

Ćwiczenie SCR-02. Komunikacja międzyprocesowa Send-Receive-Reply

SYSTEMY OPERACYJNE CZASU RZECZYWISTEGO

LABORATORIUM SYSTEMÓW STEROWNIA PRZEMYSŁOWEGO I AUTOMATYKI BUDYNKÓW

KATEDRA AUTOMATYKI NAPĘDU I URZĄDZEŃ PRZEMYSŁOWYCH
WWW.KANIUP.AGH.EDU.PL

AKADEMIA GÓRNICZO-HUTNICZA
WWW.AGH.EDU.PL

Temat: Komunikacja międzyprocesowa Send-Receive-Reply
Wymagane umiejętności: Zagadnienia synchronicznej komunikacji międzyprocesowej

Wstęp

Komunikacja i synchronizacja procesów

Jądro systemu QNX realizuje cztery mechanizmy komunikacji procesów: synchroniczny przekaz wiadomości (*message*) podczas spotkania, synchronizację za pomocą semaforów (*semaphore*), asynchroniczny przekaz depozytów (stałych wiadomości) poprzez procesy depozytowe (*proxy*) oraz sygnalizację zdarzeń (*signal*). Wszystkie mechanizmy - oprócz semaforów - mogą być wykorzystane do komunikacji procesów wykonywanych w różnych węzłach sieci. Dodatkowo, odpowiednie procesy systemowe umożliwiają buforowane przekazywanie danych między procesami poprzez potoki (*pipe*) i kolejki (FIFO) oraz kolejki wiadomości (*message queue*).

Wiadomości

Podstawowym mechanizmem komunikacji procesów jest wymiana pary wiadomości (wiadomość-odpowiedź), realizowana według schematu spotkania. Podstawowe etapy wymiany wiadomości są realizowane przez trzy funkcje systemowe: `Send()`, `Receive()` i `Reply()`. Proces nadający wiadomość musi zawsze znać tożsamość odbiorcy, którego identyfikator jest jednym z parametrów funkcji `Send()`. Proces odbierający wiadomość nie musi znać tożsamości nadawcy - zależnie od sposobu wywołania funkcji `Receive()` wiadomości mogą być odbierane od wskazanego nadawcy albo od każdego procesu. W razie jednoczesnego nadania wielu wiadomości, skierowanych przez różnych nadawców do tego samego odbiorcy, wszystkie procesy nadające są zawieszane. Przez domniemanie, kolejka procesów oczekujących (i ich wiadomości) jest obsługiwana zgodnie z algorytmem FIFO. Algorytm ten można jednak zmienić na priorytetowy, ustawiając odpowiednie znaczniki systemowe. Wszystkie wiadomości są przekazywane bezpośrednio między procesami, tzn. treść wiadomości jest kopiowana z przestrzeni adresowej nadawcy do przestrzeni adresowej odbiorcy. W czasie trwania wymiany wiadomości istnieje jednak między procesami pewien bufor, który pozwala procesowi odbierającemu wiadomość i nadającemu odpowiedź odbierać lub nadawać wiadomość fragmentami, bez potrzeby kumulowania całej treści w jednej tablicy.

Ćwiczenie SCR-02. Komunikacja międzyprocesowa Send-Receive-Reply

Przesyłanie komunikatów

Przesłanie komunikatu pomiędzy procesami jest przesłaniem pewnej liczby bajtów pomiędzy tymi procesami według ustalonego protokołu. Przesłanie komunikatu jest operacją atomowa.

Możliwość przekazywania komunikatów pomiędzy procesami jest fundamentalną własnością systemu QNX. W systemie QNX wyróżniamy następujące akcje związane z przesyłaniem komunikatów pomiędzy procesami P1 i P2:

1. Wysłanie komunikatu przez P1 do P2
2. Odbiór komunikatu przez P2
3. Przesłanie odpowiedzi zwrotnej od P2 do P1.

Wysłanie komunikatu

Komunikat do procesu P wysyła się wykonując funkcję Send. Prototyp funkcji Send podany został poniżej.

```
int Send(pid_t pid, void * msg, void * rmsg, unsigned sbytes, unsigned rbytes)
```

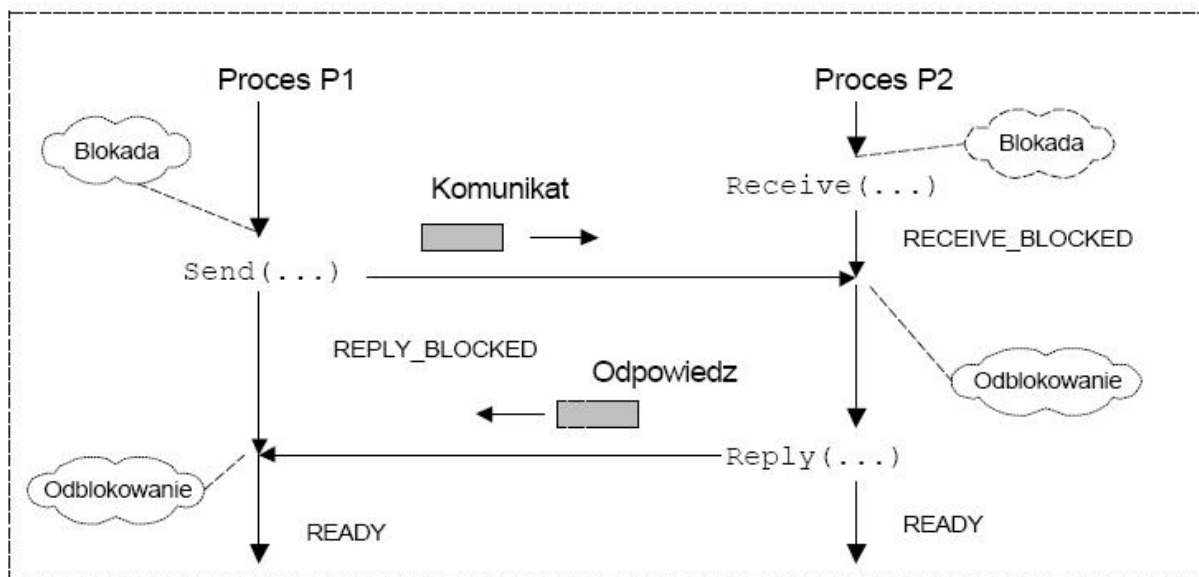
Znaczenie parametrów powyższej funkcji jest następujące:

- pid PID procesu docelowego P
- msg adres bufora danych wysyłanych
- rmsg adres bufora danych odbieranych
- sbytes liczba bajtów wysyłanych
- rbytes max. liczba bajtów odbieranych

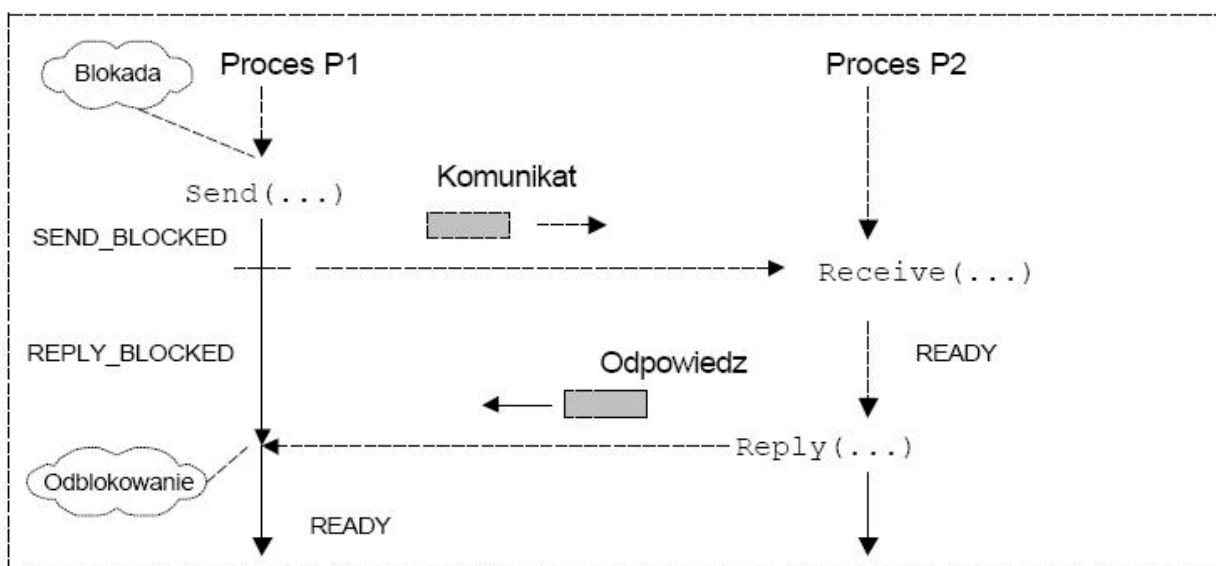
Funkcja zwraca: 0 – sukces, -1 - błąd Działanie funkcji Send jest następujące:

- 1) Komunikat msg wysyłany jest do procesu docelowego, po czym proces bieżący ulega zablokowaniu.
- 2) Proces wysyłający jest zablokowany do czasu, gdy proces docelowy nie odbierze wysłanego komunikatu i nie prześle odpowiedzi.
- 3) Po otrzymaniu odpowiedzi jest ona umieszczana w buforze rmsg i proces bieżący jest wznowiany.

Ćwiczenie SCR-02. Komunikacja międzyprocesowa Send-Receive-Reply



Wymiana komunikatów pomiędzy procesami P1 i P2. Funkcja `Receive(...)` poprzedza funkcję `Send(...)`.



Wymiana komunikatów pomiędzy procesami P1 i P2. Funkcja `Send (...)` poprzedza funkcję `Receive (...)`.

Odbiór komunikatu

Do odbioru komunikatów przez proces P służy funkcja `Receive`.

```
pid_t Receive(pid_t pid, void * msg, unsigned rbytes)
```

Znaczenie parametrów powyższej funkcji `Receive` jest następujące:

