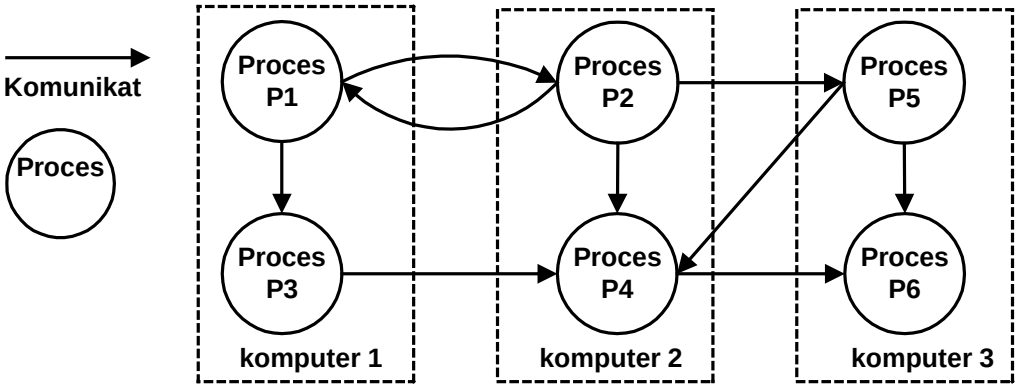
# 5. Model komunikujących się procesów, komunikaty

## *5.1 Model procesów i komunikatów*

Model procesów i komunikatów skonstruowany jest w oparciu o następujące reguły:

- Aplikacja składa się ze zbioru procesów sekwencyjnych. Procesy mogą być wykonywane równolegle na jednej lub wielu węzłach
- Proces wykonuje się sekwencyjnie i używa swej pamięci lokalnej.
- Proces komunikuje się z otoczeniem za pomocą komunikatów. Są dwie podstawowe operacje komunikacyjne: wysłanie komunikatu i odbiór komunikatu.
- Procesy mogą być przydzielone do procesorów w różny sposób. Poprawność działania aplikacji nie powinna zależeć od tego podziału.




Rys. 5-1 Aplikacja rozproszona jako zbiór komunikujących się procesów

## *5.2 Komunikaty*

Możliwość przekazywania komunikatów pomiędzy procesami jest fundamentalną własnością większości systemów operacyjnych.

Jeśli pomiędzy dwoma maszynami istnieje jakikolwiek sposób komunikacji (sieć lokalna, sieć rozległa, bezpośrednie łącze, wspólna pamięć, itd.) na pewno daje się przesłać komunikat pomiędzy procesami wykonywanymi na tych maszynach.

Przesłanie komunikatu pomiędzy procesami jest przesłaniem pomiędzy nimi pewnej liczby bajtów według ustalonego protokołu. Przesłanie komunikatu jest operacją atomową.

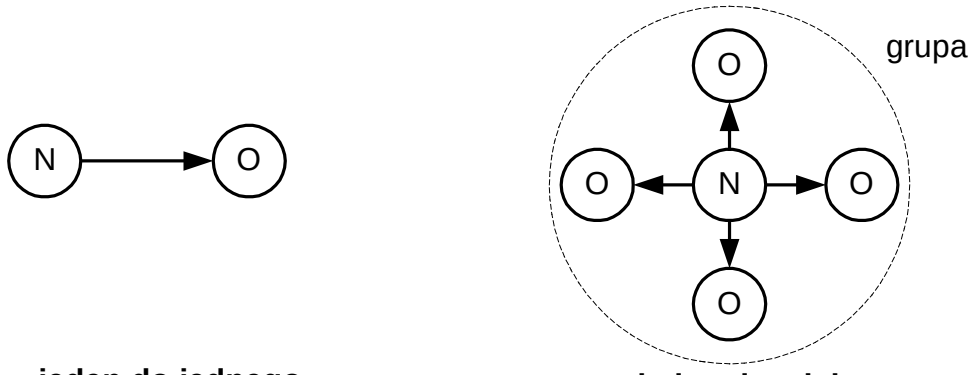
Typy komunikacji ze względu na liczbę odbiorców:

- Klient – serwer (jeden do jednego)
- Komunikacja grupowa

W komunikacji klient / serwer komunikaty wymieniane były pomiędzy dwoma procesami. Typ komunikacji jeden do jednego.

Grupa – zbiór procesów działających wspólnie w sposób określony poprzez system lub użytkownika.

Komunikacja grupowa – jeden proces wysyła komunikat odbierany przez wiele procesów należących do grupy.




**jeden do jednego** **jeden do wielu**

Rys. 5-2 Komunikacja jeden do jednego i jeden do wielu

Problemy jakie należy rozwiązać projektując komunikację:

- problem synchronizacji nadawcy i odbiorcy (kto i kiedy czeka),
- problem adresowania (jaki system adresacji),
- problem identyfikacji (czy procesy znają swoje identyfikatory),
- problem przepływu danych (w jedną czy dwie strony),
- problem zapewnienia niezawodnego przesłania przez zawodny kanał komunikacyjny.

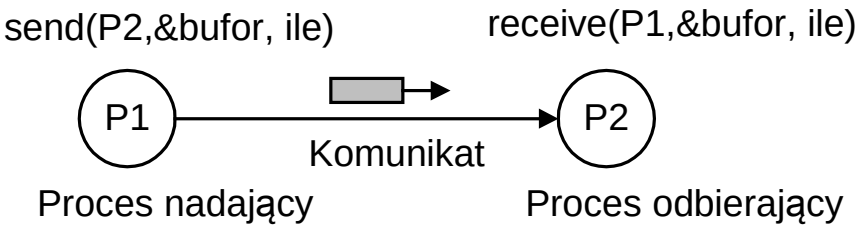
Do zapewnienia komunikacji potrzebne są przynajmniej dwie funkcje interfejsowe – wysyłająca i odbierająca komunikat.

Wysłanie komunikatu: **`send(id_odb, void *bufor, int ile)`**

Odbiór komunikatu: **`receive(id_nad, void *bufor, int ile)`**

Gdzie:

| | |
|---|---|
| **`id_odb`** | identyfikator odbiorcy |
| **`id_nad`** | Identyfikator nadawcy |
| **`*bufor`** | Adres bufora |
| **`ile`** | Liczba bajtów do przesłania |



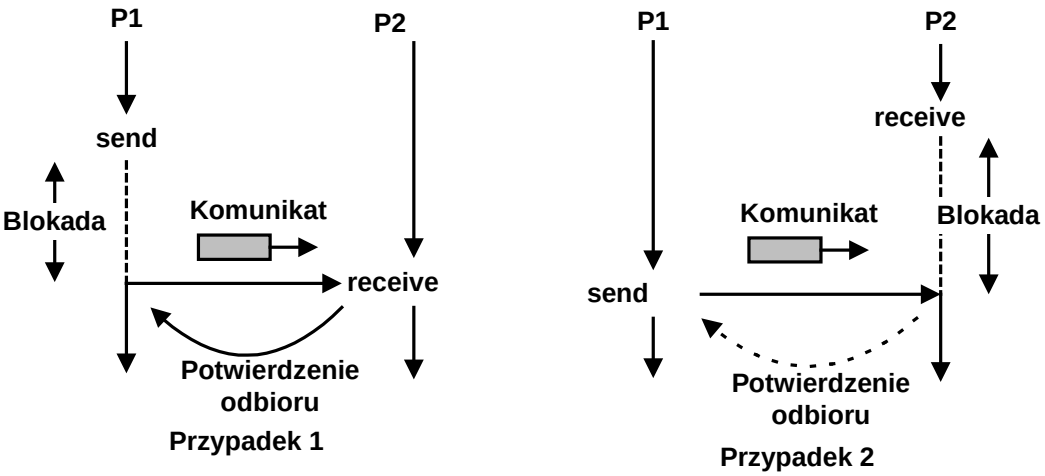

Rys. 5-3 Przesyłanie komunikatów pomiędzy procesami P1 i P2

## *5.3 Komunikacja synchroniczna i asynchroniczna*

### 5.3.1 Synchronizacja

Komunikacja synchroniczna

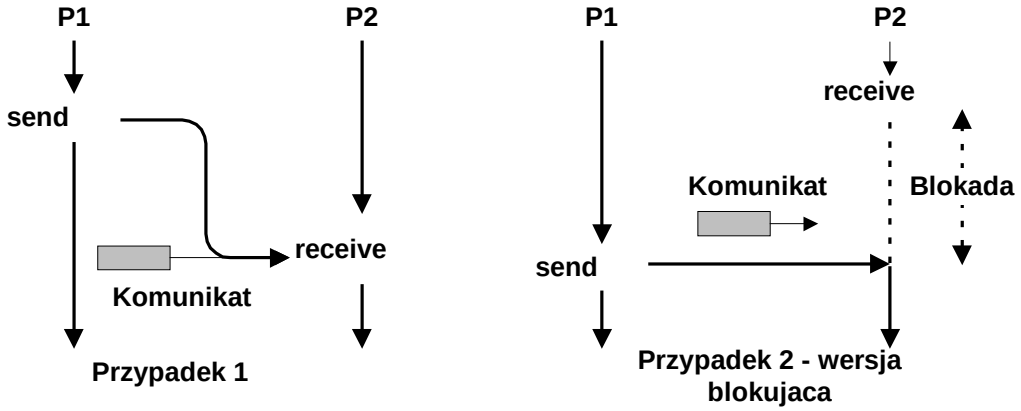
- Proces wysyłający jest blokowany do czasu otrzymania potwierdzenia że proces docelowy otrzymał wysyłany komunikat
- Gdy w momencie wykonania funkcji receive brak jest oczekującego komunikatu, proces odbierający jest wstrzymywany do czasu nadejścia jakiegoś komunikatu. Gdy jakiś komunikat oczekuje, proces odbierający nie jest blokowany.




Rys. 5-4 Komunikacja synchroniczna pomiędzy procesami P1 i P2

Komunikacja asynchroniczna

- Proces wysyłający komunikat nie jest blokowany.
- Proces odbierający jest wstrzymywany do czasu nadejścia komunikatu (wersja blokująca) lub też nie jest wstrzymywany (wersja nie blokująca). Informacja czy odebrano komunikat czy też nie, przekazywana jest jako kod powrotu funkcji odbierającej.
- Wymagane jest buforowanie




Rys. 5-5 Komunikacja asynchroniczna pomiędzy procesami P1 i P2

| Rodzaj komunikacji | Blokada przy wysyłaniu | Blokada przy odbiorze |
|---|---|---|
| Synchroniczna | Tak | Tak |
| Asynchroniczna | Nie | Tak - wersja blokująca<br>Nie - wersja nie blokująca |

Tab. 1 Definicja komunikacja synchronicznej i asynchronicznej

### 5.3.2 Buforowanie

Przy transmisji asynchronicznej konieczne jest buforowanie po stronie wysyłającej. Powstaje pytanie:

- Jaka powinna być pojemność tego bufora ?
- Co zrobić gdy bufor się przepełni ?

Postępowanie w przypadku przepełnienia bufora:

- Zablokować proces wysyłający.
- Funkcja wysyłająca komunikat kończy się błędem.

| | Synchroniczna | Asynchroniczna |
|---|---|---|
| Obsługa błędów | Testowanie kodu powrotu | Konieczna obsługa wyjątków |
| Buforowanie po stronie wysyłającej | Nie | Tak |
| Szybkość przetwarzania | Mniejsza | Większa |

Porównanie komunikacji synchronicznej i asynchronicznej

## *5.4 **Komunikacja połączeniowa i bezpołączeniowa***

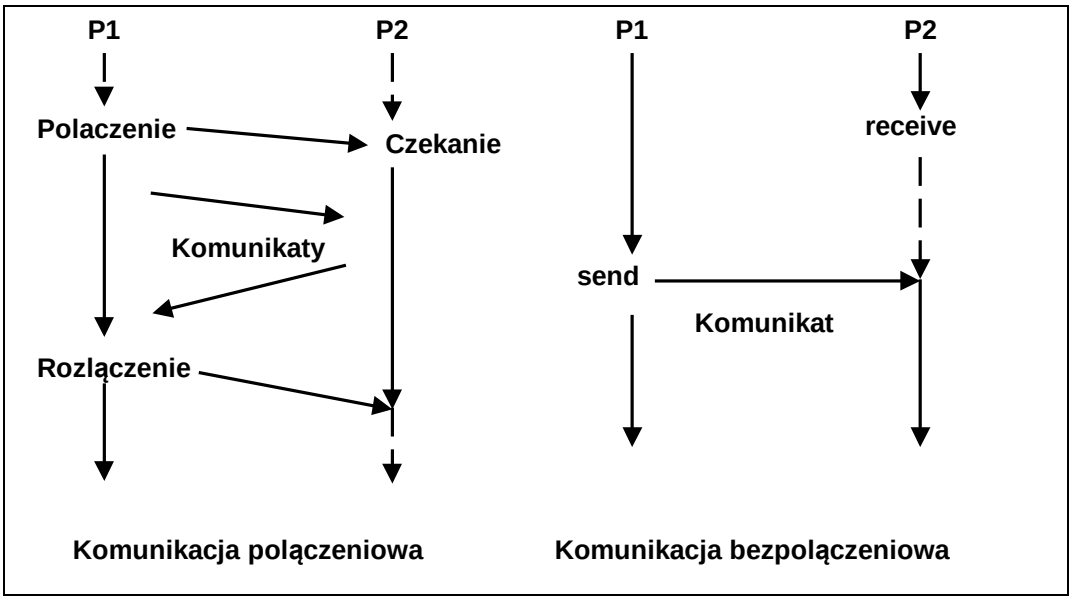
Komunikacja połączeniowa:

1. Dwa procesy ustanawiają połączenie w którym występuje informacja adresowa
2. Wymieniają dane używając tylko identyfikatora połączenia
3. Rozłączają się

Często połączenie jest kontrolowane w sensie:
- Możliwości przesyłania danych
- Poprawności danych

Komunikacja bezpołączeniowa

W każdym przesłaniu występuje pełna informacja adresowa




Rys. 5-6 Komunikacja połączeniowa i bezpolaczeniowa

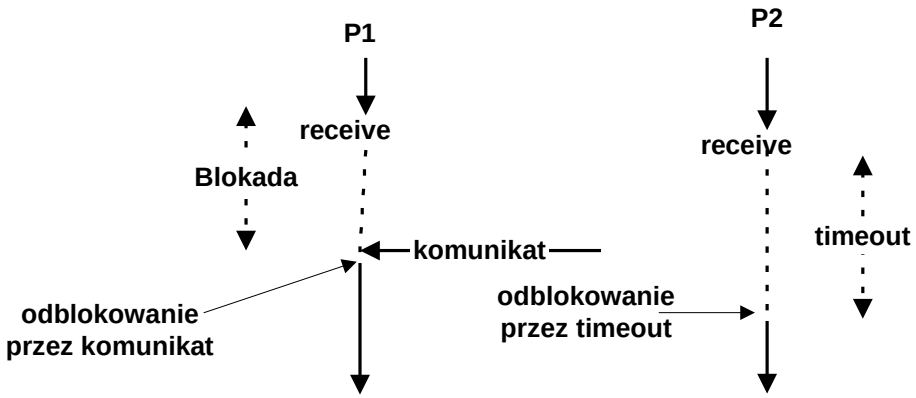
### *5.5 Przeterminowania*

Konieczność synchronizowania procesów wymusza konieczność ich blokowania. Procesy są blokowane w oczekiwaniu na pewne zdarzenie. Z różnych powodów (błędy, uszkodzenia) zdarzenie to może nigdy nie nadejść. Spowoduje to trwałą blokadę procesu. Aby temu zaradzić stosuje się przeterminowanie (ang. *Timeout*) oczekiwania.

Wysłanie komunikatu:
**`send(id_odb, void *bufor, int ile, timespec timeout)`**

Odbiór komunikatu:
**`receive(id_nad,void *bufor,int ile, timespec timeout)`**

Gdy proces nie zostanie samoistnie odblokowany po czasie **`timeout`** zrobi to system operacyjny. Funkcja zwróci kod błędu.




Rys. 5-7 Odblokowanie procesu P1 przez komunikat, procesu P2 przez timeout

### *5.6 Reprezentacja danych*

Wysyłający dane używa specyficznych typów danych jak:
znaki, liczby int, liczby real, stałe boolean, struktury, tablice, teksty róznych formatów, obiekty. Na poziomie sieci transmitowane są bajty które muszą być przesyłane jeden po drugim

Przykład:
Maszyna M1 - 32 bitowa, system "big endian"
Maszyna M2 - 16 bitowa, system "little endian"

Wniosek: transformacja danych jest konieczna gdy systemy są heterogeniczne.
Transformacja postaci danych przy IPC nazywa się przetaczaniem danych (*ang. data marshalling*)

**Kroki przetaczania danych**
Przy nadawaniu
Serializacja danych
Konwersja do zewnętrznej reprezentacji

Przy odbiorze
Konwersja z zewnętrznej do wewnętrznej reprezentacji
Deserializacja danych

## 5.7 Własności modelu procesów i komunikatów

Wydajność
Model procesu sekwencyjnego posiada wydajne narzędzia implementacyjne (kompilatory itd.). Gdy procesy wykonywane są na różnych procesorach, sprzęt jest dobrze wykorzystany.

Niezależność od fizycznej struktury maszyny.
Model procesów i komunikatów jest niezależny od tego czy procesy wykonywane są w systemie jedno, wieloprocesorowym czy rozproszonym.

Skalowalność
Model jest niezależny od liczby dostępnych procesorów.

Modularność
Problem może być podzielony na tworzone niezależnie moduły (procesy), komunikujące się poprzez dobrze zdefiniowane interfejsy (komunikaty, kanały).

Determinizm
Program jest deterministyczny gdy takie same sekwencje na wejściu, powodują takie same sekwencje na wyjściu. W modelu procesów i komunikatów łatwo osiągnąć determinizm.

Inne modele przetwarzania:
Procesy komunikują się przez wspólną pamięć