Assignment-5 Q1

Tags	ASSIGNMENT
contributors	Rhythm

Q1-Washing Machine

N students will come to get their clothes washed
• There are M functioning washing machines
• The time at which the i-th student comes is T_i seconds after the execution of the program
• The time taken to wash the i-th student's clothes is W_i seconds
• The patience of the i-th student is P_i seconds, after which he leaves without getting his clothes washed

- ***Implementation***

▼ We first create a washing machine semaphore to tell if a washing machine is free or not for a student.

```
sem_t* curr_sem;
...
if((curr_sem = sem_open("/semaphore",O_CREAT,0777,m)) == SEM_FAILED)
{
    perror("sem open");
    exit(0);
}
```

▼ I take input in a struct array that stores all the relevant details related to the student process.

▼ after taking the input i sort the data since we have to implement FCFS and it requires that the threads come in after sorted by arrival, and then student index. We use the fact that **if the threads are created in a particular order with a certain delay then they access the semaphore in that order only.**

```
typedef struct  sdetails
{
    pthread_t pid;
    int student_no;
    int arrival;
    int duration;
    int patience;
    int start_time;
    int arrived;
}td;
```

▼ I store the current real time in *startcode* variable denoting starting of the code. It will be used to calculate the timestamps of all the events. After this we create all the student threads.

```
start_code = time(0);
for(int i=0;i<n;i++)
{
    pthread_create(&student_details[i].pid,NULL,student,(void*)(&student_details[i]));
    usleep(0.5);
}
```

▼ in the student thread first the student is sleeping till its arrival time after which it attempts to lock the semaphore till it is timed out (used sem_timedwait()). the timeout has been set to its Patience time. If before the timeout the student is able to aquire the washing machine semaphore then it starts washing, else the student has to leave without washing

```
sleep(((td*)student)->arrival);
    printf("%d:Student %d arrives\n",((td*)student)->arrival,((td*)student)->student_no);
    ((td*)student)->arrived = 1;
    struct timespec ts;
    if (clock_gettime(CLOCK_REALTIME, &ts) == -1)
    {
        return NULL;
    }
    ts.tv_sec += ((td*)student)->patience;
    int s;
    // while ((s = sem_timedwait(curr_sem, &ts)) == -1 && errno == EINTR)
    //            continue;       /* Restart if interrupted by handler */
    /* Check what happened */
    if (sem_timedwait(curr_sem, &ts) == -1)
    {
        if (errno == ETIMEDOUT)
        {
            for(int i=0;i<10000;i++);
            if(sem_trywait(curr_sem) == -1)
            {
                printf("\033[0;31m");
                printf("%d:Student %d leaves without washing\n",((td*)student)->arrival + ((td*)student)->patience,((td*)student)->student_
                ((td*)student)->start_time = ((td*)student)->patience;
                pthread_mutex_lock(&not_wash_update);
                not_wash++;
                pthread_mutex_unlock(&not_wash_update);
                printf("\033[0;37m");
                return NULL;
            }
            else{
                time_t curr = time(0);
                printf("\033[0;32m");
                printf("%d:Student %d starts washing\n",(int)(curr - start_code),((td*)student)->student_no);
                ((td*)student)->start_time = (int)(curr - start_code) - ((td*)student)->arrival;
                printf("\033[0;37m");
                sleep(((td*)student)->duration);
                printf("\033[0;33m");
                printf("%ld:Student %d leaves after washing\n",time(0) - start_code,((td*)student)->student_no);
                printf("\033[0;37m");

                sem_post(curr_sem);
            }
        }
        else
            perror("sem_timedwait");
    } else{
            time_t curr = time(0);
            printf("\033[0;32m");
            printf("%d:Student %d starts washing\n",(int)(curr - start_code),((td*)student)->student_no);
            ((td*)student)->start_time = (int)(curr - start_code) - ((td*)student)->arrival;
            printf("\033[0;37m");
            sleep(((td*)student)->duration);
            printf("\033[0;33m");
            printf("%ld:Student %d leaves after washing\n",time(0) - start_code,((td*)student)->student_no);
            printf("\033[0;37m");

            sem_post(curr_sem);
            // printf("sem_timedwait() succeeded\n")
    }
    return NULL;
```

Q3-Internet Routing

A routing table is a set of rules, often viewed in table format, that is used to determine where data packets traveling over an Internet Protocol (IP) network will be directed. In this question you will be simulating the process of generating and using a routing table

- We first create a routing table for every node that stores the first forward edge to reach a destination from the source node. To create the routing table, we run dijikstra's algorithm n times for every node, calculate the shortest path and take the first edge of the path as the forward edge

```
vector<pair<int,int> > vec[n];
vector<pair<int,int> > routing[n];
dijikstras()
update_routing()
```

▼ Now we create a phthread for every edge of the node graph. Edge contains the necessary data like the client, server, their socket_fds, port for the connection and the routing table of the server node.

```
typedef struct edge_details
{
    pthread_t pid;
    int server_node;
    int client_node;
    int client_socked_fd;
    int server_socket_fd;
    int server_uses_this;
    vector<pair<int,int> > server_routing;
    int port;
}edge;
```

▼ in the thread function, a TCP connection is established between the client and the server taking help of the code given for the server.cpp and client.cpp. After the connection is established the thread server halts for reading message from the client node of the edge.

```
void *thread_function()
...
while(1)
    {
        string cmd;
        tie(cmd, received_num) = read_string_from_socket(client_socket_fd, buff_sz);
        curr_node = ((edge*)ed)->server_node;
        printf("Data received at node: %d : Source : %d : Destination : %d : Forwarded_Destination : %d : ",curr_node,((edge*)ed)->client_n
        cout <<"Message: " << cmd << "\n";
        if(curr_node == dest) continue;
        send_string_on_socket(edge_data[curr_node][((edge*)ed)->server_routing[dest].second]->client_socked_fd,cmd);
    }
...
```

▼ when the client receives message from a server, it sends the message to the server which is the respective forward node in the client's routing table. For message passing between 2 nodes of an edge we use the TCP pathway already established in the threads.

▼ So the server of the edge first reads the message from its client, after which is passes the message as a client to its forward node using the routing table I have pre calculated before.