Theme:

Date: / /
Sat Sun Mon Tue Wed Thu Fri

Lecture 8,

Q; Critical section Problem?

⇒ Critical section problem happen when multiple processes are running at the same time and they all want to access or change the same shared data.

Each process has a part of its code called the critical section where it is uses that shared data. The Problem making sure that only one process at a time can be in its critical section at a time so the shared data doesn't get messed up or given wrong result.

Each proces must ask permission to enter critical section in entry section. may follow critical section with exit section, then reminder section.

```
do{
    entry section
        critical section
    exit section
        remainder section
} while (true);
```

Theme:

Date: / /

Sat Sun Mon Tue Wed Thu Fri

8; Explain three critical section Problem solution ?

⇒ There are three critical section Problem solution. 1. Mutual Exclusion. 2. Progress. 3. Bounded waiting

Mutual exclusion: This means that only one Proces can be in critical section at a time. If one process is already using the shared resource no other proces should be allowed to enter the critical section until the 1st one finish.

Progress: If no Proces is in the critical section and some processes want to enter, then one of them should be allowed to go in without unnecessary delay. The Proceses must not block each other forever if no one is inside.

Bounded waiting: once a Proces has asked to enter the critical section, there should be a limit on how many others Proceses can go ahead of it. It shouldnot have to wait forever while others keep entering the critical section again and again

Theme:

Date: / /

Sat Sun Mon Tue Wed Thu Fri

Q; What is synchonization? Explain with example?

⇒ Synchronization in operating systems means making sure that multiple proces or threads that run at the same time work together properly without interfering with each other, especially when they share data or resources.

Example:

int a = 5; [Shared variable]

P_1	P_2
a++	a--
6	4

Q; Explain Race condition with produce & consumer example?

⇒ A Race condition happens when two or more proceses try to change the same data at the same time. and the final result depends on who finish first.

Example:

Lets The counter = 5;

- Producer wants to do counter++ (increas to 6)
- Consumer wants to do counter-- (decrease to 4)

without synceronyzation;

1. Producer read = 5
2. consumer read = 5
3. Producer add1 = 6
4. consumer remove 1 = 4
5. Producer store = 6
6. Consumer store = 4

The final result is = 4 (But it should be 5)

Theme:

Date: / /
Sat Sun Mon Tue Wed Thu Fri

Buffer example;

Producer

```
while (true)
{
while (counter = Buffer size)
Buffer[in] = next_Producer;
in = (in+1) % Buffer size
counter++;
}
```

Consumer

```
while (true)
{
while (counter = 0)
next = consumer = Buffer[out];
out = (out+1) % Buffer size
counter--;
}
```

Theme:

Date: / /

Sat Sun Mon Tue Wed Thu Fri

Peterson's Solution:

P_i P_j

```
do{
flag[i] = True;
turn = j;
while (flag[j] && turn=j);
//critical section
flag[i] = false;
// reminder section
}while (true)
```

```
do{
flag[j] = True;
turn = i;
while (flag[i] && turn = i);
// critical section
flag[j] = false;
// reminder section
}while (true)
```

Here using two variable → 1. flag [which process is ready to enter]
→ 2. turn [whos turn it is]

Here, 3 critical problem solution are worked ; (Mutual exclusion, Progress, Bounded waiting)

Theme:

Date: / /

Sat Sun Mon Tue Wed Thu Fri

Test and set lock solution

```
boolean TestAndSet (boolean * target)
{
boolean rv = * target;
* target = True;
return rv;
}
```

P1

```
do{
while (TestAndSet (&lock));
// critical section
lock = false;
// Reminder section
}while (true);
```

P2

```
do{
while (TestAndSet (&lock));
// critical section
lock = false;
// Reminder section
}while (true);
```

There are ~~one~~ ~~two~~ variable ~~Here.~~ Test

Here use just one variable → lock;

Test and set [function name]

lock 0 ⇒ CS এ কেউ নাই available

lock 1 ⇒ Unavailable

Only mutual exclusion & Progress are fulfill Here. Bounded waiting never meet Here.

Theme:

Date: / /

Sat Sun Mon Tue Wed Thu Fri

Compare & swap Instruction :

```
int compare_And_swap (int * locked 0,1)
{
int value = * locked;
if (value == 0)
* locked = 1;
return value
}
```

P1

```
do {
while (compare And swap (&lock, 0, 1) != 0);
 // CS
lock = 0;
// RS
} while (true);
```

P2

```
do{
while (compare and swap(&lock,0,1)!=0);
  // CS
 lock = 0;
 RS
}while(true);
```

The variable is lock [lock 0 = avaiable ; lock 1 = unavaiable]

Bounded waiting can't be fulfilled Here.

Date: / /

Sat Sun Mon Tue Wed Thu Fri

Theme:

Mutex lock :

P_1 P_2

```
while (true)
{
while (lock != 0);
lock = 1;
// cs
lock = 0;
RS
}
```

Atomic Instraction

```
acquire (int *L)
{
while (*L != 0);
*L = 1;
}
release (int *L)
{
*L = 0;
}
```

P_1

```
while(true)
{
acquire(&lock);
// cs
release (&lock);
//RS
}
```

Function → acquire and release .

varriable → lock

Bounded waiting can't Fullfill Here .

Theme:

Date: / /

Sat Sun Mon Tue Wed Thu Fri

Semaphore :

```
wait (semaphore s)
{
while (s<=0);
   s--;
}
signal (semaphore s)
{ s++;
}
```

P1

```
while (true)
{
wait (s);
//cs
signal (s);
//RS
}
```

P2

```
while (true)
{
wait (s);
//cs
signal (s);
//RS
}
```

function → wait and signal

variable → s [s = 0 = not available, s = 1 available]

spinlock problem slove করে।

Theme:

Date: / /

Sat Sun Mon Tue Wed Thu Fri

Lab - 9:

If else statement:

Syntax

```
if [condition]
then
    statement

else
    statement

fi
```

```
#!/binn/bash
```

If else if statement:

Syntax

```
if [condition]
then
    statement
else elif [condition]
    then
    statement

else
    statement

fi
```

AND

0 0 - 0
0 1 - 0
1 0 - 0
1 1 - 1

Theme:

0 0 | 0
0 1 | 1
1 0 | 1
1 1 | 1 → OR

Date: / /
Sat Sun Mon Tue Wed Thu Fri

operators:

- -eq or ==
- -ne !=
- -gt >
- -lt <
- -ge >=
- -le <=

☐ count 1p

~~if [$count~~

☐ a=b কি না?

```
if [ $a == $b ]
then
   echo "equal"
else
   echo "not equal"
fi
```

a = 10;

```
[ $a > 18 ] || [ $a < 30 ]
   0          1
then
   echo "True"
else
   echo "False"
fi
```

```
a=10
if [ $a > 18 ] && [ $val < 30 ]
     0              1
then
   echo "False"
else
   echo "True"
fi
```

Theme:

Date: / /
Sat Sun Mon Tue Wed Thu Fri

Check if a number is even or odd

```
#!/bin/bash
if [ $num % 2 == 0 ]
then
    echo "even"
else
    echo "odd"
fi
```

Take two numbers as input and print the greater one

```
#!/bin/bash
if [ $a > $b ]
Then
  echo "a is greather than b"
else
  echo "b is greather than a"
fi
```

Check a number is positive, negative or zero

```
#!/bin/bash
if [ $num -gt 0 ]
Then
  echo "Positive number"
elif [ $num -lt 0 ]
  echo "Negative number"
than
  echo "zero"
fi
```

Theme:

Date: / /
Sat Sun Mon Tue Wed Thu Fri

Q; Take mark as input:

A+ : 90 - 100
A : 85 - 89
B+ : 80 - 84
B : 75 - 79
C+ : 70 - 74
C : 65 - 69
D+ : 60 - 64
D : 55 - 59
F : Below 55

```
#!/bin/bash
read -P "Enter your marks: " marks
if [ $marks -gt 90 ] && [ $marks -lt 100 ];
Then
  echo "Grade A+"
elif [ $marks -gt 85 ] && [ $marks -lt 89 ];
Then
  echo "Grade A"
elif [ $marks -gt 80 ] && [ $marks -lt 84 ];
Then "Grade B+"
elif [ $marks -gt 75 ] && [ $marks -lt 79 ];
Then "GradeB"
else
  "GradeF"
fi
```

Theme:

Date: / /

Sat Sun Mon Tue Wed Thu Fri

Lecture 7:

Q: what is multilevel Queue scheduling?

⇒ Multilevel Queue scheduling is a way to manage Process by putting them into different groups based on their type or priority. Each queue can use a different scheduling method. Ex: high priority task may use Round Robin, lower priority task may use FCFS. The CPU always picks tasks from the highest priority queue first, and it only looks at the next queue if the higher is empty.

Q; What is feedback Queue scheduling?

⇒ It is a smart way to manage process where ~~task~~ tasks can move different queue based on how they behave. At first every task starts in the highest priority queue. If a task take too long it moved to a lower-priority queue. If it keep taking long, it moves even lower. This way first task finish quickly. and slow tasks get a fair chance. It keep fair balance,

Theme:

Date: / /
Sat Sun Mon Tue Wed Thu Fri

Q; what is load balancing?

⇒ In a multiprocessor system, if one cpu is overloaded with many task while other are lightly waited. the overall system performance suffers. Load balancing ensures that all processors are kept busy. This help the computer run fast and smoother.

There are 2 main types of load balancing;

1. Pull Migration, 2. Push Migration.

↓ A free cpu pull tasks from a overloded cpu.

↳ A cpu that is too busy. then it pushes some of its tasks to other cpus.

Q; what is processor affinity?

⇒ when a proces uses a cpu, that cpu stores some helpful data in its cache. The cache helps the proces run faster next time. Now if the proces is suddiently moved to different cpu, it can't use the old cache. It has start fresh. This slows things down. To avoid that, the operating system tries to keep the proces on the same cpu as much as possible. This is called proces affinity.

Theme:

Date: / /
Sat Sun Mon Tue Wed Thu Fri

There are two type of Process affinity;

1. Soft Affinity ⇒ The OS keep tries to keep the process on the same cpu. But it can move the process if needed. It's flexible.
2. Hard Affinity.

↓

The process is locked specific cpu. It's not flexible.

Q; What is latency?

⇒ In OS latency means the time delay between an event occuring and the system response to that event.

There are two types of latency; 1. Interrupt latency and Dispatch latency.

Q; what is littles law?

$n = \lambda \times w$

n = average queue length

w = average waiting time in queue.

λ = average arrival rate into queue.

Theme:

Date / /

Sat Sun Mon Tue Wed Thu Fri

Banker's Algorithm:

Q; Here; Total resource A = 10; B = 5; C = 7

Process	Allocation			Max Need			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			
	= 7	2	5						

∴ Total resource:

A = 7+3 = 10

B = 2+3 = 5

C = 5+2 = 7

Here;

Need Matrix: (Max Allocation)

	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

[Max Allocation = [Max Need] - [Allocation]]

[যে Process এর Allocation + Available = New Available]

Date: / /

Sat Sun Mon Tue Wed Thu Fri

Theme:

need ≤ available

$P_0 \rightarrow 7\ 4\ 3 \leq 3\ 3\ 2$ [F]

$P_1 \rightarrow 1\ 2\ 2 \leq 3\ 3\ 2$ [T]

$P_2 \rightarrow 6\ 0\ 0 \leq 5\ 3\ 2$ [F]

$P_3 \rightarrow 0\ 1\ 1 \leq 5\ 3\ 2$ [T]

$P_4 \rightarrow 4\ 3\ 1 \leq 7\ 4\ 3$ [T]

$P_0 \rightarrow 7\ 4\ 3 \leq 7\ 4\ 5$ [T]

$P_2 \rightarrow 6\ 0\ 0 \leq 7\ 5\ 5$ [T]

Process	Allocation			Available		
	A	B	C	A	B	C
P_0	0	1	0	3	3	2
P_1	2	0	0	5	3	2 → P_1
P_2	3	0	2	7	4	3 → P_3
P_3	2	1	1	7	4	5 → P_4
P_4	0	0	2	7	5	5 → P_0
				10	5	7 → P_2

$\therefore P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_0 \rightarrow P_2$

Q; Can the request 3,4,0 by P_4 be granted immediately :

∴ a) request ≤ need

3 4 0 ≤ 4 3 1 [T]

b) request ≤ avaiable

3 4 0 ≤ 3 3 2 [F]

Equation :

1) Allocation = Allocation + Request .

2) Available = Available - Request

3) Need = Need - Request .

Theme:

Date / /

Sat Sun Mon Tue Wed Thu Fri

Q; Can the request 2,1,0 by P_1 be granded immediately

a) Request ≤ need

2 1 0 ≤ 4 3 1 [T]

b) Request ≤ available

2 1 0 ≤ 3 3 2 [T]

Process	Allocation			Max Need			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	1	2	2
P_1	2	0	0	3	2	2	3	2	2 → P_1
P_2	3	0	2	9	0	2	5	3	3 → P_3
P_3	2	1	1	2	2	2	7	4	5 → P_4
P_4	2	1	2	4	3	3	7	5	5 → P_0
	= 9	3	5				10	5	7 → P_2

Safe state:
∴ $P_1 \to P_3 \to P_4 \to P_0 \to P_2$

For P_4, Allocation = Allocation + Request

= 0 0 2 + 2 1 0

= 2 1 2

Need Matrix:

	A	B	C
P_4	2	2	1

Here; The resource

A = 10

B = 5

C = 7

∴ New available = Available - Request

= 3 3 2 - 2 1 0

= 1 2 2

Need ≤ available

$P_0 \to$ 7 4 3 ≤ 2 2 1 [F]

$P_1 \to$ 1 2 2 ≤ 2 2 1 [T]

$P_2 \to$ 6 0 0 ≤ 4 2 1 [F]

$P_3 \to$ 0 1 1 ≤ 4 2 1 [T]

$P_4 \to$ 2 2 1 ≤ 6 3 2 [T]

$P_0 \to$ 7 4 3 ≤ 8 4 4 [T]

$P_2 \to$ 6 0 0 ≤ 8 5 4 [T]

Theme

Date

5; Can the Request 3 3 1 by P_4 be granted immediately;

a) Request ≤ need

3 3 1 ≤ 4 3 3 [T]

b) Request ≤ available

3 3 1 ≤ 3 3 2 [T]

Process	Allocation			MaxNeed			Available		
	A	B	C	A	B	C			
P_0	0	1	0	7	5	3	0	0	1
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	3	3	3	4	3	3			
	10	5	6						
	10	5	7(001)						

For, P_4 Allocation = Request + Allocation

= 3 3 1 + 0 0 2

= 3 3 3

P_4, Need matrix = Need - Request

= 4 3 3 - 3 3 1

= 1 0 0

P_4, available = Available - Request

= 3 3 2 - 3 3 1

= 0 0 1

Need ≤ available

$P_0 \rightarrow$ 7 4 3 ≤ 0 0 1 [F]

$P_1 \rightarrow$ 1 2 2 ≤ 0 0 1 [F]

$P_2 \rightarrow$ 6 0 0 ≤ 0 0 1 [F]

