Procesy

Pojęcie procesu

System operacyjny może wykonywać programy w różny sposób:

- System wsadowy — zadania (jobs)
- System z podziałem czasu — programy użytkownika (user programs) albo prace (tasks).

Terminy zadanie (job) i proces stosowane są niemal zamiennie.

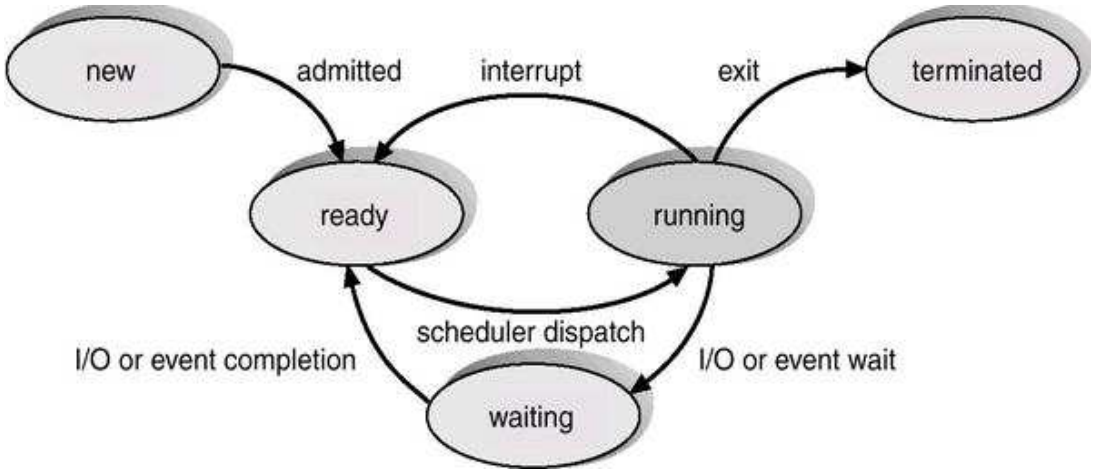
Proces — wykonujący się program. Wykonanie przebiega w sposób sekwencyjny.

Proces zawiera między innymi:

- sekcja kodu programu
- licznik rozkazów
- zawartość rejestrów procesora
- stos
- segment danych

Stan procesu

W trakcie wykonywania proces zmienia swój stan:



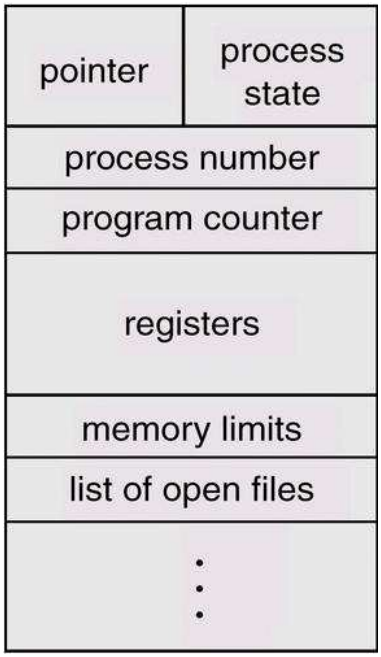
- nowy: proces został właśnie stworzony,
- wykonywany(aktywny): są wykonywane instrukcje programu,
- oczekujący: proces czeka na wystąpienie zdarzenia (np. zakończenie operacji we/wy),
- gotowy: czeka na przydział procesora,
- zakończony: zakończył właśnie działanie.

Blok kontrolny procesu — PCB

Każdy proces w systemie jest reprezentowany przez strukturę nazywaną blokiem kontrolnym procesu. Zawiera on wszystkie informacje związane z procesem.

