Page No.: 22
Date.:

Tutorial-4

Q1 Write an algorithm of producer consumer problem for infinite buffer. Explain with the help of tracing.

⇒
```
Semaphore n=0, s=1;
void producer()
{
  while(true)
  {
    produce();
    semWait(s);
    append();
    semSignal(s);
    semSignal(n);
  }
}

void consumer()
{
  while(true)
  {
    semWait(n);
    semWait(s);
    take();
    semSignal(s);
    consume();
  }
}
void main(){
parbegin(producer, consumer)
}
```

Tracing → finding semaphore values

s.count	s.queue	Ready
1		
0	P_2 |	P_2 | C_1
-1		
0		

buffer is full/empty

n.count	n.queue
0	
-1	C_1 |
0	

Q2 Write an algorithm of reader writer problem having writers have priority. Explain this with help of tracing.

⇒
```
int readcount, writecount;
void reader ()
{
  While (true) {
    semWait (z);
    semWait (rsem);
      semWait (x);
    if (readCount == 1)
      semWait (wsem);
      semSignal (rsem);
    semSignal (z);
    READUNIT ();
```

```
    SemWait (x);
      readCount --;
      if (readCount == 0) SemSignal (wsem);
      SemSignal (x);
    }
}

void writer ()
{
    while (true) {
    SemWait (y);
      WriteCount ++;
      if (writeCount == 1)
        SemWait (rsem);
      SemSignal (y);
      SemWait (wsem);
      WRITEUNIT ();
      SemSignal (wsem);
      Sem Wait (y);
        writeCount;
        if (writeCount == 0) SemSignal (rsem)
        SemSignal (y);
    }
}

void main ()
{
  readcount = writeCount = 0;
  parbegin (reader, writer);
}
```

Q3 Explain Message Passing & explain producer consumer algorithm using message passing.

⇒ Message passing refers to means of communication between:
- different threads within a process.
- different processes running on same node.
- different processes running on different node.

Message has predefined structure and message passing uses two system call: send & Receive

Send (destination, message)
Receive (source, message)

This is minimum set of operations needed for processes to engage in message passing. A process sends information in the form of message to another process designated by a destination. A process receives information by executing the receive primitive, indicating the source and the message.

```
const int
    capacity = /* buffering capacity */;
    null = /* empty message */;
int i;
```

```
void producer()
{
    message pmsg;
    while(true){
    receive(mayproduce, msg);
    pmsg = produce();
    send(mayconsume, pmsg);
    }
}

void consumer()
{
    message cmsg;
    while(true){
    receive(mayconsume, cmsg);
    consume(cmsg);
    send(mayproduce, null);
    }
}

void main()
{
    create_mailbox(mayproduce);
    create_mailbox(mayconsume);
    for(int i=1; i<=capacity; i++)
        send(mayproduce, null);

    parbegin(producer, consumer);
}
```

Q4 Write difference between semaphore & Monitor

	Semaphore	Monitor
1)	A variable used to control access to a common resource by multiple processes in a concurrent system such as a multitasking operating system.	1) A synchronization construct that allows threads to have both mutual exclusion and ability to wait (block) for a certain condition to become true.
2)	An integer variable.	2) An abstract datatype.
3)	There is no condition variable concept.	3) Has condition variables.

4) When a process requires to access the semaphore, it performs wait() & performs signal() when releasing the resource.

4) A process uses procedures to access the shared variables in the monitor.

Tutorial 3

Q1 Fetch_And_Add (x, i) is an atomic Read-Modify-Write instruction that reads the value of memory location x, increments it by the value i, and returns the old value of x. It is used in the pseudo-code shown below to implement a busy-wait lock. L is an unsigned integer shared variable initialized to 0. The value of 0 corresponds to lock being available, while any non-zero value corresponds to the lock being not available.

```
Acquire_lock(L){
while (Fetch_And_Add(L, 1))
L = 1;
}
Release_lock(L){
L = 0;
}
```

This implementation

(A) Fails as L can overflow.
(B) Fails as L can take on a non-zero value when the lock is actually available.
(C) Works correctly but may starve some processes.
(D) Works correctly without starvation.

Q2 Each process Pi, i=0,1,2,3,.....,9 is coded as follows:-

Mutex
1

repeat
P(mutex)
{ Critical section }
V(mutex)
forever

The code for P10 is identical except that it uses V(mutex) instead of P(mutex). What is the largest number of processes that can be inside the critical section at any moment (the mutex being initialized to 1)?

a) 1 b) 2 c) 3 d) None of these

Q3 Two processes, P_1 and P_2, need to access a critical section of code. Consider the following synchronization construct used by the processes:

Process P_1:
while (true)
{ w1 = true;
while (w2 == true);
Critical section
w1 = false;
}
Remainder section

Process P_2:
while (true) { w2 = true;
while (w1 == true);
Critical section
w2 = false;
}
Remainder section

Here, w1 and w2 are shared variables, which are initalized to false. Which one of the following statements is TRUE about the above construct?

a) It does not ensure mutual exclusion
b) It does not ensure bounded waiting.
c) It requires that processes enter the critical section in strict alternation
d) It does not prevent deadlocks, but ensures mutual exclusion.

Q4 The following program consists of 3 concurrent processes and 3 binary semaphores. The semaphores are initalized as S0 = 1, S1 = 0, S2 = 0.

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

How many times will P0 print 10 [2]?

a) At least twice
b) Exactly twice
c) Exactly thrice
d) Exactly once

Q5 Consider the methods used by processes P1 and P2 for accessing their critical sections whenever need as given below. The inital values of shared boolean variables S1 and S2 are randomly assigned.

Method used by P1:-
```
while (S1 == S2);
Critical section
S1 = S2;
```

Method used by P2:-
```
while (S1 != S2);
Critical section
S2 = not(S1);
```

Which of the following discribes properties achi

a) Mutual exclusion but not progress
b) Progress but not mutual exclusion.
c) Neither mutual exclusion nor progress.
d) Both mutual exclusion and progress.

Tutorial-4

A computer has 8 tape drives, with n processes competing for them. Each process may need maximum three tape drives. For eg which value of n is system deadlock free?

n = 3 max = 3
System deadlock free = 3

$P_1 = 3$	3	$P_1 = 2$
$P_2 = 2$	3	$P_2 = 2$
$P_3 = 2$	2	$P_3 = 2$
		$P_4 = 2$

Allocated Matrix

	X	Y	Z
P_0	1	2	1
P_1	2	0	1
P_2	2	2	1

Request/Need Matrix

	X	Y	Z
P_0	1	0	3
P_1	0	1	2
P_2	1	2	0

Resource Vector(R)

X	Y	Z
5	5	5

Available Vector

X	Y	Z
0	1	2

i) P_1 will proceed

2 0 1
0 1 2
2 1 3

ii) P_0 will proceed

2 1 3
1 2 1
3 3 4

iii) P_2 will Proceed

3 3 4
2 2 1
5 5 5

∴ Hence P_2 will proceed at last.

Q3 Resource (R) Vector

R1	R2	R3	R4
3	2	3	2

At t = 0

	R1	R2	R3	R4
	3	2	3	2
P1 (-)		2		
	3	0	3	2
P2 (-)			2	
	3	0	1	2
P3 (-)				1
	3	0	1	1

At t = 1

	3	0	1	1
P1 (-)				1
	3	0	0	1

At t = 2

	3	0	0	1
P2 (-)				1
	3	0	0	0
P3 (-)	2			
	1	0	0	0

At t = 3: None

	1	0	0	0

(Requests 2 units of R1, whereas we have only 1 unit of R1 so this process will be blocked (P1))

At t = 4

1 0 0 0

P2(-) 1

0 0 0 0

At t = 5

0 0 0 0

P1(+) 1 1

1 1 0 0

P3(+) 2

3 1 0 0

2

1 1 0 0

(Now Removing P1 from block and fulfilling the requirement.)

At t = 6

1 1 0 0

P2(+) 1

1 1 1 0

At t = 7

1 1 1 0

P1(+) 1

1 1 2 0

P3(-) 1

1 0 2 0

At t = 8

	1	0	2	0
P1 (-)			2	
	1	0	0	0
	1	0	2	0
P2 (+)	1			1
	2	0	4	1
P3 (-)			1	
	2	0	3	1

(P1 blocked as need 2 units of R4)

At t = 9

	2	0	3	1
P3 (+)	0	1	0	1
	2	1	3	2
P1 (-)				2
	2	1	3	0

At t = 10

	2	1	3	0
P1 (+)	1	1	0	2
	3	2	3	2

So finally | 3 | 2 | 3 | 2 |

Allocated Need

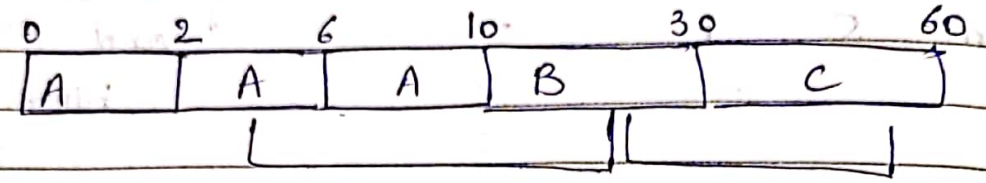
Who	Max	Has	Room
A	3	1	2
B	2	1	(1)
C	4	2	2
System	5	1	→

✓ B needs only 1 room so B will be granted and A and C will be blocked.

Tutorial-6

Q2

Process	Arrival	Service Time
A	0	10 ~~8 4~~
B	2	20
C	6	30

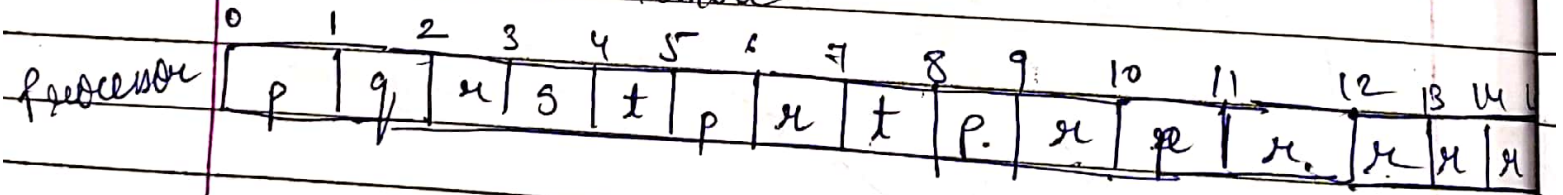


2 context switches are needed.
Context switching means how much time processes will change from processor.

Q3

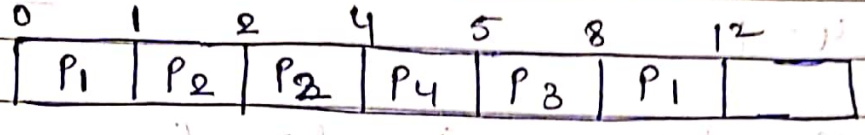
Jobid	Burst Time
p	~~4 3 7 2~~
q	~~1~~
r	~~8 7 6 5 4~~
s	~~1~~
t	~~2 1~~

Round Robin method



Process	Arrival time	Burst time
P_1	0	~~4~~
P_2	1	~~3~~ 2
P_3	2	~~3~~
P_4	4	~~1~~

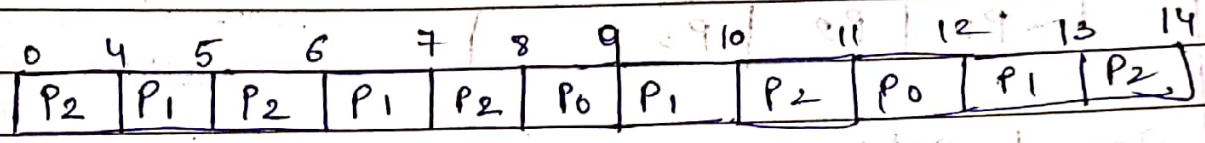
(SRTF)



Q6

Process id	Burst time	Arrival time = 0
P_0	0	2
P_1	1	~~3~~ 3
P_2	2	~~4~~ 4

(SRTF)



Turn Around time = Completion time - Arrival time

$P_0 = 12 - 0 = 12$

$P_1 = 13 - 0 = 13$

$P_2 = 14 - 0 = 14$

Average Turnaround time $= \left(\frac{12 + 13 + 14}{3}\right)$

$= 39/3$

$= 13$

Date

Page No.: 40
Date.:

Q4 Turnaround time :- Completion time - Arrival time

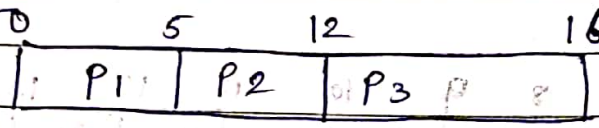
$P_1 = 12 - 0 = 12$
$P_2 = 4 - 1 = 3$
$P_3 = 8 - 2 = 6$
$P_4 = 5 - 4 = 1$

Average Turnaround Time $= (12+3+6+1)/4$
$= 21/4$
$= 5.25$

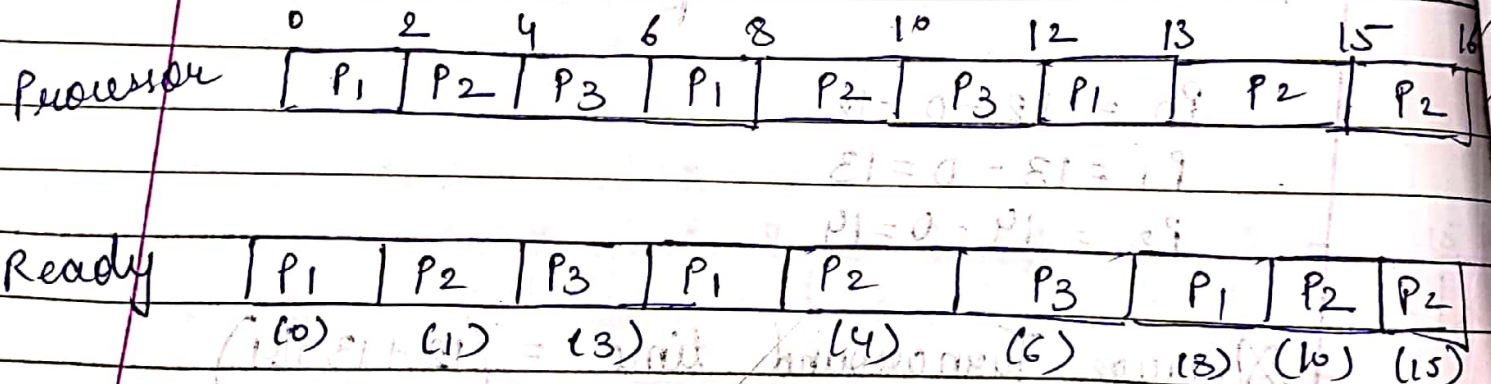
Q6

Process	Arrival time	Execution time
P_1	0	~~8 3 7 4~~
P_2	1	7 ~~8~~ 3
P_3	3	~~4 2~~

FCFS



Round Robin

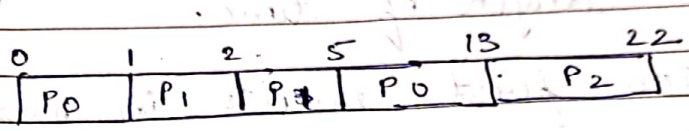


Page No. : 11
Date :

Tutorial-7

Q1.

Process Name	Arrival Time	Burst Time
P_0	0	~~8~~
P_1	1	~~4~~
P_2	2	~~9~~



Waiting time = Turnaround time - Burst time

Turnaround time = Completion Time - Arrival time

$P_0 = 13 - 0 = 13$

$P_1 = 5 - 1 = 4$

$P_2 = 22 - 2 = 20$

Waiting time

$P_0 = 13 - 9 = 4$

$P_1 = 4 - 4 = 0$

$P_2 = 20 - 9 = 11$

Average of waiting time $= \frac{4+0+11}{3}$

$= 5$

Q2 Round robin is most suitable for a time-shared operating system.
- As it uses preemption based on a clock.
- An amount of time is determined that allows each process to use the processor for that length of time.

Q3 Shortest Job first results in maximum throughput.

Date:- 18/10/19
Name:- Vinita. Dayal. Bhatia
Roll No:- 18MCA0026
Course code:- 3CA135e
Course Name:- Operating System
Practical No:- 5(a) & (b)

Aim:- To study about finding the name of the file starting with the given alphabet from the user.

Methodology:- Shell script in UNIX Programming

a) if ($check -eq 1)
then
files = 'ls|grep "^ $ch"'
echo "Files $ files"
else
echo "something wrong"
fi

b) count total number of directory

dn = 'ls -l | grep "^ d" | wc -l'
echo "Total : $ dir"

files = 'ls -l | grep "^ " | wc -l'
echo "Total files: $ files"

Signature of Teacher:-

Date:- 18/01/19
Name:- Vinita Dayal Bhatia
Roll No:- 18MCA0062
Course Code:- 3CA1352
Course Name:- Operating System
Practical No:- 06

Aim:- To learn about head, tail, grep command with menu based.

Methodology:-
echo "Enter a choice"
read choice

1) head -n 4 file1.dat
2) tail -n 4 file2.dat
3) grep -n ^ t file1.dat
4) grep -n .$ file1.dat
5) nl -s". " file1.dat
6) grep -c -E "^.{32}$" file1.dat
7) Exit

Conclusion:- Learnt about head, tail, cat, grep commands.

Signature of Teacher:-

Date :- 18/01/19
Name :- Vinita Dayal Bhatia
Roll No :- 18MCA0022
Course Code :- 3CA1352
Course Name :- Operating System
Practical No :- 07

Aim :- To check only even number of arguments are allowed.

Methodology :-

```
num = 'expr $# % 2'
if [ $num -ne 0 ]
then
    echo "Only even number of arguments are allowed"
else
    count = 1
    while [ $count -lt $# ]
    do
        cp $1 $2
        shift
        shift
        count = 'expr $count + 2'
        echo "file copied succesfully"
    done
fi
```

Conclusion :- Learnt how to allow even number of arguments.

Signature of Teacher :-

Practical No:- 07(b)

Name:- Vinita Bhatia (18MCA002)

Aim:- Shell script which receives two filenames as argument and perform the following operation.

```
echo "Enter file name"
read file1

echo "Enter second file name"
read file2

cmp $file1 $file2 > error

total='wc -c error | cut -f 7 -d " "'
echo $total

if [$total -eq 0]
then
   echo "Both file are same"
else
   echo "Both file are not same"

fi
```