

Name – Ashutosh Ardu

Regno – 20BRS1262

## OS Lab 8

Running a simple counter increment Function to check the synchronisation of two process of two separate thread but on a single cpu

### Code

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

#define NITER 1000000

int cnt = 0;

void * Count(void * a)
{
    int i, tmp;
    for(i = 0; i < NITER; i++)
    {
        tmp = cnt;
        tmp = tmp+1;
        cnt = tmp;
    }
}

int main(int argc, char * argv[])
{
    pthread_t tid1, tid2;

    if(pthread_create(&tid1, NULL, Count, NULL))
    {
        printf("\n ERROR creating thread 1");
        exit(1);
    }

    if(pthread_create(&tid2, NULL, Count, NULL))
    {
        printf("\n ERROR creating thread 2");
    }
}
```

```

        exit(1);
    }

    if(pthread_join(tid1, NULL))
    {
        printf("\n ERROR joining thread");
        exit(1);
    }

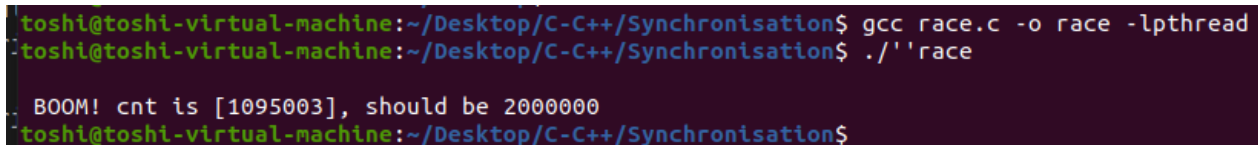
    if(pthread_join(tid2, NULL))
    {
        printf("\n ERROR joining thread");
        exit(1);
    }

    if (cnt < 2 * NITER)
        printf("\n BOOM! cnt is [%d], should be %d\n", cnt,
2*NITER);
    else
        printf("\n OK! cnt is [%d]\n", cnt);

    pthread_exit(NULL);
    return 0;
}

```

## Output



```

toshi@toshi-virtual-machine:~/Desktop/C-C++/Synchronisation$ gcc race.c -o race -lpthread
toshi@toshi-virtual-machine:~/Desktop/C-C++/Synchronisation$ ./'race

BOOM! cnt is [1095003], should be 2000000
toshi@toshi-virtual-machine:~/Desktop/C-C++/Synchronisation$

```

(As you see can the expected value was 2000000 but the final value of 'cnt' is lesser than expected, which means the two thread have not worked in synchronisation. As we wanted the processes to work in a certain precedence, the two threads run alternatively causing the generation of a random output. Because the cpu runs both the threads in a random order not in a synchronized way.)

## Using Mutex Lock to solve the above synchronisation problem

### Code

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

#define NITER 1000000
pthread_mutex_t mutex;
int cnt = 0;

void * Count(void * a)
{
    int i, tmp;
    pthread_mutex_lock(&mutex);
    for(i = 0; i < NITER; i++)
    {
        tmp = cnt;
        tmp = tmp+1;
        cnt = tmp;
    }
    pthread_mutex_unlock(&mutex);
}

int main(int argc, char * argv[])
{
    pthread_t tid1, tid2;
    pthread_mutex_init(&mutex, NULL);
    if(pthread_create(&tid1, NULL, Count, NULL))
    {
        printf("\n ERROR creating thread 1");
        exit(1);
    }

    if(pthread_create(&tid2, NULL, Count, NULL))
    {
        printf("\n ERROR creating thread 2");
        exit(1);
    }

    if(pthread_join(tid1, NULL))
    {
        printf("\n ERROR joining thread");
        exit(1);
    }

    if(pthread_join(tid2, NULL))
    {

```

```

        printf("\n ERROR joining thread");
        exit(1);
    }

    if (cnt < 2 * NITER)
        printf("\n BOOM! cnt is [%d], should be %d\n", cnt, 2*NITER);
    else
        printf("\n OK! cnt is [%d]\n", cnt);

    pthread_exit(NULL);
}

```

## Output

```

toshi@toshi-virtual-machine:~/Desktop/C-C++/Synchronisation$ gcc race.c -o race -lpthread
toshi@toshi-virtual-machine:~/Desktop/C-C++/Synchronisation$ ./'race

OK! cnt is [2000000]
toshi@toshi-virtual-machine:~/Desktop/C-C++/Synchronisation$

```

(In the above code both the processes on the both the threads have run in proper order by achieving synchronisation through the use of mutex locks)

Write C/C++ code to provide synchronization code to the below problem.

