
LUT
University
Operating Systems and System Programming
LAND
OF THE
CURIOUS

Experiment 5 – Project1

- POSIX semaphores
  - named semaphores
  - unnamed semaphores
- Unnamed semaphores are mainly used for synchronization between threads, and can also be used for synchronization between processes (generated by fork).
- Named semaphores can be used for synchronization between processes and between threads.

Experiment 5 – Project1

- Named semaphores
- *sem_t *sem_open(const char *name, int oflag);*
- *sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value);*
- *int sem_wait(sem_t *sem)*;
- *int sem_post(sem_t *sem)*;
- *int sem_close(sem_t *sem)*;
- *int sem_unlink(const char *name)*;

https://linux.die.net/man/7/sem_overview

Experiment 5 – Project1

- Observe the situation when the mutex semaphore is not used through an example (no_sem.c)
- Compile and run
- Observe the appearance rules of X and O, and analyze the reasons

```
[root@CentOS52 home]# gcc -o no_sem no_sem.c
[root@CentOS52 home]# ./no_sem & ./no_sem o
```

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main(int argc, char *argv[]){
    char message='X';
    int i=0;
    if(argc>1){
        message = argv[1][0];
    }
    for(i=0;i<10;i++){
        printf("%c",message);
        fflush(stdout);
        sleep(rand()%3);
        printf("%c",message);
        fflush(stdout);
        sleep(rand()%2);
    }
    sleep(10);
    exit(0);
}
```

no_sem.c

Experiment 5 – Project1

- Observe the situation when the mutex semaphore is used through an example (with_sem.c)
- Compile and run
- Observe the appearance rules of X and O, and analyze the reasons

```
[root@CentOS52 home]# gcc -o with_sem with_sem.c -lrt
[root@CentOS52 home]# ./with_sem & ./with_sem o
```

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<semaphore.h>
#include<fcntl.h>
#include<sys/stat.h>

int main(int argc, char *argv[]){
    char message='X';
    int i=0;
    if(argc>1){
        message = argv[1][0];
    }
    sem_t *mutex=sem_open("mysem",O_CREAT,0666,1);
    for(i=0;i<10;i++){
        sem_wait(mutex);
        printf("%c",message);
        fflush(stdout);
        sleep(rand()%3);
        printf("%c",message);
        fflush(stdout);
        sem_post(mutex);
        sleep(rand()%2);
    }
    sleep(10);
    sem_close(mutex);
    sem_unlink("mysem");
    exit(0);
}
```

with_sem.c

Experiment 5 – Project1

- Simulate playing chess with semaphores, where red and black take turns.
- Write two C language programs, black_chess.c and red_chess.c, to simulate that red moves and black moves in the process of playing chess respectively.
- Rules of movement: red first and black second; red and black take turns to move, until the 10th step, the red side wins and the black side loses.

Experiment 5 – Project1

- Set up two synchronization semaphores.
- (1) *hei:* The initial value is 1, which means that the black side has already moved, and it is the red side's turn to move (meeting the chess rule "red first, black second").
- (2) *hong*: The initial value is 0, which means that the red side has not moved yet.
- Before the red chess moves, test the semaphore *hei* to determine whether the black side has already moved. If so, it is the red side's turn to move, otherwise it is blocked and waits for the black side to move. Since the initial value of *hei* is 1, it must be the red side go first. After the red side moves, it sets the semaphore to notify the black side to move.
- Before the black side moves, it first tests the semaphore *hong* to determine whether the red side has already moved. If so, it is the turn of the black side to move, otherwise block and wait for the red side to move. Since the initial value of *hong* is 0, the black side will not move before the red side moves. After the black side moves, set the semaphore *hei* to notify the red side to move.