시스템프로그래밍

과 제 명 : Proxy 3-1

교 수 님 : 최상호 교수님

강의시간 : 월요일

학 번 : 2019202100

이 름 : 하주영

Introduction

semaphore란 여러 프로세스들이 한정된 수의 자원을 이용할 때, 한정된 수의 프로세스만 이용할 수 있게 하는 방법을 제시하는 개념이다

다른 프로세스가 이미 가지고 있는 semaphore를 요청시 그 프로세스는 휴먼 상태가 된다.

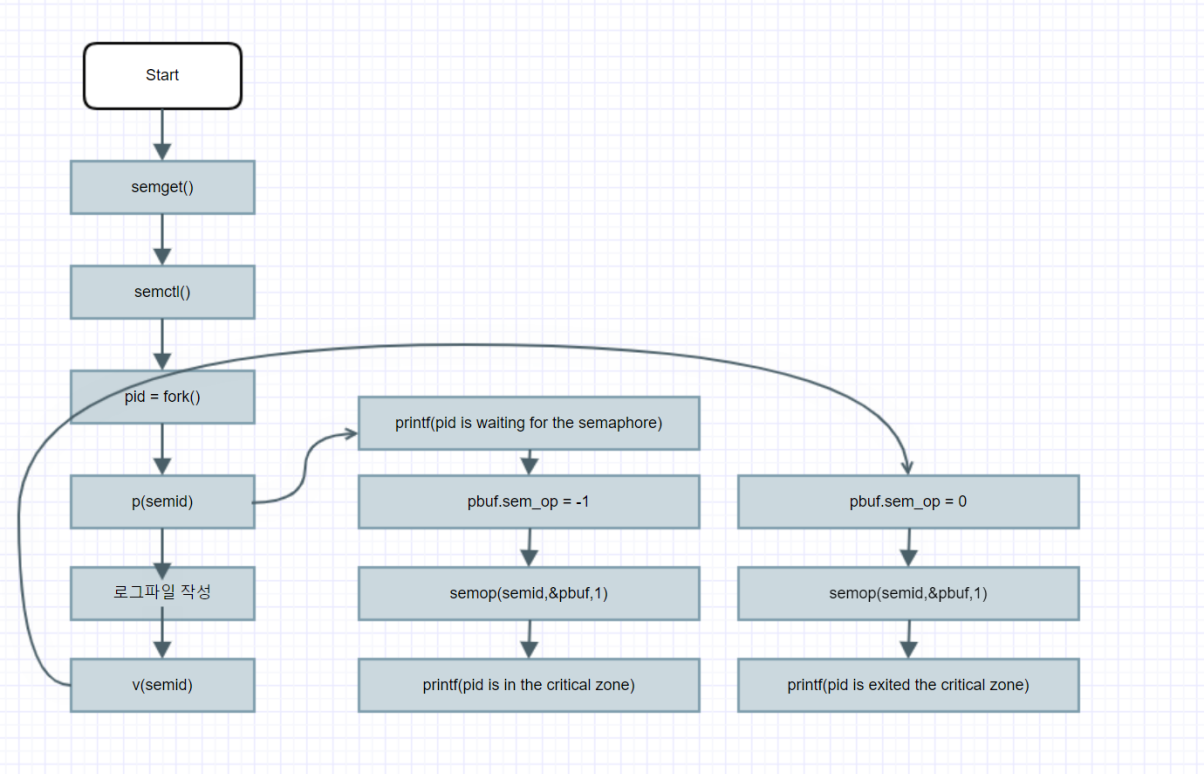
Race condition이란 여러 프로세스가 공유 데이터를 동시에 접근하는 상황을 말한다.

critical section을 실행 중일때, 그 critical section을 실행시키는 다른 프로세스가 없도록 semaphore를 사용한다.

이번 과제에서는 semaphore를 사용하여 한 번에 하나의 프로세스만 log file에 기록하도록 수정하고,

동시에 여러 프로세스가 접근했을 때를 시뮬레이션 하여 서버의 터미널에 상태를 출력하는 과제이다.

