***EX:3 PRIORITY & RR***

***CPU SCHEDULING***

***-S.Vishakan CSE-C 18 5001 196***

***Source Code:***

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include "Queue.h"**

**/\* DEFINED IN QUEUE.H**

**typedef struct{**

**char pid[10]; //process ID**

**int arrtime; //arrival time**

**int bursttime; //burst time**

**int resptime; //resp time**

**int waittime; //wait time**

**int turntime; //turnaround time**

**int comptime; //completion time**

**int nextidle; //idle time after completion**

**int remtime //remaining time**

**int status; //completed or not**

**int queued; //indicates whether or not process is in queue**

**int prio; //process priority**

**}pInfo;**

**\*/**

**pInfo \*processInput(int numprocess, int flag);**

**pInfo \*processArrivalSort(pInfo \*process, int numprocess);**

**void chartPrinter(pInfo \*process, int numprocess);**

**void chartGantt(pInfo \*process, int numprocess);**

**pInfo \*schedulerRR(pInfo \*process, int numprocess, int quantum);**

**pInfo \*schedulerPPNP(pInfo \*process, int numprocess);**

**pInfo \*schedulerPPP(pInfo \*process, int numprocess);**

**int min(int a, int b);**

**int main(void){**

**int i, numprocess, opt=1, p=0, quantum = 0;**

**while(opt != 0){**

**printf("\n\t\tCPU SCHEDULING ALGORITHMS\n");**

**printf("\n\t1. ROUND ROBIN\n\t2. PRIORITY\n\t0. EXIT\n\tYour Option -> ");**

**scanf("%d",&opt);**

**if(opt == 1){**

**printf("\n\tROUND ROBIN CPU SCHEDULER\n");**

**printf("\nEnter Time Quantum : ");**

**scanf("%d", &quantum);**

**}**

**else if(opt == 2){**

**printf("\n\tPRIORITY CPU SCHEDULER\n");**

**printf("\t1. PREEMPTIVE\n\t2. NON-PREEMPTIVE\n\tYour Option -> ");**

**scanf("%d",&p);**

**while(p>2 || p<=0){**

**printf("\n\tInvalid Option!\n");**

**printf("\t1. PREEMPTIVE\n\t2. NON-PREEMPTIVE\n\tYour Option -> ");**

**scanf("%d",&p);**

**}**

**}**

**else if(opt == 0){**

**printf("\n\tThank You!\n");**

**return 0;**

**}**

**else{**

**printf("\n\tInvalid Option!\n");**

**return 0;**

**}**

**printf("\nEnter the Number of Processes : ");**

**scanf("%d",&numprocess);**

**pInfo \*process;**

**process = (pInfo \*)malloc(sizeof(pInfo)\*numprocess);**

**if(opt == 1){**

**process = processInput(numprocess, 0);**

**process = processArrivalSort(process, numprocess);**

**process = schedulerRR(process, numprocess, quantum);**

**}**

**else if(opt == 2 && p == 1){**

**process = processInput(numprocess, 1);**

**process = processArrivalSort(process, numprocess);**

**process = schedulerPPP(process, numprocess);**

**chartPrinter(process,numprocess);**

**}**

**else if(opt == 2 && p == 2){**

**process = processInput(numprocess, 1);**

**process = processArrivalSort(process, numprocess);**

**process = schedulerPPNP(process, numprocess);**

**chartGantt(process, numprocess);**

**chartPrinter(process,numprocess);**

**}**

**else{**

**return 0;**

**}**

**}**

**return 0;**

**}**

**int min(int a, int b){**

**if(a <= b)**

**return a;**

**return b;**

**}**

**pInfo \*processInput(int numprocess, int flag){**

**pInfo \*process;**

**process = (pInfo \*)malloc(sizeof(pInfo)\*numprocess);**

**int i=0;**

**while(i<numprocess){**

**printf("\nEnter the Process ID: ");**

**scanf("%s",process[i].pid);**

**printf("\nEnter the Arrival Time of %s: ",process[i].pid);**

**scanf("%d",&process[i].arrtime);**

**printf("\nEnter the CPU Burst Time of %s: ",process[i].pid);**

**scanf("%d",&process[i].bursttime);**

**if(flag == 1){**

**printf("\nEnter the Priority of %s: ",process[i].pid);**

**scanf("%d",&process[i].prio);**

**}**

**else**

**process[i].prio = 0;**

**process[i].waittime = 0;**

**process[i].resptime = 0;**

**process[i].turntime = 0;**

**process[i].nextidle = 0;**

**process[i].status = 0;**

**process[i].comptime = 0;**

**process[i].queued = 0;**

**process[i].bufftime = 0;**

**process[i].remtime = process[i].bursttime;**

**++i;**

**}**

**return process;**

**}**

**pInfo \*processArrivalSort(pInfo \*process, int numprocess){**

**pInfo proctemp;**

**int i=0,j=0,temp=0;**

**for(i=0;i<numprocess;++i){**

**temp = i;**

**for(j=i+1;j<numprocess;++j){**

**if(process[temp].arrtime > process[j].arrtime){**

**temp = j;**

**}**

**proctemp = process[i];**

**process[i] = process[temp];**

**process[temp] = proctemp;**

**}**

**}**

**return process;**

**}**

**void chartPrinter(pInfo \*process, int numprocess){**

**int i=0;**

**float avgwait=0, avgresponse=0;**

**printf("\n---------------------------------------------------------------------\n");**

**printf("| %5s | %7s | %7s | %7s | %7s | %7s | %7s |", "P. ID", "A. Time", "B. Time", "C. Time", "T. Time", "W. Time", "R. Time");**

**printf("\n---------------------------------------------------------------------\n");**

**while(i<numprocess){**

**avgwait += process[i].waittime;**

**avgresponse += process[i].resptime;**

**printf("| %-5s | %-7d | %-7d | %-7d | %-7d | %-7d | %-7d |\n",process[i].pid, process[i].arrtime, process[i].bursttime, process[i].comptime, process[i].turntime, process[i].waittime, process[i].resptime);**

**++i;**

**}**

**avgwait/=numprocess;**

**avgresponse/=numprocess;**

**printf("---------------------------------------------------------------------\n");**

**printf("\nAverage Waiting Time : %.2f\nAverage Response Time : %.2f",avgwait, avgresponse);**

**printf("\n---------------------------------------------------------------------\n");**

**return;**

**}**

**void chartGantt(pInfo \*process, int numprocess){**

**int i = 0, numidles = 0, n = 0;**

**printf("\nGantt Chart:\n");**

**for(i=0;i<numprocess;++i){**

**if(process[i].nextidle !=0)**

**++numidles;**

**}**

**n = numprocess + numidles;**

**if(process[0].arrtime != 0) //accounting for init idle time, if any**

**++n;**

**for(i=0;i<n;++i)**

**printf("--------");**

**printf("\n|");**

**if(process[0].arrtime != 0)**

**printf(" %-5s |","IDLE");**

**for(i=0;i<numprocess;++i){**

**printf(" %-5s |",process[i].pid);**

**if(process[i].nextidle != 0){**

**printf(" %-5s |","IDLE");**

**}**

**}**

**printf("\n");**

**for(i=0;i<n;++i)**

**printf("--------");**

**if(process[0].arrtime != 0)**

**printf("\n%-5d\t",0);**

**else**

**printf("\n");**

**printf("%-5d\t",process[0].arrtime - process[i].waittime);**

**for(i=0;i<numprocess;++i){**

**printf("%-5d\t",process[i].comptime);**

**if(process[i].nextidle != 0){**

**printf(" %-5d ",process[i].comptime + process[i].nextidle);**

**}**

**}**

**printf("\n");**

**}**

**int statusChecker(pInfo \*process, int numprocess){**

**int i = 0, min = 0, flag = -1;**

**while(i < numprocess){**

**if(process[i].status == 1){**

**++i;**

**continue;**

**}**

**else{**

**flag = i;**

**break;**

**}**

**++i;**

**}**

**if(flag == -1) //returning -1 if all processes are complete.**

**return -1;**

**//i = min + 1;**

**min = flag;**

**while(process[i].status == 0 && i < numprocess){**

**if(process[i].prio < process[min].prio){**

**min = i;**

**}**

**++i;**

**}**

**return min; //returning the minimum burst time unfinished process**

**}**

**pInfo \*schedulerRR(pInfo \*process, int numprocess, int quantum){**

**Queue \*ready;**

**ready = (Queue \*)malloc(sizeof(Queue));**

**pInfo \*listGantt;**

**listGantt = (pInfo \*)malloc(sizeof(pInfo)\*100);**

**pInfo temp;**

**initQueue(ready, numprocess);**

**int currentwait = process[0].arrtime, job = 0, nextjob = 0, idletime = 0, i = 0, elapsedtime = 0;**

**while(statusChecker(process, numprocess) != -1){**

**job = nextjob;**

**elapsedtime = min(quantum, process[job].remtime);**

**if(process[job].remtime == process[job].bursttime){ //new process, first time executing**

**if(process[job].remtime <= quantum){ //if it can execute within the time quantum**

**process[job].status = 1;**

**process[job].waittime = currentwait + idletime - process[job].arrtime;**

**process[job].comptime = currentwait + elapsedtime;**

**process[job].turntime = process[job].comptime - process[job].arrtime;**

**process[job].remtime = 0;**

**//for Gantt Chart**

**temp = process[job];**

**}**

**else{ //else, incomplete process.**

**process[job].waittime = currentwait - process[job].arrtime + idletime;**

**process[job].remtime -= quantum;**

**//for Gantt Chart**

**temp = process[job];**

**temp.comptime = currentwait + quantum;**

**}**

**process[job].resptime = currentwait - process[job].arrtime;**

**}**

**else if(process[job].remtime <= quantum){ //if the process has been pre-empted before, and is executing again,**

**process[job].status = 1; //and can execute within time quantum**

**process[job].comptime = currentwait + elapsedtime;**

**process[job].turntime = process[job].comptime - process[job].arrtime;**

**process[job].waittime = process[job].turntime - process[job].bursttime;**

**process[job].remtime = 0;**

**temp = process[job];**

**}**

**else{ //otherwise.**

**process[job].remtime -= quantum;**

**process[job].waittime += (currentwait - quantum);**

**//for Gantt Chart**

**temp = process[job];**

**temp.comptime = currentwait + quantum;**

**}**

**int j = job+1; //going to the FCFS-wise next job**

**currentwait += elapsedtime;**

**//queueing algorithm**

**while(j < numprocess && process[j].arrtime <= currentwait){ //Queueing waiting processes to the RQ**

**if(process[j].status == 0 && process[j].queued == 0){**

**process[j].queued = 1;**

**enQueue(ready, j);**

**}**

**j++;**

**}**

**if(process[job].status == 0 && process[job].queued == 0){ //If the process currently executed hasn't completed yet.**

**enQueue(ready, job);**

**process[job].queued = 1;**

**}**

**j = job + 1;**

**while(process[j].arrtime <= currentwait && j < numprocess) //suppose the RQ is empty, and there is an idletime case**

**j = j + 1; //we shouldn't pick a process thats' already pre-empted and done before**

**if(isEmpty(ready) && j < numprocess){**

**idletime = process[j].arrtime - currentwait;**

**currentwait = process[j].arrtime;**

**temp.nextidle = idletime;**

**nextjob = j;**

**listGantt[i++] = temp;**

**continue;**

**}**

**else{**

**idletime = 0;**

**}**

**listGantt[i++] = temp;**

**if(!isEmpty(ready)){**

**nextjob = deQueue(ready);**

**process[nextjob].queued = 0;**

**}**

**else**

**break;**

**}**

**//check for idletimes**

**chartGantt(listGantt, i);**

**chartPrinter(process, numprocess);**

**return process;**

**}**

**pInfo \*schedulerPPNP(pInfo \*process, int numprocess){**

**int i = 0, currentwait = process[0].arrtime, idletime = 0, j = 0, nextproc = 0, iter = 1;**

**pInfo temp;**

**while(i<numprocess){**

**//++iter; //keeping no. of iterations in check**

**process[i].waittime = currentwait - process[i].arrtime + idletime;**

**process[i].resptime = process[i].waittime;**

**//printf("\n%s %d %d %d",process[i].pid, currentwait, idletime, process[i].bursttime);**

**process[i].comptime = currentwait + process[i].bursttime + idletime;**

**process[i].turntime = process[i].comptime - process[i].arrtime;**

**process[i].status = 1; //indicates process completion**

**currentwait = process[i].comptime;**

**nextproc = i;**

**for(j=0;j<numprocess;j++){ //finding the next process**

**if(process[j].status == 0){**

**if(process[j].arrtime <= process[i].comptime){**

**if(nextproc == i)**

**nextproc = j;**

**else{**

**if(process[j].prio < process[nextproc].prio)**

**nextproc = j;**

**}**

**}**

**}**

**}**

**idletime = 0; //setting default idletime = 0**

**if(nextproc == i) //if nextproc is unchanged**

**nextproc+=1;**

**if(i+1<numprocess){**

**temp = process[i+1]; //swapping the nextproc with i+1 to follow order**

**process[i+1] = process[nextproc];**

**process[nextproc] = temp;**

**//printf("\nPID : %s i Val : %d",process[i+1].pid, i+1);**

**if(process[i+1].arrtime > currentwait){ //finding idletime if any**

**idletime = process[i+1].arrtime - currentwait;**

**process[i].nextidle = idletime; //indicates the amt of idletime after process[i]**

**}**

**}**

**++i;**

**}**

**return process;**

**}**

**pInfo \*schedulerPPP(pInfo \*process, int numprocess){**

**int i = 0, currentwait = process[0].arrtime, idletime = 0;**

**int curtime = 0, nextjob = 0, j = 0, jobsdone = 0, job = 0, prevjob = 0;**

**pInfo \*queue, temp; //queue for Gantt Chart**

**queue = (pInfo \*)malloc(sizeof(pInfo)\*100);**

**while(statusChecker(process, numprocess) != -1){ //checking if all jobs are complete**

**job = nextjob;**

**int etime = process[job].bursttime + currentwait;**

**nextjob = -1;**

**//finding jobs that might trigger pre-empt**

**for(j=job+1;(process[j].arrtime < etime) && (j < numprocess) && (j!=job);j = (j+1)%numprocess){**

**if(process[j].status == 1 || j == job)**

**continue;**

**else if(nextjob == -1 && process[j].prio < process[job].prio)**

**nextjob = j;**

**//else if(temp <= process[nextjob].bursttime + process[nextjob].arrtime)**

**else if(process[j].prio < process[nextjob].prio) //Higher Priority First**

**nextjob = j;**

**else**

**continue;**

**}**

**//printf("\nLoop Chosen : %s", process[nextjob].pid);**

**if(nextjob == -1){ //if there's no job to pre-empt**

**if(process[job].bufftime == 0){ //if the process is hasn't been pre-empted before**

**process[job].resptime = currentwait - process[job].arrtime; //- idletime**

**process[job].comptime = currentwait + process[job].bursttime; //- idletime**

**}**

**else{**

**process[job].comptime = currentwait + process[job].bursttime - process[job].comptime;**

**}**

**//process[job].waittime += (currentwait - process[job].arrtime - 2\*process[job].waittime); // - idletime**

**process[job].status = 1;**

**process[job].turntime = process[job].comptime - process[job].arrtime;**

**process[job].waittime = process[job].turntime - process[job].bursttime;**

**process[job].temptime = process[job].bursttime;**

**for(j = 0;j<numprocess && process[j].arrtime <= etime; j++) //choosing the appropriate next process**

**continue; //priority - wise**

**nextjob = statusChecker(process,j);**

**if(nextjob == -1) //if there is an idle time scenario, and no process gets chosen**

**nextjob = job + 1; //choose the FCFS-wise next job**

**currentwait += process[job].remtime; //updating currentwait**

**temp = process[job];**

**}**

**else{ //pre-empt current job and queue next job**

**int stoptime = process[nextjob].arrtime;**

**if(process[job].bufftime == 0){**

**process[job].resptime = currentwait - process[job].arrtime; //- idletime**

**process[job].comptime = stoptime - currentwait;**

**}**

**temp = process[job];**

**process[job].bufftime = process[job].bursttime - stoptime;**

**process[job].remtime = process[job].remtime - (stoptime - currentwait);**

**//process[job].waittime = currentwait - process[job].arrtime; // - idletime**

**process[job].turntime = process[job].comptime - process[job].arrtime;**

**process[job].temptime = process[nextjob].arrtime - currentwait;**

**temp.comptime = process[job].temptime + currentwait;**

**currentwait += process[job].temptime;**

**}**

**//printf("\nFunct Chosen: %s",process[nextjob].pid);**

**//printf("\n%d : %d",currentwait, process[nextjob].arrtime);**

**if(process[nextjob].arrtime > temp.comptime) //finding idletime if any**

**idletime = process[nextjob].arrtime - temp.comptime;**

**//indicates the amt of idletime after process[i]**

**else**

**idletime = 0;**

**//printf("\n%d, %d",job, idletime);**

**currentwait+=idletime; //accounting waiting time to sync with existing idletimes**

**process[job].nextidle = idletime;**

**temp.nextidle = idletime;**

**queue[i++] = temp; //queueing for Gantt Chart**

**prevjob = job;**

**//printf("\n%s CurWait: %d Wait: %d Turn: %d Comp %d IDLE %d BUFF %d",process[job].pid, currentwait, process[job].waittime, process[job].turntime, process[job].comptime, process[job].nextidle, process[job].remtime);**

**}**

**chartGantt(queue,i);**

**return process;**

**}**

/\*

OUTPUT:

(base) vishakan@Legion:~/Desktop/Operating-Systems/Ex3 Job Scheduling$ gcc RoundRobin.c -o r

(base) vishakan@Legion:~/Desktop/Operating-Systems/Ex3 Job Scheduling$ ./r

CPU SCHEDULING ALGORITHMS

1. ROUND ROBIN

2. PRIORITY

0. EXIT

Your Option -> 1

ROUND ROBIN CPU SCHEDULER

Enter Time Quantum : 2

Enter the Number of Processes : 7

Enter the Process ID: p1

Enter the Arrival Time of p1: 2

Enter the CPU Burst Time of p1: 3

Enter the Process ID: p2

Enter the Arrival Time of p2: 4

Enter the CPU Burst Time of p2: 2

Enter the Process ID: p3

Enter the Arrival Time of p3: 5

Enter the CPU Burst Time of p3: 1

Enter the Process ID: p4

Enter the Arrival Time of p4: 7

Enter the CPU Burst Time of p4: 4

Enter the Process ID: p5

Enter the Arrival Time of p5: 9

Enter the CPU Burst Time of p5: 2

Enter the Process ID: p6

Enter the Arrival Time of p6: 15

Enter the CPU Burst Time of p6: 6

Enter the Process ID: p7

Enter the Arrival Time of p7: 16

Enter the CPU Burst Time of p7: 8

Gantt Chart:

--------------------------------------------------------------------------------------------------------------------------------

| IDLE | p1 | p2 | p1 | p3 | p4 | p5 | p4 | IDLE | p6 | p7 | p6 | p7 | p6 | p7 | p7 |

--------------------------------------------------------------------------------------------------------------------------------

0 2 4 6 7 8 10 12 14 15 17 19 21 23 25 27 29

----------------------------------------------------------------------------------

| P. ID | A. Time | B. Time | C. Time | T. Time | W. Time | R. Time |

----------------------------------------------------------------------------------

| p1 | 2 | 3 | 7 | 5 | 2 | 0 |

| p2 | 4 | 2 | 6 | 2 | 0 | 0 |

| p3 | 5 | 1 | 8 | 3 | 2 | 2 |

| p4 | 7 | 4 | 14 | 7 | 3 | 1 |

| p5 | 9 | 2 | 12 | 3 | 1 | 1 |

| p6 | 15 | 6 | 25 | 10 | 4 | 0 |

| p7 | 16 | 8 | 29 | 13 | 5 | 1 |

----------------------------------------------------------------------------------

