دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

گزارش کار آزمایشگاه سیستم عامل – شماره ۶

همگام سازی فرایند ها

حسنا اویارحسینی – ۹۸۲۳۰۱۰

استاد درس: جناب آقای مهندس کیخا

نیمسال دوم سال تحصیلی ۰۱-۱۴۰۰

بخش ۱) مساله خوانندگان-نویسندگان را پیاده سازی کنید.

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.

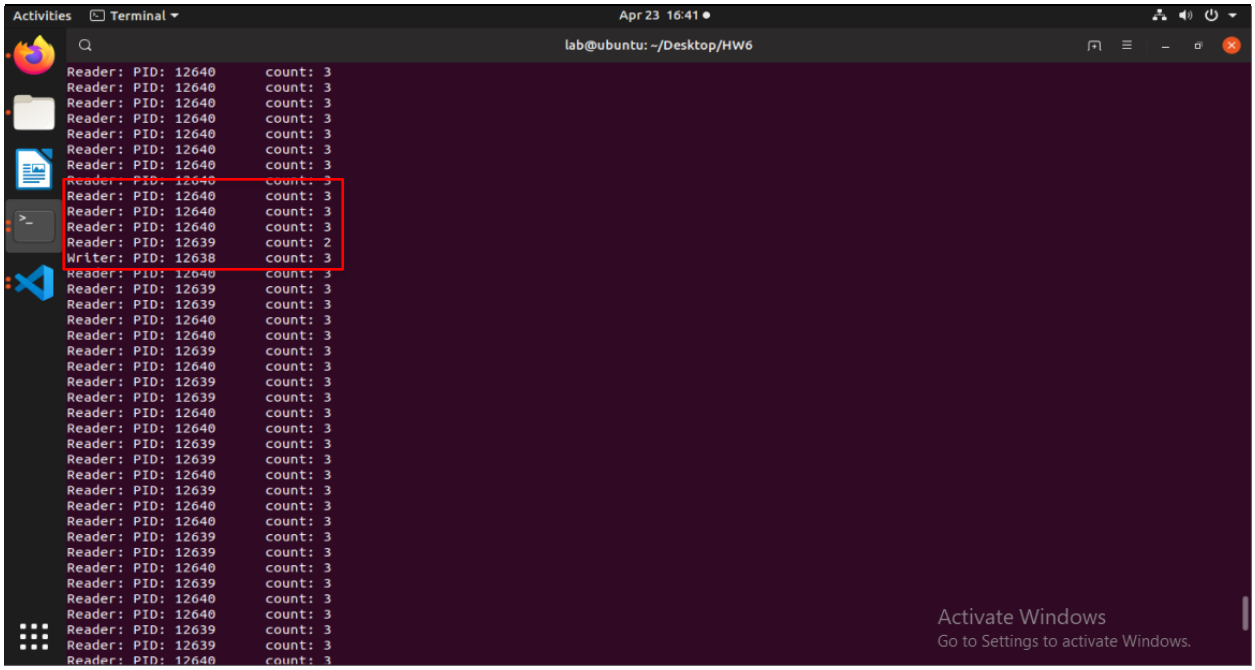
کد:

```
int main()
{

    key_t key = ۲۵۲۵;

    // shmget returns an identifier in shmid
    int shmid = shmget(key, sizeof(int), ۰٦٦٦|IPC_CREAT);

    // shmat to attach to shared memory
    int *count = (int*) shmat(shmid, (void*)۰, ۰);
    *count = ۰;

    int n۱ = fork();
    int n۲ = fork();
    if (n۱ != ۰ && n۲ != ۰){
        while(*count < ۱۰){
            (*count) ++;
            printf("Writer:\tPID: %d\tcount: %d\n", getpid(), *count);
            usleep(۵۰۰);
        }
        wait(NULL);
        wait(NULL);
        wait(NULL);
    }
    else if (n۱ == ۰ || n۲ == ۰){
        while(*count < ۱۰){
            int current_pid = getpid();
                printf("Reader:\tPID: %d\tcount: %d\n", current_pid, *count);
        }
    }

}
```

در کد بالا پردازه والد هر ۵۰۰ میکروثانیه مقدار count را یکی زیاد میکند تا جایی که count ۱۰ شود و همزمان ۳ پردازه فرزند که با دو دستور fork ایجاد شده اند مقدار این متغیر را خوانده و گزارش میدهند.

