REPORT – ASSIGNMENT 5

Name: Ashna Dua
Roll No. 2021101072

UNDERSTANDING OF Q1:

In this question, each student is to be treated as individual thread, that will be running concurrently, and the number of washing machines will be the value of the counting semaphore.

IMPLEMENTATION OF Q1:

1. Storing the arrival time, washing time and patience time of each student in the array.

   Additional elements i.e., id and washingFlag were added to the struct.

2. Creating a global array of structs (Student) as follows:

```
typedef struct Student
{
    int id;
    int arrivalTime;
    int washingTime;
    int patienceTime;
    int washingFlag;
} Student;

Student arrStudents[1000];
```

3. After storing all the details of a students, this array was sorted based on their arrival time. This was done using bubble sort.

```
// using bubble sort to sort array of structs
// on the basis of their arrival time
void sort(int n, Student arrStudents[n])
{
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (arrStudents[j].arrivalTime > arrStudents[j + 1].arrivalTime)
            {
                Student temp;
                temp.id = arrStudents[j].id;
                temp.arrivalTime = arrStudents[j].arrivalTime;
                temp.washingTime = arrStudents[j].washingTime;
                temp.patienceTime = arrStudents[j].patienceTime;
                arrStudents[j].id = arrStudents[j + 1].id;
                arrStudents[j].arrivalTime = arrStudents[j + 1].arrivalTime;
                arrStudents[j].washingTime = arrStudents[j + 1].washingTime;
                arrStudents[j].patienceTime = arrStudents[j + 1].patienceTime;
                arrStudents[j + 1].id = temp.id;
                arrStudents[j + 1].arrivalTime = temp.arrivalTime;
                arrStudents[j + 1].washingTime = temp.washingTime;
                arrStudents[j + 1].patienceTime = temp.patienceTime;
            }
        }
    }
}
```

4. A counting semaphore (washingMachine) was created with the value m (number of washing machines).
5. n (number of students) individual thread was created to serve as n students. Each thread was run concurrently.

```
if ((washingMachine = sem_open("/semaphore", O_CREAT, 0777, m)) == SEM_FAILED)
{
    perror("sem_open");
    exit(EXIT_FAILURE);
}
// initializing semaphore with value m
sem_init(&washingMachine, 0, m);

// creating n threads for each student
pthread_t students[n];
for (int i = 0; i < n; i++)
{
    pthread_create(&students[i], NULL, threadFunc, i);
    usleep(1);
}
```

6. In the threadFunc passed as an argument, sleep was applied equivalent to the arrival time. Following this, the value of the counting semaphore was checked using the sem_trywait function.
7. If the return value of the sem_trywait function is 0, then the semaphore is available, i.e., a washing machine is available to use. Post this, the value of the semaphore is decreased by 1, and the student starts washing is printed. Sleep is applied equivalent to the washing time of the student.
8. After the washing is completed, student leaves after washing is printed and the value of the semaphore is incremented using the sem_post function.
9. A signal is sent to the conditional variable using pthread_cond_signal that the washing machine is free to use.

```
// sleep till time of arrival comes
sleep(arrStudents[i].arrivalTime);
int tme2 = time(NULL);
tme2 = tme2 - tme;
int returnVal;
printf("\033[0;37m%d: Student %d arrives\033[0;37m\n", tme2, arrStudents[i].id);
// if semaphore is available, it goes inside the if condition
if (sem_trywait(washingMachine) == 0)
{
    timeWasted += tme2 - arrStudents[i].arrivalTime;
    printf("\033[0;32m%d: Student %d starts washing\033[0;37m\n", tme2, arrStudents[i].id);
    sleep(arrStudents[i].washingTime);
    tme2 = time(NULL);
    tme2 = tme2 - tme;
    printf("\033[0;33m%d: Student %d leaves after washing\033[0;37m\n", tme2, arrStudents[i].id);
    arrStudents[i].washingFlag = 1;
    // release the semaphore i.e. increment it's value
    sem_post(washingMachine);
    // send signal, that machine is available
    pthread_mutex_lock(&mutex);
    pthread_cond_signal(&cond);
    pthread_mutex_unlock(&mutex);
```