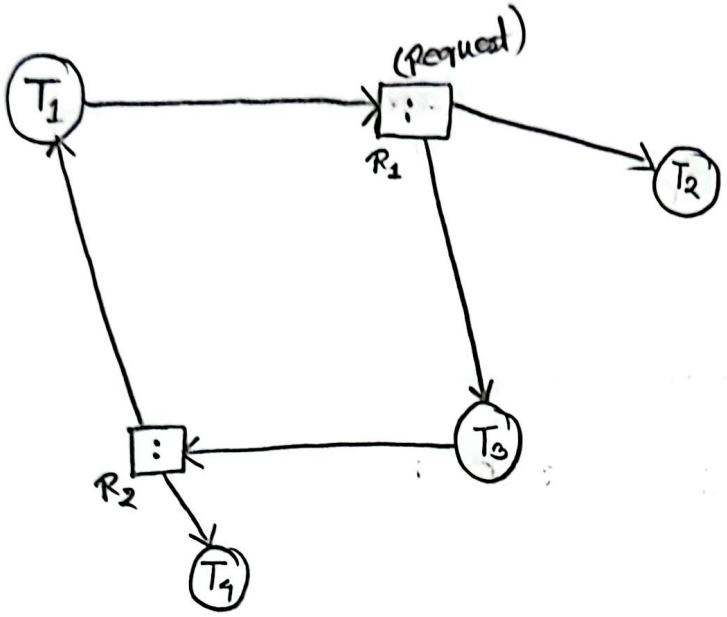
$P_3 \rightarrow$ 0 1 1 ≤ 0 0 1 [F]

$P_4 \rightarrow$ 1 0 0 ≤ 0 0 1 [F]

Theme:

Date: / /
Sat Sun Mon Tue Wed Thu Fri

Resource Allocation Graph:



	Allocation		Request		Available	
	R_1	R_2	R_1	R_2	R_1	R_2
T_1	0	1	1	0	0	0
T_2	1	0	0	0	1	0 → T_2
T_3	1	0	0	1	1	1 → T_4
T_4	0	1	0	0	1	2 → T_1
	2	2			2	2 → T_3

T তে যা ঢুকবে সেগুলো Allocation এ যাবে, আর T থেকে যেগুলো বের হবে সেগুলো Request এ যাবে

Here; Resource : 2 2

∴ request ≤ available

$T_1 \rightarrow 1\ 0 \leq 0\ 0$ [F]

$T_2 \rightarrow 0\ 0 \leq 0\ 0$ [T]

$T_3 \rightarrow 0\ 1 \leq 1\ 0$ [F]

$T_4 \rightarrow 0\ 0 \leq 1\ 0$ [T]

$T_1 \rightarrow 1\ 0 \leq 1\ 1$ [T]

$T_3 \rightarrow 0\ 1 \leq 1\ 2$ [T]

∴ safest state : $T_2 \rightarrow T_4 \rightarrow T_1 \rightarrow T_3$

Theme:

Date: / /

Sat Sun Mon Tue Wed Thu Fri

All are a memory management technique.

Q; What is Adress binding?

= Adress binding is the process of mapping logical address to Physical address in memory.

1) source code. 2) compiler, 3) object file. 4) Linker, 5) Executable file. 6) Loader. 7) CPU.

3 typer of time here; 1. compile time, 2. Load time, 3. execute time

Memory management unit.

Q; what is loading?

= Loading is a process of tranferring a program from secondary memory to main memory.

There are 2 types of loading: 1) static loading 2. Dynamic loading

1) ~~Stading~~ loading: Here, whole program together load in the RAM; It is fast but the avarage of memory are huge.

2) Dynamic loading: wherenever what need just load that. It into RAM. Its also fast but didnt waste any memory.

Theme:

Date: / /

Sat Sun Mon Tue Wed Thu Fri

Q; what is swapping?

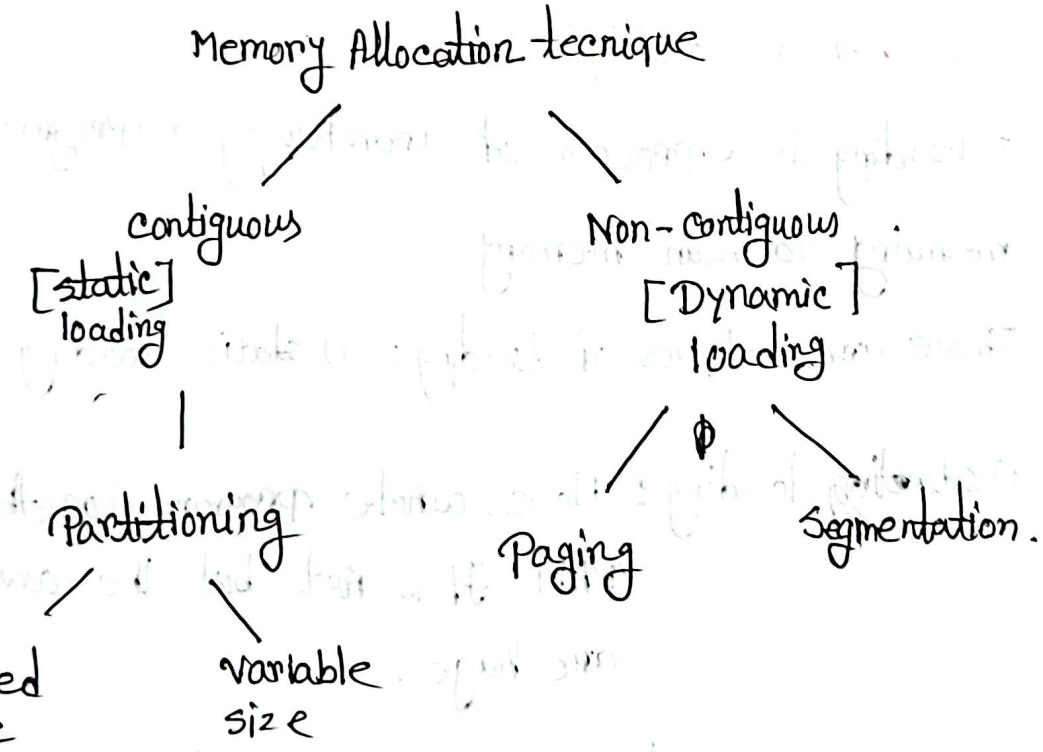
= Swapping is a memory management technique where a process temporarily moved from main memory to a backing store.

where OS removes a program from the RAM to free up the memory.

RAM → Backing store = swap in

Backing store → RAM = swap out

Q; Memory Allocation technique;



Theme:

Date: / /

Sat Sun Mon Tue Wed Thu Fri

Q; What is fragmentation?

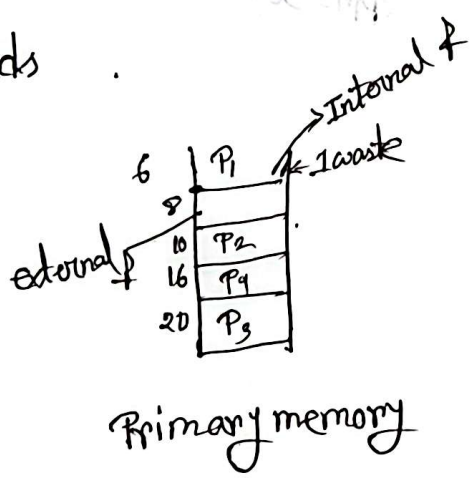
⇒ Fragmentation is a memory management problem.

There are 2 types of fragmentation:

1. External fragmentation. 2. Internal fragmentation.

External fragmentation: There is enough total free memory to run a program. but it is scattered in small, non-contiguous block. so the program can't be allocated memory as one large continuous block.

Internal fragmentation: Extra wasted inside memory block. It happen when a block is bigger than what the program actually needs.



Theme:

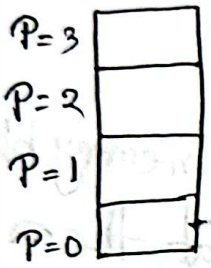
Date: / /

Sat Sun Mon Tue Wed Thu Fri

Q: what is Paging?

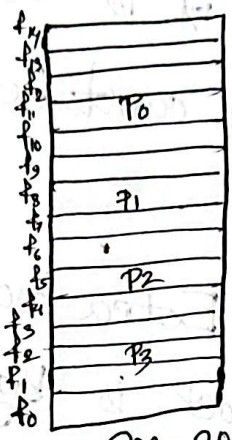
= Paging is a memory management technique that allows a Programs memory to be divided into fixed size blocks called Pages. The physical memory also divided into blocks of the same size called frames.

Page



$P_1 = 8mb$

Block size = 2mb



PM = 30 mb

∴ No of page = $8/2 = 4mb$

Page Table

P_3	f_2
P_2	f_5
P_1	f_8
P_0	f_{11}

Theme:

Date: / /

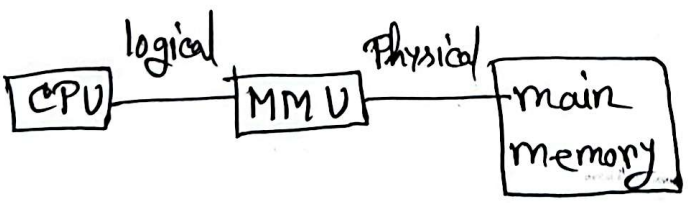
Sat Sun Mon Tue Wed Thu Fri

Q; what is segmentation?

= segmentation is a memory management technique where a program is divided into variable-size parts called segments. It tracks each segments location. It makes memory access more logical but it can suffer from external fragmentation

Q; Physical adress vs logical adress?

Physical adress	logical adress
1. Actual adress in main memory.	1. Generated by CPU.
2. Data is stored there.	2. It is used as a reference to access the memory located by CPU.
3. This adress used by MMU.	3. Its a virtual address.



Date: / /

☐ Sat ☐ Sun ☐ Mon ☐ Tue ☐ Wed ☐ Thu ☐ Fri

Theme:

Lab;

- Always variable name এর আগে $ হবে।
- echo $Name .
- read command takes input from the keyboard।
Ex: read variablename .
- Taking input in same line (Not in next line)
Ex: read -P "Enter variable"
- sum = $(expr $a + $b)
- sub = $(expr $a - $b)
- multi = $(expr $a * $b)
- div = $(expr $a / $b)
- mod = $(expr $a % $b)

Relational operators:

- eq (==)
- ne (!=)
- gt (>)
- lt (<)
- ge (>=)
- le (<=)

Theme:

Date: / /

Sat Sun Mon Tue Wed Thu Fri

If else statement:

```
if [condition]
then
    statement
else
    statement
fi
```

↳ Asciivalue check করে

switch case:

```
case expression in
    Pattern)
        statement
;;
Pattern)
        statement
;;
*)
    echo Default
;;
esac
```

If else if statement:

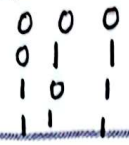
```
if [condition]
then
    statement
elif [condition]
then
    statement
else
    statement
fi
```

switch case:

```
var = "Name"
case $var in
    "karim")
        echo "karim is good"
;;
"Rahim")
        echo "Rahim is bad"
;;
*)
    echo "unknown"
;;
esac
```

Theme:

OR



AND

Date: / /

Sat Sun Mon Tue Wed Thu Fri

OR operator:

```
val = 10
if [$val -gt 18] || [ $val -lt 30]
then
        True
else
        False
fi
```

AND operator:

```
val = 10
if [$val -gt 18] && [$val -lt 30]
Then
        False
else
        True
fi
```


BD SCHOOL SHOP
ONE-STOP SCHOOL SHOPPING

Theme:

Date: / /

Sat Sun Mon Tue Wed Thu Fri

While Loop:

```
while [condition]
do
    statement
    statement
done
```

Ex:

```
n=1
while [$n -le 10]
do
    echo "$n"
    n=$((n++))
done
```

For Loop:

```
for [condition]
do
   statement
done
```

Ex:

```
for ((i=0; i<10; i++))
do
   echo $i
done
```

Pwd:

```
fo c in Pwd uname
do
 echo "command Name: $c"
 echo "command output:"
$c
done
```

Date: / /

Sat Sun Mon Tue Wed Thu Fri

Theme:

Until Loop

Syntax

```
until [condition]
Do
   statement
done
```

Ex.

```
a=0
until [$a -lt 10]
Do
   echo $a
   a='expr $a+1'
done
```

Break and con

Break

```
for((i=1; i<10; i++))
do
  if [$i -gt 5]
then
          break
fi
echo "$i"
done
```

continue;

```
for((i=1; i<10; i++))
do
 if [$i -eq 5]
 then
       continue
 fi
 echo  "$i"
 done
```

Theme:

Date: / /
Sat Sun Mon Tue Wed Thu

Using IFS (Internal File separator)

```
filename = user.txt
while IFS = ',' read name age address
do
 echo $name "lives in" $address
done < $filename
```

File existence check

```
#!/bin/bash
read -P "Enter a file name: " filename
if [-f $filename]
then
 echo "File Exist"
else
 echo "File doesn't exist"
fi
```

```
arr = (10 20 5 15 35)
min = ${arr[0]}
for num in "${arr[@]}"
do
if (( num < min ))
  then
      min = $num
fi
done
echo "Minimum value :" $min
```

Date: / /
Sat Sun Mon Tue Wed Thu Fri

Theme:

Check if a number even or odd:

```
echo "Enter a number:"
read num
if [ $num % 2 == 0 ]
 then
     echo " Its a even number"
else
   echo " Its a odd number"
fi
```

Take two number as input and print the greather one

```
echo "Enter two number:"
read a
read b
if [ $a -gt $b ]
echo "a is greather than b"
else
  echo "b is greather than a"
fi
```

Theme:

Date / /

Sat Sun Mon Tue Wed Thu Fri

Even or odd o in a Loop :

```
read -p "Enter the number:" i
For (( i=0; i<=5; i++))
do
  if [$i %2 == 0]
then
    echo "even number"$i
else
    echo " odd number "$i
fi
done
```

Even number between 1 to 20 ; and sum them

```
Sum_even=0
Sum_odd=0
Total sum=0
for ((i=0; i<=20; i++))
do
 if ((i%2==0))
then
   echo "$i is an even number"
   ((sum_even= sum_even+i))
else
   echo "$i is an odd number"
   ((sum_odd=sum_odd+i))
fi
done
```

```
total_sum=$((sum_even+sum_odd))
echo " sum of even numbers: "$sum_even
echo "sum of odd numbers: "$sum_odd
echo "sum of total number: "$totalsum
```

Date: / /

Sat Sun Mon Tue Wed Thu Fri

Theme:

Pyramid using *

```
read -p "Enter a number of rows: " rows
for ((i=1; i<=rows; i++)) [Print the pyramid]
do
 for ((j=i, j<=rows; j++)) [Print space]
do
 echo -n " "
done
 for ((k=1; k<=2*i; k++)) [Print stars]
do
 echo -n "*"
done
echo
done
```

basic pyramid

```
read -p "Enter number of rows:" rows
for ((i=1; i<=rows; i++))
do
 for ((j=1; j<=i; j++))
 do
  echo -n "*"
 done
 echo
done
```

Theme:

Date: / /

Sat Sun Mon Tue Wed Thu Fri

wc -c

```
count=$(wc -c < f1.txt)
for var in $count
do
echo "Number of character: " $var
done
```

wc -w

```
count=$(wc -w < f1.txt)
for var in count
do
echo "Number of word: " $var
done
```

File existance

```
read -P "Enter a file name: " filename
if [-f $filename]
then
echo "File exist"
else
  echo "File doesn't exist"
fi
```

wc -l

```
count=$(wc -l < f1.txt)
for var in $count
do
echo "Number of line: " $var
done
```

Date: / /

☐ Sat ☐ Sun ☐ Mon ☐ Tue ☐ Wed ☐ Thu ☐ Fri

Theme:

Fibonacci number:

```
read -p 'Enter the number of terms: " n
a=0
b=1
echo "Fibonacci series up to $n terms: "
for (( i=0; i<n; i++))
do
  echo -n "$a"
  fn=$((a+b))
  a=$b
  b=$fn
done
echo
```

Prime number:

```
read -p "Enter a number: "num
if [ $num -le 1 ]
then
 echo "$num is not a prime number"
fi
for ((i==2; i<=num; i++))
do
 if ((num % i = 0))
then
 echo "$num is not a prime number"
fi
done
echo "$num is a Prime number"
```


BD SCHOOL SHOP
ONE-STOP SCHOOL SHOPPING