



دانشگاه صنعتی امیرکبیر  
(پلی تکنیک تهران)  
دانشکده مهندسی برق

آزمایشگاه سیستم عامل  
آزمایش ۶  
همگام سازی فرایندها

نگارش  
علی بابالو  
پویا شریفی

استاد راهنما  
مهندس کیخا

خرداد ماه 1402

## بخش اول: مساله Reader-Writer:

با پیاده سازی مسئله خوانندگان و نویسندگان مشاهده میکنیم که ممکن است بین پردازش write با پردازش های read حالت مسابقه پیش بیاید زیرا ممکن است در یک لحظه پردازش write بخواند مقدار count را تغییر دهد و در همان لحظه نیز پردازش read بخواند این مقدار را بخواند به همین دلیل حالت مسابقه پیش می آید و ممکن است مقدار خواند شده ناصحیح باشد. باید با گذاشتن قفل روی ناحیه بحرانی از دسترسی همزمان خواننده و نویسنده به مقدار count جلوگیری کنیم. در این مسئله چون reader ها همزمان می توانند به مقدار count دسترسی داشته باشند در واقع می توان مسئله را به مسئله Peterson تبدیل کرد و برای خواننده با  $turn = 1$  و نویسنده با  $turn = 0$  ناحیه بحرانی را کنترل کرد. همچنین برای حل این مشکل می توان از semaphore و mutex نیز استفاده کرد.

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

int main()
{
    //key_t key = 2525;
    key_t key;
    key = ftok("/mnt/c/Users/98912/Desktop/uni/OS LAB/EXP 6", 'R');

    // shmget returns an identifier in shmid
    int shmid = shmget(key, sizeof(int), 0666|IPC_CREAT);

    // shmat to attach to shared memory
    int *count = (int*) shmat(shmid, (void*)0, 0);

    *count = 0;

    int n1 = fork();
    int n2 = fork();
```

```

if (n1 != 0 && n2 != 0){
while(*count < 10){
    (*count) ++;
    printf("Writer:\tPID: %d\tcount: %d\n", getpid(), *count);
    usleep(500);
}
    wait(NULL);
    wait(NULL);
    wait(NULL);
}
else if (n1 == 0 || n2 == 0){
while(*count < 100){
    int current_pid = getpid();
    printf("Reader:\tPID: %d\tcount: %d\n", current_pid, *count);
}
}
}
}

```

نتیجه در زیر مشخص است:

همانطور که مشخص است writer مقدار 7 را نوشته اما reader مقدار 6 را خوانده است که نشان race condition است.



فیلسوف کناری میماند و بن بست رخ میدهد. برای حل این مشکل لازم است که وقتی هر فیلسوف میخواهد غذا بخورد همزمان (در طی یک فرایند اتمیک) چوب راست و چپ را بردارد در این صورت مطمئن میشویم که بن بست رخ نمیدهد و همچنین در زمان هایی ممکن است دو فیلسوف هم در صورتی که چوب هایشان اشتراک نداشته باشند باهم غذا بخورند. در ادامه پیاده سازی این را حل را میبینیم.

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

#define N 5

sem_t pick_up;
pthread_t philosopher[N];
pthread_mutex_t chopsticks[N];
int count[N];

void pickup(int philosopher){
    pthread_mutex_lock(&chopsticks[philosopher]);
    pthread_mutex_lock(&chopsticks[(philosopher+1) % N]);
}

void putdown(int philosopher){
    pthread_mutex_unlock(&chopsticks[philosopher]);
    pthread_mutex_unlock(&chopsticks[(philosopher+1) % N]);
}

void think(int philosopher){
    printf("philosopher %d is thinking\n", philosopher);
    sleep(2);
}

void eat(int philosopher){
    printf("philosopher %d is eating with chopsticks(%d) and chopsticks(%d)\n",
           philosopher, philosopher, (philosopher+1) % N);
    sleep(2);
}

void finish(int philosopher){
    printf("philosopher %d finished eating\n", philosopher);
}

void *handle_philosopher (void *i){
```

```

int id = (int)(long) i;

count[id] = 1;
while(1){

    //think
    think(id);

    //pickup
    sem_wait(&pick_up);
    pickup(id);
    sem_post(&pick_up);

    //eat
    eat(id);

    //finish eating
    finish(id);

    //putdown
    putdown(id);

    count[id]++;
}
}

int main(){

    sem_init(&pick_up, 0 , 1);

    for (int i = 0; i < N; ++i)
    {
        if (pthread_mutex_init(&chopsticks[i], NULL))
        {
            printf("Mutex Initialize Failed\n");
            return 0;
        }
    }
    for (int i = 0; i < N; ++i)
    {
        pthread_create(&philosopher[i], NULL, handle_philosopher, (void *)i);
    }
    for (int i = 0; i < N; ++i)

```

```

{
    pthread_join(philosopher[i], NULL);
}

return 0;
}

```

برای پیاده سازی راه حل گفته شده از یک آرایه از mutex ها (یک mutex به ازای هر چوب) و یک سمافور برای اتمیک کردن فرآیند برداشتن دو چوب راست و چپ (pick up) استفاده میکنیم به این صورت که هر فیلسوفی بخواهد چوب بردارد باید روی سمافور برداشتن چوب ها wait کند و در صورتی که سمافور اجازه داد، چوب دستی ها را بردارد یعنی mutex مربوط به دو چوب دستی کناری خود را lock کند. و پس از برداشتن چوب ها signal را روی سمافور pick up صدا بزند تا نوبت برداشتن چوب به نفر بعدی برسد و بعد از اینکه غذا خورد باید چوب ها را با unlock کردن دو mutex ذکر شده آزاد کند.

خروجی کد به شکل زیر است:

```

● cavendish@LAPTOP-J4F04081:/mnt/c/Users/98912/Desktop/uni/OS LAB/EXP 6$ ls
Read-Write Reader-Writer.c a.out philosepher philosopher.c report result
○ cavendish@LAPTOP-J4F04081:/mnt/c/Users/98912/Desktop/uni/OS LAB/EXP 6$ ./philosepher
philosopher 0 is thinking
philosopher 1 is thinking
philosopher 2 is thinking
philosopher 3 is thinking
philosopher 4 is thinking
philosopher 1 is eating with chopsticks(1) and chopsticks(2)
philosopher 1 finished eating
philosopher 1 is thinking
philosopher 0 is eating with chopsticks(0) and chopsticks(1)
philosopher 2 is eating with chopsticks(2) and chopsticks(3)
philosopher 0 finished eating
philosopher 0 is thinking
philosopher 2 finished eating
philosopher 2 is thinking
philosopher 3 is eating with chopsticks(3) and chopsticks(4)
philosopher 3 finished eating
philosopher 3 is thinking
philosopher 4 is eating with chopsticks(4) and chopsticks(0)
philosopher 1 is eating with chopsticks(1) and chopsticks(2)
philosopher 4 finished eating
philosopher 4 is thinking
philosopher 1 finished eating
philosopher 1 is thinking
philosopher 0 is eating with chopsticks(0) and chopsticks(1)
philosopher 2 is eating with chopsticks(2) and chopsticks(3)
philosopher 0 finished eating
philosopher 0 is thinking
philosopher 2 finished eating
philosopher 2 is thinking
philosopher 3 is eating with chopsticks(3) and chopsticks(4)

```

