Systemy operacyjne – studia stacjonarne 2022/23

Lab 10. Rozwiązywanie klasycznych problemów synchronizacji procesów przy pomocy mechanizmów IPC - implementacja zadania klient-serwer z wykorzystaniem kolejki komunikatów. Projekt nr 3.

Do wywołania funkcji niezbędne są następujące pliki nagłówkowe:

<sys/types.h>
<sys./ipc.h>
<sys/msg.h>

1. Klucz do kolejki komunikatów.

   Do tworzenia lub do odwoływania się do kolejki komunikatów potrzebny jest tzw. klucz – liczba całkowita. Jednoznaczne klucze można utworzyć przy pomocy funkcji ftok().

pliki nagłówkowe	<sys/types.h>, <sys/ipc.h>		
prototyp	key_t ftok(const char *path, int id)		
zwracana wartość	sukces	porażka	zmiana errno
	wartość klucza	-1	nie

   path - ścieżkowa nazwa istniejącego pliku - dostępnego procesowi
   id – zwykle pojedynczy znak, który jednoznacznie identyfikuje projekt

2. Struktura komunikatu.

   Jest ograniczona na dwa sposoby. Komunikat musi być mniejszy od systemowego limitu oraz musi rozpoczynać się od wartości typu long int, używanej jako wskaźnik typu komunikatu w funkcji odbiorczej. Przykładowa postać struktury komunikatu:

```
struc moj_komunikat {
        long int mtype; /*typ komunikatu, musi być > 0 */
        char text[20];    /* przekazane dane*/
    }
```

3. Tworzenie i uzyskiwanie dostępu do kolejki komunikatów: msgget().

pliki nagłówkowe	<sys/msg.h>, <sys/types.h>, <sys/ipc.h>		
prototyp	int msgget(key_t key, int msgflg)		
zwracana wartość	sukces	porażka	zmiana errno
	identyfikator kolejki	-1	tak

   key - klucz do kolejki (różne procesy, które chcą korzystać z tej samej kolejki, muszą użyć tego samego klucza), ftok() lub IPC_PRIVATE
   shmflg – flaga określająca sposób wykonania funkcji i prawa dostępu do kolejki komunikatów:
   IPC_CREAT – utworzenie kolejki lub uzyskanie dostępu do istniejącej kolejki
   IPC_EXCL – użyta w połączeniu z IPC_CREAT zwraca błąd, jeżeli dla danego klucza istnieje już Kolejka komunikatów
   PRAWA DOSTĘPU – podobnie jak dla pliku np. 0666

   Dodatkowe informacje:
   http://www.linux.pl/man/index.php?command=msgget

4. Dodanie komunikatu do kolejki: funkcja msgsnd().

pliki nagłówkowe	<sys/msg.h>, <sys/types.h>, <sys/ipc.h>		
prototyp	int msgsnd(int msqid, const void *msg_ptr, size_t msg_sz, int msgflg);		
zwracana wartość	sukces	porażka	zmiana errno
	0	-1	tak

msqid – identyfikator kolejki (zwracany przez msgget());
*msg_ptr – wskaźnik do wysłanego komunikatu;
msg_sz – rozmiar komunikatu na który wskazuje msg_ptr bez wartości long int;
msgflg – flaga określająca reakcję na przepełnienie kolejki:
IPC_NOWAIT – funkcja powróci bez wysłania komunikatu zwracając -1;
0 – proces zostanie zawieszony w oczekiwaniu na zwolnienie miejsca w kolejce.

Dodatkowe informacje:
http://www.linux.pl/man/index.php?command=msgsnd

5. Pobranie komunikatu z kolejki: funkcja msgrcv().

pliki nagłówkowe	<sys/msg.h>, <sys/types.h>, <sys/ipc.h>		
prototyp	int msgrcv(int msqid, void *msg_ptr, size_t msg_sz, long int msgtype, int msgflg);		
zwracana wartość	sukces	porażka	zmiana errno
	Liczba pobranych bajtów	-1	tak

