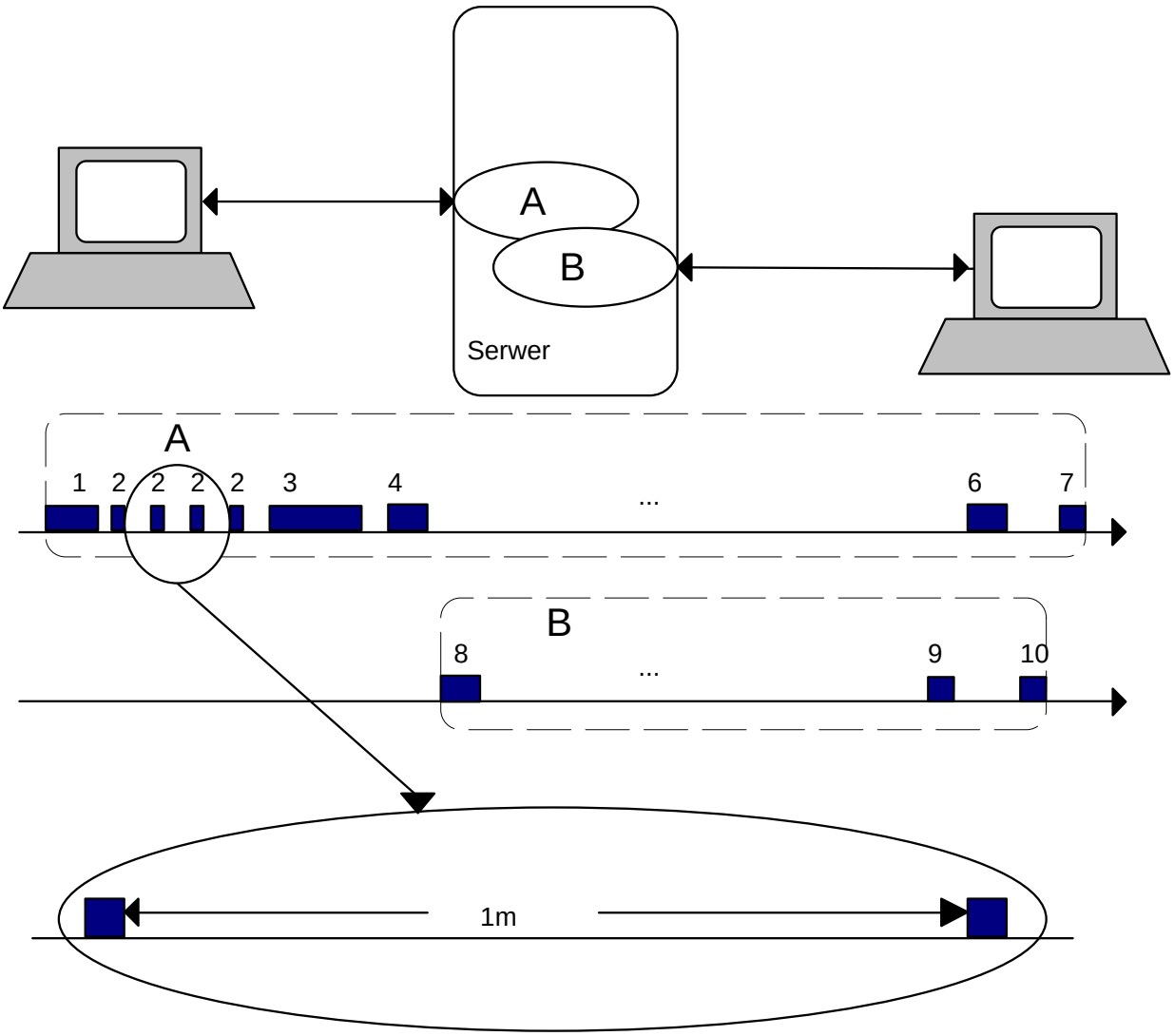
Dodatek A. Środowisko oprogramowania komunikacyjnego

Jak są obsługiwane przez serwer jednocześnie (współbieżnie) dwa terminale (stacje robocze) pracujące w sieci?



