Spis treści

Na podstawie:

- *M.J.Rochkind - Programowanie w systemie UNIX dla zaawansowanych (Advanced UNIX Programming)*
- `https://ai.ia.agh.edu.pl/wiki/pl:dydaktyka:so:2017:labs:lab_intro`
- *Graham Glass, King Ables - Linux dla programistów i użytkowników*

1 Pojęcia podstawowe potrzebne do zrozumienia tematu

Kolejki komunikatów
Jest to mechanizm pozwalający na wymianę danych pomiędzy procesami.

Semafor
Semafory służą do synchronizacji procesów korzystających ze wspólnych zasobów. Można wyróżnić semafory binarne i uogólnione. Semafory uogólnione służą do obsługi zasobów, których może być wiele, są one początkowo ustawiane na liczbę reprezentującą liczbę dostępnych zasobów danego typu. Dodatnia wartość semafora oznacza, że semafor jest dostępny. Semafory binarne kontrolują dostęp tyko do jednego zasobu, mogą mieć więc stan 0 lub 1.

Pamięć współdzielona
Jest to najszybszy mechanizm współpracy pomiędzy procesami. Polega na stworzeniu przez jeden proces segmentu pamięci współdzielonej. Do tego segmentu procesy mogą zapisywać i czytać z niego dane w zależności od praw dostępu. Do synchronizacji dostępu do pamięci współdzielonej wykorzystywane są np. semafory.

- Czym są kolejki komunikatów ?
- Czym są semafory ?
- Czym jest pamięć współdzielona ?

2 System V

2.1. Semafory, semget

W System V posługujemy się zbiorami semaforów, a nie jednym semaforem. Aby utworzyć zbiór semaforów korzystamy z funkcji semget.

```
semget-get semaphore-set identifier
#include <sys/sem.h>
int semget(
  key_t key, /* key */
  int nsems, /* size of set */
  int flags /* flags */
); /* Returns identifier or -1 on error (sets errno) */
```

Identyfikator key jest zamieniany na identyfikator zbioru semaforów, nsems to liczba semaforów w zbiorze. flags służy do konfiguracji zbioru semaforów (numerowane od 0), aby utworzyć zbiór musi być ustawiona flaga IPC_CREAT. Po wywołaniu semafory nie są jeszcze jednak gotowe do użytku, konieczne jest wywołanie semctl.

- Jak działa semget ?

2.2. semctl

To funkcja, która konfiguruje semafor (np. ustawia jego wartość początkową na zera lub inną itp.). Funkcja ta przyjmuje jako argument unię, która należy samemu zdefiniować.

```
semctl-control semaphore set
#include <sys/sem.h>
int semctl(
  int semid, /* identifier */
  int semnum, /* semaphore number */
  int cmd, /* command */
  union semun arg /* argument for command */
); /* Returns value or 0 on success; -1 on error (sets errno) */
```

```
union semun-union for semctl
union semun {
  int val; /* integer */
  struct semid_ds *buf; /* pointer to structure */
  unsigned short *array; /* array */
};
```

```
struct semid_ds-structure for semctl
struct semid_ds {
  struct ipc_perm sem_perm; /* permission structure */
  unsigned short sem_nsems; /* size of set */
  time_t sem_otime; /* time of last semop */
  time_t sem_ctime; /*
  time of last semctl */
};
```

Przykładowe utworzenie semafora i inicjalizacja wartości na 0 mogą wyglądać tak:

```
ec_neg1(semid = semget(key, 1, PERM_FILE | IPC_CREAT) )
arg.val = 0;
ec_neg1( semctl(semid, 0, SETVAL, arg) )
```

Problem z tą implementacją jest taki, że w czasie ustawiania parametrów inny wątek może zacząć korzystać z jeszcze nie gotowego semafora. Można to rozwiązać sprawdzając czas sem_otime, dopóki jest 0, inny wątek nie może korzystać z semafora.

- Do czego służy semctl ?

2.3. semop

To funkcja zarządzająca semaforami (otwiera i zamyka semafory).

```
semop-operate on semaphore set
#include <sys/sem.h>
int semop(
  int semid, /* identifier */
  struct sembuf *sops, /* operations */
  size_t nsops /* number of operations */
); /* Returns 0 on success or -1 on error (sets errno) */
```

```
struct sembuf-structure for semop
struct sembuf {
  unsigned short sem_num; /* semaphore number */
  short sem_op; /* Semaphore operation */
  short sem_flg; /* Operation flags */
};
```

Funkcja semop może działać również na całym zbiorze semaforów, ale wtedy trzeba do niej przekazać tablicę struktur sembuf, jeśli chcemy operować tylko na jednym semaforze, to wystarczy, że przekażemy tylko adres jednej struktury. sem_op to argument określający co chcemy zrobić z semaforem, jeśli jest liczbą dodatnią to zostanie ona dodana to semafora, jeśli jest liczbą ujemną to zostanie odjęta od semafora.

- Jak działa i do czego służy semop ?

2.4. Pamięć współdzielona, shmget

shmget to funkcja, która tworzy segment pamięci współdzielonej. Aby utworzyć segment należy przekazać do niej flagi IPC_CREAT oraz IPC_PRIVATE.

```
shmget-get shared memory segment
#include <sys/shm.h>
int shmget(
  key_t key, /* key */
  size_t size, /* size of segment */
  int flags /* creation flags */
); /* Returns shared-memory identifier or -1 on error (sets errno) */
```

- Jak działa i do czego służy shmget ?

2.5. shmctl

To funkcja służąca do konfiguracji pamięci współdzielonej.

```
shmctl-control shared memory segment
#include <sys/shm.h>
int shmctl(
  int shmid, /* identifier */
  int cmd, /* command */
  struct shmid_ds *data /*
  data for command */
); /* Returns 0 on success or -1 on error (sets errno) */
```

```
struct shmid_ds-structure for shmctl
struct msqid_ds {
  struct ipc_perm shm_perm; /* permission structure */
  size_t shm_segsz; /* size of segment in bytes */
  pid_t shm_lpid; /* process ID of last shared memory op */
  pid_t shm_cpid; /* process ID of creator */
  shmatt_t shm_nattch; /* number of current attaches */
  time_t shm_atime; /* time of last shmat */
  time_t shm_dtime; /* time of last shmdt */
  time_t shm_ctime; /* time of last change by shmctl */
};
```

- Jak działa i do czego służy shmctl ?

2.6. shmat, shmdt

To funkcje służące do używania pamięci współdzielonej przez procesy.

```
shmat-attach shared memory segment
#include <sys/shm.h>
void *shmat(
  int shmid, /* identifier */
  const void *shmaddr, /* desired address or NULL */
  int flags /* attachment flags */
); /* Returns pointer or -1 on error (sets errno) */
```

```
shmdt-detach shared memory segment
#include <sys/shm.h>
int shmdt(
  const void *shmaddr /* pointer to segment */
); /* Returns 0 on success or -1 on error (sets errno) */
```

- Jak działa i do czego służy shmat, shmdt ?

3 Zakleszczenie

Zakleszczenie (ang. deadlock) jest sytuacją, w której co najmniej dwie różne akcje (procesy, wątki) działają na siebie nawzajem - żadna więc nie może się zakończyć. Zakleszczenia występują, gdy wiele zadań konkuruje jednocześnie o wyłączny dostęp do zasobu.

W najprostszym przypadku, zakleszczenie powstaje pomiędzy dwoma procesami. Każdy z nich zajmuje pewien zasób, jednocześnie ubiegając się o dostęp do zasobu zajmowanego przez drugi proces.

W bardziej złożonym przypadku, więcej niż dwa procesy mogą oczekiwać na zasoby zajmowane przez kolejny proces, tworząc łańcuch cykliczny.

4 Zagłodzenie

Zagłodzenie procesu (ang. process starvation) jest sytuacją, w której proces nie może kontynuować działania, ponieważ nie może otrzymać dostępu do wymaganego zasobu.

Występuje ono najczęściej na skutek źle działającego mechanizmu szeregowania zadań, który zamiast sprawiedliwie zapewniać wszystkim procesom dostęp do zasobu, pomija niektóre z nich.

Do zjawiska inwersji priorytetów dochodzi, gdy proces o wyższym priorytecie czeka na zakończenie procesu z niższym priorytetem, a ten z kolei ulega zagłodzeniu wskutek błędnego szeregowania zadań.

- Na czym polega zakleszczenie ?
- Na czym polega zagłodzenie ?