Systemy operacyjne – studia stacjonarne 2022/23

Lab 9. Rozwiązywanie klasycznych problemów synchronizacji procesów przy pomocy mechanizmów IPC - implementacja problemu producent – konsument z wykorzystaniem pamięci dzielonej. 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.

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

Ćwiczenie 1.
Skopiuj do swojego katalogu domowego, plik pam_dzielona.c znajdujący się w katalogu:
/home/inf-prac/wojtas.jan/Dydaktyka/SO/Projekty/PAMIEC_DZIELONA
W trybie pracy krokowej przeanalizuj sposób działania funkcji związanych z pamięcią dzieloną.

Projekt nr 2.

Projekt składa się z dwóch programów uruchamianych niezależnie: producenta i konsumenta.
Producent zajmuje się produkcją towaru (np. liczb, znaków) i umieszczaniu ich we wspólnym buforze (**pamięć dzielona**), który może pomieścić tylko jedną jednostkę 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 mechanizm semaforów zaimplementuj powyższe zadanie. Dla zademonstrowania, że nie doszło do utraty lub zwielokrotnienia towaru niech producent pobiera „surowiec” (liczby, znaki) do wytwarzania towaru z pliku tekstowego, a konsument umieszcza pobrany towar w innym pliku tekstowym. Po zakończeniu działania programów (wyczerpaniu zasobów „surowca”) oba pliki tekstowe powinny być identyczne oraz powinny być usunięte użyte w zadaniu struktury systemowe: zbiór semaforów i segment pamięci dzielonej. Do symulacji różnych prędkości działania programów użyć np. funkcji sleep() z losową liczbą sekund. Zaimplementuj obsługę przerwania wywołania funkcji systemowej semop().

*Treści oznaczone kursywą pochodzą z różnych źródeł internetowych.