Rys 1 Ilustarcja współbieżności procesów

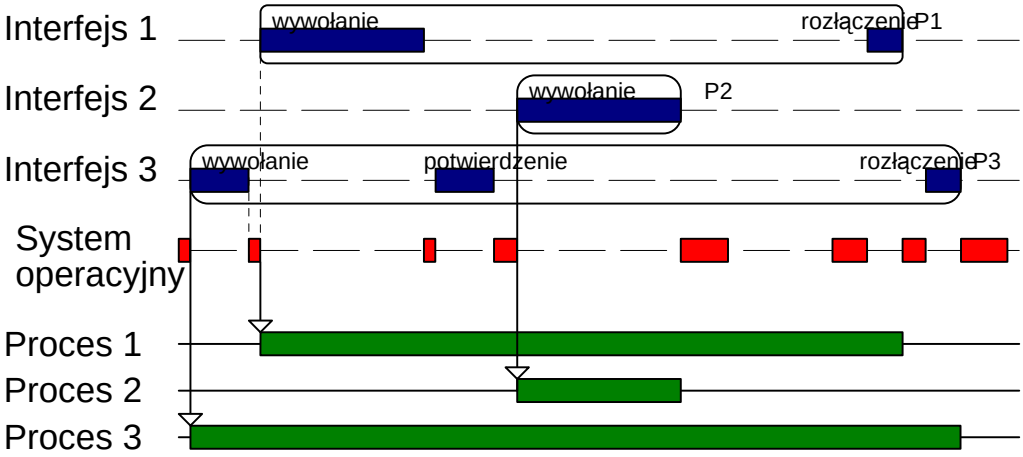
Trzeba pamiętać, że w serwerze jest równolegle nawiązywanych i rozłączanych wiele połączeń, ale byłoby to rażącą rozrzutnością każdej aktywnej parze abonentów przydzielić oddzielny procesor z pamięcią.

A.1. Procesy współbieżne: model i struktura

Jedynym rozwiązaniem jest potraktowanie wszystkich fragmentów programu niezbędnych do realizacji jednego celu jako całości. Całość ta jest nazywana **procesem**. Każdy proces działa "niezależnie" od pozostałych procesów i może:

- współpracować z innymi procesami poprzez wymianę komunikatów i informacji synchronizujących oraz
- współzawodniczyć o wszystkie zasoby systemu komputerowego, włączając w to również czas procesora.

Przykład 1:



Rys 2 Przykładowe procesy

A.2. Definicja procesu

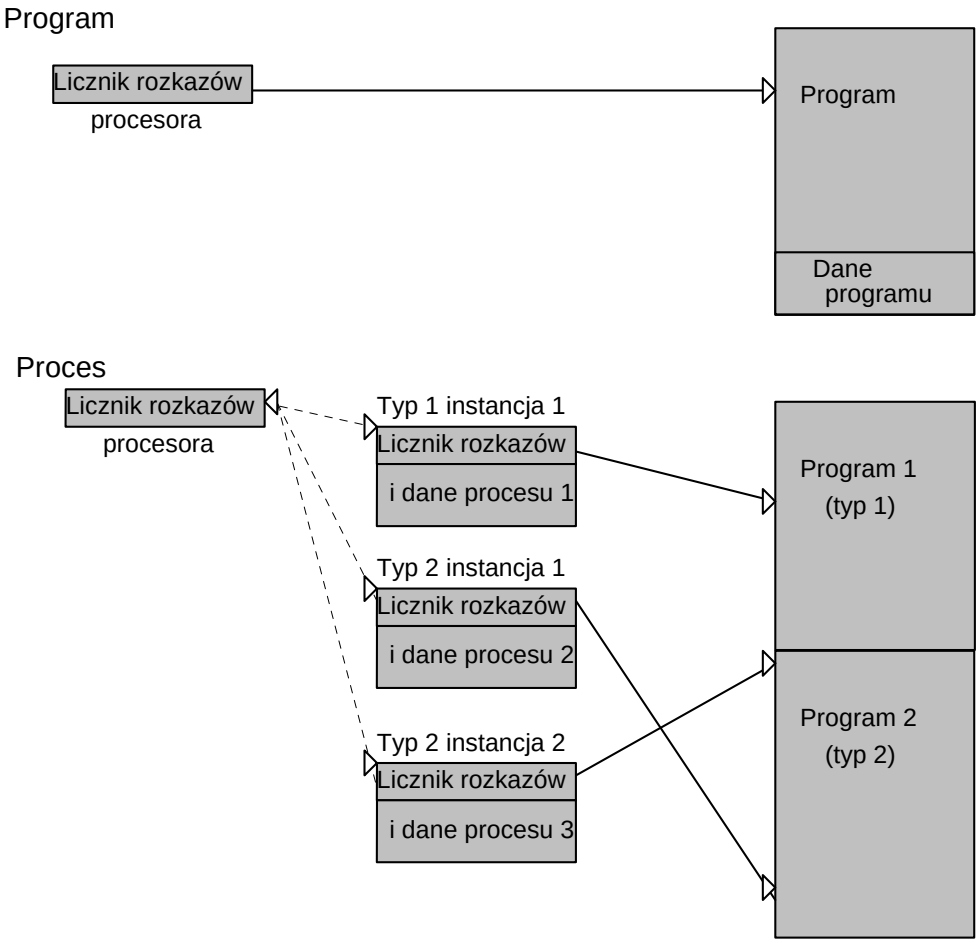
Proces (instancja procesu) jest ciągiem wykonań instrukcji jego programu, wyznaczonych kolejnymi wartościami jego licznika rozkazów, wraz z jego danymi, wskaźnikiem stosu i stosem.

Proces jest obiektem mogącym wykonywać przypisany mu program. Ten sam program może wykonywać kilka procesów, na tych samych lub różnych danych oraz "w tym samym" lub czasie "różnym czasie". Jeśli czasy wykonywania procesów, licząc od momentu rozpoczęcia procesu do momentu zakończenia procesu, pokrywają się, to mówimy o ich współbieżności. Mówiąc dalej o procesie będę miał na myśli jego **instancję**. Tekst programu przypisanego procesowi określa **typ** procesu.

Przykład 2:*(ilustrujący pojęcie procesu)*
Uczący się student jest procesorem. Następnego dnia ma kolokwium z fizyki. Czytanie podręcznika z fizyki jest procesem. Tekst podręcznika jest programem w tym procesie. W trakcie czytania student robi notatki. Notatki to dane i stos tego procesu. W pewnym momencie natrafia na fragment wymagający przypomnienia sobie sposobu obliczania pewnej całki. Student wkłada do książki z fizyki zakładkę i odkłada ją na bok pozostawiajc jednak czysty papier na notatki z matematyki. Bierze książkę z analizy matematycznej i rozwiązuje potrzebną całkę. Po jej wyznaczeniu odkłada książkę z analizy i bierze ponownie książkę z fizyki. Otwiera na stronie zaznaczonej zakładką i na podstawie notatek (zarówno z fizyki jak i z matematyki) może kontynuować uczenie się fizyki. Zakładka jest w tym przykładzie licznikiem rozkazów. Pojawienie się w drzwiach pokoju kolegi zapraszającego do baru jest typowym przerwaniem. Może to być przerwanie o krótkim albo długim czasie obsługi, w zależności od tego czy student zrezygnuje z zaproszenia czy też je przyjmie.

Typ procesu - podręcznik;
Instancja procesu - czytanie tej samej książki przez kilku studentów.

Nowe elementy:
- Stany procesu
- Komunikacja między procesami (fizyka - matematyka)
- Rola przełączania i szeregowania, czas przełączania itp.
- Współzawodnictwo o czas procesora



Rys.3 Ilustracja różnicy pomiędzy pojęciem programu i procesu

A.3. Stany procesu

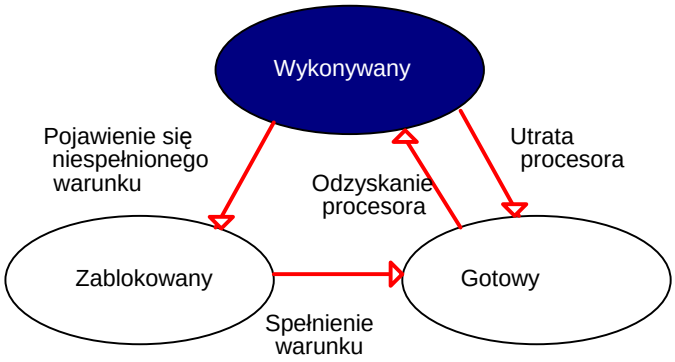
Procesy mogą znajdować się w następujących stanach:

Stan wykonywania
W stanie wykonywania w danej chwili może znajdować się tylko jeden proces. Jest to stan, w którym proces posiada procesor.

Stan gotowości
Procesy znajdujące się w stanie gotowym przeznaczone są do wykonania, gdy tylko ich priorytet będzie dostatecznie wysoki. Są to procesy oczekujące tylko na zwolnienie procesora.

Stan zablokowania (zawieszenia)
Procesy zawieszone oczekują na spełnienie warunków np. upływu czasu, dostępu do wspólnych zasobów znajdują się one w stanie oczekiwania.

Graf przejść pomiędzy stanami procesu ilustruje rysunek.



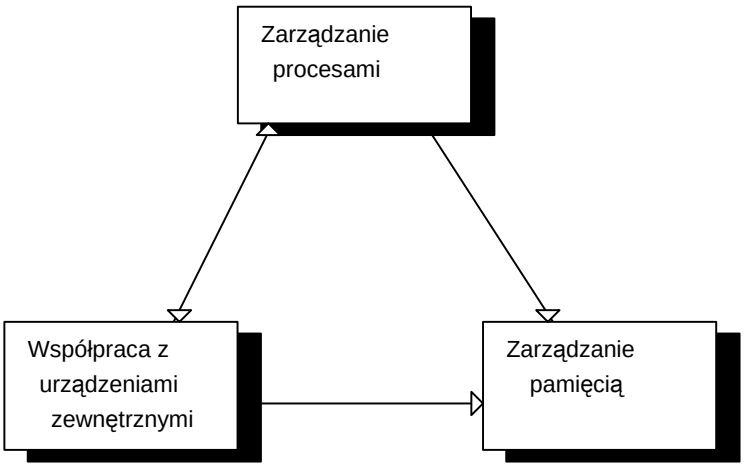
Rys. 4 Graf przejść pomiędzy stanami procesu

A.4. Usługi systemu operacyjnego

System operacyjny powinien dostarczać środków do nadzorowania zmian stanu procesów, środków do zarządzania procesami. Typowe środowisko procesów telekomunikacyjnych oferuje następujące funkcje:

- utworzenie procesu
- zakończenie procesu
- oczekiwanie na upływ czasu
- wysłanie komunikatu
- oczekiwanie na komunikat
- przydział pamięci procesowi
- odczytanie własnego identyfikatora
- zmiana priorytetu procesu itp.

Zadanie zarządzania procesami realizuje część systemu operacyjnego nazywana schedulerem (programem szeregującym). Podstawą działania schedulera są opisane dalej deskryptory procesów pozwalające identyfikować procesy oraz zapewnić im bezpieczne przejścia między stanami. Rysunek ilustruje sposób współdziałania modułów systemy operacyjnego.



Rys. 5 Elementy systemu operacyjnego

Istnieją systemy, w których funkcja utworzenia procesu nie jest oferowana. W systemach operacyjnych tego typu mamy do czynienia z procesami statycznymi. W momencie generacji systemu tworzone są wszystkie procesy, które będą istniały do czasu zakończenia pracy systemu. W systemach, w których funkcja tworzenia procesów jest oferowana musi istnieć przynajmniej jeden proces statyczny. Procesy generowane przy pomocy funkcji utworzenia procesu nazywane są procesami dynamicznymi.

Realizacja tych funkcji, a w szczególności nadzorowanie czasu trwania procesów, oraz gospodarka zasobami pamięci są tymi elementami, dzięki, którym możemy powiedzieć, że oprogramowanie komunikacyjne może funkcjonować tylko w środowisku systemu operacyjnego czasu rzeczywistego.

A.5. Identyfikacja procesów

Z każdym procesem związany deskryptor procesu, zawierający następujące informacje:

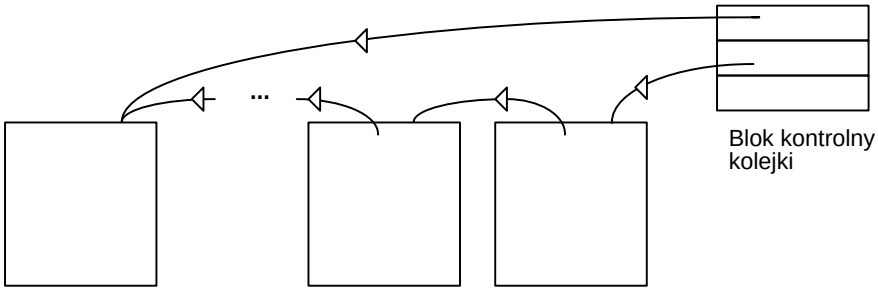
Identyfikator procesu
Licznik rozkazów procesu
Wskaźnik stosu procesu
Stos procesu
Priorytet procesu
Odwołania do posiadanych zasobów np. pamięci, adresy przekazywanych wiadomości
Żądane zasoby np. odmierzanie określonego odstępu czasu
Zużyty czas procesora
Powiązania zawierające identyfikatory procesów rodziców i procesów potomnych
Itp. np. adres następnego procesu w kolejce

Rys. 6 Identyfikator procesu

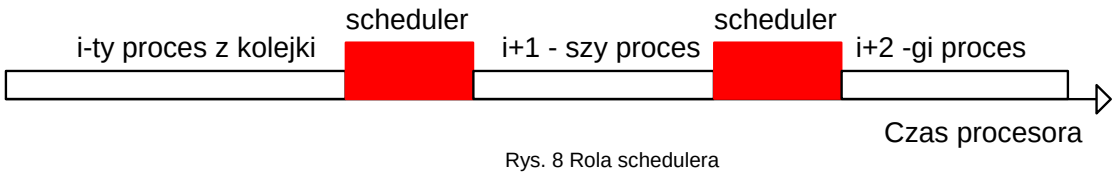
Deskryptor procesu bywa też nazywany blokiem kontrolnym bądź wektorem stanu procesu. Kolejnośc umieszczenia informacji w deskryptorze nie jest zasadnicza jednak można wyróżnić minimalną zawartość deskryptora. Są to cztery pierwsze elementy deskryptora oddzielona od opcjonalnej reszty linią przerywaną.

