Reg.no 11811930

Student Name. : kolli Sai Babu

Roll no : 56

Email: saikolli900@gmail.com

GitHub Link:

https://github.com/Code

Code:

```
// Sudesh Sharma is a Linux expert who wants to have an online system where he can handle student queries. Since there can be multiple

// requests at any time he wishes to dedicate a fixed amount of time to every request so that everyone gets a fair share of his time.

// He will log into the system from 10am to 12am only.  He wants to have separate requests queues for students and faculty.

// Implement a strategy for the same. The summary at the end of the session should include the total time he spent on handling queries

// and average query time.


#include<stdio.h>

#include<string.h>


struct process_Struct {

   char process_name[20];

   int arrival_time, burst_time, completion_time, remaining;

}temp_Struct;



void faculty_Queue(int no_of_process) {


   int count, arrival_Time, burst_Time, quantum_time;

   struct process_Struct faculty_Process[no_of_process];
```

```
for(count = 0; count < no_of_process; count++) {
   printf("Enter the details of Process[%d]", count+1);
   puts("");
   printf("Process Name : ");
   scanf("%s", faculty_Process[count].process_name);

   printf("Arrival Time : ");
   scanf("%d", &faculty_Process[count].arrival_time);

   printf("Burst Time : ");
   scanf("%d", &faculty_Process[count].burst_time);
   puts("");
}
printf("Now, enter the quantum time for FACULTY queue : ");
scanf("%d", &quantum_time);



// sorting the processes by their ARRIVAL time.
// if the ARRIVAL time is same then scheduling is based on FCFS.
for(count = 0; count < no_of_process; count++) {
   for(int x = count +1; x < count; x++){
      if(faculty_Process[count].arrival_time > faculty_Process[x].arrival_time) {
         temp_Struct = faculty_Process[count];
         faculty_Process[count] = faculty_Process[x];
         faculty_Process[x] = temp_Struct;
      }
   }
}
```

```
// initialy all the burst time is remaining and completion of process is zero.
for(count = 0; count < no_of_process; count++) {
   faculty_Process[count].remaining = faculty_Process[count].burst_time;
   faculty_Process[count].completion_time = 0;
}

int total_time, queue, round_robin[20];
total_time = 0;
queue = 0;
round_robin[queue] = 0;


int flag, x, n, z, waiting_time = 0;
do {
   for(count = 0; count < no_of_process; count++){
      if(total_time >= faculty_Process[count].arrival_time){
         z = 0;
         for(x = 0; x <= queue; x++) {
            if(round_robin[x] == count) {
               z++;
            }
         }
         if(z == 0) {
            queue++;
            round_robin[queue] == count;
         }
      }
   }
```

```
if(queue == 0) {
   n = 0;
}
if(faculty_Process[n].remaining == 0) {
   n++ ;
}
if(n > queue) {
   n = (n - 1) % queue;
}
if(n <= queue) {
   if(faculty_Process[n].remaining > 0) {
      if(faculty_Process[n].remaining < quantum_time){
         total_time += faculty_Process[n].remaining;
         faculty_Process[n].remaining = 0;
      }else {
         total_time += quantum_time;
         faculty_Process[n].remaining -= quantum_time;
      }
      faculty_Process[n].completion_time = total_time;
   }
   n++;
}
flag = 0;

for(count = 0; count < no_of_process; count++) {
   if(faculty_Process[count].remaining > 0) {
      flag++;
   }
```

```
    }
  }while(flag != 0);



  puts("\n\t\t\t*****************************************");
  puts("\t\t\t*****   ROUND ROBIN ALGORITHM OUTPUT   *****");
  puts("\t\t\t*****************************************\n");
  printf("\n|\tProcess Name\t  |\tArrival Time\t  |\tBurst Time\t |\tCompletion Time  \t|\n");


  for(count = 0; count < no_of_process; count++){
    waiting_time = faculty_Process[count].completion_time - faculty_Process[count].burst_time -
faculty_Process[count].arrival_time;


    printf("\n|\t %s\t   |\t %d\t   |\t %d\t   |\t %d\t   |\n", faculty_Process[count].process_name,
faculty_Process[count].arrival_time, faculty_Process[count].burst_time,
faculty_Process[count].completion_time);
  }


}



void student_Queue(int no_of_process) {


  int count, arrival_Time, burst_Time, quantum_time;
  struct process_Struct student_Process[no_of_process];


  for(count = 0; count < no_of_process; count++) {
    printf("Enter the details of Process[%d]", count+1);
    puts("");
    printf("Process Name : ");
```

```
    scanf("%s", student_Process[count].process_name);

    printf("Arrival Time : ");
    scanf("%d", &student_Process[count].arrival_time);

    printf("Burst Time : ");
    scanf("%d", &student_Process[count].burst_time);
}
printf("Now, enter the quantum time for STUDENT queue : ");
scanf("%d", &quantum_time);


// sorting the processes by their ARRIVAL time.
// if the ARRIVAL time is same then scheduling is based on FCFS.
for(count = 0; count < no_of_process; count++) {
    for(int x = count +1; x < count; x++){
        if(student_Process[count].arrival_time > student_Process[x].arrival_time) {
            temp_Struct = student_Process[count];
            student_Process[count] = student_Process[x];
            student_Process[x] = temp_Struct;
        }
    }
}

// initialy all the burst time is remaining and completion of process is zero.
for(count = 0; count < no_of_process; count++) {
    student_Process[count].remaining = student_Process[count].burst_time;
    student_Process[count].completion_time = 0;
}
```

```
    int total_time, queue, round_robin[20];
    total_time = 0;
    queue = 0;
    round_robin[queue] = 0;
}


int main(int argc, char const *argv[]) {
    int select_queue, no_of_process;

    puts("Please choose a queue to post your query : ");
    puts("1. FACULTY queue.");
    puts("2. STUDENT queue.");
    printf("> ");
    scanf("%d", &select_queue);

    switch(select_queue) {
        case 1 :
            printf("Enter number of process for FACULTY queue : ");
            scanf("%d", &no_of_process);

            faculty_Queue(no_of_process);

            break;

        case 2 :
            printf("Enter number of process for STUDENT queue : ");
            scanf("%d", &no_of_process);
```

```
            student_Queue(no_of_process);


            break;


      default :
            printf("Please selet the correct option by running the program again.");
   }


   return 0;
}
```

Description:

The given problem is based upon solving queries of persons of different classes i.e. Faculty and Students. Thus, these queries can be compared to different processes in terms of operating system where each process has its demands and needs resources and time for its execution. And this demands of processes are handled by the CPU. In the given scenario, Mr. Sudesh Sharma, Linux expert, can be considered as a CPU, who solves the queries of either Faculty or Student by allocating proper resources to their individual demands and processing them by allocating them time accordingly. Now, Mr. Sharma, wants to provide priority for each query based upon its class, as well as, he wants to dedicate a fixed amount of time to every request. Thus in Operating System, if we divide the requests into two separate queues i.e. Faculty and Student such that the first queue contains faculty queries has higher priority and the second contains student queries which has lower priority, then we can resolve the problem, by allocating them required resources based upon their priorities as done in the scheduling algorithm in operating systems.