Flow Chart



Pseudo Code

/*메인함수*/

```
int main() {
        int semid, i;
        semid = semget((key_t)PORTNO,IPC_CREAT|0666);
        arg.val = 1;
        semctl(semid,0,SETVAL,arg);
        while(1) {
                pid = fork();
                if(pid == 0) {
                        if(isHit()) {
                                p(semid);
                                로그파일 작성
```

```
                        v(semid);
                else {
                        p(semid);
                        로그파일 작성
                        v(semid);
                }
        }
        semctl(semid,0,IPC_RMID,arg);
}
```

/*p연산*/

```
void p(int semid) {
        printf("*PID# %d is waiting for the semphore₩n", getpid());
        struct sembuf pbuf;
        pbuf.sem_num = 0;
        pbuf.sem_op = -1;
        pbuf.sem_flg = SEM_UNDO;
        semop(semid, &pbuf,1);
        printf("*PID# %d is in the critical zone.₩n",getpid());
}
```

/*v연산*/

```
void v(int semid) {
        struct sembuf vbuf;
        vbuf.sem_num = 0;
        vbuf.sem_op = 1;
        vbuf.sem_flg = SEM_UNDO;
        semop(semid, &vbuf, 1);
        printf("*PID %d is exited the critical zone.₩n", getpid());
}
```

Result Screen

1. Server Terminal

   PID = 2672 프로세스가 임계구역에 들어가 있기 때문에 PID = 2675가 임계구역안으로 들어가지 못함. PID=2672가 임계구역에서 탈출하고 비로소 PID = 2675가 들어갈 수 있음.

```
kw2019202100@ubuntu:~/2-4$ ./proxy_cache
*PID# 2672 is waiting for the semaphore
*PID# 2672 is in the critical zone.
*PID# 2675 is waiting for the semaphore
*PID 2672 is exited the critical zone.
*PID# 2675 is in the critical zone.
*PID 2675 is exited the critical zone.
```

2. Logfile

   입력한 url 순서대로 logfile에 작성되는 것을 확인

```
[Hit] ServerPID : 2672 |fed/818da7395e30442b1dcf45c9b6669d1c0ff6b -[2022/06/01, 01:27:51]
[Hit]www.naver.com
[Hit] ServerPID : 2675 |025/15aba49d232baf9567495aa44eb8d20b2b764 -[2022/06/01, 01:27:54]
[Hit]www.daum.net
```

Consideration

두 개의 프로세스가 동시에 접근하는 시나리오를 가정하여 p연산과 v연산이 제대로 작동하는 지 체크한다. 첫번째 프로세스가 Sleep() 함수를 사용하여 p연산에 들어간 순간 그 프로세스를 잠시 재우면 두번째 프로세스도 p연산에 들어갈 수 있다.

```
void p(int semid) {
        printf("*PID# %d is waiting for the semaphore\n", getpid());
        struct sembuf pbuf;
        pbuf.sem_num = 0;
        pbuf.sem_op = -1;
        pbuf.sem_flg = SEM_UNDO;
        if((semop(semid, &pbuf,1)) == -1) {
                perror("p : sempop failed");
                exit(1);
```

```
        }
        printf("*PID# %d is in the critical zone.\n",getpid());
        sleep(5);
}
```

다음은 p연산함수를 작성한 코드이다.

Semop(semid, &pbuf, 1) 함수는 세마포어 연산함수로 남은 세마포어가 있는지 확인하고 만약세마포어가 남아있다면 sem_op = -1을 통하여 세마포어를 1 감소시킨다.

따라서 첫번째 프로세스가 세마포어를 갖고 임계구역안으로 들어간다. 이 후 두번째 프로세스는 세마포어가 없기 때문에 첫번째 프로세스가 끝날 때까지 대기한다. 첫번째 세마포어가 임계구역을 나오면서 세마포어를 반납한다. 따라서 두번째 프로세스가 임계구역 안으로 들어갈 수 있게 된다.