- `pid` PID procesu nadającego lub 0 – gdy odbiór od wszystkich procesów

Ćwiczenie SCR-02. Komunikacja międzyprocesowa Send-Receive-Reply

- rmsg adres bufora danych odbieranych
- rbytes max. liczba bajtów odbieranych

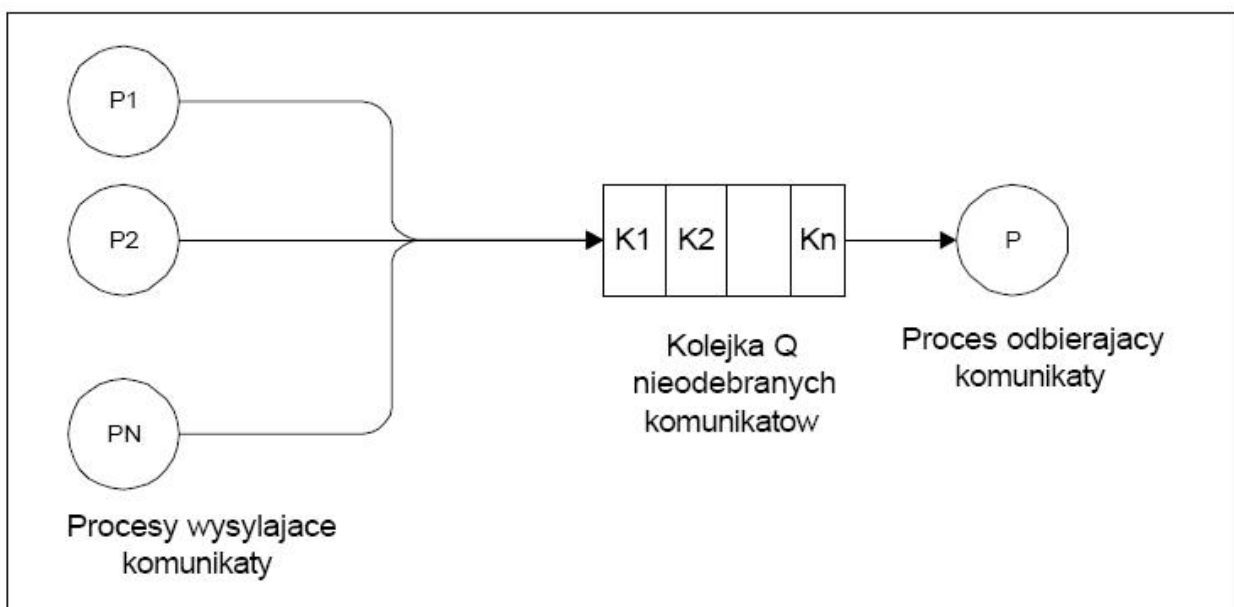
Funkcja zwraca: > 0 – PID procesu, który nadał komunikat, - 1 - błąd

Gdy zachodzi potrzeba odbioru komunikatu należy wykonać funkcję Receive. Zwykle jako parametr pid wstawiamy, 0 co oznacza że odbierane będą komunikaty od wszystkich procesów. Działanie funkcji Receive zależy od tego czy istnieją komunikaty, które zostały wcześniej wysłane do procesu P przez inne procesy. Jeśli tak było to muszą one oczekiwać w kolejce Q nieodebranych komunikatów.

Działanie funkcji jest następujące:

1. Gdy w kolejce Q są nieodebrane komunikaty pierwszy z nich usuwany jest z kolejki Q i umieszczany w buforze msg. Funkcja zwraca PID procesu, który wysłał ten komunikat. System powoduje odblokowanie procesu wysyłającego komunikat. Proces bieżący nie ulega zablokowaniu.
2. Gdy kolejka Q jest pusta proces P ulega zablokowaniu do czasu nadejścia takiego komunikatu. Zablokowany proces jest w stanie RECEIVE_BLOCKED. Istnieje możliwość zmiany uporządkowania tej kolejki na uporządkowanie według priorytetów procesów przysyłających komunikaty. Należy w tym przypadku użyć funkcji qnx_pflags(...).

Priorytetowa organizacja kolejki pociąga za sobą możliwość zagłócenia procesów.



Proces P nie odebrał komunikatów wysłanych przez procesy P1, P2, ...PN. Komunikaty oczekują w kolejce Q.

Ćwiczenie SCR-02. Komunikacja międzyprocesowa Send-Receive-Reply

Wysłanie odpowiedzi na komunikat

pid_t Reply(pid_t pid, void * msg, unsigned sbytes)

- pid PID procesu nadającego lub 0 – odbiór od wszystkich
- msg adres bufora danych wysyłanych
- sbytes maksymalna liczba bajtów wysyłanych

