

دانشگاه صنعتي امیرکبیر  
(پلی تکنیک تهران)

دانشكده مهندسی برق

آزمایشگاه سیستم عامل

آزمایش ۶

همگام سازی فرایند ها

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 پردازه write با پردازه های read حالت مسابقه پیش بیاید زیرا ممکن است در یک لحظه پردازه write بخواهد مقدار count را تغییر دهد و در همان لحظه نیز پردازه read بخواهد 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 جاوگیری کنیم. در این مسئله جون reader ها همزمان می­توانند به مقدار count دسترسی داشته باشند در واقع می­توان 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 می­توان 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);

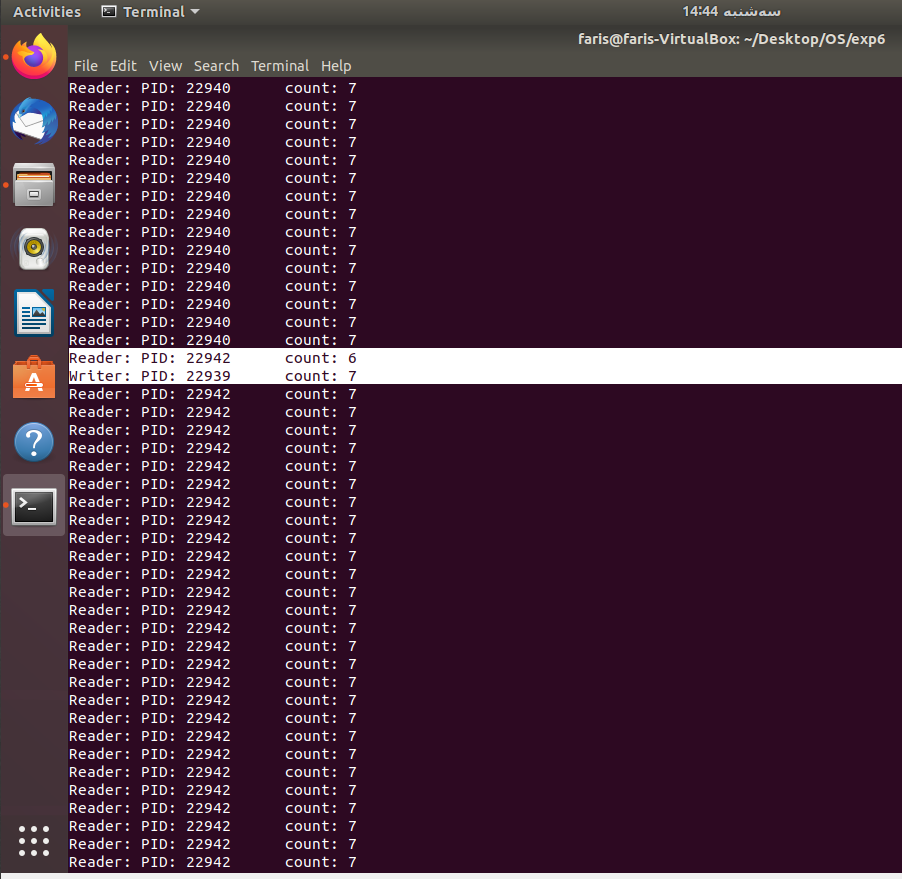
            }

    }

}

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.

**بخش دوم – مسئله شام فیلسوفان:**

در این مسئله 5 فیلسوف در دور یک میز هستند و قرار است که با 5 چوبی که روی میز است غذای وسط میز را بخورند. هر فیلسوف در کنار خود 2 فیلسوف چپ و راست را دارد و همچنین یک چوب چپ و یک چوب راست هم وجود دارد و برای آنکه بتواند بخورد باید هر دو چوب کنار خود را در اختیار داشته باشد.

**آیا ممکن است بن بست رخ دهد؟** بله این امکان وجود دارد، برای مثال زمانی که همه فیلسوف ها همزمان گرسنه شوند و هر فیلسوف چوب غذاخوری سمت راست خود را بردار در این صورت هیچ چوبی باقی نمی ماند و هر فیلسوف منتظر است تا نفر سمت چپی چوب سمت چپ را آزاد کند و به این ترتیب هر فیلسوف منتظر 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)

    {

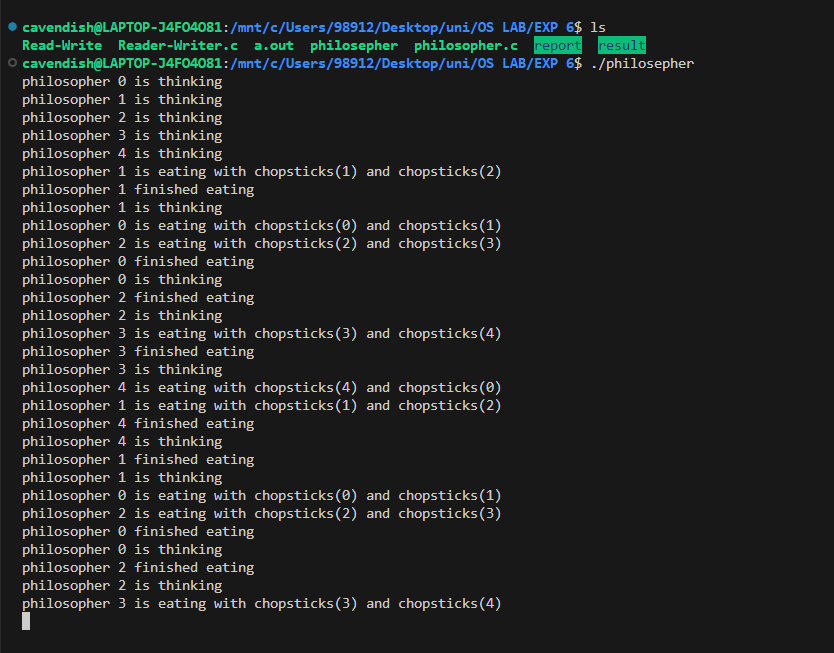
        pthread\_join(philosopher[i], NULL);

    }

    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 درصورتی که سمافور اجاره 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: