Responsible lecturer: Illés, Zoltán associate professor
Media & Educational Informatics Department

Shared memory, Semaphore

Goal: We are getting to know the quickest way to share data among processes to use the so called shared memory. We have to avoid that two or several processes modify the value of it or read data while one of them is not finished with overwriting its content. We have to synchronize their work somehow. We may use for this purpose signals but there is the possibility to use semaphores as well.

We learn about: *shmget, semctl – creates/deletes shared memory (System_V)(include sys/ipc.h, sys/shm.h); shmat, shmdt – shared memory operation (include sys/types.h, sys/shm.h); shm_open, shm_unlink – creates/deletes shared memory (POSIX) (include sys/mman.h, sys/stat.h, fcntl.h); mmap, munmap – map and unmap files into memory; semget, semctl – creates/deletes a semaphore family (System_V) (include sys/types.h, sys/ipc.h, sys/sem.h); semop, semctl – semaphore operations; sem_open, sem_unlink – creates, deletes a named semaphore (POSIX) (include fcntl.h, sys/stat.h, semaphore.h); sem_post, sem_wait – semaphore operations; sem_close – release a semaphore*

Tasks

1. Write a C program with a child process. Both the parent and the child should use a shared memory (System_V). The parent has to write a Hello message to the memory and the child has to read it out! (*What happens if you do not think of synchronization? Use a simple sleep to avoid it! Do not forget about deletion! If you still missed it you can use the ipcs and ipcrm commands (you know from message queues)!)*

```
int shmget(key_t key, size_t size, int shmflg);
//key - result of ftok or IPC_PRIVATE, size - required size of memory,
//shmflg - as before

void *shmat(int shmid, const void *shmaddr, int shmflg);
//to connect shared memory (shmid), gets back an address (shmaddr) usually
call with NULL, shmflg - usually 0, result the address you can achieve the
memory

int shmdt(const void *shmaddr);
//to release memory at address shmaddr (every process must release before
delete

int shmctl(int shmid, int cmd, struct shmid_ds *buf);
//cmd - IPC_RMID - delete, (IPC_SET etc.), buf - NULL at deletion
```

2. Modify the program and use a signal for synchronization! (*The child should wait with reading while it does not get a signal from the parent!*)

3. Modify the progam and use a POSIX semaphore to solve syncronization! (*You have to initialize the semaphore and create it at a closed state. So parent closes it for itself while it does not finish writing. After it the parent opens it and the child can see the value!*)

```
sem_t *sem_open(const char *name, int oflag);
sem_t *sem_open(const char *name, int oflag,
                       mode_t mode, unsigned int value);
//name - name of semaphore starts with „/”, oflag, mode - as before, value - 1
//means it is open, 0 it is closed, result the identifier of the semaphore

int sem_wait(sem_t *sem);
//try to close semaphore - success if it is open, waiting for it
```

Financed from the financial support ELTE won from the Higher Education Restructuring Fund of the Hungarian Government

```
int sem_post(sem_t *sem);
//release semaphore, open it

int sem_close(sem_t *sem);
//close semaphore sem

int sem_unlink(const char *name);
//delete semaphore with given name if everybody closed it
```

4. Modify the program and use System_V semaphore! (*Remember, now you have a set of semaphores! If you want to use a single semaphore you have to create a semaphore family which consists of only one element! To use an operation (to close or open) you have to give which semaphore you want to execute it!)*

```
int semget(key_t key, int nsems, int semflg);
//creates the semaphore family, key as befor with ftok, nsems – number of
semaphores, semflg – as before (must use an initialization with semctl)

int semop(int semid, struct sembuf *sops, size_t nsops);
//to set a semaphore (semid) with values sops, nsops – number of semaphores

int semctl(int semid, int semnum, int cmd, ...);
//semnum – number of semaphores, cmd – IPC-SET (set initialization value),
//IPC_RMID (delete)

struct sembuf {
           unsigned short sem_num;  /* semaphore number */
           short          sem_op;   /* semaphore operation +1, -1 */
           short          sem_flg;  /* operation flags, usually 0 */

}
```

5. Write a program in which the child process creates 50 random numbers into the shared memory. If it is finished the parent has to read them out and write them on the screen! *(You can use the shared memory as an integer array as well! So you do not have to make any conversions! Solve the problem of synchronization with semaphores!)*

6. Write a program to solve the well-known customer/producer problem! (At first *try to solve it by using System_V implementation!)*

7. Modify the customer/producer program by using POSIX implementations!

```
int shm_open(const char *name, int oflag, mode_t mode);
//name with starting „/”, oflag,mode  – as usual

int shm_unlink(const char *name);
//delete shared memory

void *mmap(void *addr, size_t lengthint „prot”, int flags ,
           int fd, off_t offset);
//addr – usually NULL, prot  - PROT_READ, PROT_WRITE – permissions, flags –
MAP_SHARED,MAP_PRIVATE, fd – file descriptor, return value the address

int munmap(void *addr, size_t length);
//delete mapping
```

Financed from the financial support ELTE won from the Higher Education Restructuring Fund of the Hungarian Government