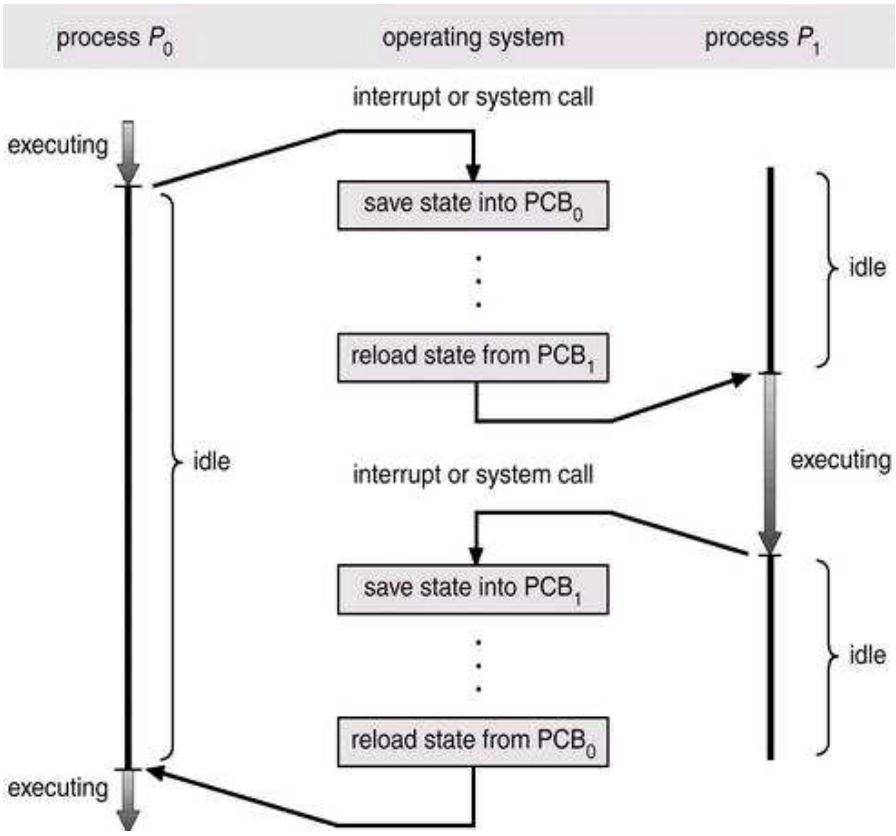
- stan procesu,
- licznik rozkazów,
- zawartość rejestrów procesora,
- informacja o szeregowaniu (sposób szeregowania, priorytet, wskaźniki do kolejek, itp),

- informacja o zarządzaniu pamięcią procesu (zależy od sposobu organizacji pamięci w systemie),
- informacje związane z rozliczeniami (zużycie zasobów, limity, konta, itp.),
- Informacje związane z we/wy (urządzenia, otwarte pliki, itp.)



Przełączanie procesora

Procesor może być przydzielany przemiennie różnym procesom.



Szeregowanie (planowanie) procesów

System utrzymuje struktury danych grupujące procesy. Są to listy albo kolejki procesów.

Zmiana stanu procesu związana jest zwykle ze zmianą kolejki (listy).

Przykładowe kolejki to:

- kolejka zadań — zawiera zwykle wszystkie procesy w systemie,
- kolejka procesów gotowych — zawiera procesy znajdujące się w pamięci i czekające na procesor,
- kolejki urządzeń — procesy czekające na urządzenia we/wy.

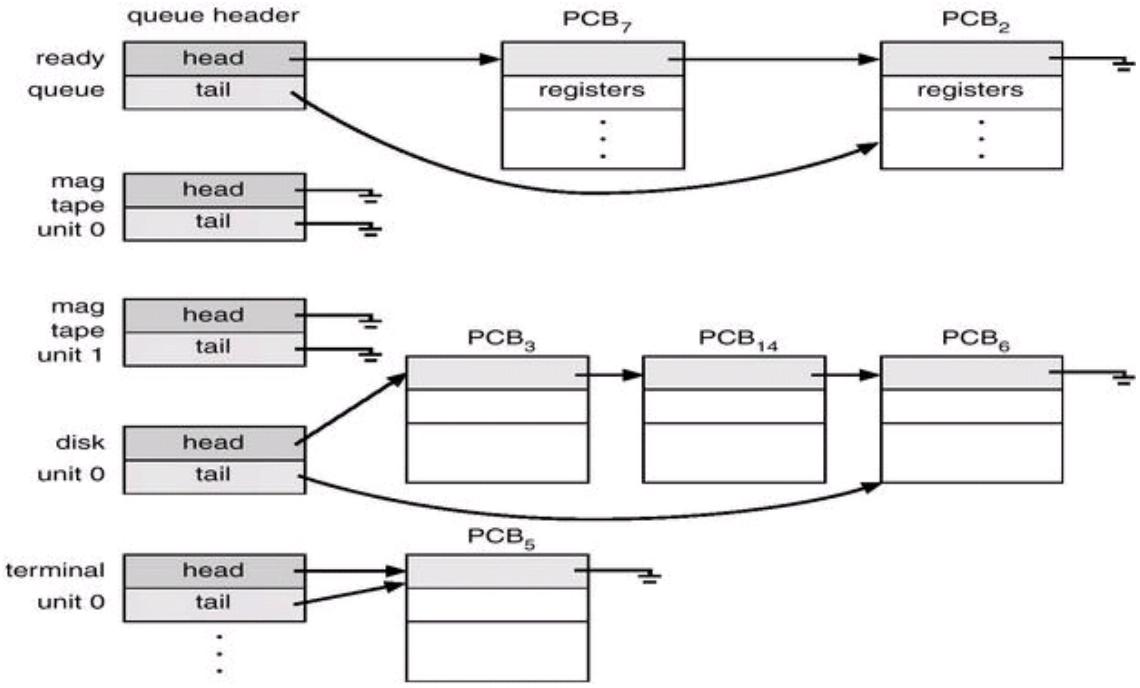
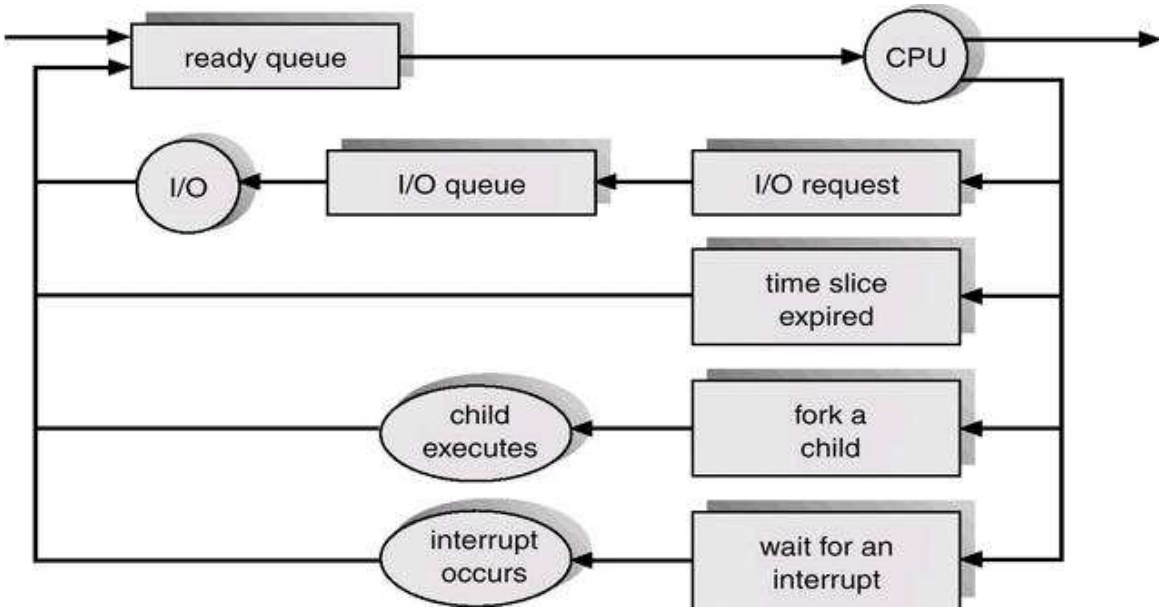


Diagram kolejek w planowaniu procesów



Planiści

- Planista długoterminowy: wybiera zadania do załadowania do pamięci i umieszczenia w kolejce gotowych do wykonania.
- Planista krótkoterminowy: wybiera zadanie z kolejki procesów gotowych i przydziela mu procesor.

