COSC 439 – Operating System, Assignment#3
Instruction – This assessment is an individual work.
Assignment Name: Working with Semaphores.

Task 1 - Synchronizing Threads Using Semaphores in C (Coding based task)

In this task, we will utilize POSIX semaphores on **Unix/Linux** to synchronize a set of concurrent POSIX threads in a C program. A semaphore is an object of type sem_t, and the relevant library functions are defined in <semaphore.h>. Additionally, we will use the POSIX thread library functions from <pthread.h> for pthread_t thread objects.

The relevant semaphore functions include:

```
int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_wait(sem_t *sem);
int sem_trywait(sem_t *sem);
int sem_post(sem_t *sem);
int sem_getvalue(sem_t *sem, int *sval);
int sem_destroy(sem_t *sem);
```

And the pthread functions include:

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void
*(*start_routine)(void *), void *arg);
int pthread_join(pthread_t thread, void **value_ptr);
void pthread_exit(void *value_ptr);
```

Consider the following incomplete program provided:

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

#include <unistd.h> /* defines _POSIX_THREADS if pthreads are available */
#ifdef _POSIX_THREADS
# include <pthread.h>
#endif

#include <semaphore.h>
void *text(void *arg);

int code[] = { 4, 6, 3, 1, 5, 0, 2 };
int main()
```

```
{
  int i;
  pthread_t tid[7];
  for (i = 0; i < 7; i++)
      pthread_create(&tid[i], NULL, text, (void*)&code[i]);
  return 0;
}

void *text(void *arg)
{
  int n = *(int*)arg;
  switch (n)
  {
      case 0:
      printf("A semaphore S is an integer-valued variable which can take
only non-negative\n");
      printf("values. Exactly two operations are defined on a
semaphore:\n\n");
      break;

      case 1:
      printf("Signal(S): If there are processes that have been suspended on
this semaphore,\n");
      printf(" wake one of them, else S := S+1.\n\n");
      break;

      case 2:
      printf("Wait(S): If S>0 then S:=S-1, else suspend the execution of
this process.\n");
      printf(" The process is said to be suspended on the semaphore
S.\n\n");
      break;

      case 3:
      printf("The semaphore has the following properties:\n\n");
      break;

      case 4:
      printf("1. Signal(S) and Wait(S) are atomic instructions. In
particular, no\n");
      printf(" instructions can be interleaved between the test that S>0
and the\n");
      printf(" decrement of S or the suspension of the calling
```

```
process.\n\n");
      break;

      case 5:
      printf("2. A semaphore must be given an non-negative initial
value.\n\n");
      break;

      case 6:
      printf("3. The Signal(S) operation must wake one of the suspended
processes. The\n");
      printf(" definition does not specify which process will be
awakened.\n\n");
      break;
  }

  pthread_exit(0);
}
```

There are a couple of problems with this program. First of all, the threads do not have a chance to complete, because the main program terminates without waiting for them. Second, the threads are not synchronized and therefore the text output is garbled.

Your task is to add semaphores to this program to synchronize the threads. You may declare the semaphores as global objects.

Task 2 - The Sleeping Barber Problem

Problem Statement: You are required to implement a solution to the Sleeping Barber Problem using semaphores for the **Unix/Linux** environment.

In a barber shop:

- There is one barber who cuts hair and sleeps when there are no customers.
- The shop has a limited number of waiting chairs (denoted by N). When all chairs are occupied, new customers leave the shop without receiving service.
- When a customer arrives and finds the barber sleeping, they wake the barber to get a haircut.
- If the barber is busy (cutting another customer's hair), the arriving customer sits in one of the waiting chairs (if available).

- When the barber finishes cutting a customer's hair, they check if there are any waiting customers. If there are customers waiting, the barber calls in the next customer. If no customers are waiting, the barber goes back to sleep.

You will simulate this scenario using semaphores to manage synchronization between the barber and customers.

Inputs:

The program will take the following inputs:

1. N: The number of waiting chairs in the barber shop.
2. M: The number of customers who will attempt to enter the barber shop for a haircut.

The customer arrival times and haircut durations will be managed **internally** by the program (won't be given as user input):

- Haircut Time: The time taken to cut a customer's hair will be a random duration between 1 to 5 seconds.
- Customer Arrival Time: Customers will arrive at random intervals between 1 to 5 seconds.

Output:

The output should show the sequence of events occurring in the system, including:

- When a customer arrives at the shop.
- If the customer finds an available waiting chair or if they leave because all chairs are occupied.
- When a customer wakes the barber if they find the barber sleeping.
- When the barber starts and finishes cutting a customer's hair.
- When a customer leaves the shop after their haircut.
- When the barber goes back to sleep because no customers are waiting.

Example Inputs:

```
N = 3  // Number of waiting chairs
M = 5  // Number of customers
```

Example Outputs:

```
Customer 1 arrived.
Customer 1 wakes up the barber.
Barber starts cutting hair of Customer 1 for 4 seconds.
Customer 2 arrived.
Customer 2 sits in waiting area (chair 1).
Customer 3 arrived.
Customer 3 sits in waiting area (chair 2).
Customer 4 arrived.
Customer 4 sits in waiting area (chair 3).
Customer 5 arrived.
No chairs available, Customer 5 leaves the shop.
Barber finishes cutting hair of Customer 1.
Barber starts cutting hair of Customer 2 for 3 seconds.
Customer 2 leaves after haircut.
Barber starts cutting hair of Customer 3 for 5 seconds.
Customer 3 leaves after haircut.
Barber starts cutting hair of Customer 4 for 2 seconds.
Customer 4 leaves after haircut.
Barber goes to sleep.
```

Submission Guidelines:

1. **For Task #1:**
    a. Submit a document explaining your code and the fixes applied.
    b. Submit the final C/C++ code file.
2. **For Task #2:**
    a. Submit the C/C++ code file.
3. **For both tasks:**
    a. Provide an additional document with instructions on how to run your code.
    b. Submit a zip file containing all your codes and documentations

Note: Late submissions will incur a flat penalty of 40% deduction if submitted after the due date.