10. In the case, the value of the sem_trywait function is not 0, then the patience time is initialized using the pthread_cond_timedwait function.

```
printf("");
struct timespec waitTime;
struct timeval now;
gettimeofday(&now, NULL);
waitTime.tv_sec = now.tv_sec + arrStudents[i].patienceTime;
waitTime.tv_nsec = (1.5)*(now.tv_usec * 1000); // could use 7 zeros

// initializing patience time of student
pthread_mutex_lock(&mutex);
returnVal = pthread_cond_timedwait(&cond, &mutex, &waitTime);
pthread_mutex_unlock(&mutex);
```

11. If this function returns 0, then the signal has been received and a washing machine is free to use, before the patience time of a student expires. This machine can now be acquired by one of the waiting students i.e., waiting threads.

```
// if returnVal == 0 then machine has become
// empty during the patience time period
if(returnVal == 0)
{
    if (arrStudents[i].washingFlag == 0)
    {
        // allocate machine to this new thread
        if (sem_trywait(washingMachine) == 0)
        {
            tme2 = time(NULL);
            tme2 = tme2 - tme;
            timeWasted += tme2 - arrStudents[i].arrivalTime;
            printf("\033[0;32m%d: Student %d starts washing\033[0;37m\n", tme2, arrStudents[i].id);
            sleep(arrStudents[i].washingTime);
            tme2 = time(NULL);
            tme2 = tme2 - tme;
            printf("\033[0;33m%d: Student %d leaves after washing\033[0;37m\n", tme2, arrStudents[i].id);
            arrStudents[i].washingFlag = 1;
            sem_post(washingMachine);
            pthread_mutex_lock(&mutex);
            pthread_cond_signal(&cond);
            pthread_mutex_unlock(&mutex);
        }
    }
}
```

12. If the function pthread_cond_timedwait returns a value other than 0, then the patience time of the student has expired, and student leaves without washing is printed.

```
else{
    int tm2 = time(NULL);
    tm2 = tm2 - tme;
    // current[currIndex] = i;
    // currIndex++;
    counter++;
    timeWasted += arrStudents[i].patienceTime;
    printf("\033[0;31m%d: Student %d leaves without washing\033[0;37m\n", tm2, arrStudents[i].id);
    arrStudents[i].washingFlag = 1;
}
```

13. After, all the threads have been created, and joined, the semaphore is unlinked.
14. In the end, the number of students who returned without washing their clothes, time wasted, and if more machines are required is printed.

OUTPUT:

Test Case 1:

```
5 2
6 3 5
3 4 3
6 5 2
2 9 6
8 5 2
2: Student 4 arrives
2: Student 4 starts washing
3: Student 2 arrives
3: Student 2 starts washing
6: Student 1 arrives
6: Student 3 arrives
7: Student 2 leaves after washing
7: Student 1 starts washing
8: Student 5 arrives
8: Student 3 leaves without washing
10: Student 1 leaves after washing
10: Student 5 starts washing
11: Student 4 leaves after washing
15: Student 5 leaves after washing
1
5
No
```

Test Case 2:

```
3 1
2 5 1
1 2 4
2 4 2
1: Student 2 arrives
1: Student 2 starts washing
2: Student 1 arrives
2: Student 3 arrives
3: Student 2 leaves after washing
3: Student 1 starts washing
4: Student 3 leaves without washing
8: Student 1 leaves after washing
1
3
Yes
```

PROBLEMS FACED AND ASSUMPTIONS:

1. Patience time is assumed to a very small quantity (few nanoseconds) more than the integer value in seconds. This is to ensure, that if the exit time (washing machine getting free) is equal to the exhaustion of the patience time, then the student can start washing clothes as well. Hence, the following is assumed.

   waitTime.tv_nsec = (1.5) * (now.tv_usec * 1000);

2. At times, the program must be compiled more than once, as the threads cannot be initialized on the CPU due to which threads which are to be waiting with their patience time, exit at the same time of their arrive. This problem is fixed if the program is compiled again. After compiling again, the correct output is displayed on the screen.