Sisteme de operare

Laborator 10

Semestrul I 2024-2025

Laborator 10

- IPC System V
  - cozi de mesaje
- POSIX threads
  - mutex-uri
  - semafoare
  - monitoare

Cozi de mesaje

- create cu *msgget*

int msgget(key_t key, int flag);

ex: flag IPC_CREAT |IPC_EXCL | 0644

- atributele lor (stocate intr-o structura *msqid_ds*) pot fi manipulate cu *msgctl*

*int msgctl(int msgid, int cmd, struct msgid_ds *buf);*

Valori *cmd*: IPC_STAT, citeste structura *msgid_ds* asociata cozii

IPC_SET, seteaza UID/GID, *mode* si dimensiunea cozii (dimensiunea cozii poate fi crescuta doar de *root*)

IPC_RMID, sterge imediat coada din sistem (si mesajele din ea; alte procese care folosesc ulterior coada primesc EIDRM)

Obs: pt SET si RMID e nevoie fie de drepturi de root fie ca

UID creator = UID proprietar

Cozi de mesaje (cont.)

- mesajele se scriu/citesc in/din coada cu *msgsnd*/*msgrcv*
- apelurile folosesc o structura definita de utilizator

```
struct msg {
        long mtype;             // tip mesaj, valoare pozitiva
        char mtext[BUFSIZE];    // date mesaj
}
```

*int msgsnd(int msgid, const void *ptr, size_t nbytes, int flag);*

*int msgrcv(int msgid, void *ptr, size_t nbytes, long type, int flag);*

- pt. type = 0 la *msgrcv* se citeste primul mesaj din coada (politica FIFO), altfel se citeste primul mesaj din coada cu acel tip

POSIX Mutex Locks

- crearea si initializarea lock-ului mutex

```
#include <pthread.h>

pthread_mutex_t mutex;

/* create and initialize the mutex lock */
pthread_mutex_init(&mutex,NULL);
```

- obtinerea si eliberarea lock-ului

```
/* acquire the mutex lock */
pthread_mutex_lock(&mutex);

/* critical section */

/* release the mutex lock */
pthread_mutex_unlock(&mutex);
```

Semafoare POSIX

- cu nume (*named*) sau fara nume (*unnamed*)
- cele cu nume pot fi folosite de procese diferite (neinrudite), cele fara nume nu

Semafoare POSIX cu nume

- creare si initializare:

```
#include <semaphore.h>
sem_t *sem;

/* Create the semaphore and initialize it to 1 */
sem = sem_open("SEM", O_CREAT, 0666, 1);
```

- alte procese acceseaza semaforul cu ajutorul numelui **SEM**. (de fapt /SEM, sunt fisiere intr-un pseudo-sistem de fisiere */dev/shm*)
- obtinerea si eliberarea semaforului:

```
/* acquire the semaphore */
sem_wait(sem);

/* critical section */

/* release the semaphore */
sem_post(sem);
```

Semafoare POSIX fara nume

- creare si initializare:

```
#include <semaphore.h>
sem_t sem;

/* Create the semaphore and initialize it to 1 */
sem_init(&sem, 0, 1);
```

Arg 2 (*pshared*): 0 – semafor partajat intre threadurile aceluiasi proces

!= 0 – partajat intre procese si plasat in shared memory

- obtinerea si eliberarea semaforului:

```
/* acquire the semaphore */
sem_wait(&sem);

/* critical section */

/* release the semaphore */
sem_post(&sem);
```

Dealocarea semafoarelor

- semafoare cu nume:

*int sem_unlink(const char *name);*

Obs: semantica de fisiere

- numele semaforului indepartat imediat
- semaforul dispare insa doar dupa ce toate procesele care l-au deschis il inchid cu

*int sem_close(sem_t *sem);*

- semafoare fara nume:

*int sem_destroy(sem_t *sem);*

Semafoare POSIX vs System V

- POSIX create si intializate intr-o singura operatie, System V create intai si apoi initializate independent
- POSIX functioneaza cu increment/decrement 1, System V foloseste valori arbitrare
- POSIX are apeluri non-blocante (tip *trywait* sau timeout), System V foloseste IPC_NOWAIT
- POSIX identifica semafoarele cu/fara nume, System V foloseste chei

Variabile conditie POSIX

- POSIX folosit uzual impreuna cu C/C++
- C/C++ nu ofera suport pentru monitoare la nivel de limbaj

=> POSIX implementeaza monitoare la nivel de biblioteca *pthreads* prin asocierea variabilelor conditie cu mutex-uri

- creare si initializare “monitor” POSIX:

```
pthread_mutex_t mutex;
pthread_cond_t cond_var;

pthread_mutex_init(&mutex,NULL);
pthread_cond_init(&cond_var,NULL);
```

Variabile conditie POSIX (cont.)

- thread care asteapta ca o conditie, **`a == b,`** sa devina adevarata:

```
pthread_mutex_lock(&mutex);
while (a != b)
     pthread_cond_wait(&cond_var, &mutex);

pthread_mutex_unlock(&mutex);
```

- thread care deblocheaza alt thread care asteapta la o variabila conditie:

```
pthread_mutex_lock(&mutex);
a = b;
pthread_cond_signal(&cond_var);
pthread_mutex_unlock(&mutex);
```

Producator-consumator POSIX

```c
#include <stdio.h>
#include <pthread.h>

#define ITEMS           8
#define BUFFER_SIZE     4
int buffer[BUFFER_SIZE];

void *producer(void*);
void *consumer(void*);

int spaces, items, tail;
pthread_cond_t space, item;
pthread_mutex_t buffer_mutex;

int main()
{
        pthread_t producer_thread, consumer_thread;
        void *thread_return;
        int result;

        spaces = BUFFER_SIZE;
        items = 0;
        tail = 0;

        pthread_mutex_init(&buffer_mutex, NULL);
        pthread_cond_init(&space, NULL);
        pthread_cond_init(&item, NULL);

        if(pthread_create(&producer_thread, NULL, producer, NULL) ||
           pthread_create(&consumer_thread, NULL, consumer, NULL))
                exit(1);

        if(pthread_join(producer_thread, &thread_return))
                exit(1);
        else
                printf("producer returns with %d\n", (int)thread_return);

        if(pthread_join(consumer_thread, &thread_return))
                exit(1);
        else
                printf("consumer returns with %d\n", (int)thread_return);
        exit(0);
}

```