**CSE 316: OPERATING SYSTEMS**



# NAME : Manikranth Reddy

# REG NO: 11809502

# ROLL NO: 12

# SECTION : K18PB

Q: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.

Sol:

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.

Code:

#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;

}