##

Experiment No. 4

##

Author: Vidyut Chakrabarti
Semester/section: IV Sem/ B
Roll no.: 68
Date of execution: 19/02/24

##

Aim

To write and execute C programs to demonstrate different CPU scheduling algorithms.

Problem Statement

(A) Consider a scenario wherein N processes (4 ≤ N ≤7) are to be scheduled on a single processor. Write a menu-driven program in C to implement the following CPU scheduling algorithms

[1] First Come First Serve (common arrival & distinct arrival)

[2] Shortest Job First (non-preemptive)

[3] Shortest Remaining Time First

[4] Round-Robin

Note: Appropriately display the contents of data structures and process state parameters performing step-by-step process.

==========

Menu Driven C program: prac4_b468.c

==========

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>

int main(){
   int s;

printf("Enter 1 for FCFS scheduling.\nEnter 2 for SJF scheduling.\nEnter 3 for SRTF scheduling.\nEnter 4 for executing Round robin algorithm.\n");

  printf("Enter your choice: ");
  scanf("%d", &s);
```

```
  switch(s){

/** [1] Executing first come first serve. (b468_fcfs.c)**/

    case 1:
system("gcc b468_fcfs.c -o fcfs.out");
system("./fcfs.out");
printf("================================================================\n");
break;

/** [2] Executing shortest job first algorithm. (b468_sjf.c) **/

    case 2:
system("gcc b468_sjf.c -o sjf.out");
system("./sjf.out");
printf("================================================================\n");
break;

/** [3] Executes b468_srtf.c. **/

    case 3:
system("gcc b468_srtf.c -o srtf.out");
system("./srtf.out");
printf("================================================================\n");
break;

/** [4] EXECUTES ROUND ROBIN (b468_rr.c) **/

case 4:
system("gcc b468_rr.c -o rr.out");
system("./rr.out");
printf("================================================================\n");
break;

default:
      printf("Invalid input.");
      printf("================================================\n");
  }
  return 0;
 }

#############################################################################

/**b468_fcfs.c **/
/** [1] EXECUTE FCFS ALGORITHM. **/

# include<stdio.h>
# include<stdlib.h>

struct Process {
```

```
    int process_id;
    int arrival_time;
    int burst_time;
    int priority;
};

void fcfs(struct Process processes[], int n){
int min_at = 100;
int adder = n;
int selected[n] = {0};
int wt, i;
int tat = 0;
float avgtat = 0;
float avgwt = 0;
printf("process   BT  AT  WT   TAT\n");
while(adder>0){
min_at = 100;
for(i =0; i<n; i++){
  if(processes[i].arrival_time< min_at && selected[i] == 0){
     min_at = processes[i].arrival_time;
}}
for(i = 0; i<n; i++){
   if(processes[i].arrival_time == min_at){
   wt = tat;
   tat = tat + processes[i].burst_time;
   printf("%d         %d    %d   %d   %d\n",i+1, processes[i].burst_time,
processes[i].arrival_time,wt - processes[i].arrival_time, tat -
processes[i].arrival_time);
   avgtat+= tat - processes[i].arrival_time;
   avgwt+= wt - processes[i].arrival_time;
   selected[i] = 1;
   adder--;
}}}
printf("\navg wt: %.2f,  avg tat: %.2f \n", avgwt/n, avgtat/n);
}

int main(){
int n;
printf("Enter the number of processes: ");
 scanf("%d", &n);
 struct Process processes[n];
    // Input arrival time and burst time for each process
    for (int i = 0; i < n; i++) {
        processes[i].process_id = i + 1;
        printf("Enter arrival time for process %d: ", i + 1);
        scanf("%d", &processes[i].arrival_time);
        printf("Enter burst time for process %d: ", i + 1);
        scanf("%d", &processes[i].burst_time);
}
fcfs(processes, n);
```

```
return 0;
}
```

```
/** b468_sjf.c **/
/** [2] EXECUTE SHORTEST JOB FIRST SCHEDULING. **/

# include<stdio.h>
# include<stdlib.h>

struct Process {
    int process_id;
    int arrival_time;
    int burst_time;
    int priority;
};

void insertionSort(struct Process prc[], int n)
{
    int i, key, j;
    struct Process k;
    for (i = 1; i < n; i++) {
        key = prc[i].burst_time;
        k = prc[i];
        j = i - 1;
        while (j >= 0 && prc[j].burst_time > key)  {
            prc[j + 1] = prc[j];
            j = j - 1;
        }
        prc[j+1] = k;
}}

void sjf(struct Process process[], int n){

int i,j, sch;
int selected[20] = {0};
struct Process p_ready[n];
int t = 0;
int select = 0;
int wt;
float avgwt = 0.0;
float avgtat = 0.0;
int tat = 0;
printf("process   BT  AT  WT   TAT\n");
while(select!=n){
int prc = 0;
for(j = 0; j<n;j++){
if(process[j].arrival_time<=t && selected[j]!=1){
  p_ready[prc] = process[j];
```

