Name: Fiza Mansoor Patel

Roll no: 191 **Div.: C**

PRNo: 0120190425 **Batch: C3**

Assignment 6

Sleeping Barber: The analogy is based upon a hypothetical barber shop with one barber. There is a barber shop which has one barber, one barber chair, and n chairs for waiting for customers if there are any to sit on the chair.

If there is no customer, then the barber sleeps in his own chair.

When a customer arrives, he has to wake up the barber.

If there are many customers and the barber is cutting a customer's hair, then the remaining customers either wait if there are empty chairs in the waiting room or they leave if no chairs are empty.

Design and implement the given scenario in such a way that the barber and customers will not get into race condition.

```
#include <stdio.h>

#include <unistd.h>  //unistd.h is the name of the header file that provides access to the POSIX operating system API,the base of the Single Unix Specification

#include <stdlib.h>

#include <time.h>

#include <pthread.h>

#include <semaphore.h>


// The maximum number of customer threads.

#define MAX_CUSTOMERS 25


// Function prototypes…

void *customer(void *num);
```

```
void *barber(void *);

void wait(int secs);

// Define the semaphores.

// waitingRoom Limits the # of customers allowed
// to enter the waiting room at one time.
sem_t waitingRoom;

// barberChair ensures mutually exclusive access to
// the barber chair.
sem_t barberChair;

// barberPillow is used to allow the barber to sleep
// until a customer arrives.
sem_t barberPillow;

// seatBelt is used to make the customer to wait until
// the barber is done cutting his/her hair.
sem_t seatBelt;

// Flag to stop the barber thread when all customers
// have been serviced.
int allDone = 0;

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

pthread_t btid;
pthread_t tid[MAX_CUSTOMERS];
```

```
int i, numCustomers, numChairs;
int Number[MAX_CUSTOMERS];

printf("Enter the number of Customers : ");
scanf("%d",&numCustomers) ;
printf("Enter the number of Charis : ");
scanf("%d",&numChairs);

// Make sure the number of threads is less than the number of
// customers we can support.
if (numCustomers > MAX_CUSTOMERS) {
printf("The maximum number of Customers is %d.\n", MAX_CUSTOMERS);
exit(-1);
}

// Initialize the numbers array.
for (i=0; i<MAX_CUSTOMERS; i++) {
Number[i] = i;
}

// Initialize the semaphores with initial values…
sem_init(&waitingRoom, 0, numChairs);
sem_init(&barberChair, 0, 1);
sem_init(&barberPillow, 0, 0);
sem_init(&seatBelt, 0, 0);

// Create the barber.
//pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start)(void *), void *arg);
/*
```

thread -location where the ID of the newly created thread should be stored, or NULL if the thread ID is not required.

attr -Is the thread attribute object specifying the attributes for the thread that is being created. If attr is NULL, the thread is created with default attributes.

start-the main function for the thread; the thread begins executing user code at this address.

arg -Is the argument passed to start.

*/

```
pthread_create(&btid, NULL, barber, NULL);


// Create the customers.
for (i=0; i<numCustomers; i++) {
pthread_create(&tid[i], NULL, customer, (void *)&Number[i]);
sleep(1);
}


// Join each of the threads to wait for them to finish.
for (i=0; i<numCustomers; i++) {
pthread_join(tid[i],NULL);
sleep(1);
}


// When all of the customers are finished, kill the
// barber thread.
allDone = 1;
sem_post(&barberPillow); // Wake the barber so he will exit.
pthread_join(btid,NULL);
}


void *customer(void *number) {
int num = *(int *)number;
```

```
// Leave for the shop and take some random amount of
// time to arrive.
printf("Customer %d leaving for barber shop.\n", num);
wait(2);
printf("Customer %d arrived at barber shop.\n", num);

// Wait for space to open up in the waiting room…
sem_wait(&waitingRoom);
printf("Customer %d entering waiting room.\n", num);

// Wait for the barber chair to become free.
sem_wait(&barberChair);

// The chair is free so give up your spot in the
// waiting room.
sem_post(&waitingRoom);

// Wake up the barber…
printf("Customer %d waking the barber.\n", num);
sem_post(&barberPillow);

// Wait for the barber to finish cutting your hair.
sem_wait(&seatBelt);

// Give up the chair.
sem_post(&barberChair);
printf("Customer %d leaving barber shop.\n", num);
}

void *barber(void *junk) {
```

```
// While there are still customers to be serviced…
// Our barber is omnicient and can tell if there are
// customers still on the way to his shop.
while (!allDone) {

// Sleep until someone arrives and wakes you..
printf("The barber is sleeping\n");
sem_wait(&barberPillow);

// Skip this stuff at the end…
if (!allDone) {

// Take a random amount of time to cut the
// customer’s hair.
printf("The barber is cutting hair\n");
wait(2);
printf("The barber has finished cutting hair.\n");

// Release the customer when done cutting…
sem_post(&seatBelt);
}
else {

printf("The barber is going home for the day.\n");
}
}

}

void wait(int secs) {
```

```
int len;


// Generate a random number…
len = (int) ((1 * secs) + 1);
sleep(len);
}
```

Output: