National University of Computer and Emerging

Sciences

Lab 11

(Operating Systems)

Instructors	Sohaib Ahmad Syed Mujtaba Hassan Rizvi
Topic	Semaphores
Semester	Spring 2023 to onwards

Department of Computer Science

FAST-NU, Lahore

Semaphores

Header Files:

- #include <semaphores.h>

Datatype for Semaphore variables:

- sem_t

Methods for handling semaphores:

- int sem_init (sem_t *sem, int pshared, unsigned int value);
- int sem_wait(sem_t *sem);
- int sem_post(sem_t *sem);

Previous Knowledge:

```
wait(S) {
        while( S <= 0);
        S--;
}

signal(S) {
        s = 1;
}
```

Note: We use wait before signal when our semaphore variable (S) is initialed to 1. If it is initialized to 0 then we have to use signal before wait.

Usage:

```
sem_t S;

// 2nd Argument: 0 for thread sync. And Non Zero value for process sync
// 3rd Argument: Initial value for semaphore variable
sem_init(&S, 0, 1);
```

Thread 1	Thread 2

worker1(){
sem_wait(&S);
CRITICAL SECTION
sem_post(&S)
}
	worker2(){
	sem_wait(&S);
	CRITICAL SECTION
	sem_post(&S)
	}

Example Program with 2 Threads without synchronization:

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

void *worker1();
void *worker2();

int shared_variable = 10;

int main(int argc, char *argv[])
{
   pthread_t thread1, thread2;
   pthread_create(&thread1, NULL, worker1, NULL);
   pthread_create(&thread2, NULL, worker2, NULL);
   pthread_join(thread1, NULL);
   pthread_join(thread2, NULL);
   return 0;
}

void *worker1()
{
   int x = shared_variable;
   printf("WORKER1 - Before updation - Shared Variable: %d and x: %d\n", shared_variable, x);
   x++;
   shared_variable = x;
   printf("WORKER1 - After updation - Shared Variable: %d and x: %d\n", shared_variable, x);
}

void *worker2()
{
   int y = shared_variable;
   printf("WORKER2 - Before updation - Shared Variable: %d and y: %d\n", shared_variable, y);
   y += 10;
   shared_variable = y;
   printf("WORKER2 - After updation - Shared Variable: %d and y: %d\n", shared_variable, y);
}
```

Output:

```
irtiza@Irtiza:/mnt/c/Users/m7irt/OneDrive/Desktop$ gcc main.c -o main -lpthread
irtiza@Irtiza:/mnt/c/Users/m7irt/OneDrive/Desktop$ ./main
WORKER1 - Before updation - Shared Variable: 10 and x: 10
WORKER1 - After updation - Shared Variable: 11 and x: 11
WORKER2 - Before updation - Shared Variable: 11 and y: 11
WORKER2 - After updation - Shared Variable: 21 and y: 21
irtiza@Irtiza:/mnt/c/Users/m7irt/OneDrive/Desktop$ ./main
WORKER1 - Before updation - Shared Variable: 10 and x: 10
WORKER1 - After updation - Shared Variable: 11 and x: 11
WORKER2 - Before updation - Shared Variable: 10 and y: 10
WORKER2 - After updation - Shared Variable: 20 and y: 20
irtiza@Irtiza:/mnt/c/Users/m7irt/OneDrive/Desktop$ ./main
WORKER1 - Before updation - Shared Variable: 10 and x: 10
WORKER1 - After updation - Shared Variable: 11 and x: 11
WORKER2 - Before updation - Shared Variable: 10 and y: 10
WORKER2 - After updation - Shared Variable: 20 and y: 20
irtiza@Irtiza:/mnt/c/Users/m7irt/OneDrive/Desktop$ ./main
WORKER1 - Before updation - Shared Variable: 10 and x: 10
WORKER1 - After updation - Shared Variable: 11 and x: 11
WORKER2 - Before updation - Shared Variable: 10 and y: 10
WORKER2 - After updation - Shared Variable: 20 and y: 20
irtiza@Irtiza:/mnt/c/Users/m7irt/OneDrive/Desktop$ ./main
WORKER1 - Before updation - Shared Variable: 10 and x: 10
WORKER1 - After updation - Shared Variable: 11 and x: 11
WORKER2 - Before updation - Shared Variable: 10 and y: 10
WORKER2 - After updation - Shared Variable: 20 and y: 20
irtiza@Irtiza:/mnt/c/Users/m7irt/OneDrive/Desktop$ ./main
WORKER1 - Before updation - Shared Variable: 10 and x: 10
WORKER1 - After updation - Shared Variable: 11 and x: 11
WORKER2 - Before updation - Shared Variable: 11 and y: 11
WORKER2 - After updation - Shared Variable: 21 and y: 21
irtiza@Irtiza:/mnt/c/Users/m7irt/OneDrive/Desktop$ ./main
WORKER1 - Before updation - Shared Variable: 10 and x: 10
WORKER1 - After updation - Shared Variable: 11 and x: 11
WORKER2 - Before updation - Shared Variable: 11 and y: 11
WORKER2 - After updation - Shared Variable: 21 and y: 21
irtiza@Irtiza:/mnt/c/Users/m7irt/OneDrive/Desktop$ ./main
```

You can see how we get non-consistent results each time we run the program, without synchronization. Our program's threads are suffering from race condition.

Example Program with 2 Threads with synchronization:

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

void *worker1();
void *worker2();

int shared_variable = 10;
sem_t semaphore_variable;

int main(int argc, char *argv[])
```

```
{
    sem_init(&semaphore_variable, 0, 1);

    pthread_t thread1, thread2;
    pthread_create(&thread1, NULL, worker1, NULL);
    pthread_create(&thread2, NULL, worker2, NULL);
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    return 0;
}

void *worker1()
{
    sem_wait(&semaphore_variable);
    int x = shared_variable;
    printf("WORKER1 - Before updation - Shared Variable: %d and x: %d\n", shared_variable, x);
    x++;
    shared_variable = x;
    printf("WORKER1 - After updation - Shared Variable: %d and x: %d\n", shared_variable, x);
    sem_post(&semaphore_variable);
}

void *worker2()
{
    sem_wait(&semaphore_variable);
    int y = shared_variable;
    printf("WORKER2 - Before updation - Shared Variable: %d and y: %d\n", shared_variable, y);
    y += 10;
    shared_variable = y;
    printf("WORKER2 - After updation - Shared Variable: %d and y: %d\n", shared_variable, y);
    sem_post(&semaphore_variable);
}
```

Output:

```
irtiza@Irtiza:/mnt/c/Users/m7irt/OneDrive/Desktop$ gcc main.c -o main -lpthread
irtiza@Irtiza:/mnt/c/Users/m7irt/OneDrive/Desktop$ ./main
WORKER1 - Before updation - Shared Variable: 10 and x: 10
WORKER1 - After updation - Shared Variable: 11 and x: 11
WORKER2 - Before updation - Shared Variable: 11 and y: 11
WORKER2 - After updation - Shared Variable: 21 and y: 21
irtiza@Irtiza:/mnt/c/Users/m7irt/OneDrive/Desktop$ ./main
WORKER1 - Before updation - Shared Variable: 10 and x: 10
WORKER1 - After updation - Shared Variable: 11 and x: 11
WORKER2 - Before updation - Shared Variable: 11 and y: 11
WORKER2 - After updation - Shared Variable: 21 and y: 21
irtiza@Irtiza:/mnt/c/Users/m7irt/OneDrive/Desktop$ ./main
WORKER1 - Before updation - Shared Variable: 10 and x: 10
WORKER1 - After updation - Shared Variable: 11 and x: 11
WORKER2 - Before updation - Shared Variable: 11 and y: 11
WORKER2 - After updation - Shared Variable: 21 and y: 21
irtiza@Irtiza:/mnt/c/Users/m7irt/OneDrive/Desktop$ ./main
WORKER1 - Before updation - Shared Variable: 10 and x: 10
WORKER1 - After updation - Shared Variable: 11 and x: 11
WORKER2 - Before updation - Shared Variable: 11 and y: 11
WORKER2 - After updation - Shared Variable: 21 and y: 21
irtiza@Irtiza:/mnt/c/Users/m7irt/OneDrive/Desktop$ ./main
WORKER1 - Before updation - Shared Variable: 10 and x: 10
WORKER1 - After updation - Shared Variable: 11 and x: 11
WORKER2 - Before updation - Shared Variable: 11 and y: 11
WORKER2 - After updation - Shared Variable: 21 and y: 21
irtiza@Irtiza:/mnt/c/Users/m7irt/OneDrive/Desktop$ ./main
WORKER1 - Before updation - Shared Variable: 10 and x: 10
WORKER1 - After updation - Shared Variable: 11 and x: 11
WORKER2 - Before updation - Shared Variable: 11 and y: 11
WORKER2 - After updation - Shared Variable: 21 and y: 21
irtiza@Irtiza:/mnt/c/Users/m7irt/OneDrive/Desktop$ |
```

Now each time we run the program, we get consistent results.

Question # 1 (10 Marks) Take X, Y, Z, W from the user. Use semaphore to implement the following Program:

Thread T1	**Thread T2**
Input (X,Y):	Input (W,Z):
X1= Z+2;	Z1= X1*2;
Y1=Z1*5;	W1=Y1+5;
S1=X1+Y1;	S2=Z1+W1;
Printf(“x=%d”,S1);	Printf(“x=%d”,S2);