```
  prc++;
} }
insertionSort(p_ready , prc);

int select_id;
sch = p_ready[0].process_id;
for(i = 0; i<n;i++){
if(process[i].process_id == sch){
select_id = i;
selected[i] = 1;
break;
}}
wt = tat;
tat = tat + process[select_id].burst_time;
printf("%d         %d    %d   %d   %d\n",sch,
process[select_id].burst_time, process[select_id].arrival_time,wt -
process[select_id].arrival_time, tat - process[select_id].arrival_time);
avgtat+= tat - process[select_id].arrival_time;
avgwt+= wt - process[select_id].arrival_time;
select++;
t+=process[select_id].burst_time;
}
printf("\navg wt: %.2f,  avg tat: %.2f \n", avgwt/n, avgtat/n);
}

int main(){
int n;
printf("Enter the number of processes: ");
 scanf("%d", &n);
 struct Process processes[n];
    // Input arrival time and burst time for each process
    for (int i = 0; i < n; i++) {
        processes[i].process_id = i + 1;
        printf("Enter arrival time for process %d: ", i + 1);
        scanf("%d", &processes[i].arrival_time);
        printf("Enter burst time for process %d: ", i + 1);
        scanf("%d", &processes[i].burst_time);
    }
sjf(processes, n);
return 0;
}

################################################################################

/** b468_srtf.c**/
/** [3] EXECUTES SHORTEST REMAINING TIME FIRST SCHEDULING. **/

#include<stdlib.h>
#include "queue.h"
```

```
// Structure to represent a process
struct Process {
    int process_id;
    int arrival_time;
    int burst_time;
    int remaining_time;
    int wt;
    int tat;
};

void insertionSort(struct Process arr[], int n)
{
    int i, key, j;
    struct Process k;
    for (i = 1; i < n; i++){
        key = arr[i].remaining_time;
        k = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j].remaining_time > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j+1] = k;
}}

void srtsched(struct Process processes[], int n){
int ct;
int total_time = 0;
int turnaround_time = 0;
int waiting_time = 0;

// Calculate the total time required for all processes
    for (int i = 0; i < n; i++) {
        total_time += processes[i].burst_time;
        processes[i].remaining_time = processes[i].burst_time;
    }
for(ct = 0; ct<= total_time; ct++){
struct Process cur_proces[n];
int j = 0;
for(int i = 0; i< n; i++){
if(processes[i].arrival_time <= ct && processes[i].remaining_time != 0){
cur_proces[j] = processes[i];
j++;
}}

insertionSort(cur_proces, j);

int proces = cur_proces[0].process_id;
for(int h = 0; h<n; h++){
```

```
if(proces == processes[h].process_id){
processes[h].remaining_time = processes[h].remaining_time -1;
if(processes[h].remaining_time == 0){
turnaround_time += ct+1 - processes[h].arrival_time;
waiting_time+= ct - processes[h].arrival_time - (processes[h].burst_time
1);
processes[h].wt = ct - processes[h].arrival_time –
(processes[h].burst_time - 1);
processes[h].tat = ct+1 - processes[h].arrival_time;
}}
}}
printf("\npid    arrival    burst    wait     tat\n");
for(int h = 0; h<n; h++){
printf("%d       %d         %d        %d       %d\n", h+1,
processes[h].arrival_time, processes[h].burst_time, processes[h].wt,
processes[h].tat);
}
printf("\ntotal turnaround time: %d\n", turnaround_time);
printf("total waiting time: %d\n", waiting_time);
printf("\naverage waiting time: %f", (float)waiting_time/n);
printf("\naverage turnaround time: %f\n", (float)turnaround_time/n);
}

int main(){
struct Queue cqu;
createQueue(&cqu, 20);
int n;
// Input the number of processes
 printf("Enter the number of processes: ");
 scanf("%d", &n);
 struct Process processes[n];
    // Input arrival time and burst time for each process
    for (int i = 0; i < n; i++) {
        processes[i].process_id = i + 1;
        printf("Enter arrival time for process %d: ", i + 1);
        scanf("%d", &processes[i].arrival_time);
        printf("Enter burst time for process %d: ", i + 1);
        scanf("%d", &processes[i].burst_time);
    }
srtsched(processes, n);
return 0;
}

#############################################################################

/** b468_rr.c **/
/** EXECUTES ROUND ROBIN **/

#include<stdio.h>
#include "queue.h"
```

```
struct ans{
int id;
int bt;
int at;
int wt;
int tat;
};

void rrsched(struct Process process[], int n, int q){
struct ans schd[20];
struct Queue cqu;
createQueue(&cqu, 20);
int i;
int done = 0;
printf("\n======= READY QUEUE VISUALIZATION =======\n\n");
for(i=0; i<n; i++){
if(process[i].arrival_time == 0){
enqueue(&cqu, process[i]);
process[i].selected = 1;
printf("%d====>", process[i].process_id);
}}
int ct = 0;
while(done!=n){
struct Process r_pr = dequeue(&cqu);
if(r_pr.remaining_time>q){
ct+=q;
int inter_id = r_pr.process_id;
int id;
for(int h =0; h<n; h++){
if(process[h].process_id== inter_id){
id = h;
}}
process[id].remaining_time -= q;
for(i = 0;i<n;i++){
if(process[i].arrival_time<=ct && process[i].remaining_time!=0 &&
process[i].selected !=1){
printf("%d====>", process[i].process_id);
enqueue(&cqu, process[i]);
process[i].selected = 1;
}}
enqueue(&cqu, process[id]);
printf("%d====>", process[id].process_id);
}
else{
ct += r_pr.remaining_time;
int inter_id = r_pr.process_id;
int id;
for(int h =0; h<n; h++){
if(process[h].process_id== inter_id){
id = h;
```

```
}}
process[id].remaining_time = 0;
process[id].tat = ct - process[id].arrival_time;
process[id].wt = ct - process[id].arrival_time - (process[id].burst_time);
struct ans a;
a.id = id+1;
a.bt = process[id].burst_time;
a.at = process[id].arrival_time;
a.tat = process[id].tat;
a.wt = process[id].wt;
schd[done] = a;
done +=1;
}}
printf("\n\n======Processes Scheduled=======\n\n");
printf("pid    bt    at    wt    tat\n");
for(i = 0;i<n;i++){
printf("%d     %d      %d      %d      %d\n", schd[i].id, schd[i].bt,
schd[i].at, schd[i].wt, schd[i].tat);
}}

int main(){
int n;
int q;
printf("\nExecuting Round Robin scheduling...\n");
// Input the number of processes
 printf("Enter the number of processes: ");
 scanf("%d", &n);
 printf("Enter the quantum: ");
 scanf("%d", &q);
 struct Process processes[n];
    // Input arrival time and burst time for each process
    for (int i = 0; i < n; i++) {
        processes[i].process_id = i + 1;
        printf("Enter arrival time for process %d: ", i + 1);
        scanf("%d", &processes[i].arrival_time);
        printf("Enter burst time for process %d: ", i + 1);
        scanf("%d", &processes[i].burst_time);
        processes[i].remaining_time = processes[i].burst_time;
        processes[i].selected = 0;
    }
rrsched(processes, n, q);
return 0;
}

################################################################################

/** queue.h (used in srtf and round robin) **/

#include<stdio.h>
```

```
struct Process {
    int process_id;
    int arrival_time;
    int burst_time;
    int remaining_time;
    int wt;
    int tat;
    int selected;
};

struct Queue {
struct Process q[30]; //define max as 30.
int CAP; //capacity
int front;
int back;
};

void createQueue(struct Queue *cqu, int capacity) {
cqu -> front = cqu -> back = -1;
cqu -> CAP = capacity;
}
int isEmpty(struct Queue cqu) {
if(cqu.front == -1) {
return 1;
}
return 0;
}
int isFull(struct Queue cqu) {
if((cqu.front == 0 && cqu.back == cqu.CAP) || (cqu.front == cqu.back + 1))
{
return 1;
}
return 0;
}
int enqueue(struct Queue *cqu,struct Process element) {
if(isFull(*cqu)) {
return -1;
}
cqu -> back = (cqu -> back + 1) % cqu -> CAP;
cqu->q[cqu->back]=element;
if(cqu ->front == -1) {
cqu -> front = 0;
}
return 1;
}
struct Process dequeue(struct Queue *cqu){
struct Process data;
struct Process null;
if(isEmpty(*cqu)){
printf("queue is empty");
```

```
return null;
}
data = cqu->q[cqu->front];
if(cqu->front==cqu->back){
cqu->front=cqu->back=-1;
}
else{
cqu->front=(cqu->front+1)%(cqu->CAP);
}
return data;
}
```

##

EXECUTION TRACE:

==========

[1] FIRST COME FIRST SERVE:

// bt, at, priority

Example: {{2, 0, 5}, {3, 2, 1}, {5, 3, 4}, {2, 1, 3}, {1, 4, 2}};

