

CS314:Operating Systems Laboratory

Lab 2 Report Group 10

Utkarsh Prakash - 180030042

Shriram Ghadge - 180010015

1. Introduction

In this lab we were expected to implement semaphore using pthreads conditional variable and mutex locks. Our implementation of semaphore is termed as zemaphore. We also had to synchronize the usage of 3 threads using zemaphore.

2. Zemaphore Implementation

Our implementation of zemaphore consists of three parts: one, is the value of the semaphore (value), second is the condition variable (cond) and third is the mutex lock (lock).

a. struct zemaphore :

```
typedef struct zemaphore{
    int value;
    pthread_cond_t cond;
    pthread_mutex_t lock;
} zem_t;
```

Here we defined the structure of zemaphore, with pthread mutex lock, pthread condition variable and the value of the zemaphore.

b. zem_init :

```
void zem_init(zem_t *s, int value){
    s->value = value;
    s->cond = PTHREAD_COND_INITIALIZER;
    pthread_mutex_init(&s->lock, NULL);
}
```

Here we initialize the pthread mutex lock and pthread condition variable. We also initialize the value of the semaphore to the value passed as argument.

c. zem_down (zem_wait):

```
void zem_down(zem_t *s) // Zem_wait
{
    pthread_mutex_lock(&s->lock);
    while (s->value <= 0)
        pthread_cond_wait(&s->cond, &s->lock);
    s->value--;
    pthread_mutex_unlock(&s->lock);
}
```

Here we first lock the mutex lock of the semaphore. This is done because the value of the semaphore is a critical section of the code and we want it's value to be modified by one thread at a time. Then we send the thread to wait if the value of semaphore is less or equal to 0. Once the thread wakes up we decrement the value of the semaphore and unlock the lock associated with the semaphore.

d. zem_up (zem_post) :

```
void zem_up(zem_t *s) // Zem_post
{
    pthread_mutex_lock(&s->lock);
    s->value++;
    pthread_cond_signal(&s->cond);
    pthread_mutex_unlock(&s->lock);
}
```

Here we first lock the mutex lock of the semaphore. Then we increment the value of the semaphore. We then signal (wake) any of the thread waiting on the same semaphore (conditional variable of the same semaphore). Then we unlock the mutex lock associated with the semaphore.


```

void *justprint(void *data)
{
    int thread_id = *((int *)data);
    if (thread_id == 0)
        zem_down(&z1);
    else if (thread_id == 1)
        zem_down(&z2);
    else
        zem_down(&z3);
    for(int i=0; i < NUM_ITER; i++)
    {
        printf("This is thread %d\n", thread_id);
        if (thread_id == 0 && i!= NUM_ITER-1) {
            zem_up(&z2);
            zem_down(&z1);
        }
        else if (thread_id == 0 && i== NUM_ITER-1) {
            zem_up(&z2);
        }
        else if (thread_id == 1 && i!= NUM_ITER-1) {
            zem_up(&z3);
            zem_down(&z2);
        }
        else if (thread_id == 1 && i== NUM_ITER-1) {
            zem_up(&z3);
        }
        else if (thread_id == 2 && i!= NUM_ITER-1) {
            zem_up(&z1);
            zem_down(&z3);
        }
        else {

        }
    }
    return 0;
}

```