خروجی کد:

در خروجی میتوان race condition را که باعث خواندن مقدار ناصحیح توسط پردازه های reader شده است را مشاهده کرد:

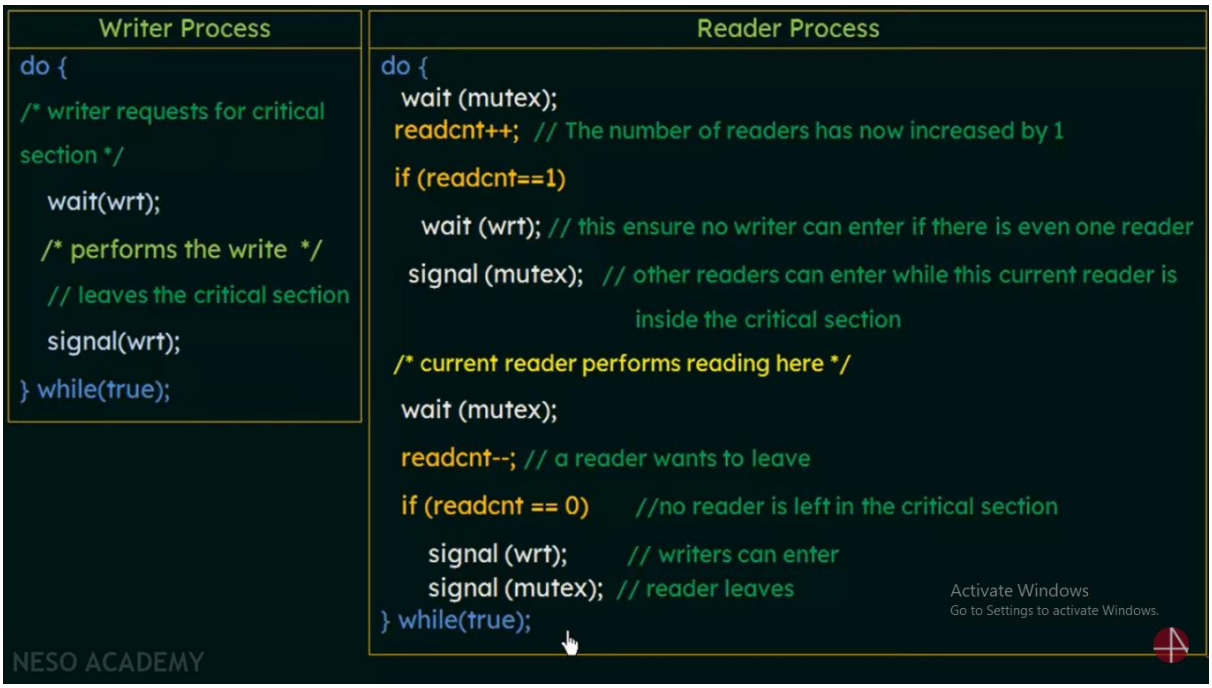


برای حل این مشکل میتوانیم از mutex و semaphore استفاده کنیم. در ادامه به توضیح یکی از راه حل های ممکن میپردازیم.

ابتدا یک سمافور مشترک بین تمام پردازه ها تعریف میکنیم (نام این سمافور را wrt) میگذاریم. از این سماور استفاده میکنیم تا کاری کنیم که همزمان روی count خواندن و نوشتن رخ ندهد به این منظور در پردازه Write قبل ازنوشتن ابتدا روی این سمافور wait میکنیم و زمانی که سمافور آزاد شد وارد ناحیه بحرانی میشویم و Count را تغییر میدهیم و پس از تغییر signal را صدا میزنیم. از طرف دیگر در پردازه های read نیز قبل از خواندن count روی سمافور wrt صبر مکینیم تا مطمئن شویم پردازه write در لحظه خواندن کاری انجام نمیدهد سپس داده را خوانده و پس از اتمام کار signal را صدا میزنیم تا سمافور آزاد شود.

البته با این کار، خواندن همزمان چند پردازه read را نیز محدود کرده ایم، در صورتی که خواندن همزمان مشکل race condition ایجاد نمیکرد پس راه بهتر این است که از سمافور دومی به نام mutex و متغیر کمکی به نام (readcnt) استفاده کنیم در واقع readcnt تعداد پردازه هایی که قصد خواندن count را دارند نشان میدهد و mutex از رخ دادن حالت مسابقه روی readcnt جلوگیری میکند. در پردازه های read ابتدا روی این سمافور mutex منتظر میمانیم و سپس هر زمان که سمافور را بدست آوردیم به متغیر readcnt، یکی اضافه کنیم و بعد روی wrt صبر کرده و mutex را آزاد میکنیم. با این کار پردازه های Read دیگر نیز میتوانند count را بخوانند. و در نهایت پس از خواندن به کمک mutex مقدار readcnt را به صورت ایمن یکی کم میکنیم و چک میکنیم اگر این متغیر صفر شد، یعنی دیگر پردازه ای نمیخواهد count را بخواند، پس signal را روی wrt صدا میزنیم تا این سمافور را آزاد کنیم. شبه کد راه حل گفته شده را میتوان در شکل ۱ مشاهده کرد.

*لازم به ذکر است در این راه حل اولویت پردازنده read را بیشتر از write در نظر گرفتیم.



شکل ۱ شبه کد راه حل حالت مسابقه در مسئله خوانندگان و نویسندگان

بخش ۲) مساله فیلسوف های غذاخور

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 راه های مختلفی وجود دارد در ادامه به بررسی دو راه میپردازیم:

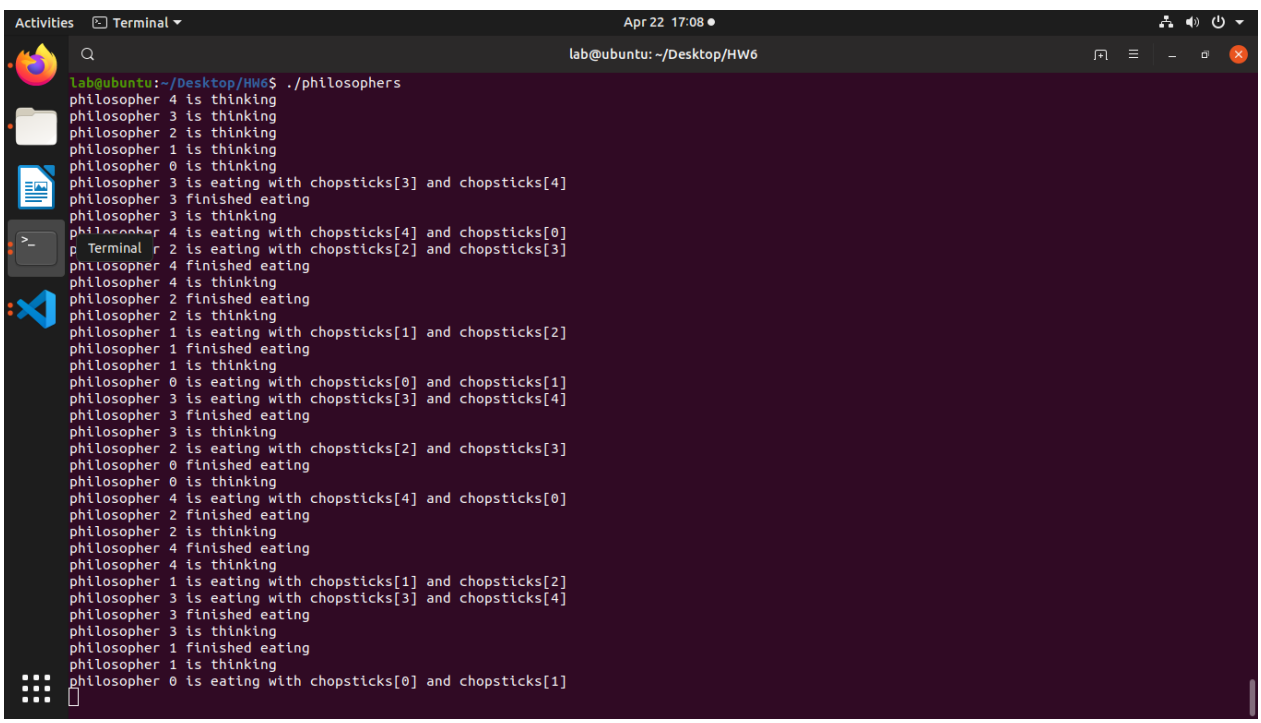
- به هر فیلسوف فقط زمانی اجازه دهیم چوب ها را بردارد که فیلسوف دیگری چوبی بر نداشته است برای این کار باید برداشتن چوب ها را در ناحیه بحرانی مربوط به هر ترد انجام دهیم. در این حالت مشکل بن بست حل میشود اما این راه حل لزوما بهینه ترین حالت نیست زیرا در هر لحظه حداکثر یک فیلسوف میتواند غذا بخورد
- راه حل بهتر این است که هر فیلسوف اگر 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 (مثلا فیلسوف ۴ و ۲) در ادامه پیاده سازی این را حل را میبینیم.

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.

کد:

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

void *handle_philosopher(void * i);
void think(int philosopher);
void eat(int philosopher);
void finish(int philosopher);
void pickup(int philosopher);
void putdown(int philosopher);

sem_t pick_up;
pthread_t philosopher[۵];
```

```c
pthread_mutex_t chopsticks[٥];
int count[٥];

int main(){

    sem_init(&pick_up, ٠ , ١);

    for (int i = ٠; i < ٥; ++i)
    {
        if (pthread_mutex_init(&chopsticks[i], NULL))
        {
            printf("Mutex Initialize Failed\n");
            return ٠;
        }
    }
    for (int i = ٠; i < ٥; ++i)
    {
        pthread_create(&philosopher[i], NULL, handle_philosopher, (void *)i);
    }
    for (int i = ٠; i < ٥; ++i)
    {
        pthread_join(philosopher[i], NULL);
    }
}

void *handle_philosopher (void *i){

    int id = (int)(long) i;

    count[id] = ١;
    while(١){
        think(id);

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

        eat(id);
        finish(id);
        putdown(id);
        count[id]++;
    }
}
```

```c
void pickup(int philosopher){
    pthread_mutex_lock(&chopsticks[philosopher]);
    pthread_mutex_lock(&chopsticks[(philosopher+۱)%۵]);
}

void putdown(int philosopher){
    pthread_mutex_unlock(&chopsticks[philosopher]);
    pthread_mutex_unlock(&chopsticks[(philosopher+۱)%۵]);
}

void think(int philosopher){
    printf("philosopher %d is thinking\n", philosopher);
    sleep(۲);
}

void eat(int philosopher){
    printf("philosopher %d is eating with chopsticks[%d] and chopsticks[%d]\n",
philosopher, philosopher, (philosopher+۱)%۵);
    sleep(۲);

}

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

خروجی کد: