Assignment Report

On

Design of Operating Systems (CSE 4049)

Submitted by

Name	**: Saswat Mohanty**
Reg. No.	**: 1941012407**
Branch	**: CSE**
Semester	**: 5th**
Section	**: D**
Session	**: 2021-2022**
Admission Batch	**: 2019**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

FACULTY OF ENGINEERING & TECHNOLOGY (ITER)

SIKSHA 'O' ANUSANDHAN DEEMED TO BE UNIVERSITY

BHUBANESWAR, ODISHA – 751030

CONTENT

Assignment No.	Name of the Assignment	Remarks
1	Process management	
2	Process scheduling	
3	Process Synchronization	
4	Deadlocks and methods of handling deadlock	
5	Memory management Strategies	
6	End term project	

Assignment #No. 1

On

Design of Operating Systems (CSE 4049)

Submitted by

Name	**: Saswat Mohanty**
Reg. No.	**: 1941012407**
Semester	**: 5th**
Branch	**: CSE**
Section	**: D**
Session	**: 2021-2022**
Admission Batch	**: 2019**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

FACULTY OF ENGINEERING & TECHNOLOGY (ITER)

SIKSHA 'O' ANUSANDHAN DEEMED TO BE UNIVERSITY

BHUBANESWAR, ODISHA – 751030

P.S. Assignment - 1

Q1) Write a c program to create a child process using fork() system call. The child process will print the message "child" with its process identifier and then continue in an indefinite loop. The parent process will print the message "Parent" with its process identifier and then continue in an indefinite loop.

a) Run the program and trace the state of both process.

b) Terminate the child process. Then trace the state of processors.

c) Run the program and trace the state of both processes. Terminate the parent process. Then trace the state of processes.

d) Modify the program so that the parent process after displaying the message will wait for child process to complete its task. Again run the program and trace the state of both processes.

e) Terminate the child process. Then trace the state of processes.

Program :-

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main() {

    if (fork() == 0) {

        printf("child %d \n", getpid());

        while (1);

    }

    else {

        printf("Parent %d \n", getpid());
        wait(NULL);
        while (1); }
```

```
    return 0;
}
void a() {
    if (fork() == 0) {
        printf ("child : %d \n", getpid());
    }
    else
        printf ("Parent: %d \n", getpid());
}
void b() {
    if (fork() == 0) {
        printf ("child : %d \n", getpid());
        exit(0);
    }
    else
        printf ("Parent : %d \n", getpid());
}
void c() {
    if (fork() == 0) {
        printf ("child : %d \n", getpid());
    else {
        printf ("Parent: %d \n", getpid());
        exit(0);
    }
}
void d() {
    if (fork() == 0) {
        printf ("child: %d \n", getpid());
    }
    else {
```

```
        printf("Parent: %d \n", getpid());
        wait(NULL);
    }
}
void c(){
    if(fork() == 0) {
        printf("child: %d \n", getpid());
        exit(0);
    }
    else {
        printf("Parent: %d \n", getpid());
        wait(NULL);
    }
}
int main(){
    a();
    b();
    c();
    d();
    e();
    return 0;
}
```

Output :-

a) Parent : 29171
   child : 29172

b) Parent : 29218
   child: 29219

c) Parent : 29491

child : 29492

d) Parent : 29640

child : 29641

e) Parent : 29772

child : 29773

Q2) Trace the output of the following program segment :

a)
```
int main()
{
  if (fork() == 0)
    fprintf("1");
  else
    fprintf("2");
  fprintf("3");
  return 0;
}
```

output :- 2313

b)
```
int main()
{
  if (fork() == 0)
    fprintf("1");
  else
    fprintf("2");
  fprintf("3");
}
```

output :- 1323

timeout : the monitored command dumped core.

c) int main()

```
{
  pid_t pid;
  int i = 5;
  pid = fork();
  i = i + 1;
  if (pid == 0)
  {
    printf ("child : %d", i);
  }
  else
  {
    wait (NULL);
    printf ("Parent : %d", i);
  }
  return 0;
}
```

output :- child : 6
Parent : 6

d) int main()

```
{
  pid_t pid;
  int i = 5;
  pid = vfork ();
  i = i + 1;
  if (pid == 0)
  {
    printf ("child : %d", i);
    exit(0);
  }
  else
  {
    printf ("Parent : %d", i);
  }
  return 0;
}
```

output :- child : 6
Parent : 7

e)
```
int main()
{
  pid_t pid;
  int i = 5;
  pid = fork();
  if (pid == 0)
  {
    i = i + 1;
    printf ("child : %d", i);
  }
  else
  {
    wait (NULL);
    printf ("Parent : %d", i);
  }
  return 0;
}
```

output :- child : 6
Parent : 5

f)
```
int main()
{
  pid_t pid;
  int i = 5;
  pid = vfork();
  if (pid == 0)
  {
    i = i + 1;
    printf ("child : %d", i);
    exit(0);
  }
```

```
    else {
      printf("Parent: %d", i);
    }
    return 0;
}
```

output :- child : 6
Parent : 6

g) int main()

```
{
  int i = 5;
  if(fork() == 0)
  {
    printf("child: %d", i);
  }
  else
  {
    printf("Parent: %d", i);
  }
  return 0;
}
```

output :- Parent : 5
child : 5

h) int main()

```
{
  int i = 5;
  if(vfork() == 0)
  {
    printf("child: %d", i);
    exit(0);
  }
  else
  {
    printf("Parent: %d", i);
  }
  return 0;
}
```

output :- child : 5
Parent : 5

i) int main()

```
{
  if (fork() == 0)
  {
    printf("1");
  }
  else
  {
    wait(NULL);
    printf("2");
    printf("3");
  }
  return 0;
}
```

output :- 1 2 3

j) int main()

```
{
  if (vfork() == 0)
  {
    printf("1");
    exit(0);
  }
  else
  {
    printf("2");
    printf("3");
  }
  return 0;
}
```

output :- 1 2 3

k) int main()

```
{
  pid_t c1;
  int n = 10;
  c1 = fork();
```

```
if (c1 == 0)
{
  printf(" child \n");
   n = 20
   printf (" n = %d \n", n);
}
else
{
   wait (NULL);
   printf (" Parent \n");
   printf(" n = %d \n ",n);
}
return 0;
}
```

output :- child
n = 20
Parent
n = 10

l) int main()

```
{
  pid_t c1;
  int n = 10;
  c1 = vfork();
  if (c1 == 0)
  {
    printf (" child \n");
    n = 20;
    printf (" n = %d \n", n);
    exit(0);
  }
  else
  {
    printf (" Parent \n");
    printf (" n = %d \n", n);
  }
  return 0;
}
```

output :- child
n = 20
Parent
n = 20

m) int main()

```
{ int i = 5;
  fork();
  i = i + 1;
  fork();
  fprintf(stderr, "%d", i);
  return 0;
}
```

output :- 6 6 6 6

n) int main()

```
{
  pid_t pid;
  int i = 5;
  pid = vfork();
  if (pid == 0)
  {
    printf("child: %d", i);
    exit(0);
  }
  else
  {
    i = i + 1;
    printf("Parent: %d", i);
  }
  return 0;
}
```

output :- child : 5
Parent : 6

o) int main()

```
{
  int i = 5;
  if (fork() == 0)
    i = i + 1;
```

```
else
    i = i - 1;
    fprintf (stderr, "%d", i);
    return 0;
}
```

output :- 46

b)
```
int main()
{
    int i = 5;
    if (vfork() == 0)
    {
        i = i + 1;
        _exit (0);
    }
    else
        i = i + 1;
    fprintf (stderr, "%d", i);
    return 0;
}
```

output :- 5

c)
```
int main()
{
    int j, i = 5;
    for (j = 1; j < 3; j++)
    {
        if (fork() == 0)
        {
            i = i + 1;
            break;
        }
        else
            wait (NULL);
    } fprintf (stderr, "%d", i); return 0;
}
```

output :- 6 6 5

g) int main()
{
int j, i = 5
for (j = 1; j < 3; j++)
{
if (fork() != 0)
{ i = i + 1;
break;
}
}
fprintf (stderr, "%d", i);
return 0;
}

output :- 4 4 5

h) int main()
{
if (fork() == 0)
if (fork())
printf ("a \n");
return 0;
}

output :- 1

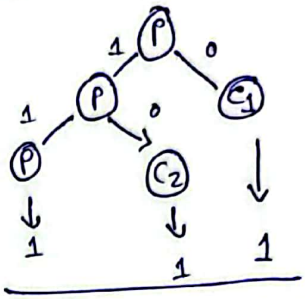
i) void fun1 () {
fork();
fork();
printf ("a \n");
}
int main() {
fun1 ();
printf ("a \n");
return 0;
}

output :-
1
1
1
1
1
1
1
1
11

Q3) Trace the following program segment & determine how many processes are created. Draw a graph that shows how the processes are related.

a)
```
int main()
{
    if(fork() && fork());
    printf("1");
    return 0;
}
```

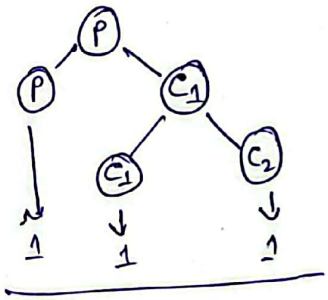
output :- 1 1 1



Print (111) => O/P = 111

b)
```
int main()
{
    if (fork() || fork());
    printf("1");
    return 0;
}
```

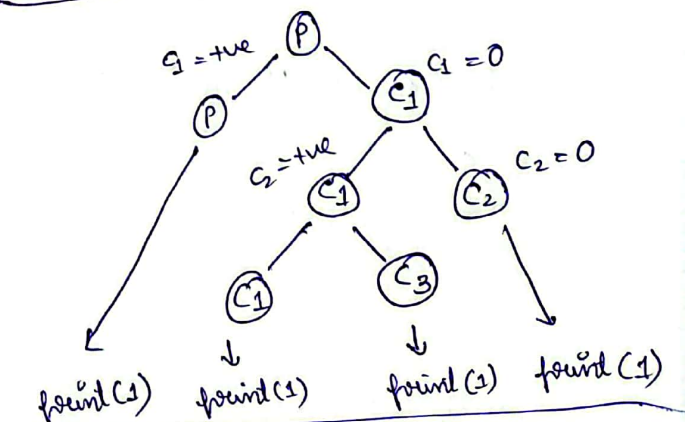
output :- 1 1 1



Print (111) => O/P = 111

c)
```
int main()
{
    pid_t c1, c2;
    c2 = 0;
    c1 = fork();
    if (c1 == 0)
        c2 = fork();
    if (c2 > 0)
        fork();
    printf("1");
    return 0;
}
```

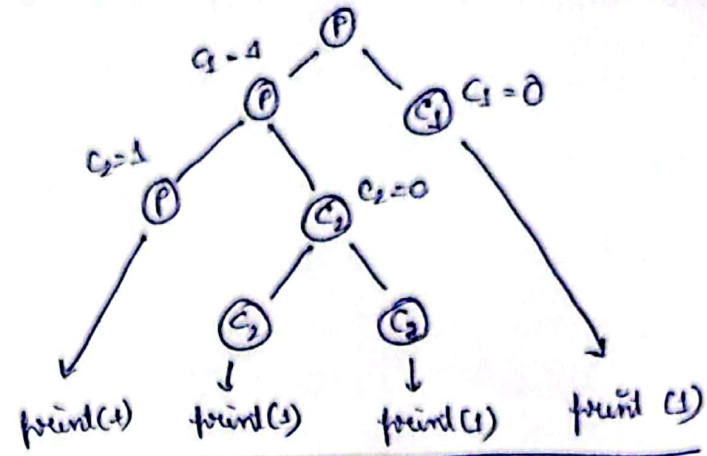
output :- 1 1 1 1



O/P = 1 1 1 1

d) int main()
{
  pid_t c1=1, c2=1;
  c1 = fork();
  if (c1 != 0)
    c2 = fork();
  if (c2 == 0)
    fork();
  printf("1");
  return 0;
}

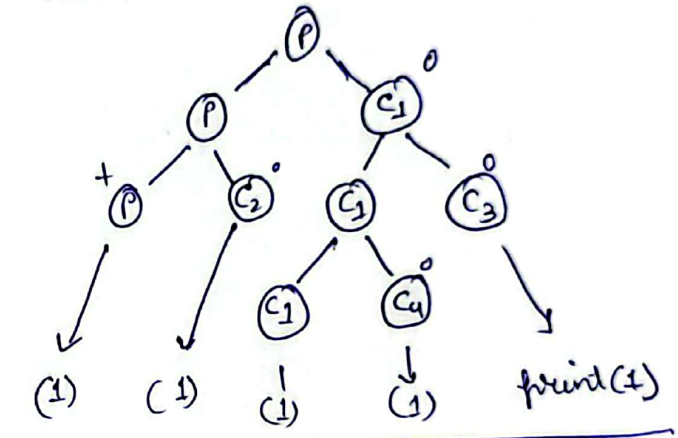
output :- 1 1 1 1



O/P = 1 1 1 1

e) int main()
{
  if (fork() || fork())
    fork();
  printf("1");
  return 0;
}

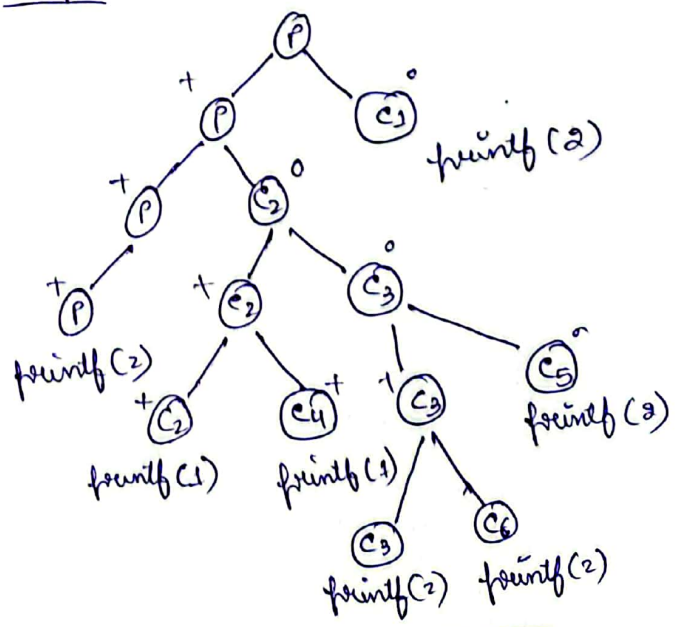
output :- 1 1 1 1 1



O/P = 1 1 1 1 1

f) int main()
{
  if (fork() && (!fork()))
  {
    if (fork() || fork())
    {
      fork();
    }
  }
  printf("2");
  return 0;
}

output :- 2 2 2 2 2 2 2



O/P = 2 2 2 2 2 2 2

Assignment #No. 2

On

Design of Operating Systems (CSE 4049)

Submitted by

Name	**: Saswat Mohanty**
Reg. No.	**: 1941012407**
Semester	**: 5th**
Branch	**: CSE**
Section	**: D**
Session	**: 2021-2022**
Admission Batch	**: 2019**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

FACULTY OF ENGINEERING & TECHNOLOGY (ITER)

SIKSHA 'O' ANUSANDHAN DEEMED TO BE UNIVERSITY

BHUBANESWAR, ODISHA – 751030

P.S. Assignment - 2

Q1) Consider the set of processes with arrival time (in milliseconds), CPU burst time (in milliseconds), and priority (low number implies high priority) as shown below:

Process	Arrival time	Burst time	Priority
P_1	0	10	3
P2	0	1	1
P3	0	2	3
P4	0	1	4
P5	0	5	2

a) Find the average turnaround time, average waiting time and average response time for each of the following scheduling algorithms along with their Gantt chart illustrating the execution of these processes:

> FCFS, (consider the order specified in the table

> SJF (Non preemptive)

> SRTF

> Non preemptive, Priority based scheduling,

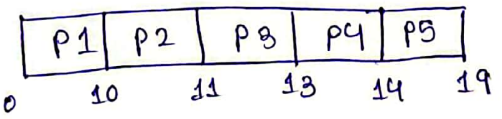
> Preemptive Priority based scheduling

> Round robin scheduling (quantum = 4 ms)

b) Which of the algorithms result in the minimum average waiting time over all processes?

(Ans) a) FCFS :-

Gantt chart →



Process	TAT	WT	RT
P1	10	0	0
P2	11	10	10
P3	13	11	11
P4	14	13	13
P5	19	14	14

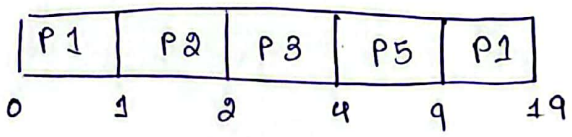
Avg. TAT = 67/5 = 13.4 ms

avg. WT = 48/5 = 9.6 ms

avg. RT = 48/5 = 9.6 ms

SJF (Non preemptive) :-

Gantt chart →



Process	TAT	WT	RT
P1	19	9	9
P2	1	0	0
P3	4	2	2
P4	2	1	1
P5	9	4	4

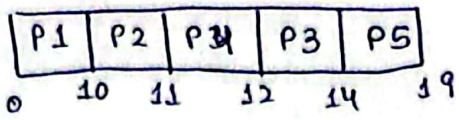
avg. TAT = 35/5 = 7 ms

avg. WT = 16/5 = 3.2 ms

avg. RT = 16/5 = 3.2 ms

SRTF :-

Gantt chart →



Process	WT	TAT	RT
P1	0	10	0
P2	10	11	10
P3	12	14	12
P4	11	12	11
P5	14	19	14

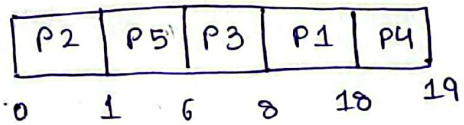
avg. WT = 47/5 = 9.4 ms

avg. TAT = 66/5 = 13.2 ms

avg. RT = 9.4 ms

Non-preemptive, Priority based scheduling :-

Gantt chard →



Process	TAT	WT	RT
P1	18	8	8
P2	1	0	0
P3	8	6	6
P4	19	18	18
P5	6	1	1

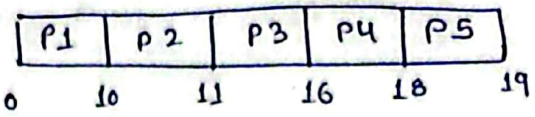
avg. TAT = 52/5 = 10.4 ms

avg. WT = 33/5 = 6.6 ms

avg. RT = 33/5 = 6.6 ms

Preemptive, Priority based scheduling :-

Gantt chart →



Process	TAT	WT	RT
P1	10	0	0
P2	11	10	10
P3	18	16	16
P4	19	18	18
P5	16	11	11

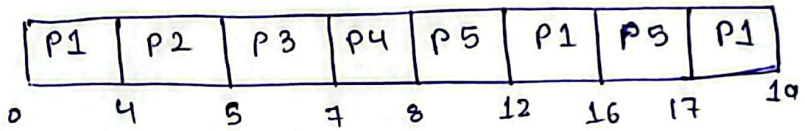
avg. TAT = 14.8 ms

avg. WT = 11 ms

avg. RT = 11 ms

Round robin scheduling (quantum = 5ms) :-

Gantt chart →



Process	BT	TAT	WT	RT
P1	~~10~~ 2	19	9	9
P2	~~10~~ 0	5	4	4
P3	~~8~~ 0	7	5	5
P4	~~7~~ 0	8	7	7
P5	~~5~~ 1	17	12	12

avg TAT = 11.2 ms

avg WT = 7.4 ms

avg. RT = 7.4 ms

b) SJF has the minimum average waiting time over all processes.

Q3) Consider the set of processes with arrival time (in milliseconds), CPU burst time (in milliseconds), and priority (low number implies high priority) as shown below.

Process	Arrival time	Burst time	Priority
P1	0	4	3
P2	0	2	1
P3	1	3	2
P4	2	2	4

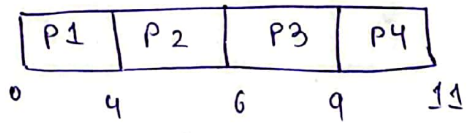
a) Find the average turnaround time, average waiting time and average response time for each of the following scheduling algorithm along with their gantt chart illustrating the execution of these processes:

- FCFS, (consider the order specified in the table)
- SJF (Non preemptive)
- SRTF
- Non preemptive, Priority based scheduling
- Preemptive priority based scheduling.
- Round robin scheduling (quantum = 2 ms)

b) which of the algorithms results in the minimum average waiting time over all processes?

(Ans) a) FCFS :-

gantt chart →



Process	TAT	WT	RT
P1	4	0	0
P2	6	4	4
P3	8	5	5
P4	9	7	7

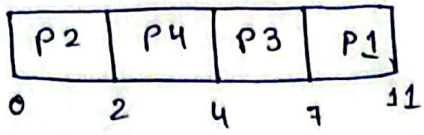
avg. TAT = 6.75 ms

avg. WT = 4 ms

avg. RT = 4 ms

SJF (non-preemptive) :-

Gantt chart →

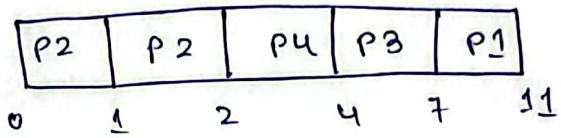


Process	TAT	WT	RT
P1	11	7	7
P2	2	0	0
P3	6	3	3
P4	2	0	0

avg TAT = 5.25ms
avg. WT = 2.5ms
avg. RT = 2.5ms

SRTF :-

Gantt chart →

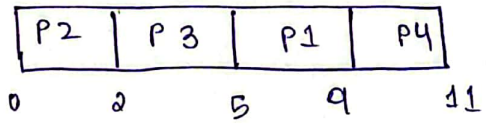


Process	BT	TAT	WT	RT
P1	~~4~~ 0	11	7	7
P2	~~2~~ 0	2	0	0
P3	~~6~~ 0	6	3	3
P4	~~2~~ 0	2	0	0

avg TAT = 5.25ms
avg WT = 2.5ms
avg RT = 2.5ms

Non-preemptive, priority based scheduling :-

Gantt chart →

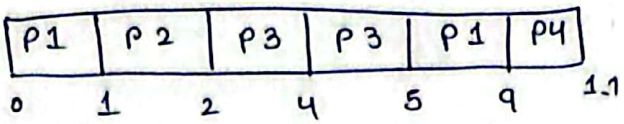


Process	CT	TAT	WT	RT
P1	9	9	5	5
P2	2	2	0	0
P3	5	4	1	1
P4	11	9	7	7

avg TAT = 6ms
avg WT = 3.25ms
avg. RT = 3.25ms

Preemptive, Priority based scheduling :-

Gantt chart →

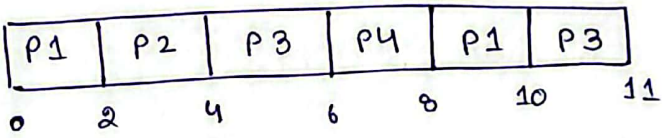


Process	BT	CT	TAT	WT	RT
P1	~~4~~ 0	9	9	5	5
P2	~~2~~ ~~1~~ 0	2	2	0	0
P3	~~3~~ 0	5	4	1	1
P4	~~1~~ 0	11	9	7	7

avg TAT = 6ms
avg WT = 3.25ms
avg RT = 3.25ms

Round robin scheduling :-

Gantt chart →



Process	BT	CT	TAT	WT	RT
P1	~~4~~ ~~2~~ 0	10	10	6	0
P2	~~2~~ 0	4	4	2	2
P3	~~3~~ ~~1~~ 0	11	10	7	3
P4	~~1~~ 0	8	6	4	4

avg TAT = 7.5ms
avg WT = 4.75ms
avg RT = 2.25ms

b) SJF & SRTF scheduling algorithms results in the minimum average waiting time over all processes.

Q3) consider the set of processes with arrival time (in miliseconds), CPU burst time (in miliseconds), and priority (high number implies high priority) as shown below :

Process	Arrival time	Burst time	Priority
P1	0	11	1
P2	0	8	0
P3	12	2	3
P4	2	6	2
P5	9	16	4

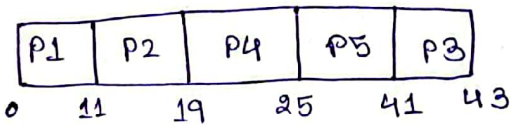
a) find the average turnaround time, average waiting time and average response time for each of the following scheduling algorithm along with their Gantt chart illustrating the execution of these processes:

- FCFS, (consider the order specified in the table)
- SJF (Non preemptive)
- SRTF
- Non preemptive, Priority based scheduling
- Preemptive, Priority based scheduling
- Round robin scheduling (quantum = 5ms)

b) which of the algorithms results in the minimum average waiting time over all process?

(Ans) a) FCFS:-

Gantt chart →



Process	CT	TAT	WT	RT
P1	11	11	0	0
P2	19	19	11	11
P3	43	31	29	29
P4	25	23	17	17
P5	41	32	16	16

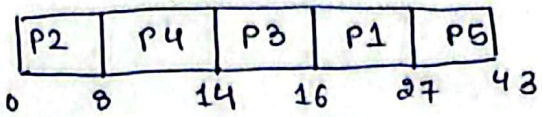
avg TAT = 23.2 ms

avg WT = 14.6 ms

avg RT = 14.6 ms

SJF (Non-preemptive) :-

Gantt chart →



Process	CT	TAT	WT	RT
P1	27	27	16	16
P2	8	8	0	0
P3	16	4	2	2
P4	14	12	6	6
P5	43	34	18	18

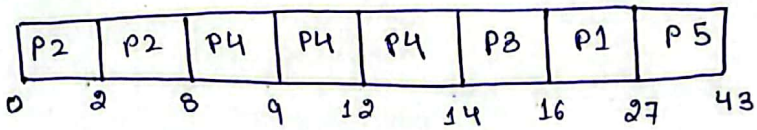
avg TAT = 17 ms

avg WT = 8.4 ms

avg RT = 8.4 ms

SRTF :-

Gantt chart →



Process	BT	CT	TAT	WT	RT
P1	~~11~~ 0	27	27	16	16
P2	~~8~~ ~~6~~ 0	8	8	0	0
P3	~~2~~ 0	16	4	2	2
P4	~~8~~ ~~8~~ ~~2~~ 0	14	12	6	6
P5	~~18~~ 0	43	34	18	18

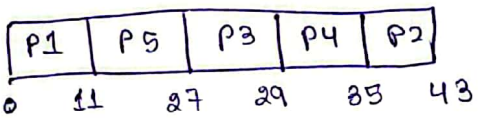
avg TAT = 17 ms

avg WT = 8.4 ms

avg RT = 8.4 ms

Non-preemptive priority based scheduling :-

Gantt chart →



Process	CT	TAT	WT	RT
P1	11	11	0	0
P2	43	43	35	35
P3	29	17	15	15
P4	35	33	27	27
P5	27	18	2	2

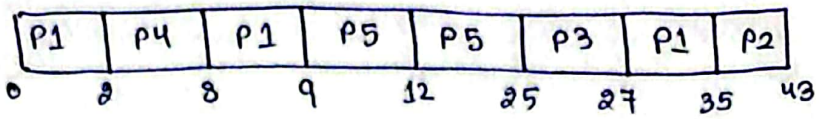
avg TAT = 24.4 ms

avg WT = 15.8 ms

avg RT = 15.8 ms

Preemptive priority based scheduling :-

Gantt chart →

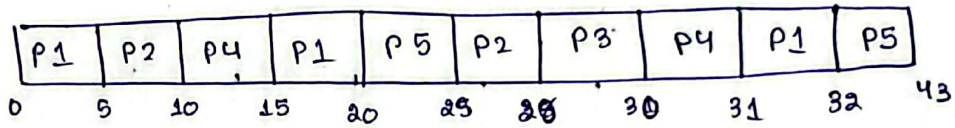


Process	CT	TAT	WT	RT
P1	35	35	24	0
P2	43	43	35	35
P3	27	15	13	13
P4	8	6	0	0
P5	25	16	0	0

avg TAT = 23ms
avg WT = 14.4ms
avg RT = 9.6ms

Round Robin scheduling :-

gantt chart →



Process	CT	TAT	WT	RT
P1	32	32	21	0
P2	28	28	20	5
P3	30	18	16	28
P4	31	29	23	10
P5	43	34	18	20

avg TAT = 28.8ms
avg WT = 19.6ms
avg RT = 12.6ms

b) Both SRTF & SJF have the minimum average waiting time over all processes.

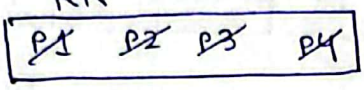
Q4) consider a multilevel feedback queue scheduling (MFLBQ) with three queues q1, q2 and q3. q1 and q2 use round-robin algorithm with time quantum equal to 3 and 5 miliseconds respectively. q3 uses first-come first-serve algorithm. Assume the arrival time of all processes as 0. A process entering the ready queue will put in queue 0. Processes in queue q1, q2 will be denoted to lower priority queue, if not

completed on specified time quantum. find the average waiting time. (A.W.T) and average turnaround time (A.T.A.T) for executing the following process ?

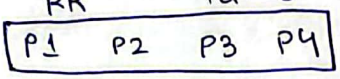
Processes	Burst time
P1	8
P2	22
P3	4
P4	12

(Ans)

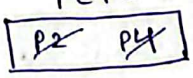
Q1 RR TQ = 3



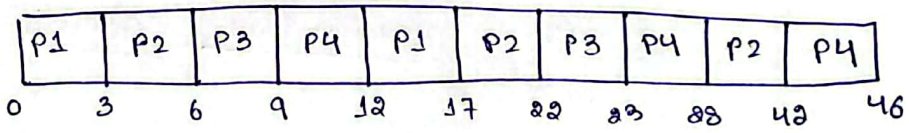
Q2 RR TQ = 5



Q3 FCFS



Gantt chart →



Process	AT	BT	CT	TAT	WT	RT
P1	0	8	17	17	9	0
P2	0	22	42	42	20	3
P3	0	4	23	23	19	6
P4	0	12	46	46	34	9

avg TAT = 29.5 ms

avg WT = 20.5 ms

avg RT = 4.5 ms

Q3) consider three CPU-intensive processes, which require 10, 20 and 30 time units and arrive at times 0, 2 and 6, respectively. How many

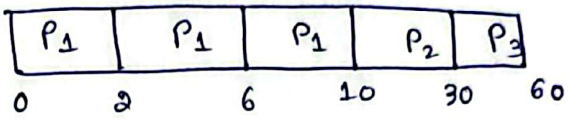
context switches are needed if the operating system implements a shortest remaining time first scheduling algorithm? Do not count the context switches at time zero and at the end..

(Ans)

Process	BT	AT
P1	10	0
P2	20	2
P3	30	6

SRTF (shortest remaining time first scheduling)

Gantt chart →



only two context switches are required i.e. P_1 to P_2 and P_2 to P_3

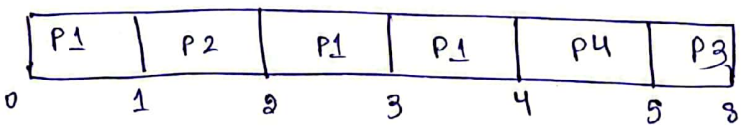
Q6) consider the following four processes with arrival times (in milliseconds) and their length of CPU bursts (in milliseconds) as shown below:

Process	Arrival time	Burst time
P1	0	3
P2	1	1
P3	3	3
P4	4	x

find the value of x, such that the average waiting time of the process is 1 millisecond, if the processes execute on a single processor using SRTF scheduling.

(Ans) Let x = 1

Gantt chart



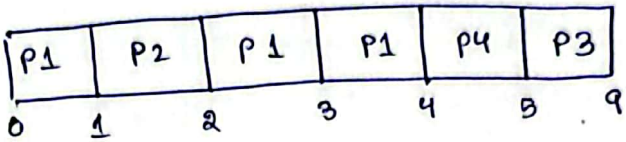
Process	AT	BT	CT	TAT	WT	RT
P1	0	3	4	4	1	0
P2	1	1	2	1	0	0
P3	3	3	8	5	2	2
P4	4	x=1	5	1	0	0

avg = $\frac{3}{4}$

= 0.75 ms

Let x = 2

Gantt chart →



Process	AT	BT	CT	TAT	WT	RT
P1	0	3	4	4	1	0
P2	1	1	2	1	0	0
P3	3	3	9	6	3	3
P4	4	x=2	6	2	0	0

avg = $\frac{4}{4}$

= 1 ms

when the value of x is 2, the average waiting time of the process is 1 millisecond.

Q7) How the base time quantum is related to the static priority? Find the base time quantum for the processes having highest, lowest and default static priority

(Ans) Base time quantum (in milliseconds) = $\begin{cases} (140 - \text{static priority}) \times 20 & \text{if } SP < 120 \\ (140 - \text{static priority}) \times 5 & \text{if } SP \geq 120 \end{cases}$

Base time quantum for highest priority

HP = 100

BTQ = (140 - (static priority)) × 20

= (140 - 100) × 20

= 40 × 20 = 800 ms

Base time quantum for lowest priority

LP = 139

BTQ = (140 - SP) × 5

= (140 - 139) × 5 = 5 ms

Base time quantum for default static priority

D.P. = 120

BTQ = (140 - 120) × 5

= 20 × 5 = 100 ms

Q8) what will be the dynamic priority of the following process?

Process	static priority	average sleep time
P1	110	250 ms
P2	120	700 ms
P3	132	15

(Ans)

Process	static priority	avg. sleep time	Bonus
P1	110	250 ms	2
P2	120	700 ms	7
P3	132	15	10

Dynamic priority,

P1 = max (100, min (static priority - bonus + 5, 139)

= max(100, min(110-8+5, 139))

= max(100, min(113, 139))

= max(100, 113)

$P_1 = 113$

P_2 = max(100, min(static priority - bonus + 5, 139))

= max(100, min(120 - 7 + 5, 139))

= max(100, min(118, 139))

= max(100, 118)

$P_2 = 118$

P_3 = max(100, min(132 - 10 + 5, 139))

= max(100, min(127, 139))

= max(100, 127)

$P_3 = 127$

Q9) What will be the minimum average sleep time for a process, so that the following processes will be considered as an interactive process by the scheduler in Linux system?

Process	static priority
P1	130
P2	108
P3	124
P4	132.

(Ans) A process is interactive if it satisfies the following condition.

bonus - 5 ≥ static priority / 4 - 28

P_1

bonus - 5 $\geq \frac{130}{4} - 28$

$\Rightarrow$ bonus $\geq \frac{130}{4} - 28 + 5$

$\Rightarrow$ bonus $\geq 33 - 23$

$\Rightarrow$ bonus ≥ 10

avg sleep time of P_1 is 1 sec

P_2

bonus $\geq \frac{108}{4} - 23$

$\Rightarrow$ bonus $\geq 27 - 23$

$\Rightarrow$ bonus ≥ 4

Average sleep time of P_2 is $\geq 400 < 500$ msec.

P_3

bonus $\geq \frac{124}{4} - 23$

$\Rightarrow$ bonus $\geq 31 - 23$

$\Rightarrow$ bonus ≥ 8

Average sleep time of P_3 is $\geq 800 < 900$ msec

P_4

bonus $\geq \frac{132}{4} - 23$

$\Rightarrow$ bonus $\geq 33 - 23$

$\Rightarrow$ bonus $\geq$ 10

So, avg. sleep time of P_4 is 1 sec.

Q10) which of the following process can not be considered as an interactive process scheduler in Linux system ?

Process	static priority
P1	136
P2	104
P3	116

(Ans) A process is interactive when the bonus is between 1 to 10.

P1

$$\text{bonus} - 5 \geq \frac{\text{static priority}}{4} - 28$$

$$\Rightarrow \text{bonus} \geq \frac{136}{4} - 28 + 5$$

$$\Rightarrow \text{bonus} \geq \frac{136}{4} - 23 \geq 11$$

$\therefore$ P_1 is not an interactive process.

P2

$$\text{bonus} \geq \frac{104}{4} - 28 + 5$$

$$\Rightarrow \text{bonus} \geq 26 - 23$$

$$\Rightarrow \text{bonus} \geq 3$$

$\therefore$ P_2 is an interactive process.

P_3

bones $\geq \frac{116}{4} - 28+5$

$\Rightarrow$ bones $\geq 29-23$

$\Rightarrow$ bones ≥ 6

$\therefore$ P_3 is an interactive process.

Assignment #No. 3

On

Design of Operating Systems (CSE 4049)

Submitted by

Name	**: Saswat Mohanty**
Reg. No.	**: 1941012407**
Semester	**: 5th**
Branch	**: CSE**
Section	**: D**
Session	**: 2021-2022**
Admission Batch	**: 2019**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

FACULTY OF ENGINEERING & TECHNOLOGY (ITER)

SIKSHA 'O' ANUSANDHAN DEEMED TO BE UNIVERSITY

BHUBANESWAR, ODISHA – 751030

P.S. Assignment - 3

Q1) A counting semaphores S is initialized to 10. 8 wait operations followed by 6 signal operations are carried out. what is the final value of the semaphores S?

(Ans) Initially, we have semaphore value = 10

after 8 wait operations, we get the semaphore value as = 10 - 8 = 2

now, we perform 6V operation, here semaphore value will be =

= 2 + 6 = 8

Q2) Two concurrent processes P and Q are accessing their critical sections by using Boolean variables S and T as follows:

```
Process P                          Process Q
while (true)                       while (true)
{                                  {
  // Entry section ?                   // Entry section ?
        Print (1);                         Print(0);
        Print (1);                         Print(0);
  // Exit section ?                    // Exit section ?
}                                  }
```

compare the entry section and exit section of Process P and Q with suitable semaphore operations using the two Boolean semaphores S and T. Also suggest the initial values of S and T such that execution of the processes will print the sequences 00110011....

(Ans) initial value S = 0, T = 1

Process P	Process Q

while (true)
{
wait (S);
Print (1);
Print (1);
signal (T);
}
	while (true)
	{
	wait (T);
	Print (0);
	Print (0);
	signal (S);
	}

Q3) Two concurrent processes P and Q are executing the following instructions. Synchronize the execution of P and Q with suitable semaphore operations and provide initialization of the semaphore values, so that the final outcome will be in the following order:-

a) 1 3 2 4 b) 3 1 2 4

Process P	Process Q
print (1);	print (3);
print (2);	print (4);

(Ans) a) S = 1, T = 0

process P	process Q
wait (S);	wait (T);
print (1);	print (3);
signal (T);	signal (S);
wait (S);	wait (T);
print (2);	print (4);
signal (T);	signal (S);

b) S = 0 , T = 1

Process P	Process Q
wait (S);	wait (T);
print (1);	print (3);
print (2);	signal (S);
signal (T);	wait (T);
	print (4);
	signal (S);

Q4) Assume the following 3 concurrent processes that use 3 binary semaphores S0, S1, S2 initialized as S0 = 1 , S1 = 0, S2 = 0. How many maximum and minimum number of times will process P0 print '0'? Justify your answer

Process P0	Process P1	Process P2
while (true) {	wait(S1);	wait(S2);
wait (S0);	signal (S0);	signal (S0);
print (0);		
signal (S1);		
signal (S2);		
}		

(Ans) minimum = 2

maximum = 3

Q5) Let 4 concurrent processes P1, P2, P3, P4 are accessing their critical sections by using Boolean semaphores S1, S2, S3 and S4. Write the entry section and exit section of all processes using the semaphores with suitable initialization, such that P1 will complete its critical section before P2 and P3, P2 and P3 will complete their critical section in any order before P4.

(Ans) $S_1 = 0$. $S_2 = 0$ $S_3 = 0$ $S_4 = 0$

P_1	P_2	P_3	P_4
wait (s1);	wait(s2);	wait (s3);	wait (s4);
CS	CS	CS	CS
signal (s2);	signal (s3);	signal (s4);	signal (s1);

OR

P_1	P_2	P_3	P_4
wait (s1);	wait (s2);	wait (s3);	wait (s4);
CS	CS	CS	CS
signal (s3);	signal (s4);	signal (s2);	signal (s1);

06) Assume that val is an atomic integer in a Linux system. What is the value of val after the following operations have been completed?

```
atomic_set (&val, 10);
atomic_sub (8, &val);
atomic_inc (&val);
atomic_inc (&val);
atomic_add (6, &val);
atomic_sub (3, &val);
```

(Ans) As given in the question we will find the value of val,

atomic_set (&val, 10);

→ 10

atomic_sub (8, &val);

→ 10 − 8 = 2

atomic_inc(&val);

→ 2 + 1 = 3

atomic_inc (&val);

→ 3 + 1 = 4

atomic_add (6, &val);

→ 6+4 = 10

atomic_sub (3, &val);

→ 10 - 3 = 7

Q7) Define the compare-and-swap hardware instruction. Specify a solution to critical section problem using compare_and_swap instruction and explain how the solution will satisfy all the three requirements.

(Ans) boolean CompareAndSwap (boolean *target, boolean expected, boolean new)

```
{
    boolean rv = *target;
    if (*target == expected)
        *target = new;
    return rv;
}
```

A solution to critical section problem using compare_and_swap instruction which satisfies all three requirements.

```
do
{
    waiting[i] = T
        key = T
    while (key == T && waiting[i] == T)
        key = (&lock, F, T);
    waiting[i] = F;

    j = (i+1) % n;
    while (j != i && waiting[j] == F).

        j = (j+1) % n;
    if (i == j)
        lock = False;
```

```
        else
            waiting[j] = False.
                    RS
    } while (TRUE);
```

Q8) write a monitor solution for the bounded buffer producer consumer problem.

(Ans) Monitor PC

```
{
  int c;
  condition full, empty;
  void put_item (item p)
  {
    if (c == N)
        full.wait();
    insert (item P);
    c = c+1;
    if (c == 1)
        empty.signal(1);
  }
  void get_item()
  {
    if (c == 0)
      empty.wait();
    Remove (item P);        // item P = Buffer [c];
    c = c-1;                //          out = (out+1)/N
    if (c == N-1)
        full.signal();
  }
  initialization code()
  {
    int c = 0;
  }
}
```

```
Producer ()
{
  while (1)
  {
     producer (item p);
      PC. put_item (item p);
   }
}
consumer()
{
   while (1)
    {
       PC. get_item ();
        consumer(item p);
   }
}
```

09) write a solution using semaphores for a Reader's Writers problem in which writer has higher priority than reader. once a writer is ready, that writer performs its write as soon as possible. In other words, if a writer is waiting to access the object, no new readers may start reading.

(Ans)

```
semaphore mutex = 1
int AW = 0
int AR = 0
int WR = 0
int WW = 0
int OKread = 0
int OKwrite = 0
```

Reader process	Writer process

Reader process()
{
while(1)
{
	Writer process()
	{
	while (1)
	{

```
wait (Mutex);
if (AW + WW == 0)
{
   signal (OKread);
    AR = AR + 1;
}
else
    WR = WR + 1
signal (Mutex);
wait (OK read);
// Reading the database
// Exit code
 wait (Mutex);
 AR = AR - 1;
 if (AR == 0 && WW > 0)
 {
   signal (OKwrite);
   AW = AW + 1;
    WW = WW - 1;
 }
 signal (Mutex);
}
```

```
wait (Mutex);
if (AR + AW == 0)
{
   signal (OKwrite);
   AW = AW + 1;
}
else
   WW = WW + 1;
signal (Mutex);
wait (OK write);
// write the database
wait (Mutex);
AW = AW - 1;
if (WW > 0)
{
   signal (OKwrite);
   AW = AW + 1;
    WW = WW - 1;
}
else
{
   while (WR > 0)
   {
      signal (OKread);
      AR = AR + 1;
      WR = WR - 1;
   }
   signal (Mutex);
}
}
```

Q10) The sleeping - Barber Problem. A barbershop consists of a waiting room with n chairs and a barber room with one barber chair. If there are no customers ... to be served, the barber goes to

sleep. If a customer enters the barbershop and all chairs are occupied, then the customer leaves the shop. If the barber is busy but chairs are available, then the customer sits in one of the free chairs. If the barber is asleep, the customer wakes up the barber.

a) write a solution using semaphores to coordinate the barber & the customers.

b) write a solution using monitor to coordinate the barber & the customer.

(Ans) Barber = 0

cus = 0

FC = N

M = 1

Barber code

```
while (1)
{
    wait (cus);
    wait (M);
    FC++;
    signal (Barber);
    signal (M);
}
```

customer code

```
while (1)
{   wait (M);
    if (FC > 0) {
        FC--;
    signal (cus);
    signal (M);
    wait (Barber);
    }
    else {
        signal (M);
    }
}
```

Monitor solution :-

```
barbershop
 waiting = 0;
 customer = 0;
 barber = 0

procedure seek - customer()
 begin if waiting = 0
          wait (customers);
  waiting = waiting - 1;
  signal (barber);
   end seek - costomer;

