**Zadanie 2**

Napisać program, który rozwidli się na dwa procesy komunikujące się z wykorzystaniem mechanizmu pamięci dzielonej. Synchronizacja dostępu do obiektu pamięci dzielonej oraz wymiana informacji o tym obiekcie ma być zapewniona przez mechanizm komunikatów synchronicznych.

Uwagi szczegółowe:

W pierwszej kolejności napisany program powinien rozwidlić się na dwa procesy, z których jeden przyjmie rolę serwera (będzie odbierał komunikaty), a drugi klienta (będzie wysyłał komunikaty). Łatwiej jest napisać program przypisując rolę serwera procesowi macierzystemu, a klienta potomnemu.

Serwer (i tylko serwer) powinien utworzyć kanał komunikacyjny do odbioru komunikatów. Identyfikator kanału i pid serwera musza być znane klientowi. Klient czyli proces potomny może zawsze pobrać pid serwera poprzez wywołanie getppid(). Pozostaje problem przekazania identyfikatora kanału. Najłatwiejszym rozwiązaniem jest utworzenie kanału przed rozwidleniem i zapamiętanie jego identyfikatora w zmiennej. Proces potomny odziedziczy ta zmienną, natomiast kanał komunikacyjny nie jest dziedziczony. Po utworzeniu kanału serwer czeka na otrzymanie komunikatu (jest zablokowany).

Klient (i tylko klient) powinien utworzyć polaczenie do kanału utworzonego przez serwer. Jest to niezbędne aby móc wysłać komunikat. Pierwszy argument funkcji tworzącej to połączenie to numer węzła w sieci, na którym uruchomiony jest serwer, jeśli jest lokalnie to podajemy 0.

Klient powinien przygotować obiekt pamięci dzielonej zawierający dane mające być przetworzone przez serwer. Załóżmy, że dane te to tablica liczb całkowitych, a serwer pełnić będzie usługi sortowania. Oczywiście to tylko przykład, a można zaimplementować coś dużo ciekawszego i nieco bardziej skomplikowanego. Klient musi kolejno:

1. Utworzyć obiekt pamięci dzielonej
2. Ustalić wielkość tego obiektu
3. Zamapować go na własna przestrzeń adresową
4. Wypełnić danymi (np. wylosować)

Po przygotowaniu obiektu i danych klient formatuje odpowiedni komunikat z informacjami o obiekcie (nazwa, ilość danych, kod żądanej operacji, itp.). Uwaga, w obrębie systemu obiekt pamięci dzielonej identyfikowany jest poprzez jego nazwę. Identyfikatory są lokalne dla procesów. Klient wysyła komunikat i zostaje zablokowany.

Komunikat dociera do serwera i go odblokowuje. Serwer powinien otworzyć obiekt pamięci dzielonej, zamapować go na własną przestrzeń adresową i wykonać żądaną operację na zawartych w nim danych (np. sortowanie). Po wykonaniu operacji serwer odmapowuje obiekt i go zamyka, a następnie wysyła odpowiedź do klienta informując go o wykonanej operacji.

Odpowiedź dociera do klienta i odblokowuje go. Klient może np. wyświetlić posortowane przez serwer dane.

Pamiętamy, ze procesy muszą po sobie posprzątać, a przede wszystkim klient powinien usunąć obiekt pamięci dzielonej. Inaczej pozostanie on w systemie.

Najłatwiej zacząć pisanie programu od rozwidlenia, następnie dodać mechanizm komunikatów, a na samym końcu pamięć dzieloną.

Gdzie szukać pomocy w systemowym helpie:

1. **Komunikaty synchroniczne:** QNX Neutrino RTOS -> System Architecture -> The QNX Neutrino Microkernel -> QNX Neutrino IPC -> Synchronous message passing
2. **Pamięć dzielona:** QNX Neutrino RTOS -> System Architecture -> The QNX Neutrino Microkernel -> QNX Neutrino IPC -> Shared memory
3. **Rozwidlenie procesów:** QNX Neutrino RTOS -> System Architecture -> Process Manager -> Process management

Niezbędne funkcje:

fork() – rozwidlenie procesów

MsgSend() – wysłanie komunikatu

MsgReceive() – odebranie komunikatu

MsgReply() – wysłanie odpowiedzi na komunikat

ChannelCreate() – utworzenie kanału

ChannelDestroy() – usunięcie kanału

ConnectAttach() – podłączenie do kanału

ConnectDetach() – odłączenie od kanału

shm\_open() – utworzenie lub jeśli istnieje otwarcie obiektu pamięci dzielonej

close() – zamknięcie obiektu pamięci dzielonej

mmap() – zamapowanie obiektu pamięci dzielonej na przestrzeń adresową procesu

munmap() – odmapowanie

shm\_unlink() – usunięcie obiektu pamięci dzielonej

ftruncate() – ustalenie wielkości obiektu pamięci dzielonej