Funkcja zwraca: > 0 – PID procesu, który nadał komunikat, 1 - błąd

Warunkowy odbiór komunikatu

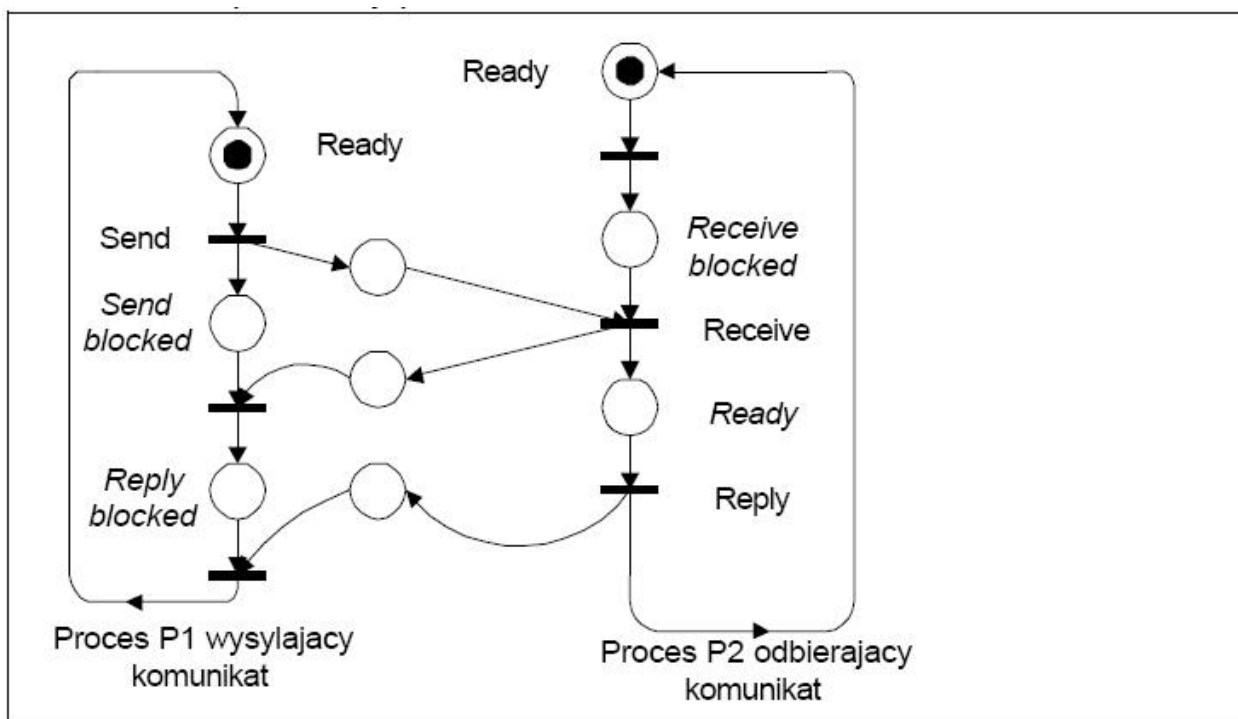
pid_t Creceive(pid_t pid, void * msg, unsigned rbytes)

Funkcja nie powoduje zablokowania procesu bieżącego, gdy brak komunikatów.

- pid > 0 - PID procesu nadającego 0 - odbiór od wszystkich procesów
- rmsg adres bufora danych odbieranych
- rbytes max. liczba bajtów odbieranych

Funkcja zwraca: 0 – PID procesu, który nadał komunikat - 1 - błąd

Na poniższym rysunku za pomocą sieci Petriego przedstawiono wymianę komunikatów pomiędzy procesami P1 i P1.



Ćwiczenie SCR-02. Komunikacja międzyprocesowa Send-Receive-Reply

Nazwy

Proces P1 może wysłać komunikat do P2 o ile zna jego PID. Bez dodatkowych mechanizmów tylko procesy będące w relacji macierzysty / potomny mogą się w ten sposób komunikować. Aby umożliwić komunikowanie się procesów niezwiązanych, w systemie QNX wprowadzono mechanizm nazw. Procesy mogą rejestrować swoje nazwy w systemie. Nazwy są zwykłymi łańcuchami. Inne procesy mogą zwrócić się do systemu o PID procesu podając jego nazwę. Mechanizm ten działa także przez sieć i nosi nazwę lokalizacji.

Rejestracja nazwy

Proces rejestruje się poprzez wykonanie funkcji:

```
int qnx_name_attach(nid_t nid, char * name)
```

- nid Identyfikator węzła, na którym nazwa jest rejestrowana (0 – węzeł bieżący)
- name Nazwa rejestrowanego procesu

Gdy nazwa zaczyna się od znaku / jest widoczna w całej sieci. Nazwa może być też zarejestrowana na określonym węźle. Funkcja zwraca:

- 0 - identyfikator nazwy (*ang. name ID*) – liczba typu int używana w innych funkcjach. -1 - gdy rejestracja się nie udała.

Lokalizacja nazwy

PID procesu o znanej nazwie uzyskuje się poprzez wywołanie funkcji:

```
pid_t qnx_name_locate(nid_t nid, char * name, unsigned size, NULL)
```

- nid Identyfikator węzła, na którym nazwa jest rejestrowana (0 – węzeł bieżący)
- name Nazwa lokalizowanego procesu
- size Długość komunikatu, który będzie przesyłany pomiędzy procesami

Funkcja zwraca:

- 0 - identyfikator procesu -1 - gdy lokalizacja się nie udała.

Gdy nid = 0 proces będzie lokalizowany najpierw na bieżącym węźle a potem w sieci. Gdy nid # 0 proces będzie lokalizowany tylko na węźle o numerze nid. Gdy lokalizowany proces leży na innym węźle automatycznie tworzone jest połączenie wirtualne (*ang. Virtual Circuit*) pomiędzy węzłami.

Ćwiczenie SCR-02. Komunikacja międzyprocesowa Send-Receive-Reply

Wyrejestrowanie nazwy

Nazwa kasowana jest poprzez wywołanie funkcji:

```
int qnx_name_detach(nid_t nid, int name_id)
```

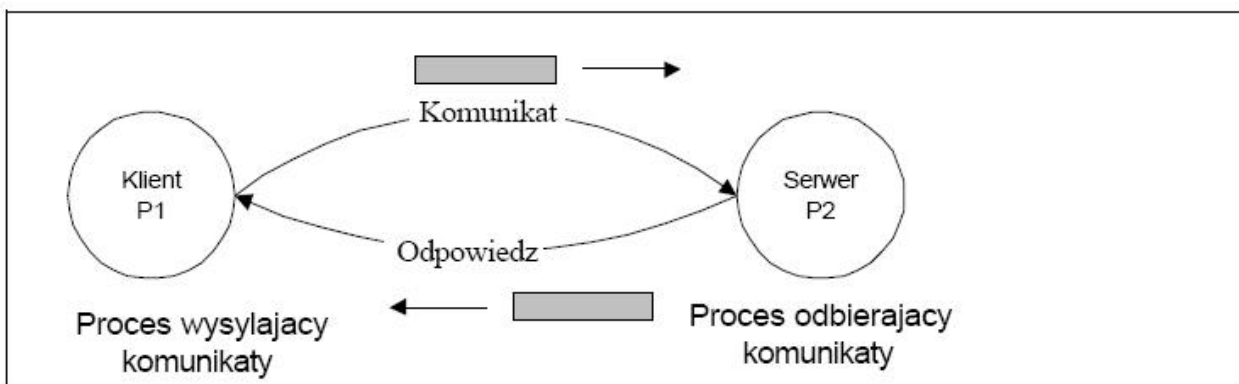
- nid Identyfikator węzła, na którym nazwa jest kasowana (0 – węzeł bieżący)
- name_id Identyfikator kasowanej nazwy – liczba zwracana przez funkcję qnx_name_attach

Funkcja powoduje usunięcie nazwy o identyfikatorze name_id z bazy nazw. Funkcja zwraca:

0 - gdy wyrejestrowanie się udało -1 - gdy operacja się nie udała

Przykład komunikacji

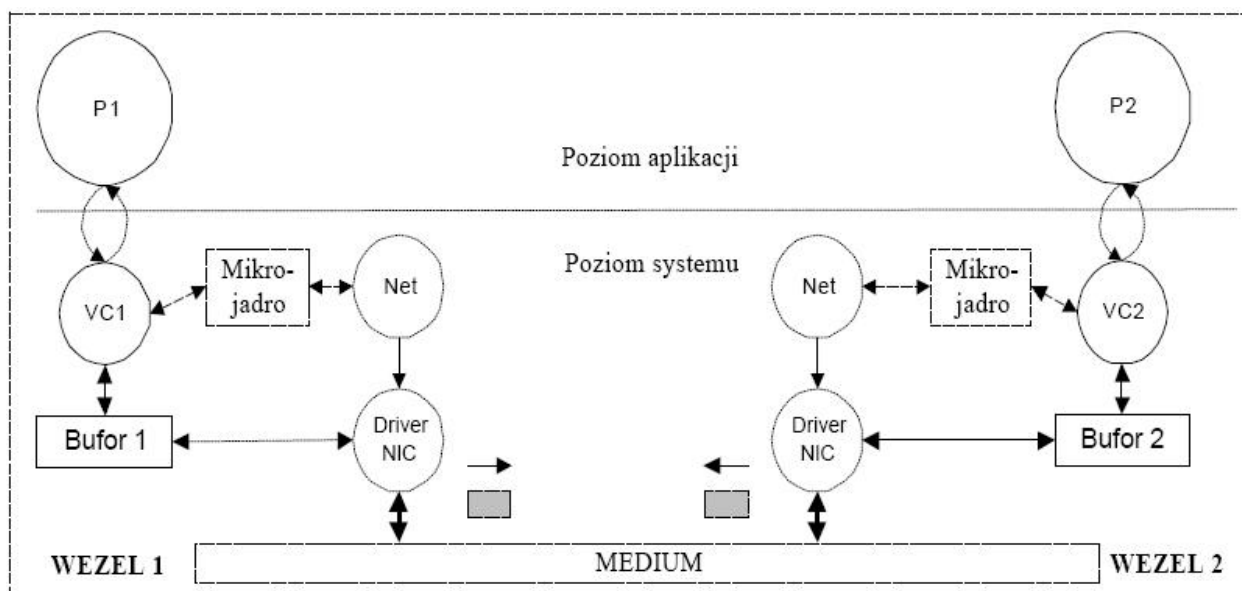
Przykład procesu klienta P1 i serwera P2 wymieniające komunikaty. Procesy mogą być uruchomione na tym samym lub oddzielnych węzłach połączonych siecią.



Przekazywanie komunikatów przez sieć – połączenia wirtualne

System QNX jest od podstaw zbudowany jako rozproszony system sieciowy. Z punktu widzenia programisty komunikacja pomiędzy procesami zlokalizowanymi na jednym węźle nie różni się od komunikacji przez sieć.

Ćwiczenie SCR-02. Komunikacja międzyprocesowa Send-Receive-Reply



Połączenie wirtualne pomiędzy procesami P1 i P2.

Połączenie wirtualne jest tworzone podczas lokalizacji procesu serwera. Tworzone są procesy wirtualne VC1 i VC2, które są pośrednikami w transmisji komunikatów. Proces VC1 reprezentuje proces P2 na węźle 1 a proces VC2 reprezentuje proces P1 na węźle 2. Każdy z procesów wirtualnych dysponuje buforem na komunikaty. Każde połączenie wirtualne musi utrzymywać następujące informacje:

1. PID procesu lokalnego i zdalnego.
2. VID procesu lokalnego i zdalnego
3. NID (numery węzłów) lokalnego i zdalnego

Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z synchroniczną komunikacją międzyprocesową (IPC) w systemie QNX i realizacja programów komunikujących się na jednym węźle sieci i poprzez sieć.

Program ćwiczenia.

1. Realizacja programu odbierającego wiadomości (przy użyciu funkcji Receive() i Reply())
2. Realizacja programu wysyłającego (przy użyciu funkcji Send())
3. Uruchomienie programów na lokalnym węźle (komputerze)
4. Uruchomienie jednego z programów na lokalnym węźle, drugiego na zdalnym.