A.6. Przydział procesora

Podstawą przydziału procesora jest kolejka procesów gotowych (deskryptorów tych procesów) oczekujących na procesor (rys 7) . Kolejka jest obsługiwana przez proces nazywany schedulerem (planistą). Proces ten pobiera z kolejki deskryptory procesów według charakteryzujących scheduler zasad i przydziela pobranemu procesowi procesor (rys.8). W podobnych kolejkach oczekują na spełnienie warunków dalszej realizacji deskryptory procesów zablokowanych.



Rys. 7 Kolejka deskryptorów procesów oczekujących na przydział procesora



Rys. 8 Rola schedulera

Wywłaszczenie procesu - odebranie procesowi procesora w wyniku zajścia zdarzenia zewnętrznego w stosunku do procesu

Alorytmy szeregowania procesów

1. kolejkowy bez wywłaszczania
2. okrężny (round-robin) (z wywłaszczaniem)
3. priorytetowy - statyczny i dynamiczny (bez wywłaszczaniem i z wywłaszczaniem)

Wady i zalety tych algorytmów:

ad. 1.

- - nieznany czas oczekiwania na procesor
- + prostota

ad. 2.

- - wywłaszczanie
- - niesprawiedliwość

ad.3.

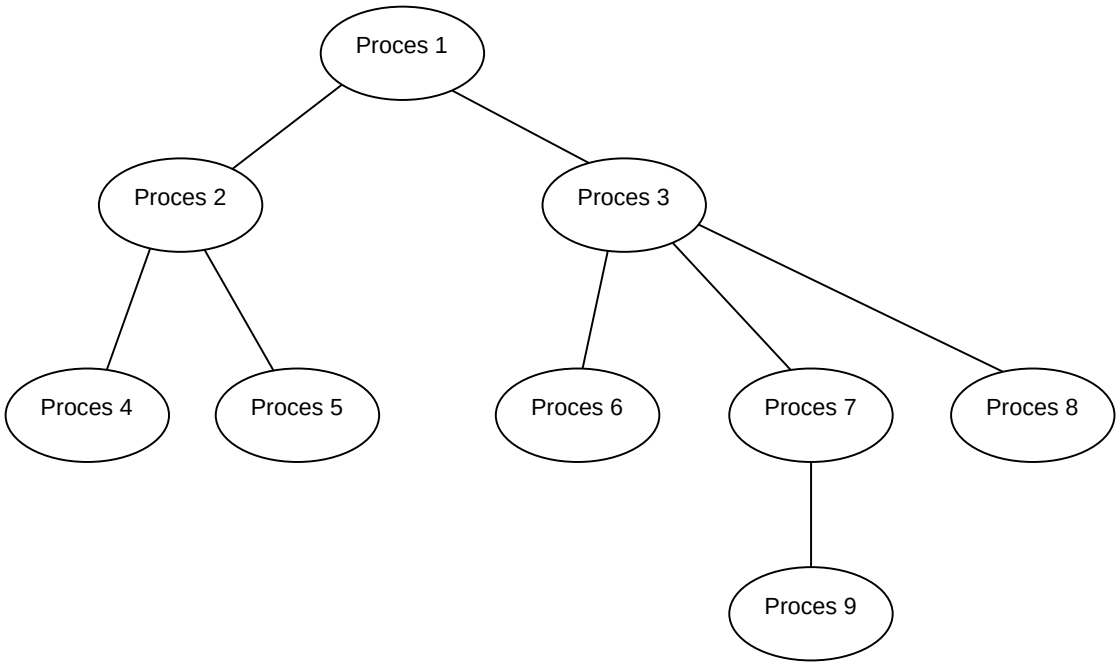
- - złożoność
- - wywłaszczanie
- + sprawiedliwość

A.7. Hierarchia procesów dynamicznych

Operacje kreacji procesu i zakończenia procesu pozwalają na tworzenie hierarchii procesów dynamicznych. Proces, który zrealizuje funkcję utworzenia nowego procesu jest nazywany procesem rodzica, utworzony przez niego proces jest nazywany procesem potomka. Pojawia się też dodatkowa funkcja zakończenia procesu potomnego.

Rysunek ilustruje przykładową hierarchię procesów.

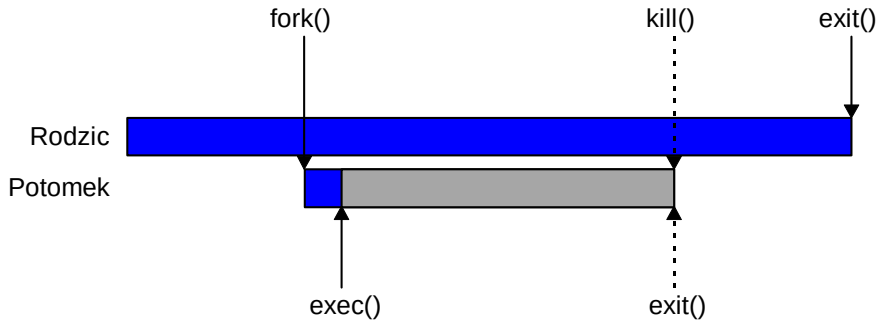
W grupie procesów będących potomkami tego samego rodzica istnieje pełna wiedza o wszystkich członkach "rodziny".
Istnieją też różnego rodzaju ograniczenia w komunikacji i likwidacji procesów.



Rys. 9 Ilustracja hierarchii procesów

UNIX - funkcje:

fork() - utworzenie kopii procesu rodzica
exec() - zmiana programu (kodu) procesu
exit() - zakończenie procesu



Rys. 10 Generowanie procesów w systemie NIX

A.8. Komunikacja pomiędzy procesami

Procesy mogą wzajemnie się komunikować przekazując sobie wiadomości. Wiadomość w najprostszym przypadku jest rekordem o następującej strukturze:

```
Type
  (* Typ identyfikatora procesu *)
  Typ_Id     : Array[1..Idmax] Of Byte;
  Wiadomosc Record
    Nadawca  : Typ_Id;
    Odbiorca : Typ_Id;
    Tresc    : Array [1..Nmax] Of Byte;
  End;
```

Do nadawania wiadomości przeznaczona jest procedura:

Send(W : Wiadomosc);

Natomiast do odbioru wiadomości przeznaczona jest procedura:

Receive(W : Wiadomosc);

Wyróżnia się przy tym dwa podstawowe mechanizmy komunikacji:

- komunikacja bez potwierdzeń i
- komunikacja z potwierdzeniami.

W pierwszym przypadku proces nadawcy nie oczekuje na przekazanie komunikatu procesowi odbiorcy, lecz kontynuuje swoją realizację. W drugim przypadku proces nadawcy jest zawieszany do czasu otrzymania potwierdzenia odbioru wysłanego komunikatu. Drugi mechanizm wykorzystywany jest przede wszystkim w środowisku rozproszonym (wieloprocesorowym), gdy z procesem związany jest dodatkowo procesor. Natomiast mechanizm pierwszy jest wykorzystywany przede wszystkim w środowisku zwartym opartym o jeden procesor i przy pewnych ograniczeniach na zasięg komunikacji miedzy procesami (komunikacja ograniczona do "rodziny"). Taki sposób komunikacji procesów jest nazywany w literaturze angielsko języcznej message passing.

Mechanizm ten jest ekwiwalentny z mechanizmami komunikacji procesów opartymi o semafory oraz monitory. Zapewnia komunikację bez zagrożenia wzajemnym wykluczeniem. W odróżnieniu od obu wymienionych mechanizów komunikacji można go stosować róznież w środowisku rozproszonym. W środowisku tym identyfikator procesu ma identyfikator rozszerzony o nazwę systemu komputerowego, w którym jest wykonywany, o nazwę grupy systemów:

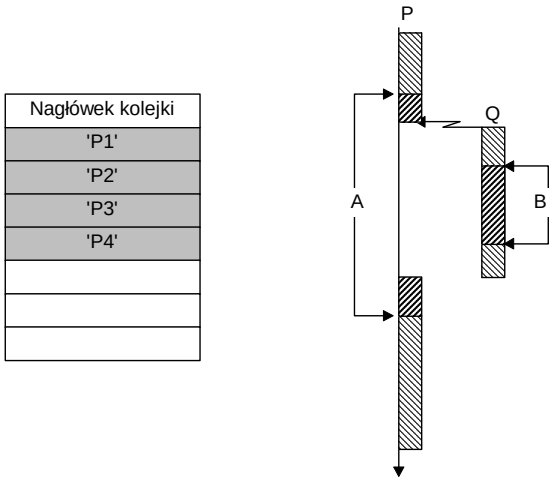
Proces@Komputer.Lokalizacja

Operacje tego typu są już dostępne w systemie operacyjnym UNIX.

A.9. Dostęp do wspólnych danych

Ilustracja problemu:

Spooler - kolejka z nazwami plików do drukowania. Proces P zapisuje do kolejki nazwę pliku 'P5' do drukowania. Odpowiadającej tej operacji fragment programu jest oznaczony literą A. W trakcie realizacji fragmentu A pojawiło się przerwanie Q, którego obsługa polega na wpisaniu do tej samej kolejki nazwy kolejnego pliku do drukowania.



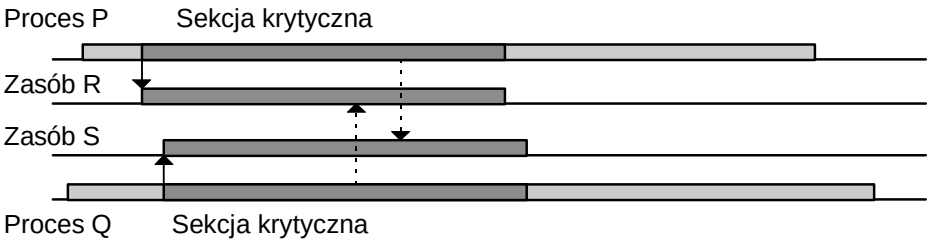
Rys. 11 Dostęp do wspólnego zasobu

Pojawia się pytanie: z jakimi danymi będzie kontynuowany proces P po wykonaniu obsługi przerwnia Q? Rozwiązaniem problemu jest przyjęcie założenia, że fragment procesu, który korzysta ze zmiennych globalnych (wspólnych dla wielu procesów) nie powinien być przerwany. Tylko jeden proces może mieć dostęp do zmiennej globalnej, aż do czasu zakończenia operacji na tej zmiennej. Fragment A procesu P i fragment B procesu Q **wykluczają się wzajemnie**.Oba fragmenty nazywamy **sekcjami krytycznymi** procesów P i Q. Najprostszym rozwiązaniem jest zablokowanie przerwań na czas trwania sekcji krytycznej, lecz rozwiązanie to niepotrzebnie blokuje całą komunikację z otoczeniem, mimo że procesy wielu potencjalnych przerwań nie będą korzystały z tej zmiennej globalnej.

Uogólnienie:
Mamy do dyspozycji dwa zasoby: R i S

```
Proces P
Begin
     żądanie R        (* A *)
     korzystanie z R  (* A *)
     żądanie S        (* A *)
     korzystanie z R i S
     zwolnienie R
     zwolnienie S
End;
```

```
Proces Q
 Begin
  żądanie S          (* B *)
  korzystanie z S    (* B *)
  żądanie R          (* B *)
  korzystanie z R i S
  zwolnienie S
  zwolnienie R
 End;
```



W rozwiązaniu problemu sekcji krytycznej przyjęto kilka wymagań, które powinny być spełnione przez to rozwiązanie:

1. wymaganie bezpieczeństwa - dwa procesy nie mogą jednocześnie znajdować się w sekcji krytycznej,
2. wymaganie żywotności - jeśli proces zamierza wejść do sekcji krytycznej, to po skończonym czasie z niej wyjdzie.
3. wymaganie asynchroniczności - nie można niczego zakładać o względnych prędkościach wykonywania się programów poszczególnych procesów (np. zakładać, że obsługa klawiatury jest wolniejsza niż obsługa operacji dyskowej).

Rozwiązaniem problemu wzajemnego wykluczania jest semafor. Semafor jest zmienną całkowitą o nieujemnych wartościach. Po nadaniu semaforowi wartości początkowej jedynymi dozwolonymi operacjami są:

wait(semafor) - czekaj na semafor, oznaczana też w literaturze jako P(semafor),

signal(semafor) - prześlij sygnał do s, oznaczana jako V(semafor).

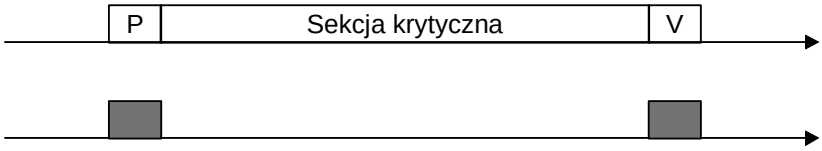
Definicje operacji semaforowych

wait (s) - if s>0 then s:=s-1
else "zawieszenie";

signal (s) - if "istnieje proces zawieszony przez s"
then "uruchomienie tego procesu"
else s:=s+1,

Operacje te są **operacjami atomowymi**, tzn. są nieprzerywalne. Dzięki temu, że tylko te operacje są nieprzerywalne komunikacja z otoczeniem jest zablokowana tylko na czas ich realizacji.

Ilustracja:



Blokowanie systemu przerwań

Przykład wykorzystania:

```
var s:semaphore;
process p1;
  begin
     wait(s);
     s_kryt1;
     signal(s);
     lok_1;
  end;

process p2;
  begin
     wait(s);
     s_kryt2;
     signal(s);
     lok_2;
  end;

begin
  s:=1;
  p1;
  p2;
end;
```

Proces oczekujący na dostęp do sekcji krytycznej jest w stanie zablokowania i jego deskryptor musi być zapisany w kolejce związanej z semaforem chroniącym zasobu.

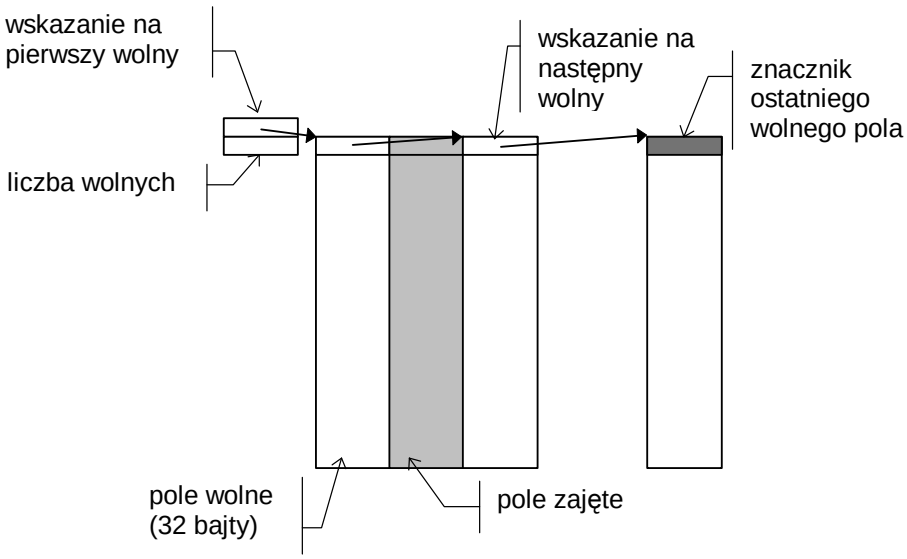
A.10. Moduł zarządzania pamięcią operacyjną procesów

Oferowane funkcje:

void inicjuj (ns_pole w_pole);
ns_pole pobierz (ns_pole w_pole);
void zwróć (ns_pole w_pole);

Struktura tablicy pól:

```
struct s_pola {
        struct s_pola *ns_pole;
        char  pole[32];
        } pola[512];
```



Plik nagłówkowy (*.h)

```
#define l_pol 5
struct s_pola {
 struct s_pola *ns_pole;
 char pole[32];
 } pola[l_pol+1];

struct s_pola *w_pierwsze;
int l_wolnych;
void inic(void);
struct s_pola* pobierz(void);
void zwroc(struct s_pola *z_pole);
```

Plik podstawowy

```
#include <stdio.h>;
#include <c:\docs\unimor\c\centrala\pamiec.h>

void inic()
{
  int n;

  w_pierwsze=&pola[0];
  for (n=1;n<l_pol;n++)
    pola[n-1].ns_pole=&pola[n];
  pola[l_pol].ns_pole=NULL;
  l_wolnych=l_pol+1;
}

struct s_pola* pobierz()
{
  struct s_pola *chwil;

  chwil=w_pierwsze;
  if (w_pierwsze!=NULL)
  {
    w_pierwsze=w_pierwsze->ns_pole;
    l_wolnych--;
  }
  return(chwil);
}
```

```
void zwroc(struct s_pola *z_pole)
{
  z_pole->ns_pole=w_pierwsze;
  w_pierwsze=z_pole;
  l_wolnych++;
}

main()
{
  struct s_pola *pobrane[3];

  inic();
  pobrane[0]=pobierz();
  zwroc(pobrane[0]);
  pobrane[0]=pobierz();
  pobrane[1]=pobierz();
  zwroc(pobrane[1]);
  zwroc(pobrane[0]);

  return 0;
}
```