Multilevel queue both fcfs

#include <stdio.h>

#define MAX 100

typedef struct {

int pid, arrival, burst, completion, turnaround, waiting;

} Process;

void inputProcesses(Process sys[], int \*sysCount, Process user[], int \*userCount);

void fcfsScheduling(Process p[], int n, int startTime);

void displayProcesses(Process p[], int n, const char \*queueName);

int main() {

Process systemQueue[MAX], userQueue[MAX];

int sysCount = 0, userCount = 0;

inputProcesses(systemQueue, &sysCount, userQueue, &userCount);

fcfsScheduling(systemQueue, sysCount, 0);

fcfsScheduling(userQueue, userCount, systemQueue[sysCount - 1].completion);

displayProcesses(systemQueue, sysCount, "System Queue");

displayProcesses(userQueue, userCount, "User Queue");

return 0;

}

void inputProcesses(Process sys[], int \*sysCount, Process user[], int \*userCount) {

int n, type;

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

scanf("%d", &n);

printf("\nEnter Process Details: (PID Arrival Burst Type[0=System, 1=User])\n");

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

Process temp;

printf("Process %d: ", i + 1);

scanf("%d %d %d %d", &temp.pid, &temp.arrival, &temp.burst, &type);

if (type == 0) {

sys[\*sysCount] = temp;

(\*sysCount)++;

} else {

user[\*userCount] = temp;

(\*userCount)++;

}

}

}

void fcfsScheduling(Process p[], int n, int startTime) {

if (n == 0) return;

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (p[j].arrival > p[j + 1].arrival) {

Process temp = p[j];

p[j] = p[j + 1];

p[j + 1] = temp;

}

}

}

int time = startTime;

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

if (time < p[i].arrival) {

time = p[i].arrival; // CPU idle time handled

}

p[i].completion = time + p[i].burst;

p[i].turnaround = p[i].completion - p[i].arrival;

p[i].waiting = p[i].turnaround - p[i].burst;

time = p[i].completion;

}

}

void displayProcesses(Process p[], int n, const char \*queueName) {

if (n == 0) {

printf("\nNo processes in %s.\n", queueName);

return;

}

printf("\n%s Execution Order:\n", queueName);

printf("PID Arrival Burst Completion Turnaround Waiting\n");

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

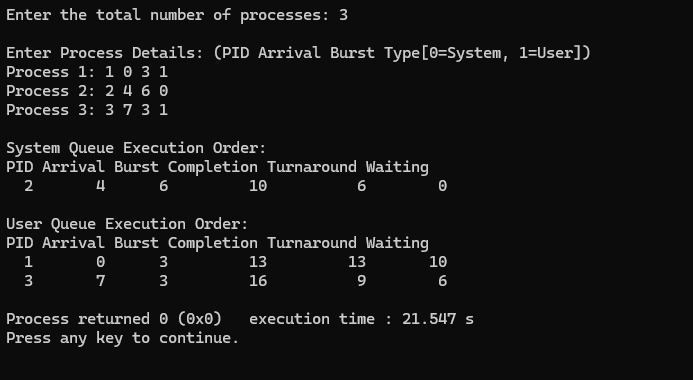
printf("%3d %7d %6d %10d %10d %8d\n",

p[i].pid, p[i].arrival, p[i].burst,

p[i].completion, p[i].turnaround, p[i].waiting);

}

}



Multilevel queue (fcfs and round robin)

#include <stdio.h>

#define MAX 100

typedef struct { int pid, arrival, burst, completion, turnaround, waiting, remaining; } Process;

void inputProcesses(Process sys[], int \*sysCount, Process user[], int \*userCount);

void fcfsScheduling(Process p[], int n, int startTime); void roundRobinScheduling(Process p[], int n, int startTime, int timeQuantum);

void displayProcesses(Process p[], int n, const char \*queueName);

int main() { Process systemQueue[MAX], userQueue[MAX]; int sysCount = 0, userCount = 0, timeQuantum;

inputProcesses(systemQueue, &sysCount, userQueue, &userCount);

printf("Enter the time quantum for Round Robin: ");

scanf("%d", &timeQuantum);

fcfsScheduling(systemQueue, sysCount, 0);

roundRobinScheduling(userQueue, userCount, systemQueue[sysCount - 1].completion, timeQuantum);

displayProcesses(systemQueue, sysCount, "System Queue (FCFS)");

displayProcesses(userQueue, userCount, "User Queue (Round Robin)");

return 0;

}

void inputProcesses(Process sys[], int \*sysCount, Process user[], int \*userCount) { int n, type; printf("Enter the total number of processes: "); scanf("%d", &n);

printf("\nEnter Process Details: (PID Arrival Burst Type[0=System, 1=User])\n");

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

Process temp;

printf("Process %d: ", i + 1);

scanf("%d %d %d %d", &temp.pid, &temp.arrival, &temp.burst, &type);

temp.remaining = temp.burst;

if (type == 0) {

sys[\*sysCount] = temp;

(\*sysCount)++;

} else {

user[\*userCount] = temp;

(\*userCount)++;

}

}

}

void fcfsScheduling(Process p[], int n, int startTime) { if (n == 0) return;

int time = startTime;

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

if (time < p[i].arrival) {

time = p[i].arrival;

}

p[i].completion = time + p[i].burst;

p[i].turnaround = p[i].completion - p[i].arrival;

p[i].waiting = p[i].turnaround - p[i].burst;

time = p[i].completion;

}

}

void roundRobinScheduling(Process p[], int n, int startTime, int timeQuantum) { if (n == 0) return;

int time = startTime, remainingProcesses = n;

while (remainingProcesses > 0) {

int allIdle = 1;

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

if (p[i].remaining > 0 && p[i].arrival <= time) {

allIdle = 0;

if (p[i].remaining > timeQuantum) {

time += timeQuantum;

p[i].remaining -= timeQuantum;

} else {

time += p[i].remaining;

p[i].completion = time;

p[i].turnaround = p[i].completion - p[i].arrival;

p[i].waiting = p[i].turnaround - p[i].burst;

p[i].remaining = 0;

remainingProcesses--;

}

}

}

if (allIdle) time++;

}

}

void displayProcesses(Process p[], int n, const char \*queueName) { if (n == 0) { printf("\nNo processes in %s.\n", queueName); return; }

printf("\n%s Execution Order:\n", queueName);

printf("PID Arrival Burst Completion Turnaround Waiting\n");

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

printf("%3d %7d %6d %10d %10d %8d\n",

p[i].pid, p[i].arrival, p[i].burst,

p[i].completion, p[i].turnaround, p[i].waiting);

}

}