Planista krótkoterminowy działa bardzo często (musi być szybki).

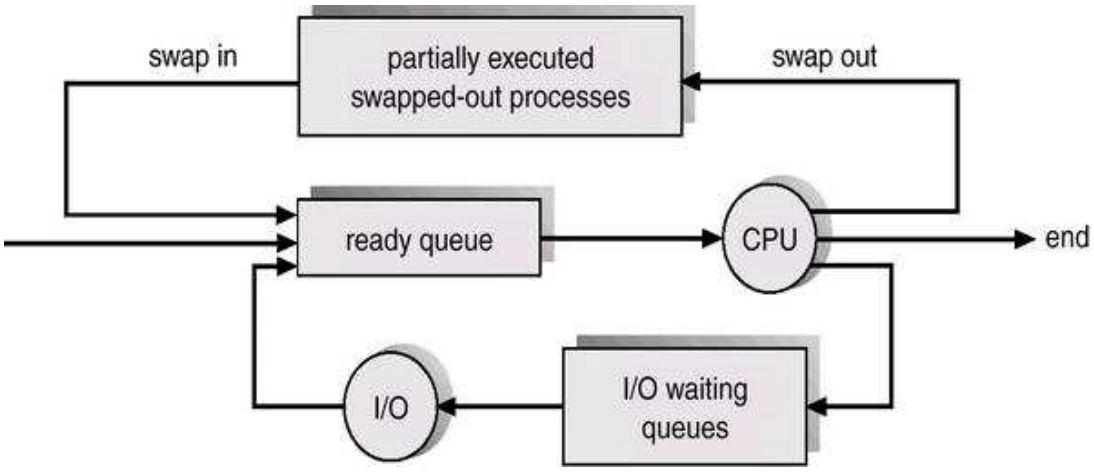
Planista długoterminowy działa znacznie rzadziej (może być wolniejszy). Decyduje o poziomie wieloprogramowości.

Procesy bywają albo:

- ukierunkowane na we/wy — dużo krótkich odcinków wykorzystania CPU,
- ukierunkowane na obliczenia — niewiele długich odcinków wykorzystania CPU.

Planista musi odpowiednio dobrać proporcje.

Niekiedy występuje trzeci rodzaj planisty — średnioterminowy: wybiera procesy do czasowego usunięcia z pamięci.



Przełączanie kontekstu

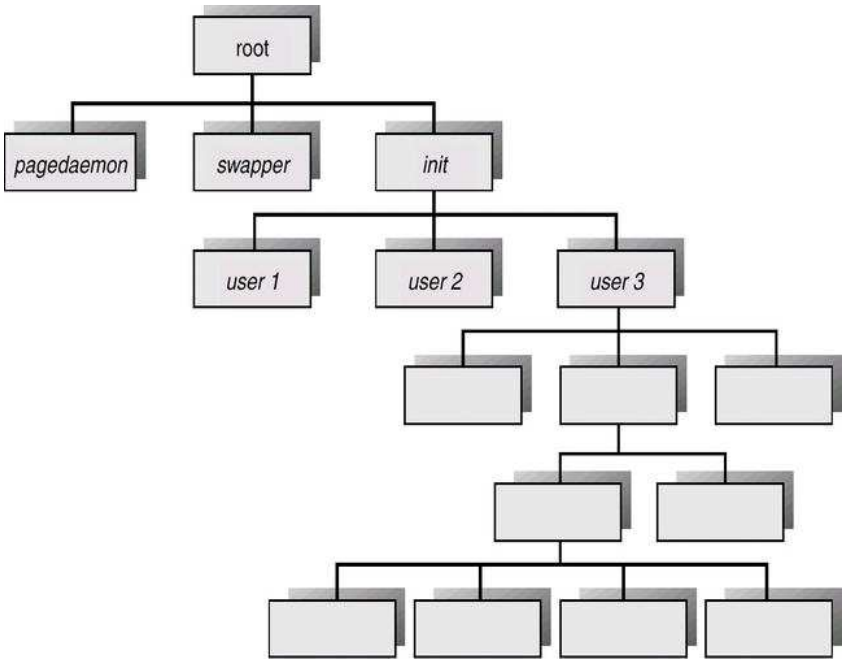
W momencie przekazywania procesora innemu procesowi system musi zapamiętać stan przerywanego procesu i odtworzyć stan procesu wznawianego.