In a Company, goods are manufactured and stored in a sharable warehouse. Goods are distributed to the distributor by the company. Goods can't be manufactured when warehouse is full, and at the same time, Goods can't be distributed when warehouse is empty. Implement the above scenario using appropriate operating system concept.

## Without Synchronisation

## Code

```

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

#define NITER 1000000
pthread_mutex_t mutex;

```

```

int cnt = 0;
int item,max,cgoods,pgoods;

void * produce(void * a){
    for(int i=0;i<pgoods;i++){
        if(item>max){
            item=max;
            printf("Overflow\n");
            break;
        }
        item+=1;
        sleep(0.005);
    }
}

void * consume(void * a){
    if(item<=0) printf("Cant consume\n");
    else{
        for(int i=0;i<cgoods;++i){
            if(item<=0){
                printf("Underflow\n");
                break;
            }
            item-=1;
            sleep(0.005);
        }
    }
}

int main(int argc, char * argv[]){
    pthread_t tid1, tid2;
    item=0;
    printf("Max limit for goods\n");
    scanf("%d",&max);
    printf("Goods to be manufactured\n");
    scanf("%d",&pgoods);
    printf("Goods to be consumed\n");
    scanf("%d",&cgoods);
    if(pthread_create(&tid1, NULL, produce, NULL)){
        printf("\n ERROR creating thread 1");
        exit(1);
    }
    if(pthread_create(&tid2, NULL, consume, NULL)){
        printf("\n ERROR creating thread 2");
        exit(1);
    }
    if(pthread_join(tid1, NULL)){
        printf("\n ERROR joining thread");
        exit(1);
    }
    if(pthread_join(tid2, NULL)){
        printf("\n ERROR joining thread");
        exit(1);
    }
}

```

```

    }
    printf("Items left %d\n",item);
    pthread_exit(NULL);
}

```

## Output

```

toshi@toshi-virtual-machine:~/Desktop/C-C++$ gcc tempCodeRunnerFile.c -o tempCodeRunnerFile -lpthread
toshi@toshi-virtual-machine:~/Desktop/C-C++$ ./tempCodeRunnerFile
Max limit for goods
100
Goods to be manufactured
50
Goods to be consumed
48
Underflow
Items left 49
toshi@toshi-virtual-machine:~/Desktop/C-C++$

```

(In the above code the output ends up in a chaos as there is no synchronisation between the process on the two threads. Even though there shouldn't be any underflow as goods produced > goods consumed)

## Using Synchronisation

### Code

```

#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>

#define NITER 1000000
pthread_mutex_t mutex;
int cnt = 0;
int item,max,pgoods,cgoods;

void * produce(void * a){
    pthread_mutex_lock(&mutex);
    for(int i=0;i<pgoods;i++){
        if(item>max){
            item=max;
            printf("Overflow\n");
            break;
        }
        item+=1;
    }
}

```

```

    }
    pthread_mutex_unlock(&mutex);
}

void * consume(void * a){
    if(item<=0) printf("Cant consume\n");
    else{
        pthread_mutex_lock(&mutex);
        for(int i=0;i<cgoods;++i){
            if(item<=0){
                printf("Underflow\n");
                break;
            }item-=1;
        }pthread_mutex_unlock(&mutex);
    }
}

int main(int argc, char * argv[]){
    pthread_t tid1, tid2;
    item=0;
    printf("Max limit for goods\n");
    scanf("%d",&max);
    pthread_mutex_init(&mutex,NULL);
    printf("Goods to be manufactured\n");
    scanf("%d",&pgoods);
    printf("Goods to be consumed\n");
    scanf("%d",&cgoods);
    if(pthread_create(&tid1, NULL, produce, NULL)){
        printf("\n ERROR creating thread 1");
        exit(1);
    }
    if(pthread_create(&tid2, NULL, consume, NULL)){
        printf("\n ERROR creating thread 2");
        exit(1);
    }
    if(pthread_join(tid1, NULL)){
        printf("\n ERROR joining thread");
        exit(1);
    }

    if(pthread_join(tid2, NULL)){
        printf("\n ERROR joining thread");
        exit(1);
    }
    printf("Items left %d\n",item);
    pthread_exit(NULL);
}

```

## Output

```
toshi@toshi-virtual-machine:~/Desktop/C-C++$ gcc tempCodeRunnerFile.c -o tempCodeRunnerFile -lpthread
toshi@toshi-virtual-machine:~/Desktop/C-C++$ ./'tempCodeRunnerFile
Max limit for goods
100
Goods to be manufactured
50
Goods to be consumed
48
Items left 2
toshi@toshi-virtual-machine:~/Desktop/C-C++$ |
```

(The Synchronisation problem in the previous code is solved by the mutex lock used in this codes)