Systemy operacyjne – studia stacjonarne 2023/24

Lab 9. Rozwiązywanie klasycznych problemów synchronizacji procesów przy pomocy mechanizmów IPC - implementacja problemu producent – konsument z wykorzystaniem pamięci dzielonej, kolejki komunikatów i semaforów. Projekt nr 2.

Segmenty pamięci dzielonej są dołączane do przestrzeni adresowych różnych procesów, dzięki czemu dane w nich zawarte stają się dostępne dla tych procesów. Operacje wejścia/wyjścia przy użyciu segmentu pamięci dzielonej są znacznie wydajniejsze/szybsze niż przy użyciu plików. Użycie segmentu pamięci do komunikacji pomiędzy procesami wymaga ich synchronizacji np. przy pomocy semaforów.

PAMIĘĆ DZIELONA

Do wywołania funkcji niezbędne są następujące pliki nagłówkowe:

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

1. Klucz do segmentu pamięci dzielonej.

   Do tworzenia lub do odwoływania się do segmentu pamięci dzielonej 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. Tworzenie i uzyskiwanie dostępu do segmentu pamięci dzielonej: shmget().

pliki nagłówkowe	<sys/types.h>, <sys/ipc.h>, <sys/shm.h>		
prototyp	int shmget (key_t key, size_t size, int shmflg)		
zwracana wartość	sukces	porażka	zmiana errno
	identyfikator pamięci dzielonej	-1	tak

   key - klucz do segmentu pamięci dzielonej (różne procesy, które chcą korzystać z tego samego segmentu, muszą użyć tego samego klucza), ftok() lub IPC_PRIVATE

   size– rozmiar pamięci w bajtach (0 jeżeli segment istnieje)

   shmflg – flaga określająca sposób wykonania funkcji i prawa dostępu do segmentu pamięci dzielonej:

   IPC_CREAT – utworzenie segmentu pamięci dzielonej lub uzyskanie dostępu do istniejącego segmentu

   IPC_EXCL – użyta w połączeniu z IPC_CREAT zwraca błąd, jeżeli dla danego klucza istnieje już segment pamięci dzielonej

   PRAWA DOSTĘPU – podobnie jak dla pliku np. 0666

   Dodatkowe informacje:

   http://www.linux.pl/man/index.php?command=shmget

3. Dołączenie (zyskanie dostępu) pamięci dzielonej: funkcja shmat().

pliki nagłówkowe	<sys/types.h>, <sys/ipc.h>, <sys/shm.h>		
Prototyp	void *shmat(int shm_id, const void *shm_addr, int shmflg)		
zwracana wartość	sukces	porażka	zmiana errno
	wskaźnik do pierwszego bajtu pamięci	-1	tak

shm_id – identyfikator segmentu pamięci (zwracany przez shmget())

shm_addr – adres pod którym pamięć zostanie dołączona do bieżącego procesu (zazwyczaj 0)

shmflg – flaga będąca zbiorem znaczników bitowych (zazwyczaj 0):

SHM_RDN – w połączeniu z shm_addr kontroluje adres

SHM_RDONLY – dołączona pamięć tylko do odczytu

Segment pamięci dzielonej można przyłączyć wielokrotnie do tego samego procesu, w różne miejsca pamięci wirtualnej procesu.

Dodatkowe informacje:

http://www.linux.pl/man/index.php?command=shmat

4. Odłączenie pamięci dzielonej: funkcja shmdt().

pliki nagłówkowe	<sys/types.h>, <sys/ipc.h>, <sys/shm.h>		
Prototyp	int shmdt(const void *addr)		
zwracana wartość	sukces	porażka	zmiana errno
	0	-1	tak

addr – wskaźnik do adresu zwróconego przez shmat()

Po odłączeniu segmentu pamięci dzielonej od procesu dane w segmencie pozostają niezmienione, nawet gdy nie jest on już przyłączony do innego procesu. Dane te można później odczytać w innym procesie. Funkcje exit() i exec() odłączają wszystkie przyłączone do procesu segmenty pamięci dzielonej.

Dodatkowe informacje:

http://www.linux.pl/man/index.php?command=shmdt

5. Sterowanie pamięcią dzieloną: funkcja shmctl().

pliki nagłówkowe	<sys/types.h>, <sys/ipc.h>, <sys/shm.h>		
prototyp	int shmctl(int shm_id, int cmd, struct shmid_ds *buf)		
zwracana wartość	sukces	porażka	zmiana errno
	0 lub wartość żądana przez cmd	-1	tak

shm_id – identyfikator segmentu pamięci (zwracany przez shmget())

cmd – operacje na segmencie pamięci:

IPC_RMID – usuwa segment pamięci dzielonej

IPC_SET – na podstawie wartości struktury wskazywanej przez argument buf ustawienie pól shm_perm.uid, shm_perm.gid, shm_perm.mode w strukturze informacyjnej shmid_ds

IPC_STAT – przekazanie w argumencie buf bieżącej zawartości struktury shmid_ds

buf – wskaźnik do struktury zawierającej tryby i zezwolenia

```
struct shmid_ds{
        uid_t shm_perm.uid;
        uid_t shm_perm.gid;
        mode_t shm_perm.mode;}
```

Segment jest usuwany gdy nie jest przyłączony do żadnego procesu.

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

6. Użyteczne komendy:

ipcs –m	podaje informacje nt. segmentów pamięci (patrz man ipcs)
ipcrm –m shmid	usuwa segment pamięci o numerze shmid
ipcs -l	podaje informacje dotyczące limitów

KOLEJKA KOMUNIKATÓW

7. 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

8. 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*/
    }
```

9. 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

10. 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

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

11. 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

12. 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

13. 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.

Skopiuj do swojego katalogu domowego, pliki .c znajdujące się w katalogu:
~suwada.anna/SO/P_K-kk
W trybie pracy krokowej przeanalizuj sposób działania programów i wykorzystaj je do realizacji Projektu 2.

Projekt nr 2.

Projekt składa się z trzech programów: mainp, producent i konsument. Uruchamiany jest program mainp, który uruchamia procesy producent i konsument. Producent produkuje liczby – PID procesu i umieszcza je we wspólnym buforze (**pamięć dzielona**), który może pomieścić MAX jednostek towaru naraz. Konsument pobiera towar (nie niszcząc bufora) i konsumuje go.
Aby panowała harmonia, muszą być spełnione dwa warunki:
- każda wyprodukowana jednostka towaru musi zostać skonsumowana,
- żadna jednostka towaru nie może być skonsumowana dwa razy (nawet jeśli konsument jest szybszy niż producent).

Wykorzystując mechanizmy IPC – pamięć dzieloną, kolejkę komunikatów i ewentualnie semafory zaimplementuj powyższe zadanie.