msqid – identyfikator kolejki (zwracany przez msgget());
*msg_ptr – wskaźnik do odebranego komunikatu;
msg_sz – rozmiar komunikatu na który wskazuje msg_ptr bez wartości long int;
msgtype =0 pobierany pierwszy dostępny komunikat;
>0 (np.11) pobierany pierwszy dostępny komunikat danego typu;
<0 (np.-6) pobierany pierwszy dostępny komunikat, którego typ ma wartość taką samą lub mniejszą niż absolutna wartość msgtype;
msgflg – flaga:
0 – proces zostanie zawieszony w oczekiwaniu na właściwy komunikat.
IPC_NOWAIT - Wywołanie nie będzie wstrzymywać pracy procesu, jeśli w kolejce nie ma komunikatów odpowiedniego typu. Wywołanie systemowe zgłosi wówczas błąd, przypisując zmiennej errno wartość ENOMSG.
MSG_NOERROR - Spowoduje obcięcie komunikatu, jeśli jego dane są dłuższe niż msgsz bajtów.

Dodatkowe informacje:
http://www.linux.pl/man/index.php?command=msgrcv

6. Obsługa kolejek komunikatów: funkcja msgctl().

pliki nagłówkowe	<sys/msg.h>, <sys/types.h>, <sys/ipc.h>		
prototyp	int msgctl(int msqid, int cmd, struct msqid_ds *buf);		
zwracana wartość	sukces	porażka	zmiana errno
	0	-1	tak

msqid – identyfikator kolejki (zwracany przez msgget);
cmd – operacje na kolejce:
IPC_RMID – usuwa kolejkę

IPC_SET – ustawia wartości związane z kolejką komunikatów na dane określone w strukturze msgid_ds (jeżeli proces ma odpowiednie zezwolenia);

IPC_STAT – ustawia dane w strukturze msgid_ds tak, aby otrzymały wartości związane z kolejką komunikatów

Dodatkowe informacje:
http://www.linux.pl/man/index.php?command=msgctl

7. Użyteczne komendy:

ipcs –q	podaje informacje nt. kolejek komunikatów (patrz man ipcs)
ipcrm –q msqid	usuwa kolejkę o numerze msqid
ipcs -l	podaje informacje dotyczące limitów

Uwaga: jeżeli nie nadano praw do czytania polecenie ipcs nie pokaże kolejki – taką kolejkę można usunąć poleceniem ipcrm podając jej msqid.

Ćwiczenie 1.
Skopiuj do swojego katalogu domowego, pliki s.c oraz k.c znajdujący się w katalogu:
/home/inf-prac/wojtas.jan/Dydaktyka/SO/Projekty/KOLEJKA_KOMUNIKATOW
W trybie pracy krokowej przeanalizuj sposób działania funkcji związanych z kolejkami komunikatów.

Projekt nr 3.

Projekt składa się z dwóch programów uruchamianych niezależnie: serwer i klient.
Proces klient wysyła do procesu serwera ciąg znaków. Serwer odbiera ten ciąg znaków i przetwarza go zmieniając w nim wszystkie litery na duże, a następnie wysyła tak przetworzony ciąg znaków z powrotem do klienta. Klient odbiera przetworzony ciąg znaków i wypisuje go na ekranie. Posługując się mechanizmem kolejki komunikatów, należy zaimplementować powyższe zadanie typu klient-serwer z możliwością obsługi wielu klientów jednocześnie. W rozwiązaniu użyć jednej kolejki komunikatów. Zastosować odpowiednie etykietowanie komunikatów w celu rozróżniania w kolejce danych dla serwera oraz danych dla poszczególnych klientów.

Serwer tworzy kolejkę komunikatów i oczekuje na komunikaty od klientów. Serwer usuwa kolejkę po otrzymaniu sygnału zdefiniowanego przez użytkownika (np. SIGINT).
Klient to aplikacja wielowątkowa (patrz laboratorium 7, ćwiczenie nr 4). Za wysyłanie komunikatów odpowiada watek 1, za odbieranie wątek2. Watki działają asynchronicznie tzn. wątek 1 może wysłać kilka komunikatów zanim wątek 2 odbierze jakikolwiek komunikach zwrotny od serwera. Należy obsłużyć błąd przepełnienia kolejki oraz przepełnienia komunikatu.
Dla funkcji systemowych zaprogramować obsługę błędów w oparciu o funkcję perror() i zmienną errno.

*Treści oznaczone kursywą pochodzą z różnych źródeł internetowych.