IPC: pamięć wspólna, semafory - materiały pomocnicze

Uwaga: podstawowe aspekty IPC Systemu V oraz POSIX, np. sposób tworzenia kluczy czy identyfikacji obiektów, zostały już przedstawione w materiałach pomocniczych do ćwiczenia 6. Nie będą one ponownie omawiane w niniejszym dokumencie.

Semafory

Semafory są jednym z najważniejszych mechanizmów synchronizacji dostępu do współdzielonych zasobów. Może na rozumieć jako zmienne licznikowe dla których zdefiniowane dwie atomowe operacje:

- Dekrementacja semafora: powoduje zmniejszenie jego semafora o 1, pod warunkiem że wyjściowa wartość była >0. Jeśli wyjściowa wartość semafora wynosi 0, dekrementacja powoduje zablokowanie wykonującego ją procesu. Proces będzie zablokowany do momentu zwiększenia wartości semafora (przez inny proces). Po odblokowaniu proces zmniejszy wartość semafora o 1.
- Inkrementacja semafora: powoduje zwiększenie wartości semafora o 1. Jeśli semafor blokuje jeden lub więcej procesów, zwiększenie jego wartości spowoduje odblokowanie jednego oczekującego procesu.

Semafory w IPC systemu V.

W systemie V semantyka semaforów jest rozszerzona w stosunku do klasycznej ich definicji:

1. Semafory IPC systemu V można zmniejszać lub powiększać o wartości większe niż 1. Jednak po każdej operacji wartość semafora musi być >=0. Operacja która narusza ten warunek jest blokowana. Blokada trwa do momentu, gdy wartość semafora pozwoli wykonać operację z zachowaniem warunku wartości końcowej >=0.
2. W systemie V można wykonać operacje na wielu semaforach równocześnie. Operacje te wykonywane są w sposób atomowy, tzn. wykonywane są wszystkie wskazane operacje (jeśli jest to możliwe) lub wszystkie wskazane operacje są blokowane (jeśli ze względu na aktualne wartości semaforów nie można wykonać wszystkich operacji na raz).
3. System V definiuje operację blokowania procesu do momentu, gdy semafor przyjmie wartość 0.

Operacje na semaforach systemu V są zdefiniowane w pliku nagłówkowym `sys/sem.h` Dodatkowo warto również dołączyć pliki nagłówkowe `sys/ipc.h` i `sys/types.h`

Aby stworzyć zbiór semaforów korzystamy z funkcji `semget`

```
int semget(key_t key, int nsems, int flag);
```

Funkcja ta zwraca identyfikator zbioru semaforów, który można następnie wykorzystać w innych funkcjach operujących na semaforach. Argument `key` wskazuje klucz zbioru semaforów, zaś w argumencie `flag` przekazywane są flagi modyfikujące proces tworzenia obiektu IPC (patrz materiały do ćwiczenia 6). Liczba semaforów do utworzenia przekazywana jest w argumencie `nsems` Funkcję `semget` można również wykorzystać do pobrania identyfikatora istniejącego już zbioru semaforów (np. utworzonego przez inny proces). Wówczas wartość `nsems` powinna wynosić 0. Bezpośrednio po utworzeniu, wartości semaforów są niezdefiniowane. Należy je zainicjalizować, np. za pomocą funkcji `semctl` (patrz poniżej).

Do wykonania operacji na zbiorze semaforów służy funkcja `semop`

```
int semop(int semid, struct sembuf *sops, unsigned nsops);
```

Funkcja ta wykonuje operacje na zbiorze semaforów o identyfikatorze `semid` Liczba operacji do wykonania jest zdefiniowana w argumencie `nsops` Same operacje przekazywane w tablicy `sops` (zawierającej `nsops` elementów). Każda operacja zdefiniowana jest jako struktura postaci:

```
struct sembuf
{
   unsigned short sem_num;
   short sem_op;
   short sem_flg;
};
```

gdzie:

- `sem_num` to numer semafora (w zbiorze) na którym należy wykonać operację,

- `sem_op` to operacje do wykonania,
- `sem_flg` to flagi operacji.

Wartość `sem_op < 0` oznacza operację zmniejszenia wartości semafora o `sem_op` Wartość `sem_op > 0` oznacza operację zwiększenia wartości semafora o `sem_op` Wartość `sem_op == 0` oznacza operację oczekiwania, aż wartość semafora będzie wynosić 0. Dla pola `sem_flg` zdefiniowano dwie istotne flagi: flaga `IPC_NOWAIT` oznacza, iż operacja nie powinna blokować procesu. Jeśli operacji nie można wykonać (ze względu na wartość semafora) i ustawiona jest flaga `IPC_NOWAIT`, funkcja `semop` zwróci błąd. Flaga `SEM_UNDO` oznacza, że w przypadku zakończenia procesu operacja wykonana na semaforze powinna zostać cofnięta.