```
vidyut@vidyut-VirtualBox:~/Desktop/B_68/prac4$ ./prac4_b468.out
Enter 1 for FCFS scheduling.
Enter 2 for SJF scheduling.
Enter 3 for SRTF scheduling.
Enter 4 for executing Round robin algorithm.
Enter your choice: 1

Executing FCFS Scheduling...
Enter the number of processes: 5
Enter arrival time for process 1: 0
Enter burst time for process 1: 2
Enter arrival time for process 2: 2
Enter burst time for process 2: 3
Enter arrival time for process 3: 3
Enter burst time for process 3: 5
Enter arrival time for process 4: 1
Enter burst time for process 4: 2
Enter arrival time for process 5: 4
Enter burst time for process 5: 1

process   BT  AT  WT   TAT
1         2    0   0   2
4         2    1   1   3
2         3    2   2   5
3         5    3   4   9
5         1    4   8   9

avg wt: 3.00,  avg tat: 5.60
```

```
============================================================

[2] SHORTEST JOB FIRST:
// bt, at , priority
 Example: {{2, 0, 5}, {3, 2, 1}, {5, 3, 4}, {2, 1, 3}, {1, 0, 2}};

vidyut@vidyut-VirtualBox:~/Desktop/B_68/prac4$ ./prac4_b468.out
Enter 1 for FCFS scheduling.
Enter 2 for SJF scheduling.
Enter 3 for SRTF scheduling.
Enter 4 for executing Round robin algorithm.
Enter your choice: 2

Executing SJF Scheduling...
Enter the number of processes: 5
Enter arrival time for process 1: 0
Enter burst time for process 1: 2
Enter arrival time for process 2: 2
Enter burst time for process 2: 3
Enter arrival time for process 3: 3
Enter burst time for process 3: 5
Enter arrival time for process 4: 1
Enter burst time for process 4: 2
Enter arrival time for process 5: 0
Enter burst time for process 5: 1

process   BT  AT  WT   TAT
5         1    0   0   1
1         2    0   1   3
4         2    1   2   4
2         3    2   3   6
3         5    3   5   10

avg wt: 2.20,  avg tat: 4.80
============================================================

[3] SHORTEST REMAINING TIME FIRST:
// bt, at
 Example: {{10, 0}, {6, 2}, {3, 4}, {7, 6}}; //EXAM QUESTION(ut-1)

vidyut@vidyut-VirtualBox:~/Desktop/B_68/prac4$ ./prac4_b468.out
Enter 1 for FCFS scheduling.
Enter 2 for SJF scheduling.
Enter 3 for SRTF scheduling.
Enter 4 for executing Round robin algorithm.
Enter your choice: 3

Executing SRTF scheduling...
Enter the number of processes: 4
Enter arrival time for process 1: 0
Enter burst time for process 1: 10
```

```
Enter arrival time for process 2: 2
Enter burst time for process 2: 6
Enter arrival time for process 3: 4
Enter burst time for process 3: 3
Enter arrival time for process 4: 6
Enter burst time for process 4: 7

pid    arrival    burst    wait     tat
1       0         10       16       26
2       2         6        3       9
3       4         3        0       3
4       6         7        5       12

total turnaround time: 50
total waiting time: 24

average waiting time: 6.000000
average turnaround time: 12.500000

========================================================================
```

[4] ROUND ROBIN:
// bt, at, priority
Example: {{10, 0}, {6, 2}, {3, 4}, {7, 6}}; //EXAM QUESTION(ut-1)

```
vidyut@vidyut-VirtualBox:~/Desktop/B_68/prac4$ ./prac4_b468.out
Enter 1 for FCFS scheduling.
Enter 2 for SJF scheduling.
Enter 3 for SRTF scheduling.
Enter 4 for executing Round robin algorithm.
Enter your choice: 4

Executing Round Robin scheduling...
Enter the number of processes: 4
Enter the quantum: 4
Enter arrival time for process 1: 0
Enter burst time for process 1: 10
Enter arrival time for process 2: 2
Enter burst time for process 2: 6
Enter arrival time for process 3: 4
Enter burst time for process 3: 3
Enter arrival time for process 4: 6
Enter burst time for process 4: 7

====== READY QUEUE VISUALIZATION =======

1====>2====>3====>1====>4====>2====>1====>4====>

======Processes Scheduled=======
```

```
pid   bt    at      wt     tat
3     3      4      4       7
2     6      2      13      19
1     10     0      13      23
4     7      6      13      20

================================================================
Average tat: 17.25
Average wt: 10.75
================================================================
```