Multilevel queue scheduling

#include <stdio.h>

#define MAX 10

typedef struct {

int pid; // Process ID

int arrival\_time; // Arrival Time

int burst\_time; // Burst Time

int remaining\_time; // Remaining Time for Round Robin

} Process;

void fcfs(Process queue[], int n, const char\* queue\_name) {

int waiting\_time[MAX], turnaround\_time[MAX], completion\_time[MAX];

int total\_wt = 0, total\_tat = 0;

printf("\nScheduling %s Queue (FCFS)\n", queue\_name);

printf("PID\tArrival\tBurst\tCompletion\tTurnaround\tWaiting\n");

completion\_time[0] = queue[0].arrival\_time + queue[0].burst\_time;

turnaround\_time[0] = completion\_time[0] - queue[0].arrival\_time;

waiting\_time[0] = turnaround\_time[0] - queue[0].burst\_time;

for (int i = 1; i < n; i++) {

if (completion\_time[i - 1] < queue[i].arrival\_time)

completion\_time[i] = queue[i].arrival\_time + queue[i].burst\_time;

else

completion\_time[i] = completion\_time[i - 1] + queue[i].burst\_time;

turnaround\_time[i] = completion\_time[i] - queue[i].arrival\_time;

waiting\_time[i] = turnaround\_time[i] - queue[i].burst\_time;

}

for (int i = 0; i < n; i++) {

printf("%d\t%d\t%d\t%d\t%d\t%d\n", queue[i].pid, queue[i].arrival\_time,

queue[i].burst\_time, completion\_time[i], turnaround\_time[i], waiting\_time[i]);

total\_wt += waiting\_time[i];

total\_tat += turnaround\_time[i];

}

printf("Average Waiting Time: %.2f\n", (float)total\_wt / n);

printf("Average Turnaround Time: %.2f\n", (float)total\_tat / n);

}

void roundRobin(Process queue[], int n, int quantum, const char\* queue\_name) {

int waiting\_time[MAX], turnaround\_time[MAX], completion\_time[MAX];

int total\_wt = 0, total\_tat = 0;

int remaining\_processes = n;

int time = 0;

for (int i = 0; i < n; i++) {

queue[i].remaining\_time = queue[i].burst\_time;

}

printf("\nScheduling %s Queue (Round Robin, Quantum = %d)\n", queue\_name, quantum);

printf("PID\tArrival\tBurst\tCompletion\tTurnaround\tWaiting\n");

while (remaining\_processes > 0) {

for (int i = 0; i < n; i++) {

if (queue[i].remaining\_time > 0) {

if (queue[i].remaining\_time > quantum) {

time += quantum;

queue[i].remaining\_time -= quantum;

} else {

time += queue[i].remaining\_time;

completion\_time[i] = time;

turnaround\_time[i] = completion\_time[i] - queue[i].arrival\_time;

waiting\_time[i] = turnaround\_time[i] - queue[i].burst\_time;

total\_wt += waiting\_time[i];

total\_tat += turnaround\_time[i];

queue[i].remaining\_time = 0;

remaining\_processes--;

}

}

}

}

for (int i = 0; i < n; i++) {

printf("%d\t%d\t%d\t%d\t%d\t%d\n", queue[i].pid, queue[i].arrival\_time,

queue[i].burst\_time, completion\_time[i], turnaround\_time[i], waiting\_time[i]);

}

printf("Average Waiting Time: %.2f\n", (float)total\_wt / n);

printf("Average Turnaround Time: %.2f\n", (float)total\_tat / n);

}

int main() {

int n, sys\_count = 0, user\_count = 0;

int quantum;

Process system\_queue[MAX], user\_queue[MAX], processes[MAX];

printf("Enter the number of processes: ");

scanf("%d", &n);

for (int i = 0; i < n; i++) {

int type;

printf("Enter PID, Arrival Time, Burst Time, Type (0 for System, 1 for User) for Process %d: ", i + 1);

scanf("%d %d %d %d", &processes[i].pid, &processes[i].arrival\_time, &processes[i].burst\_time, &type);

if (type == 0)

system\_queue[sys\_count++] = processes[i];

else

user\_queue[user\_count++] = processes[i];

}

int ch;

printf("enter choice");

scanf("%d",&ch);

switch(ch)

{

case 1:

fcfs(system\_queue, sys\_count, "System");

fcfs(user\_queue, user\_count, "User");

break;

case 2:

printf("Enter Time Quantum for Round Robin: ");

scanf("%d", &quantum);

roundRobin(system\_queue, sys\_count, quantum, "System");

fcfs(user\_queue, user\_count, "User");

break;

}

return 0;

}

Output:

