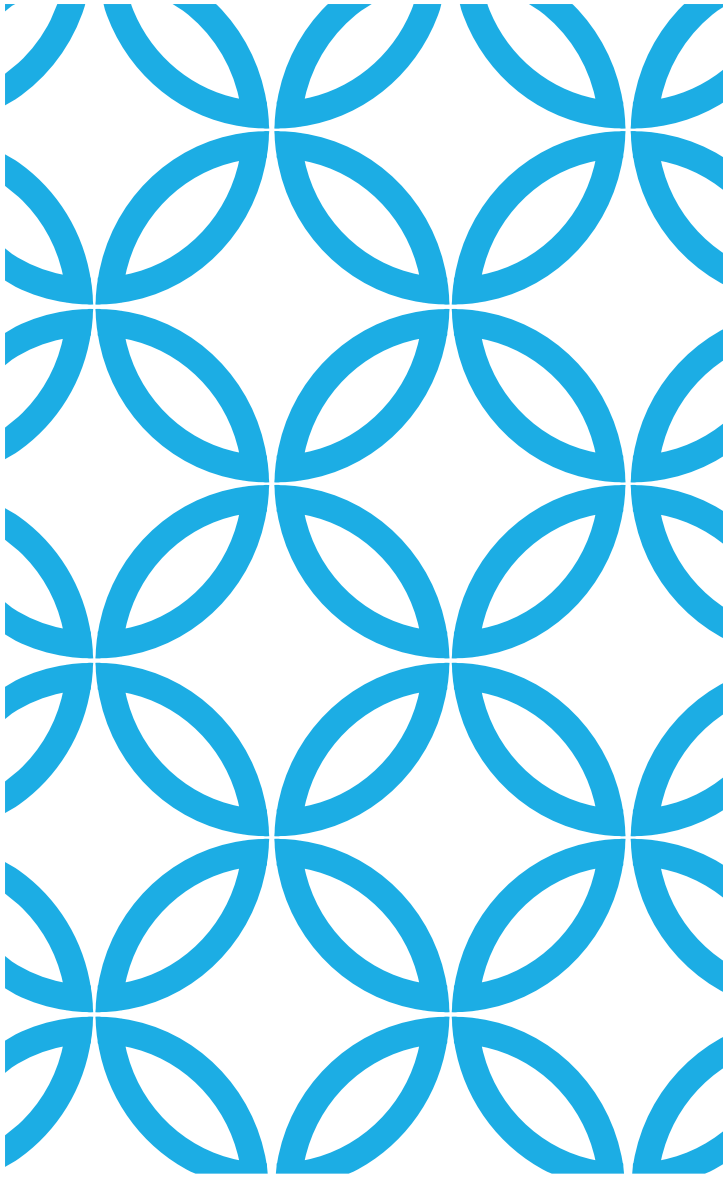

PROCESS SYNCHRONIZATION

Lab 05

YOU NEED!

Three things are necessary:

1. a text editor
2. g++ compiler
3. C++ libraires:
   - iostream
   - unistd.h
   - pthread.h
   - semaphore.h

SEMAPHORE

Semaphore is an integer variable which is accessed or modified by using two atomic operations: wait() and signal().

In this lab, we learn about process synchronization using semaphores to understand the implementation of sem_wait() and sem_signal() and avoid a race condition among threads.

EXAMPLE

The following program creates two threads: one to increment the value of a shared variable and second to decrement the value of the shared variable.

Both threads make use of a semaphore variable so that only one of the threads is executing in its critical section.

```
#include <iostream>
#include <unistd.h>
#include "pthread.h"
int shared=1; //shared variable
void *fun1(void *) {
    int x;
    x=shared;//thread1 reads value of shared variable
    printf("Thread1 reads the value as %d\n",x);
    x++;  //thread1 increments its value
    printf("Local update by Thread1: %d\n",x);
    sleep(1); //thread1 is preempted by thread 2
    shared=x; //thread one updates the value of shared variable
    printf("Value of shared variable updated by Thread1 is: %d\n",shared);
  }
// ***************************************
   void *fun2(void *){
      int y;
      y=shared;//thread2 reads value of shared variable
      printf("Thread2 reads the value as %d\n",y);
      y--;  //thread2 increments its value
      printf("Local update by Thread2: %d\n",y);
      sleep(1); //thread2 is preempted by thread 1
      shared=y; //thread2 updates the value of shared variable
      printf("Value of shared variable updated by Thread2 is: %d\n",shared);
     }
  // ***************************************
```

```
int main() {
        pthread_t thread1, thread2;
        pthread_create(&thread1, NULL, fun1, NULL);
        pthread_create(&thread2, NULL, fun2, NULL);
        pthread_join(thread1, NULL);
        pthread_join(thread2,NULL);
        printf("Final value of shared is %d\n",shared);
        return 0;
 }
```

POSSIBLE OUTPUT

```
modhi@ubuntu:~$ g++ sync.cpp -lpthread
modhi@ubuntu:~$ ./a.out
Thread2 reads the value as 1
Local update by Thread2: 0
Thread1 reads the value as 1
Local update by Thread1: 2
Value of shared variable updated by Thread2 is: 0
Value of shared variable updated by Thread1 is: 2
Final value of shared is 2
modhi@ubuntu:~$
```

- **The final value of the variable shared should be 1! It is not! Why?**
- **How to ensure that only one thread is running its critical section at any given time?**

```c
#include<semaphore.h>
sem_t s; //semaphore variable
int shared=1;
void *fun1(void *)
{
    int x;
    sem_wait(&s); //executes wait operation on s
    x=shared;//thread1 reads value of shared variable
    printf("Thread1 reads the value as %d\n",x);
    x++;  //thread1 increments its value
    printf("Local update by Thread1: %d\n",x);
    sleep(1); //thread1 is preempted by thread 2
    shared=x; //thread one updates the value of shared variable
    printf("Value of shared variable updated by Thread1 is: %d\n",shared);
    sem_post(&s);
}
// ************************************
 void *fun2(void *)
{
    int y;
    sem_wait(&s);
    y=shared;//thread2 reads value of shared variable
    printf("Thread2 reads the value as %d\n",y);
    y--;  //thread2 increments its value
    printf("Local update by Thread2: %d\n",y);
    sleep(1); //thread2 is preempted by thread 1
    shared=y; //thread2 updates the value of shared variable
    printf("Value of shared variable updated by Thread2 is: %d\n",shared);
    sem_post(&s);
}
// ************************************

 int main()
 {
     sem_init(&s,0,1);
     pthread_t thread1, thread2;
     pthread_create(&thread1, NULL, fun1, NULL);
     pthread_create(&thread2, NULL, fun2, NULL);
```

- Using semaphore s, the final value of the variable "shared" will be 1.
- When any thread executes the wait operation the value of "s" becomes zero.
- The other thread is cannot execute the wait operation on "s" successfully. It will not read the inconsistent value of the shared variable.
- This ensures that only one thread is running its critical section at any given time.
- **How to enforce a specific order of execution?**

```
modhi@ubuntu:~$ ./a.out
Thread2 reads the value as 1
Local update by Thread2: 0
Value of shared variable updated by Thread2 is: 0
Thread1 reads the value as 0
Local update by Thread1: 1
Value of shared variable updated by Thread1 is: 1
Final value of shared is 1
modhi@ubuntu:~$
```

```c
#include "pthread.h"
#include<semaphore.h>
sem_t s; //semaphore variable
int shared=1;
void *fun1(void *)
{
    int x;
    x=shared;//thread1 reads value of shared variable
    printf("Thread1 reads the value as %d\n",x);
    x++;  //thread1 increments its value
    printf("Local update by Thread1: %d\n",x);
    sleep(1); //thread1 is preempted by thread 2
    shared=x; //thread one updates the value of shared variable
    printf("Value of shared variable updated by Thread1 is: %d\n",shared);
    sem_post(&s);
}
// ************************************
 void *fun2(void *)
{
    int y;
    sem_wait(&s);
    y=shared;//thread2 reads value of shared variable
    printf("Thread2 reads the value as %d\n",y);
    y--;  //thread2 increments its value
    printf("Local update by Thread2: %d\n",y);
    sleep(1); //thread2 is preempted by thread 1
    shared=y; //thread2 updates the value of shared variable
    printf("Value of shared variable updated by Thread2 is: %d\n",shared);
}
// ************************************

 int main()
 {
     sem_init(&s,0,0);
     pthread_t thread1, thread2;
     pthread_create(&thread1, NULL, fun1, NULL);
     pthread_create(&thread2, NULL, fun2, NULL);
     pthread_join(thread1, NULL);
```

- **Can you explain what happened?**

```
modhi@ubuntu:~$ ./a.out
Thread1 reads the value as 1
Local update by Thread1: 2
Value of shared variable updated by Thread1 is: 2
Thread2 reads the value as 2
Local update by Thread2: 1
Value of shared variable updated by Thread2 is: 1
Final value of shared is 1
modhi@ubuntu:~$
```

EXERCISE

1) Given four threads, show how to use semaphores to force the execution order T1, T2, T3, T4.

T1	T2	T3	T4
cout<<" one";	cout<<" two";	cout<<" three";	cout<<" four";

2) Write a C++ program to illustrate that.