Ta operacja (przełączanie kontekstu) stanowi narzut — w tym czasie procesor nie wykonuje innej użytecznej pracy.

Czas operacji zależy od stopnia wsparcia sprzętowego.

Tworzenie procesu

Proces macierzysty (ang. parrent) tworzy procesy potomne (children), które z kolei tworzą następne.



Różne systemy różnie traktują sprawę wyposażenia nowego procesu w zasoby (CPU, pamięć, otwarte pliki, urządzenia we/wy):

- rodzic i potomek współdzielą wszystkie zasoby,

- potomek współdzieli podzbiór zasobów rodzica,
- potomek i rodzic nie współdzielą zasobów.

Dwa podstawowe warianty sposobu wykonania:

- proces macierzysty czeka na zakończenie pracy wszystkich lub części procesów potomnych,
- procesy rodzica i dziecka wykonują się współbieżnie.

Podobnie dwie możliwe strategie traktowania przestrzeni adresowej:

- proces potomny jest kopią procesu macierzystego,
- Proces potomny ładuje i wykonuje nowy program.

W systemie UNIX stworzenie procesu potomnego wykonującego inny program jest dwuetapowe:

- `fork` — tworzy identyczną kopię procesu. Oba procesy wykonywane są od miejsca wywołania `fork`,
- `exec` — powoduje zastąpienie przestrzeni adresowej procesu kodem innego programu (funkcja, z której nie ma powrotu !).

Taki tryb umożliwia wykonanie pewnych czynności pomiędzy wywołaniem `fork` i `exec`.

Kończenie procesu

Proces wykonuje ostatnią instrukcję i za pomocą `exit` zleca systemowi operacyjnemu usunięcie go:

- w tym momencie może nastąpić przekazanie danych do procesu macierzystego (musiał on użyć funkcji `wait`),

- system operacyjny odbiera procesowi wykorzystywane przez niego zasoby.

Możliwe jest zakończenie awaryjne (`abort`) procesu zwykle przez proces przodka (ojca, dziadka, itd.).

Konieczna jest do tego znajomość identyfikatora procesu — w momencie tworzenia potomka jego identyfikator przekazywany jest rodzicowi.

Przyczyny mogą być różne:

- potomek nadużył przydzielonych zasobów,
- zadanie realizowane przez potomka stało się zbędne,
- proces macierzysty kończy się (wtedy system nie pozwala na działanie potomków - zakończenie kaskadowe).

Procesy współpracujące

Procesy w systemie mogą być niezależne albo mogą współpracować.

Proces współpracujący może wpływać na inne procesy w systemie i/lub podlegać wpływowi innych procesów.

Inaczej — proces dzielący dane z innymi procesami jest procesem współpracującym.

Zalety:

- dzielenie informacji, np. pliku.
- przyspieszenie obliczeń (wymaga wielu elementów przetwarzających),
- modularność,
- wygoda.

Konieczne mechanizmy komunikacji i synchronizacji.

Problem producenta i konsumenta

Popularny paradygmat procesów współpracujących: proces producenta produkuje informacje wykorzystywane (konsumowane) przez procesy konsumenta.

- brak praktycznych ograniczeń na wielkość bufora,
- praktyczne ograniczenie wielkości bufora.

Rozwiązanie z wykorzystaniem pamięci dzielonej:

```
#define BUFFER_SIZE 10
Typedef struct {
 ...
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

Wielkość bufora jest ograniczona do BUFFER_SIZE pozycji.

Schemat procesu producenta:

```
item nextProduced;

while(1){
     /* produkuj kolejny element */
     while(((in +1) % BUFFER_SIZE) == out)
                ; /* nic nie rób */
     buffer[in] = nextProduced;
     in = (in + 1) % BUFFER_SIZE;
}
```

Schemat procesu konsumenta:

```
item nextConsumed;
while(1){
      while(in == out)
            ; /* nic nie rób */
      nextConsumed = buffer[out];
      out = (out+1) % BUFFER_SIZE;
      /* konsumuj kolejny element */
}
```

Komunikacja między procesami (IPC)

Mechanizm do komunikowania się i synchronizowania działania procesów.

Komunikacja przez system komunikatów, bez odwoływania się do wspólnych zmiennych.

Podstawowe operacje:

- `send(message)` - wysłanie komunikatu o stałej/zmiennej długości,
- `receive(message)` - odebranie komunikatu.

Aby procesy P i Q mogły sie komunikować muszą:

- ustanowić połączenie,
- wymienić komunikaty.

Dwa poziomy implementacji:

- poziom fizyczny (pamięć wspólna, szyna, sieć) chwilowo nas nie interesuje,
- poziom logiczny (czyli własności logiczne).

Problemy implementacyjne:

- Jak ustanawia się połączenie?
- Czy łącze może dotyczyć więcej niż dwóch procesów?
- Ile łączy można ustanowić między każdymi dwoma procesami?
- Jaka jest pojemność łącza?
- Czy łącze akceptuje komunikaty o stałej czy zmiennej długości? Jaka jest ta długość?
- Czy łącze jest jednokierunkowe (proces albo nadaje albo odbiera)czy dwukierunkowe (proces nadaje i odbiera naprzemian)?

Podstawowe warianty implementacyjne to:

- komunikacja bezpośrednia albo pośrednia,
- komunikacja symetryczna albo asymetryczna,
- buforowanie automatyczne albo jawne,
- wysyłanie przez tworzenie kopii albo odsyłaczy,
- komunikaty stałej albo zmiennej długości.

Komunikacja bezpośrednia

Każdy z procesów odwołuje się do partnera jawnie:

- `send(P, message)` — wyślij komunikat do procesu P;
- `receive(Q, message)` — odbierz komunikat od procesu Q.

Podstawowe własności tego typu komunikacji:

- łącze jest ustanawiane automatycznie (wystarczy znać identyfikator),
- łącze dotyczy dokładnie dwóch procesów,

- między parą procesów istnieje dokładnie jedno łącze,
- łącze może być jednokierunkowe ale zwykle jest dwukierunkowe.

Komunikacja pośrednia

Komunikaty są umieszczane i pobierane z abstrakcyjnych skrzynek pocztowych (niekiedy zwanych portami).

Każda skrzynka ma jednoznaczny identyfikator.

Procesy mogą się komunikować tylko jeśli mają wspólną skrzynkę.

Własności połączenia są następujące:

- połączenie jest ustanowione tylko wtedy, gdy procesy mają wspólną skrzynkę,
- łącze może być związane z więcej niż dwoma procesami,
- każda para procesów może mieć wiele łączy,
- łącza mogą być jedno i dwukierunkowe.

Skrzynka może być własnością systemu albo procesu. Wtedy proces jest właścicielem skrzynki i odbiorcą komunikatów.

Typowe operacje to tworzenie skrzynki, wysyłanie i odbieranie informacji, likwidowanie skrzynki pocztowej.

Operacje elementarne przyjmują postać:

- `send(A, message)` — wyślij komunikat do skrzynki A;
- `receive(A, message)` — odbierz komunikat ze skrzynki A.

Rozważmy następującą sytuację:

- P1, P2, P3 współdzielą skrzynkę.

- P1 wysyła komunikat natomiast P2 i P3 wykonują `receive(A, message)`.
- Kto odbierze komunikat?

Istnieje kilka możliwych rozwiązań:

- Pozwolić na łącza tylko między dwoma procesami.
- Pozwolić na wykonanie `receive` co najwyżej jednemu procesowi naraz.
- Dopuścić aby system wybrał jednego z adresatów w sposób niedeterministyczny (można ewentualnie poinformować nadawcę, kto odebrał).

Synchronizacja operacji

Operacje wysyłania i odbierania komunikatów mogą być albo blokujące albo nieblokujące.