Na koniec warto również zwrócić uwagę na funkcję `semctl`, która pozwalającej wykonać pewne dodatkowe operacje na zbiorze semaforów. Funkcja ta ma sygnaturę:

```
int semctl(int semid, int semnum, int cmd, union semun arg);
```

gdzie:

- `semid` to identyfikator zbioru semaforów,
- `semnum` to numer semafora w zbiorze,
- `cmd` to operacja do wykonania,
- `arg` to unia bitowa przekazująca pewne dodatkowe argumenty.

Funkcja `semctl` pozwala wykonać szereg operacji na semaforze, z który najistotniejszymi są:

- `SETVAL` ustawienie wartości semafora na liczbę przekazaną w polu `arg.val`
- `GETVAL` pobranie wartości semafora,
- `IPC_RMID` usunięcie zbioru semaforów z systemu.

Semafory w IPC POSIX.

Semafory POSIXa realizują klasyczną semantykę semaforów. Aby z nich korzystać do programu należy dołączyć pliki nagłówkowe: `semaphore.h`, `sys/stat.h` oraz `fcntl.h` Ponadto program należy zlinkować z biblioteką `pthread`

Do utworzenia semafora służy funkcja `sem_open`

```
sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value);
```

Funkcja ta zwraca adres semafora lub `SEM_FAILED` w przypadku wystąpienia błędu. Argument `name` określa nazwę semafora, zaś argument `oflag` określa tryb otwarcia (patrz materiały do ćwiczenia 6). Argument `value` określa początkową wartość semafora. Inkrementację semafora (o wartość 1) realizuje funkcja `sem_post`

```
int sem_post(sem_t *sem);
```

Funkcja ta przyjmuje wskaźnik na semafor i zwraca 0 w przypadku sukcesu oraz -1 w przypadku wystąpienia błędu. Dekrementacja semafora realizowana jest analogicznie zdefiniowaną funkcją `sem_wait`

```
int *sem_wait(sem_t *sem);
```

Funkcja ta posiada również wariant nieblokującysem_trywait

```
int *sem_trywait(sem_t *sem);
```

Jeśli dekrementacja semafora nie jest możliwa (semafor ma wartość 0) funkcja `sem_trywait` nie blokuje procesu, lecz zwraca wartość -1 i ustawia zmienną `errno` na `EAGAIN`.
POSIX pozwala również odczytać aktualną wartość semafora. Służy do tego funkcja `sem_getvalue`

```
int sem_getvalue(sem_t *sem, int *valp);
```

Aktualna wartość semafora zapisywana jest pod adresem wskazywanym przez argument `valp`
Po zakończeniu pracy z semaforem należy go zamknąć. Służy do tego funkcja `sem_close`

```
int sem_close(sem_t *sem);
```

Semafor usuwamy za pomocą funkcji `sem_unlink`.

Funkcje `sem_getvalue` oraz `sem_close` zwracają 0 w przypadku sukcesu oraz -1 w przypadku wystąpienia błędu.

Pamięć wspólna

Co do zasady, każdy proces w systemie posiada odrębną wirtualną przestrzeń adresową. Zmiany zawartości pamięci dokonywane przez jeden proces nie są widoczne w innych procesach. Mechanizm pamięci wspólnej stanowi wyjątek od tej zasady - umożliwia podłączenie segmentu pamięci do przestrzeni adresowej wielu procesów. Pozwala to na komunikację między procesami bez konieczności dodatkowego kopiowania danych. Mechanizm ten nie zapewnia jednak żadnej synchronizacji dostępu do wspólnej pamięci. Z reguły konieczne więc będzie wykorzystanie dodatkowych mechanizmów synchronizacji (np. semaforów) aby zapewnić prawidłową sekwencję odczytów/zapisów z/do pamięci wspólnej.

Pamięć wspólna w IPC systemu V.

W IPC systemu V operacje na pamięci wspólnej są zdefiniowane w pliku nagłówkowym `sys/shm.h` Dodatkowo warto również dołączyć pliki nagłówkowe `sys/ipc.h` oraz `sys/types.h`

Aby stworzyć segment pamięci wspólnej korzystamy z funkcji `shmget`

```
int shmget(key_t key, size_t size, int shmflg);
```

Funkcja ta zwraca identyfikator segmentu pamięci wspólnej. Argument `key` oznacza klucz segmentu pamięci wspólnej a argument `flag` przekazuje flagi modyfikujące proces tworzenia obiektu IPC (patrz materiały do ćwiczenia 6). Rozmiar segmentu pamięci wspólnej przekazywany jest w argumencie `size` Funkcja ta pozwala również uzyskać identyfikator istniejącego już segmentu pamięci wspólnej. Wówczas argument `size` powinien mieć wartość 0. Dołączenie segmentu pamięci wspólnej do przestrzeni adresowej procesu realizuje funkcja `shmat`

```
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

Funkcja ta zwraca adres pod którym dołączono segment pamięci wspólnej. W przypadku błędu zwracana jest wartość `(void *) -1`. W argumencie `semid` przekazywany jest identyfikator segmentu. Argument `shmaddr` pozwala wskazać adres, pod którym system powinien dołączyć segment pamięci wspólnej. Zaleca się by miał on wartość `NULL`, oznaczającą, że system operacyjny sam dobierze odpowiedni adres. W argumencie `shmflg` przekazywane są flagi modyfikujące działanie funkcji `shmat` Najciekawszą z nich jest flaga `SHM_RDONLY`, pozwalająca dołączyć segment pamięci w trybie tylko do odczytu. Po zakończeniu pracy z segmentem należy go odłączyć, korzystając z funkcji `shmdt`

```
int shmdt(const void *shmaddr);
```

Funkcja ta przyjmuje w argumencie adres zwrócony przez funkcję `shmat` i zwraca 0 w przypadku powodzenia oraz -1 w przypadku błędu. Na koniec warto również wspomnieć o funkcji `shmctl`

```
void *shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

Pozwala ona, między innymi, usunąć segment pamięci wspólnej z systemu. W tym celu należy ją wywołać przekazując jako drugi argument polecenie `IPC_RMID` (argument `buf` jest wówczas ignorowany). Po wykonaniu tego polecenia nie będzie możliwe dołączenie segmentu do kolejnych procesów. Segment zostanie usunięty po odłączeniu przez wszystkie procesy, które uprzednio dołączyły go do swojej przestrzeni adresowej.

Pamięć wspólna w IPC POSIX.

Aby korzystać z pamięci wspólnej w IPC POSIX do programu należy dołączyć pliki nagłówkowe: `sys/mman.h`, `sys/stat.h` oraz `fcntl.h` Ponadto program należy zlinkować z biblioteką `rt`

Do utworzenia segmentu pamięci wspólnej (lub otwarcia istniejącego już segmentu) służy funkcja `shm_open`

```
int shm_open(const char *name, int oflag, mode_t mode);
```

Funkcja ta zwraca deskryptor plików reprezentujący segment pamięci wspólnej. W przypadku błędu zwracana jest wartość -1. Argument `name` określa nazwę segmentu, zaś argument `oflag` określa tryb otwarcia (patrz materiały do ćwiczenia 6). Po utworzeniu segmentu należy określić jego rozmiar. W tym celu należy skorzystać z funkcji `ftruncate`

```
int ftruncate(int fd, off_t length);
```

W pierwszym argumencie przekazywany jest deskryptor zwrócony przez funkcję `shm_open`. W drugim argumencie przekazywany jest pożądany rozmiar segmentu (w bajtach). Funkcja ta zwraca 0 w przypadku powodzenia oraz -1 w przypadku wystąpienia błędu. Otwarty segment pamięci wspólnej należy dołączyć do przestrzeni adresowej procesu. Operację tą można zrealizować za pomocą funkcji `mmap`

```
void *mmap(void *addr, size_t len, int prot, int flags, int fd, off_t offset);
```

Funkcja ta zwraca adres dołączonego segmentu lub, w przypadku błędu, wartość `(void *) -1`. W argumentach przekazywane są:

- `addr` określa adres pod którym powinien zostać dołączony segment pamięci wspólnej; zaleca się przekazanie wartości `NULL`, wskazującej, że system sam powinien dobrać odpowiedni adres,

- `len` to liczba bajtów segmentu mapowanych do przestrzeni adresowej procesu,
- `prot` określa prawa dostępu do mapowanej pamięci; prawa te są określane flagami `PROT_READ` (odczyt), `PROT_WRITE` (zapis),`PROT_EXEC` (prawo wykonania), `PROT_NONE` (brak uprawnień),
- flags - specyfikacja użycia segmentu (np. MAP_SHARED, MAP_PRIVATE, MAP_FIXED)
- `fd` jest deskryptorem plików zwróconym przez funkcję `shm_open`,
- `offset` określa przesunięcie mapowanego obszaru względem początku segmentu pamięci wspólnej; z reguły przyjmuje wartość 0.

Po zakończeniu pracy z segmentem pamięci wspólnej należy go odłączyć od przestrzeni adresowej procesu. Służy do tego funkcja `munmap`

```
int munmap(void *addr, size_t len);
```

W argumencie `addr` należy przekazać adres, pod którym segment został dołączony a w argumencie `len` rozmiar segmentu. Odłączony segment można następnie oznaczyć do usunięcia. Służy do tego funkcja `shm_unlink`

```
int shm_unlink(const char *name);
```

W argumencie `name` przekazywana jest nazwa segmentu do usunięcia. Po wykonaniu funkcji `shm_unlink` nie będzie już możliwe otwarcie tego segmentu funkcją `shm_open`. Co więcej, po odłączeniu go przez wszystkie procesy, które uprzednio dołączyły go do swojej przestrzeni adresowej, zostanie on usunięty z zasobów systemu. Funkcje `munmap` i `shm_unlink` zwracają 0 w przypadku powodzenia oraz -1 w przypadku błędu.

Ostatnia modyfikacja: wtorek, 7 maja 2019, 14:41

◄ Zadania - Zestaw 6

Przejdź do...

Zadania - Zestaw 7 ►