 1 proces może zapełniać bufor}
              bufor [j]:= p;
              j:= jmodN+1;
              sygnalizuj_bin (ochrona_j);
```

```
            sygnalizuj (pełne); {sygnalizuj wypełnienie}
      end
end;

process konsument(i=1,..K)
var p: porcja;
begin
      while true do begin
            czekaj (pełne)       {jeśli bufor jest pusty czekaj,}
                                 {w przeciwnym razie obniż semafor}
            czekaj_bin (ochrona_i);    {tylko 1 proces może pobierać}
            p:= bufor [i];
            i:= imodN+1;
            sygnalizuj_bin (ochrona_i);
            sygnalizuj (wolne); {sygnalizuj wypełnienie}
            konsumuj(p);
      end
end;
```

Współpraca - model czytelników i pisarzy

N-elementowy bufor, semafor całkowity (miejsce) i 2 semafory binarne (wejście)
W czytelni może być dowolna liczba czytelników ale tylko 1 pisarz.

- jeśli czeka pisarz to wstrzymuje wejścia kolejnych czytelników
- jeśli bufor jest pusty, czytelnicy czekają, aż pisarze wytworzą nowe elementy
- jeśli czekają czytelnicy to wstrzymują kolejnego pisarza

czytelnik
czekaj (czyt)
    jeśli nie czekają pisarze
        wpuść wszystkich czekających czytelników
podnieś (czyt)
    jeśli czekają pisarze, wpuść wszystkich czytelników i zapewnij sobie wejście

pisarz
czekaj (pisz)
    jeśli nie czekają czytelnicy
        wpuść po kolei wszystkich czekających pisarzy
podnieś (pisz)
    jeśli czekają czytelnicy, wpuść kolejno pisarzy

Problem czytelników i pisarzy*)

```
const  C=?;                          {liczba czytelników}
       P=?;                          {liczba pisarzy}
var    ac: integer:= 0;              {aktywni czytelnicy – czytający i czekający}
       cc: integer:= 0;              {czytający czytelnicy}
       ap: integer:= 0;              {aktywni pisarze – piszący i czekający}
       pp: integer:= 0;              {piszący pisarze }
       CZYT: semaphore;= 0;          {wstrzymywanie czytelników}
       PIS: semaphore;= 0;           {wstrzymywanie pisarzy}
       OCHRONA: bin_semaphore:=1     {ochrona zmiennych}
       EXCL: bin_semaphore:=1        {wykluczanie pisarzy – wchodzą pojedynczo!}

process czytelnik (i=1,..C)
begin
       while true do begin
       własne_sprawy;
       czekaj_bin (OCHRONA);         {chroń zmienne w sekcji krytycznej?}
              ac:= ac+1;             {doszedł nowy czytelnik}
              if ap=0 then           {sprawdź czy nie czekają pisarze}
                     while cc<ac do begin       {skoro nie czekają pisarze
                            sygnalizuj (CZYT);  {wpuść czekających czytelników}
                            cc:= cc+1;          {i zapewnij wejście sobie
                     end
```

```
        sygnalizuj_bin (OCHRONA);

        czekaj (CZYT);              {sprawdź, czy możesz wejść do czytelni}
        czytanie;
        czekaj_bin (OCHRONA);       {chroń zmienne w sekcji krytycznej?}
              cc:= cc-1;            {wyszedł czytelnik}
              ac:= ac-1;
              if cc=0 then                  {sprawdź czy czytelnia jest pusta}
                    while pp<ap do begin        {skoro nie czekają czytelnicy
                          sygnalizuj (PIS);     {wpuść czekających pisarzy}
                          pp:= pp+1;
                    end
        sygnalizuj_bin (OCHRONA);
        end
end;

process pisarz (i=1,..P)
begin
        while true do begin
        własne_sprawy;

        czekaj_bin (OCHRONA);               {chroń zmienne w sekcji krytycznej}
              ap:= ap+1;                    {doszedł nowy pisarz}
```

```
            if ac=0 then                {sprawdź czy nie czekają czytelnicy}
                  while pp<ap do begin       {skoro nie czekają pisarze
                        sygnalizuj (PIS);    {wpuść czekających pisarzy}
                        pp:= pp+1;           {i zapewnij wejście sobie
                  end
      sygnalizuj_bin (OCHRONA);

      czekaj (PIS);               {sprawdź, czy możesz wejść do czytelni}
      czekaj_bin (EXCL);
      czytanie;                   {tylko jeden pisarz jest w czytelni!}
      sygnalizuj_bin (EXCL);

      czekaj_bin (OCHRONA);       {chroń zmienne w sekcji krytycznej?}
            pp:= pp-1;            {wyszedł pisarz}
            ap:= ap-1;
            if pp=0 then          {sprawdź czy czytelnia jest pusta}
                  while cc<ac do begin       {skoro nie czekają pisarze
                        sygnalizuj (CZYT);   {wpuść czekających czytelników}
                        cc:= cc+1;
                  end
      sygnalizuj_bin (OCHRONA);
      end
end;
```

Współzawodnictwo

Wzajemne wykluczanie – warunek poprawnej synchronizacji współzawodnictwa

- schemat dostępu do zasobu współdzielonego (*shared resource*)

  …własne sprawy…
  protokół wstępny – naleganie (czekanie na dostęp)
  sekcja krytyczna – realizacja dostępu do zasobu unikatowego
  protokół końcowy – zwolnienie zasobu
  …własne sprawy…
- **warunek bezpieczeństwa** – w każdej chwili w sekcji krytycznej może przebywać tylko jeden proces
- **postulat żywotności** – proces nalegający na pewno uzyska dostęp

Wspomaganie synchronizacji na poziomie architektury rzeczywistej:

instrukcje niepodzielne: testuj i ustaw (*test and set*)
porównaj i przestaw (*compare and swap*)
zamień i dodaj (*exchange and add*)

→ uproszczenie mechanizmów wzajemnego wykluczania

Wzajemne wykluczanie

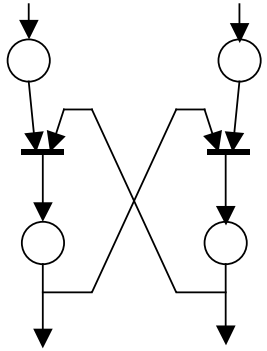
Wzajemne wykluczanie (ang. *mutual exclusion*) – uniemożliwia jednoczesny dostęp procesów do dzielonego zasobu, ale **nie chroni przed**:

- *blokadą* (ang. *deadlock*) (brak bezpieczeństwa)
  → każdy proces z podzbioru procesów jest wstrzymywany w oczekiwaniu na zdarzenie, które może spowodować tylko inny proces z tego podzbioru (uniemożliwienie dostępu do sekcji krytycznej)
- *zagłodzeniem* (ang. *starvation*) (brak żywotności)
  → proces nie zostaje wznowiony, mimo że zdarzenie na które czeka występuje nieskończenie wiele razy (brak dostępu do sekcji krytycznej)

rozwiązania – rekonstrukcja algorytmu

- *zapobieganie blokadzie* – problem wykrycia blokad w programie
- *zapobieganie zagłodzeniu* – „kontroler ruchu" umożliwiający każdemu procesowi wejście do sekcji krytycznej

Schemat blokady

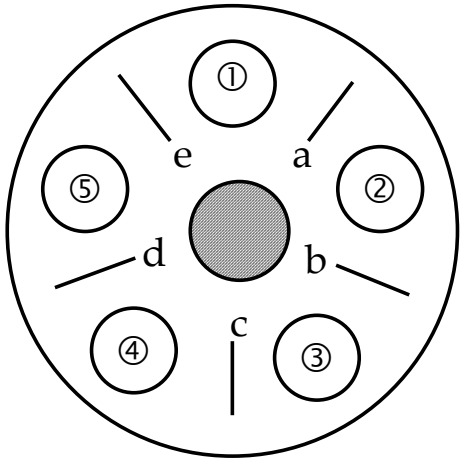
proces A	schemat przepływu (sieć Petri)	proces B
…		…
czytaj XB		czytaj XA
XA:=f(XB)		XB:=g(XA)
zapisz XA		zapisz XB
…		…

! problemu nie można rozwiązać przez wprowadzenie nadzorcy

! należy usunąć blokadę przez zmianę algorytmu

Model zagłodzenia

problem ucztujących filozofów (wersja chińska)



- aby zjeść trzeba użyć 2 pałeczki
- można podjąć tylko 2 pałeczki (nie można „zająć" pałeczki)
- czas jedzenia jest skończony
- po zaspokojeniu głodu obie pałeczki muszą być odłożone

Mimo to jednego można zagłodzić:

zmowa przeciw 1:
sekwencja zagłodzenia 1
rozwiązanie:
ten, który zjadł musi wyjść
i ustawiamy kolejkę

$2(a,b) \mid 4(c,d) \rightarrow 2(a,b) \mid 5(e,d) \rightarrow 3(b,c) \mid 5(e,d)$
$\rightarrow 2(a,b) \mid 5(e,d) \rightarrow 2(a,b) \mid 4(c,d) \ldots$

Mechanizmy synchronizacji procesów

Niskopoziomowe mechanizmy synchronizacji:

- *instrukcje niepodzielne*
- *aktywne oczekiwanie na dostęp* (ang. *busy waiting, spin lock*)
  → naleganie wymagające aktywności procesora
- *blokowanie przerwań*
  → dominacja procesu aktywnego – sekcja krytyczna = czas procesora
- *semafor* – zmienna współdzielona.

Mechanizmy systemu operacyjnego:

- *instrukcje warunkowe*
- *kolejka* – zgłoszenie żądania i oczekiwanie na potwierdzenie
  → kolejka prosta – wyklucza zagłodzenie, uniemożliwia priorytety
  → kolejka z priorytetami – nie wyklucza ryzyka zagłodzenia
  → kilka kolejek – wymaga ustalenia zasad sprawiedliwości
- *semafory*
  → algorytm Dijkstry
- *monitory, muteksy, futeksy*

Przerwania

Sygnalizacja zdarzeń zewnętrznych wymagających obsługi programowej
Upływ czasu – impuls generowany przez zliczenie cykli zegara procesora
Zablokowanie przerwań (polecenie *disable interrupt, DI*)
- uniemożliwienie zgłoszenia żądań obsługi

Synchronizacja przez blokadę przerwań:

- każde zdarzenie synchronizujące jest sygnalizowane przerwaniem
- podjęcie obsługi zdarzenia rozpoczyna polecenie zablokowania *DI*
- do zakończenia obsługi niemożliwe zgłoszenie innych zdarzeń
- zakończenie obsługi – odblokowanie przerwań

Wady:

- dominacja procesu, który podjął obsługę
- wykluczenie możliwości synchronizacji procesów niezwiązanych
  – ignorowane żądania synchronizacji
  – oczekiwanie na odblokowanie przerwań

Aktywne oczekiwanie (1)

test-and-set

```
wait:  tas  lock          ; test dostępności dzielonej pamięci
       bmi  wait          ; testuj ponownie, gdy zablokowane
       ...                ; sekcja krytyczna
       clr.b  lock        ; zwolnij dostęp dla innych procesów
       ...
```

compare-and-swap

```
       moveq  #$80, d2    ; ustaw zmienną testującą (1000 0000B)
loop:  clr.w  d1          ; przygotuj blokadę
       cas.w  d1, d2, lock ; zablokuj, gdy jest dostęp
       bne  loop          ; powtórz testowanie gdy zablokowane
       ...                ; sekcja krytyczna
       clr.w  lock        ; zwolnij dostęp dla innych procesów
```

Synchronizacja metodą aktywnego oczekiwania (*busy waiting*)
(Motorola 68020+)

Aktywne oczekiwanie (2)

test-and-set

```
        movl  zakaz, %eax          ; przygotuj blokadę
wait:   xchg  %eax, dostep         ; test dostępności dzielonej pamięci
        cmpl  zakaz, %eax
        je  wait                   ; testuj ponownie, gdy zablokowane
        ...                        ; sekcja krytyczna (blokada dostępu)
        movl  %eax, wolne          ; zwolnij dostęp dla innych procesów
        movl %eax, dostep
```

compare-and-swap

```
        ...
        movl  zakaz, %ebx          ; przygotuj blokadę
wait:   movl  klucz, %ebx          ; przygotuj klucz dostępu
        cmpxchg  %ebx, dostep      ; zablokuj, gdy jest dostęp (eax=dostep)
        jne  wait                  ; powtórz testowanie gdy zablokowane
        ...                        ; sekcja krytyczna
        movl  klucz, %eax          ; przygotuj klucz dostępu
        movl  %eax, dostep         ; zwolnij dostęp dla innych procesów
```

Synchronizacja metodą aktywnego oczekiwania (Intel 80x86)

Kolejki i semafory

Kolejka – ustalona sekwencja umożliwienia dostępu do sekcji krytycznej

- testuj: ***zakaz***: wstrzymaj akcję, ***zezwolenie***: wejdź do sekcji krytycznej

Semafor – ustalona reguła dostępu procesów oczekujących do sekcji krytycznej

- semafor binarny – dwustanowy (podniesiony – opuszczony)
  → odczytaj i blokuj (IA-32/Pentium: *xchg* lub *cmpxchg*)
  - o jeśli zablokowany wstrzymaj akcję,
    jeśli odblokowany wejdź do sekcji krytycznej

- semafor ogólny – wielostanowy, realizacja różnych uprawnień dostępu
  → odczytaj i blokuj (IA-32/Pentium: *cmpxchg* lub *xadd*)
  - o jeśli za nisko przywróć stan i wstrzymaj akcję
  - o jeśli zezwolenie opuść o jednostkę i wejdź do sekcji krytycznej

  → wykonaj sekcję krytyczną
  → podnieś o jednostkę (IA-32/Pentium: *cmpxchg* lub *xadd*)

Funkcje systemu operacyjnego

funkcje systemowe (ang. *system functions*)

- funkcje *nadzorowania* procesu
  - tworzenie (ang. *creation*) środowiska wykonania procesu
  - aktualizacja struktur danych procesu (ang. *context update*)
  - synchronizacja i przełączanie (ang. *switching*) procesów
  - szeregowanie (ang. *scheduling*)
  - raportowanie (ang. *accounting*)
- funkcje *ochrony* procesu
  - przydział zasobów (ang. *resource allocation*)
  - ochrona zasobów (ang. *resource protection*)
  - zarządzanie pamięcią (ang. *memory management*)
  - obsługa wyjątków (ang. *exception handling*)

wspomaganie użytkownika (ang. *user functions*)

- sterowanie i utrzymanie kontroli nad programem (ang. *program control*)
- obsługa wejścia/wyjścia (ang. *I/O handling*)
- obsługa plików (ang. *file system manipulation*)

System operacyjny