Average Waiting Time : 2.43

Average Response Time : 0.71

----------------------------------------------------------------------------------

CPU SCHEDULING ALGORITHMS

1. ROUND ROBIN

2. PRIORITY

0. EXIT

Your Option -> 2

PRIORITY CPU SCHEDULER

1. PREEMPTIVE

2. NON-PREEMPTIVE

Your Option -> 1

Enter the Number of Processes : 7

Enter the Process ID: P1

Enter the Arrival Time of P1: 2

Enter the CPU Burst Time of P1: 3

Enter the Priority of P1: 4

Enter the Process ID: P2

Enter the Arrival Time of P2: 4

Enter the CPU Burst Time of P2: 2

Enter the Priority of P2: 4

Enter the Process ID: p3

Enter the Arrival Time of p3: 5

Enter the CPU Burst Time of p3: 1

Enter the Priority of p3: 1

Enter the Process ID: P4

Enter the Arrival Time of P4: 7

Enter the CPU Burst Time of P4: 4

Enter the Priority of P4: 2

Enter the Process ID: P5

Enter the Arrival Time of P5: 9

Enter the CPU Burst Time of P5: 2

Enter the Priority of P5: 7

Enter the Process ID: P6

Enter the Arrival Time of P6: 15

Enter the CPU Burst Time of P6: 6

Enter the Priority of P6: 6

Enter the Process ID: P7

Enter the Arrival Time of P7: 16

Enter the CPU Burst Time of P7: 8

Enter the Priority of P7: 3

Gantt Chart:

----------------------------------------------------------------------------------------

| IDLE | P1 | p3 | P2 | P4 | P2 | P5 | IDLE | P6 | P7 | P6 |

----------------------------------------------------------------------------------------

0 2 5 6 7 11 12 14 15 16 24 29

--------------------------------------------------------------------------------

| P. ID | A. Time | B. Time | C. Time | T. Time | W. Time | R. Time |

---------------------------------------------------------------------------------

| P1 | 2 | 3 | 5 | 3 | 0 | 0 |

| P2 | 4 | 2 | 12 | 8 | 6 | 2 |

| p3 | 5 | 1 | 6 | 1 | 0 | 0 |

| P4 | 7 | 4 | 11 | 4 | 0 | 0 |

| P5 | 9 | 2 | 14 | 5 | 3 | 3 |

| P6 | 15 | 6 | 29 | 14 | 8 | 0 |

| P7 | 16 | 8 | 24 | 8 | 0 | 0 |

--------------------------------------------------------------------------------

Average Waiting Time : 2.43

Average Response Time : 0.71

--------------------------------------------------------------------------------

CPU SCHEDULING ALGORITHMS

1. ROUND ROBIN

2. PRIORITY

0. EXIT

Your Option -> 2

PRIORITY CPU SCHEDULER

1. PREEMPTIVE

2. NON-PREEMPTIVE

Your Option -> 2

Enter the Number of Processes : 7

Enter the Process ID: P1

Enter the Arrival Time of P1: 2

Enter the CPU Burst Time of P1: 3

Enter the Priority of P1: 4

Enter the Process ID: P2

Enter the Arrival Time of P2: 4

Enter the CPU Burst Time of P2: 2

Enter the Priority of P2: 4

Enter the Process ID: P3

Enter the Arrival Time of P3: 5

Enter the CPU Burst Time of P3: 1

Enter the Priority of P3: 1

Enter the Process ID: P4

Enter the Arrival Time of P4: 7

Enter the CPU Burst Time of P4: 4

Enter the Priority of P4: 2

Enter the Process ID: P5

Enter the Arrival Time of P5: 9

Enter the CPU Burst Time of P5: 2

Enter the Priority of P5: 7

Enter the Process ID: P6

Enter the Arrival Time of P6: 15

Enter the CPU Burst Time of P6: 6

Enter the Priority of P6: 6

Enter the Process ID: P7

Enter the Arrival Time of P7: 16

Enter the CPU Burst Time of P7: 8

Enter the Priority of P7: 3

Gantt Chart:

------------------------------------------------------------------------

| IDLE | P1 | P3 | P2 | P4 | P5 | IDLE | P6 | P7 |

------------------------------------------------------------------------

0 2 5 6 8 12 14 15 21 29

----------------------------------------------------------------------------------

| P. ID | A. Time | B. Time | C. Time | T. Time | W. Time | R. Time |

----------------------------------------------------------------------------------

| P1 | 2 | 3 | 5 | 3 | 0 | 0 |

| P3 | 5 | 1 | 6 | 1 | 0 | 0 |

| P2 | 4 | 2 | 8 | 4 | 2 | 2 |

| P4 | 7 | 4 | 12 | 5 | 1 | 1 |

| P5 | 9 | 2 | 14 | 5 | 3 | 3 |

| P6 | 15 | 6 | 21 | 6 | 0 | 0 |

| P7 | 16 | 8 | 29 | 13 | 5 | 5 |

----------------------------------------------------------------------------------

Average Waiting Time : 1.57

Average Response Time : 1.57

----------------------------------------------------------------------------------

CPU SCHEDULING ALGORITHMS

1. ROUND ROBIN

2. PRIORITY

0. EXIT

Your Option -> 0

Thank You!

\*/

**QUEUE.H DEFINITION:**

**typedef struct{**

**char pid[10]; //process ID**

**int arrtime; //arrival time**

**int bursttime; //burst time**

**int resptime; //resp time**

**int waittime; //wait time**

**int turntime; //turnaround time**

**int comptime; //completion time**

**int nextidle; //idle time after completion**

**int remtime; //remaining time**

**int temptime; //current comp. time for PPP**

**int bufftime; //buffering time for PPP**

**int status; //completed or not**

**int queued; //indicates whether or not process is in queue**

**int prio; //process priority**

**}pInfo;**

**typedef struct{**

**int \*pid;**

**int front, rear;**

**int capacity, size;**

**}Queue;**

**void initQueue(Queue \*q, int cap){**

**q->front = q->rear = -1;**

**q->capacity = cap;**

**q->pid = (int \*)malloc(sizeof(int)\*cap);**

**q->size = 0;**

**}**

**int isEmpty(Queue \*q){**

**if(q->size==0)**

**return 1;**

**return 0;**

**}**

**int isFull(Queue \*q){**

**if(q->size==q->capacity)**

**return 1;**

**return 0;**

**}**

**void enQueue(Queue \*q, int p){**

**if(isFull(q))**

**printf("\nQueue is Full! Cannot Enqueue any further!\n");**

**else{**

**q->rear=(q->rear+1)%q->capacity;**

**q->pid[q->rear]=p;**

**q->size++;**

**}**

**if(q->front==-1)**

**q->front=0;**

**}**

**int deQueue(Queue \*q){**

**int pop;**

**pop = -1; //default, in case no ID returned**

**if(isEmpty(q)){**

**printf("\nQueue is Empty! Cannot Dequeue any further!\n");**

**return pop;**

**}**

**pop= q->pid[q->front];**

**if(q->rear==q->front){**

**q->front=q->rear=-1;**

**}**

**else**

**q->front=(q->front+1)%q->capacity;**

**q->size--;**

**return pop;**

**}**

**void displayQueue(Queue \*q){**

**int i=0;**

**int ptr=q->front;**

**if(isEmpty(q))**

**printf("\nQueue is Empty!");**

**else{**

**printf("\nQueue Elements:\n\t");**

**while(i<q->size){**

**printf("%d ",q->pid[ptr]);**

**i++;**

**ptr=(ptr+1)%q->capacity;**

**}**

**}**

**}**