Operacje blokujące oznaczają wykonanie synchroniczne.

Operacje nieblokujace oznaczają wykonanie asynchroniczne.

Buforowanie

Kolejka komunikatów związana z połączeniem może być zaimplementowana jako:

- długości 0 — nadawca czeka aż odbiorca odbierze komunikat, spotkanie procesów (randez-vous);
- o pojemności ograniczonej — nadawca może być zmuszony do zaczekania na miejsce w kolejce;

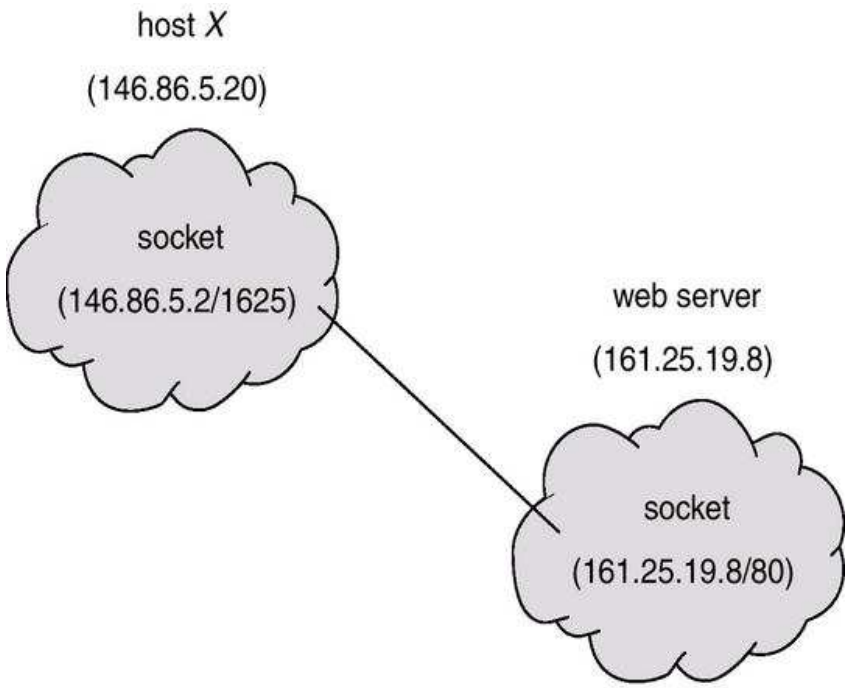
- o pojemności nieograniczonej — nadawca nigdy nie jest opóźniany.

Istnieją też bardziej wyrafinowane schematy komunikacji, np opóźnianie nadawcy do chwili otrzymania odpowiedzi.

Komunikacja typu klient-serwer

- gniazda,
- zdalne wywołanie procedur (RPC),
- zdalne wywołanie metod (Java).

Gniazda (sockets)



Gniazdo (socket) jest końcowym punktem komunikacji.

Gniazdo może być traktowane jako złożenie adresu IP z numerem portu.

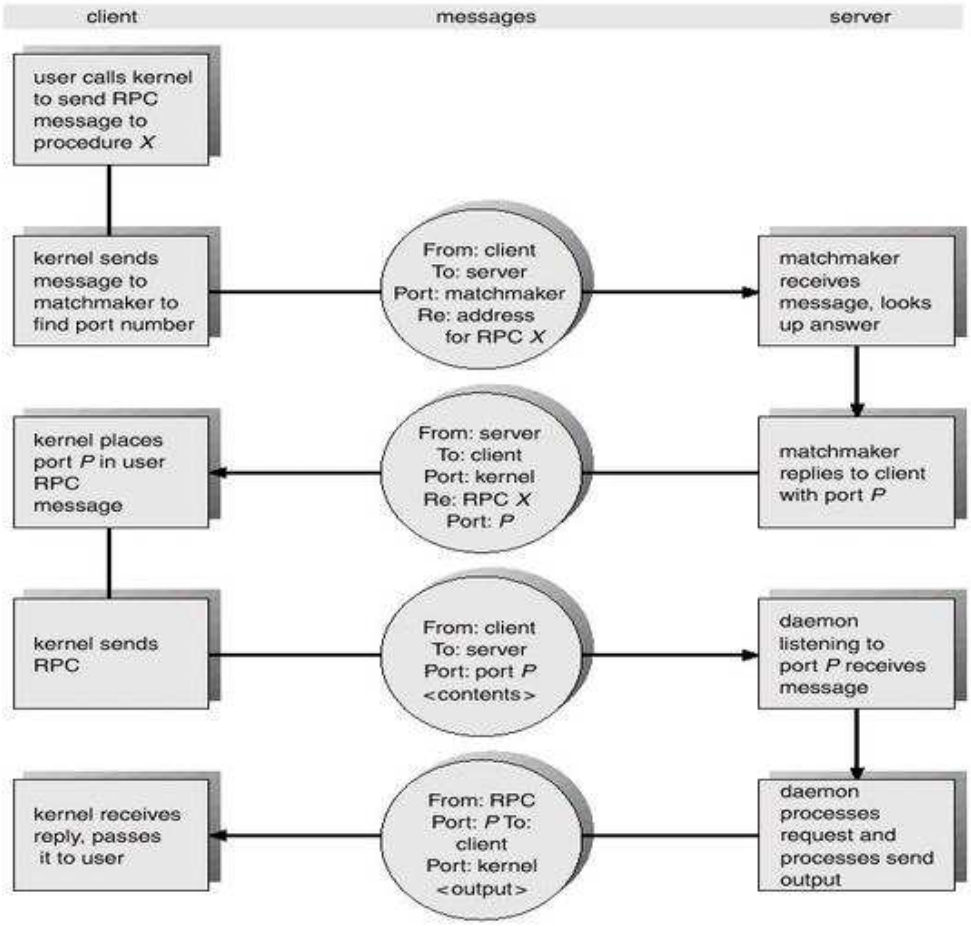
Np. gniazdo 161.25.19.8:1625 odnosi się do portu 1625 węzła sieciowego 161.25.19.8

Komunikacja odbywa się pomiędzy parą gniazd.

Zdalne wywołanie procedur (RPC)

Uogólnienie pojęcia wywołania procedury na środowisko sieciowe.

Poniższy schemat ilustruje przypadek ogólniejszy — numer portu ustalany jest przez demona spotkań.



Łącznik (stub) strony klienta pośredniczy w wywołaniu procedury.

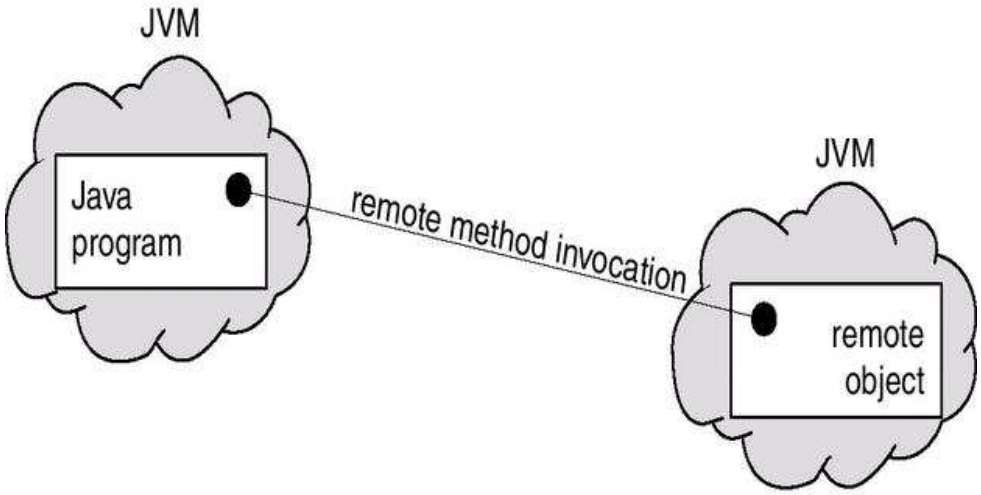
Łącznik lokalizuje serwer oraz zajmuje się konwersją parametrów do postaci sieciowej.

Łącznik strony serwera rozpakowuje parametry i realizuje wywołanie, po czym realizuje odesłanie wyniku.

Zdalne wywołanie metody (RMI, JAVA)

Procedura zbliżona do RPC.

Pozwala na wywołanie metody działającej na zdalnym obiekcie.



Marshalling parametrów