procedure get_haircut()
 begin if waiting < chairs
  {
     waiting = waiting + 1;
      signal (customers);
       wait (barber);
  }
  end get - haircut;
end barbershop;
```

Assignment #No. 4

On

Design of Operating Systems (CSE 4049)

Submitted by

Name	**: Saswat Mohanty**
Reg. No.	**: 1941012407**
Semester	**: 5th**
Branch	**: CSE**
Section	**: D**
Session	**: 2021-2022**
Admission Batch	**: 2019**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

FACULTY OF ENGINEERING & TECHNOLOGY (ITER)

SIKSHA 'O' ANUSANDHAN DEEMED TO BE UNIVERSITY

BHUBANESWAR, ODISHA – 751030

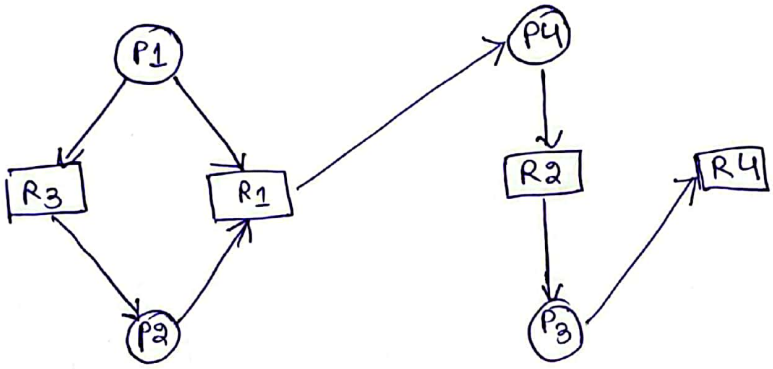
P.S. Assignment -4

Q1) consider a system with four processes P1, P2, P3 and P4, resources types R, R2, R3, R4 each one with a single instance. Draw the resource allocation graph corresponding to following resource allocation state:

- P1 requests for R3 and R1
- P2 holds R3 and requests for R1
- P3 holds R2 and requests for R1
- P4 holds R1 and requests for R2

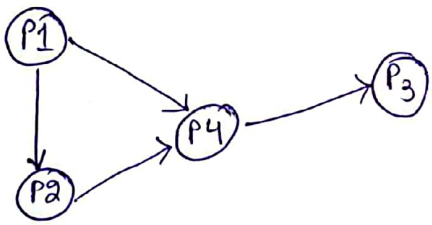
Using wait graph, check whether the system is in deadlock or not? If so how many processes are involved in deadlock?

(Ans)



the system is not in deadlock. [No cycle]

wait for graph



Q2) consider a system with 12 tapedrives & 3 processes: P0, P1, P2. P0 requires 4 tapedrives, P1 requires 10 tapedrives, P2 requires 9 tapedrives for completing their task. suppose at time t0, P0 is holding 2 tapedrives, P1 is holding 5 tapedrives and P2 is holding 2 tapedrives then check whether the current resource allocation state is safe or not. If yes specify the safe sequence.

(Ans) given :- 12 tapedrive → Resource

Process → P0, P1, P2

	Request
P0	4
P1	10
P2	9

	Allocated	Req.	Available	Req < available	New available
P0	2	2	3	true	5
P1	5	5		true	10
P2	2	7		true	12

Yes, current allocation is safe

Safe sequence :- P0, P1, P2

03) consider a system has 10 units of a resource type and 5 resources the current resource allocation state is given as follows:

Process	Used	Max
P0	2	5
P1	1	6
P2	2	6
P3	1	2
P4	1	4

If P2 will request for 2 more instances of the same resource type, check the request can be granted immediately or not?

(Ans)

Process	Used	Max	Req	Available	Req < available	New available
P0	2	5	3	3	True	5
P1	1	6	5		True	6
P2	2	6	4		True	8
P3	1	2	1		True	9
P4	1	4	3		True	10

Safe sequence :- P0, P1, P2, P3, P4

P2 will be granted with 2 more additional instances if it request as there are 3 more instances available.

Q4) consider a system with two resources A and B. A has 6 instances and B has 3 instances. Can the system execute the following processes without any deadlock ? If yes specify the safe sequence.

Process	Allocation		Max	
	A	B	A	B
P0	1	1	2	2
P1	1	0	4	2
P2	1	0	3	2
P3	0	1	1	1
P4	2	1	6	3

(Ans)

Process	Allocation		Max		Need		available		need < available		new available	
	A	B	A	B	A	B	A	B	A	B	A	B
P0	1	1	2	2	1	1	1	0	False	True	2	2
P1	1	0	4	2	3	2			F	T	4	2
P2	1	0	3	2	2	2			F	T	3	2
P3	0	1	1	1	1	0			T	T	1	1
P4	2	1	6	3	4	2			F	F	6	3

Safe sequence :- $< P_3, P0, P2, P1, P4 >$

Q5) Consider the following resource allocation state with 3 processes & 3 resources :

	Allocation			Max		
	X	Y	Z	X	Y	Z
P0	0	0	1	7	4	3
P1	3	2	0	6	2	0
P2	2	1	1	3	3	3

there are 3 instances of type X, 2 instances of type Y and 2 instances of type Z, still available.

a) find the content of the need matrix.

b) Is the system in a safe state?

c) If P_0 will request for 2 more instances of type Z, can the request be granted immediately or not?

(Ans) a)

	Allocation			Max			Need			available			new available		
	X	Y	Z	X	Y	Z	X	Y	Z	X	Y	Z	X	Y	Z
P0	0	0	1	7	4	3	7	4	2	3	2	2	8	5	4
P1	3	2	0	6	2	0	3	0	0				6	4	2
P2	2	1	1	3	3	3	1	2	2				8	5	3

b) Safe sequence :- $\{ P_1, P2, P0 \}$

c) No, P0 can't be assigned with 2 more instance of 2 as P0 already needs 2 instance of Z and are available with only 3 instance of 2

Q6) Consider the following resource allocation state with 4 processes & 4 resources, and an available vector of <0100>

	Allocation				Request			
	A	B	C	D	A	B	C	D
P0	0	1	1	0	0	0	1	0
P1	0	1	0	1	1	0	0	1
P2	1	2	0	0	0	0	0	1
P3	0	0	1	2	0	0	0	0
P4	1	0	1	0	0	1	0	0

a) Find the initial number of instances available for each resource type

b) Check whether the system is deadlock free or not

c) If P_0 is assigned with 1 more instance of type B, then check whether the system is deadlock free or not.

(Ans)

	Allocation				Request				New available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	1	1	0	0	0	1	0	1	2	3	2
P1	0	1	0	1	1	0	0	1	1	2	3	3
P2	1	2	0	0	0	0	0	1	2	5	3	3
P3	0	0	1	2	0	0	0	0	0	0	1	2
P4	1	0	1	0	0	1	0	0	1	1	2	2

a) available

A → 0

B → 1

C → 0

D → 0

b) Safe sequence :- $\langle P_3, P_4, P_0, P_1, P_2 \rangle$

Deadlock free.

a) $P_0 \rightarrow$ 1 more type B

	Allocation				Request				available				new available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
													0	2	2	2
P_0	0	2	1	0	0	0	1	0	0	0	0	0	2	5	3	3
P_1	0	1	0	1	1	0	0	1					1	4	2	2
P_2	1	2	0	0	0	0	0	1					0	0	1	2
P_3	0	0	1	2	0	0	0	0					2	4	3	2
P_4	1	0	1	0	0	1	0	0								

safe sequence :- $< P_3, P_0, P_2, P_4, P_1 >$

deadlock free

Assignment #No. 5

On

Design of Operating Systems (CSE 4049)

Submitted by

Name	**: Saswat Mohanty**
Reg. No.	**: 1941012407**
Semester	**: 5th**
Branch	**: CSE**
Section	**: D**
Session	**: 2021-2022**
Admission Batch	**: 2019**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

FACULTY OF ENGINEERING & TECHNOLOGY (ITER)

SIKSHA 'O' ANUSANDHAN DEEMED TO BE UNIVERSITY

BHUBANESWAR, ODISHA – 751030

P.S. Assignment - 5

Q1) Given four memory locations of 200k, 600k, 400k, 700k (in order). How the first-fit, Best-fit, Worst-fit algorithms place processes of 312k, 517k, 212k, 526k (in order). Which algorithm makes the most efficient use of memory?

(Ans) Memory partition → 200k, 600k, 400k, 700k

Processes → 312k, 517k, 212k, 526k

First fit

312k in 600k
517k in 700k
212k must wait
526k must wait

Best Fit

312k in 400k
517k in 600k
212k in 700k
526k must wait

Worst Fit

312k in 700k
517k in 600k
212k in 400k
526k must wait.

Q2) Using Page size of 16 bytes a physical memory of 2048 byte and logical memory of 128 bytes.

a) Find the number of bits required to represent logical address
b) Find the number of bits required to represent logical address
c) Find the number of entries in the page table.

d) Find the total no. of frames

e) Find the physical address of the logical address 80 with the following page table :

8
6
5
2
3
1
4
7

(Ans) a) No. of bits required to represent logical address

2^n = 128 bytes

$\Rightarrow n = 17$

b) No. of bits required to represent physical address

2^n = 2048 bytes

$\Rightarrow n = 21$

c) No. of entries in page table

$2^{17} / 16$ bytes

$\Rightarrow 2^{17} / 2^{14} = 2^3$

d) Total no. of frames = $2^{21} / 2^{14} = 2^7$

e)

0	8
1	6
2	5
3	2
4	3
5	1
6	4
7	7

20/16 = 1

20 % 16 = 4

6 x 16 + 4 = 100

Q3) How many numbers of pages are required for a process having size 3005 bytes with a page size of 200 bytes?

(Ans) No. of pages = $\frac{\text{Process size}}{\text{Page size}}$

$= \frac{3005}{200} = 40$ page + 5 bytes

Q4) With a page size of 2048 bytes, find the amount internal fragmentation occurs for storing a process of size 72766 bytes.

(Ans) No. of page = 35 page + 1086 bytes

Internal fragmentation = 2048 - 1086 = 962 bytes.

Q5) Consider a machine with 64mb physical memory and a 32-bit virtual address space. If the page size is 4KB, how many entries will there be in a conventional single level page table and in an inverted page table?

(Ans) No. of pages = $2^{32}/2^{12} = 2^{20}$

No. of entries = 101 conventional single-level

2^{26} = Total physical memory

2^{12} page size = frame size

$2^{29}/2^{12} = 2^{17}$ = Total no. of frame

Number of entries = 128k inverted page table

Q6) In paging scheme, if the page size is 2KB and process size is 83412 bytes. Then find the number of pages required and the size of internal fragmentation.

(Ans) No. of page = $\frac{83412 \text{ bytes}}{2000 \text{ bytes}}$ = 41 page + 1412 bytes

Internal fragmentation = 2000 - 1412 = 588 bytes.

07) A specific editor has 200K of program text, 15K of initial stack, 50K of initialized data, and 70K of bookkeeping code. If five processes are started simultaneously, how much physical memory is needed if shared program text is used?

(Ans) Total physical memory needed = 200 + 15 + 50 + 70 = 335K

08) If the hit ratio of a translational look A-side Buffer (TLB) is 80% & it takes 15 nanoseconds to search the TLB and 150 nanoseconds to access the main memory, what is the effective access time?

(Ans) $\alpha = 80\%$, $\varepsilon = 15$ nS

150 ns for memory access

$EAT = 0.80\,(15+150) + 0.20\,(15+100+100) = 175$ nS

09) A computer system implements 8KB pages and a 32-bit physical address space. Each page table entry contains a valid bit, a dirty bit, three permissions bits, and the frame numbers. If the maximum size of the page table of a process is 24 kilobytes. Find the length of the virtual address supported by the system in bits.

(Ans) Page size $= 2^{13}$ bytes = framesize

No. of frame $= 2^{32} / 2^{13} = 2^{19}$

All entries can be addressed by 23 bits

$8 \times 2^{20} = 2^{23}$

Virtual address = 23 + 13 = 36 bits.

010) Consider the byte addressable system with physical address space of 128 byte, logical address space of 64 bytes & a page size of 8 byte. The page-table is specified as follows.

4
5
1
3

a) Find the number of bits required to represent logical address.

b) Find the number of bits required to represent the physical address.

c) Find the physical address of the logical address 12

d) Find the physical address in hexadecimal representation of the logical address (35)x

(Ans) a) No. of bits required to represent logical address

$2^n = 64$ bytes

$\Rightarrow n = 16$

b) $2^n = 128$ bytes

$\Rightarrow n = 17$

c)

0	4
1	5
2	1
3	3

$12/16 = 0$

$12 \% 16 = 12$

$\therefore 16 \times 12 + 12 = 204$

d) $35 \rightarrow 0011 \quad 0101$

Page size $16 kk = 2^{14}$. So #4 bits are offset.

$0100 \quad 0000 \quad 0011 \quad 0101 \rightarrow 4035$

Q11) consider the following segment table

segment	Base	Length
0	219	600
1	2300	100
2	-90	110
3	1327	400
4	1950	50

what are the physical addresses for the following logical address ? (6)

a) 0, 430

b) 1, 10

c) 2, 100

d) 2, 500

(Ans) a) 0, 430

219 + 430 = 649

b) 1, 10

2300 + 10 = 2310

c) 2, 100

90 + 100 = 190

d) 2, 500

Illegal reference

End Term Project

On

Design of Operating Systems (CSE 4049)

Submitted by

Name	**: Saswat Mohanty**
Reg. No.	**1941012407**
Semester	**: 5th**
Branch	**: CSE**
Section	**: D**
Session	**: 2021-2022**
Admission Batch	**: 2019**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

FACULTY OF ENGINEERING & TECHNOLOGY (ITER)

SIKSHA 'O' ANUSANDHAN DEEMED TO BE UNIVERSITY

BHUBANESWAR, ODISHA – 751030

Objective of this Assignment:

To design a CPU scheduler for simulating a few CPU scheduling policies

Project Description:

Program to provides an interface to the user to implement the following scheduling policies as per the choice provided:

1. First Come First Served (FCFS)
2. Shortest Job First (SJF)
3. Shortest Remaining Time First (SRTF)
4. Round Robin (RR)

Programs

FCFS

FCFS.h

```
void FCFS();
```

FCFS.c

```
#include <stdio.h>
#include <stdlib.h>
#include "FCFS.h"
typedef struct node
{
    int no;
    float at, bt, pc, tat, wt, rt, rd;
    struct node *next;
} NODE;

void create_insert(NODE **p, int no, float at, float bt, float *fr)
{
    NODE *q, *r = *p;
    q = (NODE *)malloc(sizeof(NODE));
    q->no = no;
    q->at = at;
    q->bt = bt;
    q->rt = *fr - at;
    q->pc = *fr + bt;
    q->tat = q->pc - at;
    q->wt = q->tat - bt;
```

```
    q->rd = q->tat / bt;
    *fr = *fr + bt;
    q->next = NULL;
    if (*p == NULL)
        *p = q;
    else
    {
        while (r->next != NULL)
            r = r->next;
        r->next = q;
    }
}

void gantt_chart(NODE *p, int process)
{
    int i;
    NODE *r =  p;
    printf("\n\nGannt Chart:\n");
    for (i = 1; i <= process; i++)
        printf("-------------------------");
    printf("\n");

    for (i = 1; i <= process; i++)
    {
        printf("|\tP%d\t", p->no);
        p = p->next;
    }
    printf("|\n");

    for (i = 1; i <= process; i++)
        printf("-------------------------");

    printf("\n");
    printf("%.1f      \t", r->at);
    for (i = 1; i <= process; i++)
    {
        printf("%.1f      \t", r->pc);
        r = r->next;
    }
}

void display(NODE *p, int process)
{
    float ttat, twt, trd, trt, tbt;
    ttat = twt = trd = trt = tbt = 0;
```

```
    printf("\n\n\nProcess Details:\n");
    printf("************************************\n");
    printf("Pro\tArTi\tBuTi\tTaTi\tWtTi\tRTi\n");
    printf("************************************\n");
    while (p != NULL)
    {
        printf("%d\t", p->no);
        printf("%.2f\t", p->at);
        printf("%.2f\t", p->bt);
        printf("%.2f\t", p->tat);
        printf("%.2f\t", p->wt);
        printf("%.2f\n", p->rt);
        ttat += p->tat;
        twt += p->wt;
        trd += p->rd;
        trt += p->rt;
        tbt += p->bt;

        p = p->next;
    }
    printf("\n\n\nOverall Details:\n");
    printf("Average Turn Around Time: %.2f\n", ttat / process);
    printf("Average Waiting Time: %.2f\n", twt / process);
    printf("Average Response Time: %.2f\n", trt / process);
}

void FCFS()
{
    NODE *head = NULL;
    int process, i;
    float arrival_time, burst_time, first_response;
    printf("First- Come, First-Served (FCFS) Scheduling\n");
    printf("Enter Number of Processes\n");
    scanf("%d", &process);
    for (i = 1; i <= process; i++)
    {
        printf("\nEnter the Details for Process %d: \n", i);
        printf("Arrival Time: ");
        scanf("%f", &arrival_time);
        printf("Burst Time: ");
        scanf("%f", &burst_time);
        if (i == 1)
            first_response = arrival_time;
        create_insert(&head, i, arrival_time, burst_time, &first_response);
    }
```

```
    printf("\n<----------------------------------------------------------------------------  START ----------
----------------------------->\n");
    gantt_chart(head, process);
    display(head, process);
    printf("\n<----------------------------------------------------------------------------  END ------------------
--------------------------->\n");
}
```

SJF

SJF.h

```
void SFJ();
```

SJF.c

```
#include <stdio.h>
#include "SJF.h"
struct time
{
	int p, art, but, wtt, tat, st;
};
int process(struct time a[], int pro, int t)
{
	int i, minpro, mintime = 999;
	for (i = 0; i < pro; i++)
	{
		if (a[i].art <= t && a[i].st == 0)
		{
			if (mintime > a[i].but)
			{
				mintime = a[i].but;
				minpro = i;
			}
		}
	}
	a[minpro].st = 1;
	return minpro;
}
void ganttchart(struct time a[], int gc[], int pro)
{
```

```
        int i, x = 0;
        printf("Gantt Chart\n");
        printf("0");
        for (i = 0; i < pro; i++)
        {
                x = x + a[gc[i]].but;
                printf(" -> [P%d] <- %d", a[gc[i]].p, x);
        }
        printf("\n");
        return;
}
void SFJ()
{
        int i, pro, curpro, t = 0, gc[100];
        struct time a[100];
        float avgwt = 0, avgtt = 0;
        printf("Shortest-Job-First (SJF) Scheduling\n");
        printf("Enter Number of Processes\n");
        scanf("%d", &pro);
        for (i = 0; i < pro; i++)
        {
                printf("Enter Arrival Time & Burst Time for Process P%d\n", i);
                a[i].p = i;
                scanf("%d%d", &a[i].art, &a[i].but);
                a[i].st = 0;
        }
        for (i = 0; i < pro; i++)
        {
                curpro = process(a, pro, t);
                a[curpro].wtt = t - a[curpro].art;
                a[curpro].tat = a[curpro].art + a[curpro].but;
                t = t + a[curpro].but;
                avgwt = avgwt + a[curpro].wtt;
                avgtt = avgtt + a[curpro].tat;
                gc[i] = curpro;
        }
        printf("\n<----------------------------------------------------------------------------  START ----------
-------------------------------->\n");
        printf("****************************************\n");
        ganttchart(a, gc, pro);
        printf("****************************************\n");
        printf("\n\n\nProcess Details:\n");
        printf("****************************************\n");
        printf("Pro\tArTi\tBuTi\tTaTi\tWtTi\n");
        printf("****************************************\n");
```

```
        for (i = 0; i < pro; i++)
        {
                printf("%d\t%d\t%d\t%d\t%d\n", a[i].p, a[i].art, a[i].but,
a[i].tat, a[i].wtt);
        }
        avgwt = avgwt / pro;
        avgtt = avgtt / pro;
        printf("\n\n\nOverall Details:\n");
        printf("Average Waiting Time : %.2f\n", avgwt);
        printf("Average Turnaround Time : %.2f\n", avgtt);
        printf("\n<----------------------------------------------------------------------------  END ----------
----------------------------->\n");
}
```

SRTF

SRTF.h

```
void SRTF();
```

SRTF.c

```
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>
#include "SRTF.h"
struct process_struct
{
    int pid;
    int at;
    int bt;
    int ct, wt, tat, rt, start_time;
} ps[100];

int findmax(int a, int b)
{
    return a > b ? a : b;
}

int findmin(int a, int b)
{
    return a < b ? a : b;
```

```
}

void SRTF()
{

    int n;
    float bt_remaining[100];
    bool is_completed[100] = {false}, is_first_process = true;
    int current_time = 0;
    int completed = 0;
    ;
    float sum_tat = 0, sum_wt = 0, sum_rt = 0, total_idle_time = 0, length_cycle,
prev = 0;
    float cpu_utilization;

    int max_completion_time, min_arrival_time;
    printf("Shortest-Remaining-Time-First (SRTF) Scheduling\n");
    printf("Enter Number of Processes\n");
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        printf("\nEnter Process %d Arrival Time: ", i);
        scanf("%d", &ps[i].at);
        ps[i].pid = i;
    }

    for (int i = 0; i < n; i++)
    {
        printf("\nEnter Process %d Burst Time: ", i);
        scanf("%d", &ps[i].bt);
        bt_remaining[i] = ps[i].bt;
    }

    while (completed != n)
    {
        int min_index = -1;
        int minimum = INT_MAX;
        for (int i = 0; i < n; i++)
        {
            if (ps[i].at <= current_time && is_completed[i] == false)
            {
                if (bt_remaining[i] < minimum)
                {
                    minimum = bt_remaining[i];
                    min_index = i;
```

```
                }
                if (bt_remaining[i] == minimum)
                {
                    if (ps[i].at < ps[min_index].at)
                    {
                        minimum = bt_remaining[i];
                        min_index = i;
                    }
                }
            }
        }

        if (min_index == -1)
        {
            current_time++;
        }
        else
        {
            if (bt_remaining[min_index] == ps[min_index].bt)
            {
                ps[min_index].start_time = current_time;
                total_idle_time += (is_first_process == true) ? 0 :
(ps[min_index].start_time - prev);
                is_first_process = false;
            }
            bt_remaining[min_index] -= 1;
            current_time++;
            prev = current_time;
            if (bt_remaining[min_index] == 0)
            {
                ps[min_index].ct = current_time;
                ps[min_index].tat = ps[min_index].ct - ps[min_index].at;
                ps[min_index].wt = ps[min_index].tat - ps[min_index].bt;
                ps[min_index].rt = ps[min_index].start_time - ps[min_index].at;

                sum_tat += ps[min_index].tat;
                sum_wt += ps[min_index].wt;
                sum_rt += ps[min_index].rt;
                completed++;
                is_completed[min_index] = true;
            }
        }
    }
    max_completion_time = INT_MIN;
    min_arrival_time = INT_MAX;
```

```
    for (int i = 0; i < n; i++)
    {
        max_completion_time = findmax(max_completion_time, ps[i].ct);
        min_arrival_time = findmin(min_arrival_time, ps[i].at);
    }
    length_cycle = max_completion_time - min_arrival_time;
    printf("\n<----------------------------------------------------------------------------  START ----------
---------------------------->\n");
    printf("\n\n\nProcess Details:\n");
    printf("*************************************\n");
    printf("Pro\tArTi\tBuTi\tTaTi\tWtTi\tRTi\n");
    printf("*************************************\n");
    for (int i = 0; i < n; i++)
        printf("%d\t%d\t%d\t%d\t%d\t%d\n", ps[i].pid, ps[i].at, ps[i].bt,
ps[i].tat, ps[i].wt, ps[i].rt);
    printf("*************************************\n");
    printf("\n");
    printf("\n\n\nOverall Details:\n");
    printf("\nAverage Turn Around time= %f ", (float)sum_tat / n);
    printf("\nAverage Waiting Time= %f ", (float)sum_wt / n);
    printf("\nAverage Response Time= %f ", (float)sum_rt / n);
    printf("\n<----------------------------------------------------------------------------  END ----------
--------------------------->\n");
}
```

RR

RR.h

```
void RR();
```

RR.c

```
#include <stdio.h>
#include "RR.h"
struct times
{
    int p, art, but, wtt, tat, rnt;
};
void sortart(struct times a[], int pro)
{
    int i, j;
    struct times temp;
```

```
    for (i = 0; i < pro; i++)
    {
        for (j = i + 1; j < pro; j++)
        {
            if (a[i].art > a[j].art)
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
    return;
}
void RR()
{
    int i, j, pro, time, remain, flag = 0, ts;
    struct times a[100];
    float avgwt = 0, avgtt = 0;
    printf("Round Robin (RR) Scheduling\n");
    printf("Enter Number of Processes\n");
    scanf("%d", &pro);
    remain = pro;
    for (i = 0; i < pro; i++)
    {
        printf("Enter arrival time and Burst time for Process P%d : ", i);
        scanf("%d%d", &a[i].art, &a[i].but);
        a[i].p  =  i;
        a[i].rnt = a[i].but;
    }
    sortart(a, pro);
    printf("Enter Time Quantum Number : ");
    scanf("%d", &ts);
    printf("\n<----------------------------------------------------------------------------  START ----------
----------------------------->\n");
    printf("\n**************************************\n");
    printf("Gantt Chart\n");
    printf("0");
    for (time = 0, i = 0; remain != 0;)
    {
        if (a[i].rnt <= ts && a[i].rnt > 0)
        {
            time = time + a[i].rnt;
            printf(" -> [P%d] <- %d", a[i].p, time);
            a[i].rnt = 0;
```

```
            flag = 1;
        }
        else if (a[i].rnt > 0)
        {
            a[i].rnt = a[i].rnt - ts;
            time = time + ts;
            printf(" -> [P%d] <- %d", a[i].p, time);
        }
        if (a[i].rnt == 0 && flag == 1)
        {
            remain--;
            a[i].tat = time - a[i].art;
            a[i].wtt = time - a[i].art - a[i].but;
            avgwt = avgwt + time - a[i].art - a[i].but;
            avgtt = avgtt + time - a[i].art;
            flag = 0;
        }
        if (i == pro - 1)
            i = 0;
        else if (a[i + 1].art <= time)
            i++;
        else
            i = 0;
    }
    printf("\n\n");
    printf("\n\n\nProcess Details:\n");
    printf("***************************************\n");
    printf("Pro\tArTi\tBuTi\tTaTi\tWtTi\n");
    printf("***************************************\n");
    for (i = 0; i < pro; i++)
    {
        printf("P%d\t%d\t%d\t%d\t%d\n", a[i].p, a[i].art, a[i].but, a[i].tat,
a[i].wtt);
    }
    printf("***************************************\n");
    avgwt = avgwt / pro;
    avgtt = avgtt / pro;
    printf("\n\n\nOverall Details:\n");
    printf("Average Waiting Time : %.2f\n", avgwt);
    printf("Average Turnaround Time : %.2f\n", avgtt);
    printf("\n<----------------------------------------------------------------------------  END ----------
-------------------------->\n");
}
```

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "FCFS.h"
#include "SJF.h"
#include "SRTF.h"
#include  "RR.h"
int main()
{
    int choice;
    printf("CPU scheduler for simulating a few CPU scheduling policies.\n");
    while (1)
    {
        printf("\nChosse Option\n1->FCFS\n2->SJF\n3->SRTF\n4->RR\n5->Exit\n");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            FCFS();
            break;
        case 2:
            SFJ();
            break;
        case 3:
            SRTF();
            break;
        case 4:
            RR();
            break;
        case 5:
            exit(0);
        default:
            printf("Wrong Input\n");
            break;
        }
    }
}
```

Output Console:

```
CPU scheduler for simulating a few CPU scheduling policies.

Chosse Option
1->FCFS
2->SJF
3->SRTF
4->RR
5->Exit
```

Test Cases:

1. Consider the set of processes with arrival time (in milliseconds), CPU burst time (in milliseconds), and time quantum = 4ms as shown below.

Process	Arrival time	Burst Time
P1	0	10
P2	0	1
P3	0	2
P4	0	1
P5	0	5

Input Choice 1:

```
1
First- Come, First-Served (FCFS) Scheduling
Enter Number of Processes
5

Enter the Details for Process 1:
Arrival Time: 0
Burst Time: 10

Enter the Details for Process 2:
Arrival Time: 0
Burst Time: 1

Enter the Details for Process 3:
Arrival Time: 0
Burst Time: 2

Enter the Details for Process 4:
Arrival Time: 0
Burst Time: 1

Enter the Details for Process 5:
Arrival Time: 0
Burst Time: 5
```

```
<------------------------------------------------  START ------------------------------------->

Gannt Chart:
-------------------------------------------------------------------------------
|       P1      |       P2      |       P3      |       P4      | P5      |
-------------------------------------------------------------------------------
0.0             10.0            11.0            13.0            14.0            19.0

Process Details:
**************************************
Pro     ArTi    BuTi    TaTi    WtTi    RTi
**************************************
1       0.00    10.00   10.00   0.00    0.00
2       0.00    1.00    11.00   10.00   10.00
3       0.00    2.00    13.00   11.00   11.00
4       0.00    1.00    14.00   13.00   13.00
5       0.00    5.00    19.00   14.00   14.00



Overall Details:
Average Turn Around Time: 13.40
Average Waiting Time: 9.60
Average Response Time: 9.60

<------------------------------------------------  END --------------------------------------->
```

Input Choice 2:

```
2
Shortest-Job-First (SJF) Scheduling
Enter Number of Processes
5
Enter Arrival Time & Burst Time for Process P0
0 10
Enter Arrival Time & Burst Time for Process P1
0 1
Enter Arrival Time & Burst Time for Process P2
0 2
Enter Arrival Time & Burst Time for Process P3
0 1
Enter Arrival Time & Burst Time for Process P4
0 5
```

```
<-------------------------------------------------  START ------------------------------------->
**************************************
Gantt Chart
0 -> [P1] <- 1 -> [P3] <- 2 -> [P2] <- 4 -> [P4] <- 9 -> [P0] <- 19
**************************************



Process Details:
**************************************
Pro     ArTi    BuTi    TaTi    WtTi
**************************************
0       0       10      10      9
1       0       1       1       0
2       0       2       2       2
3       0       1       1       1
4       0       5       5       4



Overall Details:
Average Waiting Time : 3.20
Average Turnaround Time : 3.80

<-------------------------------------------------  END --------------------------------------->
```

Input Choice 3:

```
3
Shortest-Remaining-Time-First (SRTF) Scheduling
Enter Number of Processes
5

Enter Process 0 Arrival Time: 0

Enter Process 1 Arrival Time: 0

Enter Process 2 Arrival Time: 0

Enter Process 3 Arrival Time: 0

Enter Process 4 Arrival Time: 0

Enter Process 0 Burst Time: 10

Enter Process 1 Burst Time: 1

Enter Process 2 Burst Time: 2

Enter Process 3 Burst Time: 1

Enter Process 4 Burst Time: 5
```

```
<-----------------------------------------------  START ----------------------------------------->



Process Details:
**************************************
Pro     ArTi    BuTi    TaTi    WtTi    RTi
**************************************
0       0       10      19      9       9
1       0       1       1       0       0
2       0       2       4       2       2
3       0       1       2       1       1
4       0       5       9       4       4
**************************************




Overall Details:

Average Turn Around time= 7.000000
Average Waiting Time= 3.200000
Average Response Time= 3.200000
<-----------------------------------------------  END ------------------------------------------->
```

Input Choice 4:

```
4
Round Robin (RR) Scheduling
Enter Number of Processes
5
Enter arrival time and Burst time for Process P0 : 0 10
Enter arrival time and Burst time for Process P1 : 0 1
Enter arrival time and Burst time for Process P2 : 0 2
Enter arrival time and Burst time for Process P3 : 0 1
Enter arrival time and Burst time for Process P4 : 0 5
Enter Time Quantum Number : 4
```

```
<-------------------------------------------  START ------------------------------------->

**************************************
Gantt Chart
0 -> [P0] <- 4 -> [P1] <- 5 -> [P2] <- 7 -> [P3] <- 8 -> [P4] <- 12 -> [P0] <- 16 -> [P4] <- 17 -> [P0] <- 19



Process Details:
**************************************
Pro     ArTi    BuTi    TaTi    WtTi
**************************************
P0      0       10      19      9
P1      0       1       5       4
P2      0       2       7       5
P3      0       1       8       7
P4      0       5       17      12
**************************************


Overall Details:
Average Waiting Time : 7.40
Average Turnaround Time : 11.20

<-------------------------------------------  END ------------------------------------->
```

- ⇨ On analysing the results of the algorithm, the minimum average waiting time is Shortest Job First (SJF) and shortest Remaining Time First (SRTF).

2. Consider the set of processes with arrival time (in milliseconds), CPU burst time (in milliseconds), and time quantum =2ms as shown below.

Process	Arrival time	Burst Time
P1	0	4
P2	0	2
P3	1	3
P4	2	2

Input Choice 1:

```
1
First- Come, First-Served (FCFS) Scheduling
Enter Number of Processes
4

Enter the Details for Process 1:
Arrival Time: 0
Burst Time: 4

Enter the Details for Process 2:
Arrival Time: 0
Burst Time: 2

Enter the Details for Process 3:
Arrival Time: 1
Burst Time: 3

Enter the Details for Process 4:
Arrival Time: 2
Burst Time: 2
```

```
<-----------------------------------------------  START ----------------------------------------->


Gannt Chart:
-------------------------------------------------------------------
|       P1      |       P2      |       P3      |       P4      |
-------------------------------------------------------------------
0.0             4.0             6.0             9.0             11.0


Process Details:
**************************************
Pro     ArTi    BuTi    TaTi    WtTi    RTi
**************************************
1       0.00    4.00    4.00    0.00    0.00
2       0.00    2.00    6.00    4.00    4.00
3       1.00    3.00    8.00    5.00    5.00
4       2.00    2.00    9.00    7.00    7.00



Overall Details:
Average Turn Around Time: 6.75
Average Waiting Time: 4.00
Average Response Time: 4.00

<-----------------------------------------------  END ------------------------------------------->
```

Input Choice 2:

```
2
Shortest-Job-First (SJF) Scheduling
Enter Number of Processes
4
Enter Arrival Time & Burst Time for Process P0
0 4
Enter Arrival Time & Burst Time for Process P1
0 2
Enter Arrival Time & Burst Time for Process P2
1 3
Enter Arrival Time & Burst Time for Process P3
2 2
```

```
<------------------------------------------------  START ------------------------------------->
**************************************
Gantt Chart
0 -> [P1] <- 2 -> [P3] <- 4 -> [P2] <- 7 -> [P0] <- 11
**************************************



Process Details:
**************************************
Pro     ArTi    BuTi    TaTi    WtTi
**************************************
0       0       4       4       7
1       0       2       2       0
2       1       3       4       3
3       2       2       4       0



Overall Details:
Average Waiting Time : 2.50
Average Turnaround Time : 3.50

<------------------------------------------------  END --------------------------------------->
```

Input Choice 3:

```
3
Shortest-Remaining-Time-First (SRTF) Scheduling
Enter Number of Processes
4

Enter Process 0 Arrival Time: 0

Enter Process 1 Arrival Time: 0

Enter Process 2 Arrival Time: 1

Enter Process 3 Arrival Time: 2

Enter Process 0 Burst Time: 4

Enter Process 1 Burst Time: 2

Enter Process 2 Burst Time: 3

Enter Process 3 Burst Time: 2
```

```
<-------------------------------------------------  START ------------------------------------->



Process Details:
**************************************
Pro     ArTi    BuTi    TaTi    WtTi    RTi
**************************************
0       0       4       11      7       7
1       0       2       2       0       0
2       1       3       6       3       3
3       2       2       2       0       0
**************************************




Overall Details:

Average Turn Around time= 5.250000
Average Waiting Time= 2.500000
Average Response Time= 2.500000
<-------------------------------------------------  END ------------------------------------->
```

Input Choice 4:

```
4
Round Robin (RR) Scheduling
Enter Number of Processes
4
Enter arrival time and Burst time for Process P0 : 0 4
Enter arrival time and Burst time for Process P1 : 0 2
Enter arrival time and Burst time for Process P2 : 1 3
Enter arrival time and Burst time for Process P3 : 2 2
Enter Time Quantum Number : 2
```

```
<-------------------------------------------------  START ------------------------------------->

**************************************
Gantt Chart
0 -> [P0] <- 2 -> [P1] <- 4 -> [P2] <- 6 -> [P3] <- 8 -> [P0] <- 10 -> [P2] <- 11




Process Details:
**************************************
Pro     ArTi    BuTi    TaTi    WtTi
**************************************
P0      0       4       10      6
P1      0       2       4       2
P2      1       3       10      7
P3      2       2       6       4
**************************************



Overall Details:
Average Waiting Time : 4.75
Average Turnaround Time : 7.50

<-------------------------------------------------  END ------------------------------------->
```

- ⇨ On analysing the results of the algorithm, the minimum average waiting time is Shortest Job First (SJF) and shortest Remaining Time First (SRTF).