Muhammad Hamza 0213-BSCS-22 CS&I

19 September 2024 Thursday

Operating Systems:

↳ Theory ⟶ sessional
↳ Lab ↳ 50%
↳ 20% Mid
30% { Surprise Quiz
Announced
Assignment

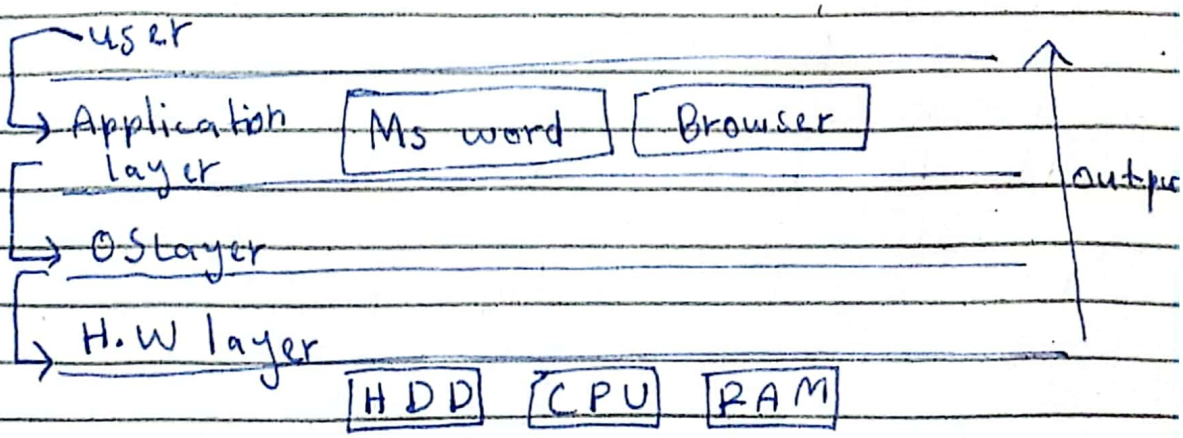
Book: Operating Systems concepts 7th edition sibber dharg

Definitions:

Layered approach:
Hardware layer

HDD | RAM | CPU

⊙ O.S layer is ontop of H.W Layer



OS is a software that directly ~~manages~~ handle the H.W and provides interface for application programs

- No universaly aggreed definition of OS

imp mcq

OS layer : drivers | embedded system

system programmer work on this level. Too much knowledge of low level

- software programmer (engineer) work on application program level

- Any software that interect with hardware is a system program.

- Two type of bug
  - Syntax error
  - logical error

∴ For an OS, we can't have a single .exe file. It always have multiple files.

- Every OS has multiple process.

Processes

↳ Multiple process

↳ kernel • authorized kernel have top priority.

↳ User process
↳ OS //

∴ 0 mean top priority.

∴ An kernel is also a OS.
coordinate all processes.

↳ Files ← virtual resource
↳ CPU ← H.W resource

∴ kernel take care of resources

⊙ **Kernel** is the most authorative process of an OS that works to control other processes & coordinates activities among them.

Kernel Panic
Blue screen of Death is kernel Panic

⊙ A situation in which even the kernel doesn't know how to respond is called **Kernel Panic**.

∴ Linux have most lower kernel panic problem i.e very rare

Roles of an OS

↳ Resource allocator (allocate resources in controlled manner) (allocated according to priority)
↳ control program (make sure all process return resources in timely manner)

∴ two resources (H.W & virtual)

↳ Bootstrap
↳ Boot Loader

bootloader program

∴ motherboard has small program called BIOS.
∴ C drive is bootable as it has O.S.

∴ when computer turn on, CPU loads bootloader in RAM. bootloader scans HDD & give list to use which O.S. Then selected OS loaded in RAM (only some files copied/loaded in RAM) & control given to OS.

Bootloader program is a S.W that scans H.DD for O.S & then loads that OS from HDD to RAM.

↳ EEPROM (updateable BIOS)
↳ ROM (fixed BIOS)

- Fedora → iso 4gb
  - ↳ stable version

Dual Boot or VM (vmware & workstation)

Bios → H.W → virtualization

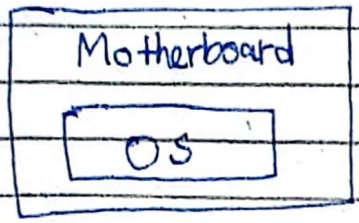
f process

25 September 2024 Wednesday

↳ Bootloader
↳ Bootstrap program
↳ BIOS
} same things

: basic input output system

- small software on Motherboard. It scans H.DD.



- A partition is bootable if it have bootable OS installed on it.

- BIOS load OS files to RAM
: No O.S found error due to BIOS.

- BIOS two types
↳ ROM
↳ EEPROM

BIOS is a small S.W that lods OS from H.D.D to memory. It is embedded into mother board. Two types 1- ROM 2- EEPROM

Interrupt vs Polling

in 1980, OS designed started.
Polling mechanism used in early computers.

In polling, OS scans all ports to see if any thing attached to it.

Old as used Polling.
Now interrupt used. It has device controller.
All same ports have only one controller.

Interrupt have highest priority in systems.

Interrupt is a mechanism of informing the OS that an event has occured. All the modern OS are interrupt based. Interrupt have highest priority in system. (as see interrupt if ping received & stop all other processes)

imp definition

Diff b/w device controller & device driver

Device driver is a software. Device controller is a hardware. Device controller has a small buffer (memory) inside it to store data temporarily. Device driver serve as a bridge between OS & device controller. Device controller interpret the signals/data being sent to devices.

∴ device controller keep data in buffer temporarily.

An interrupt that was never handled by OS is called **Lost interrupt**

Two type of interrupt
↳ H.W interrupts/interrupts
↳ S.W interrupts/monitor call

An interrupt that is generated by a SW is called **S.W interrupt / monitor call.**

∴ printf("Hello world") is S.W interrupt.
∴ interrupt through coding is S.W interrupt (API call)

An **trap** is a S.W generated interrupt specifically for error handling / exception handling

∴ interrupts have same priority.

Five states of a Process

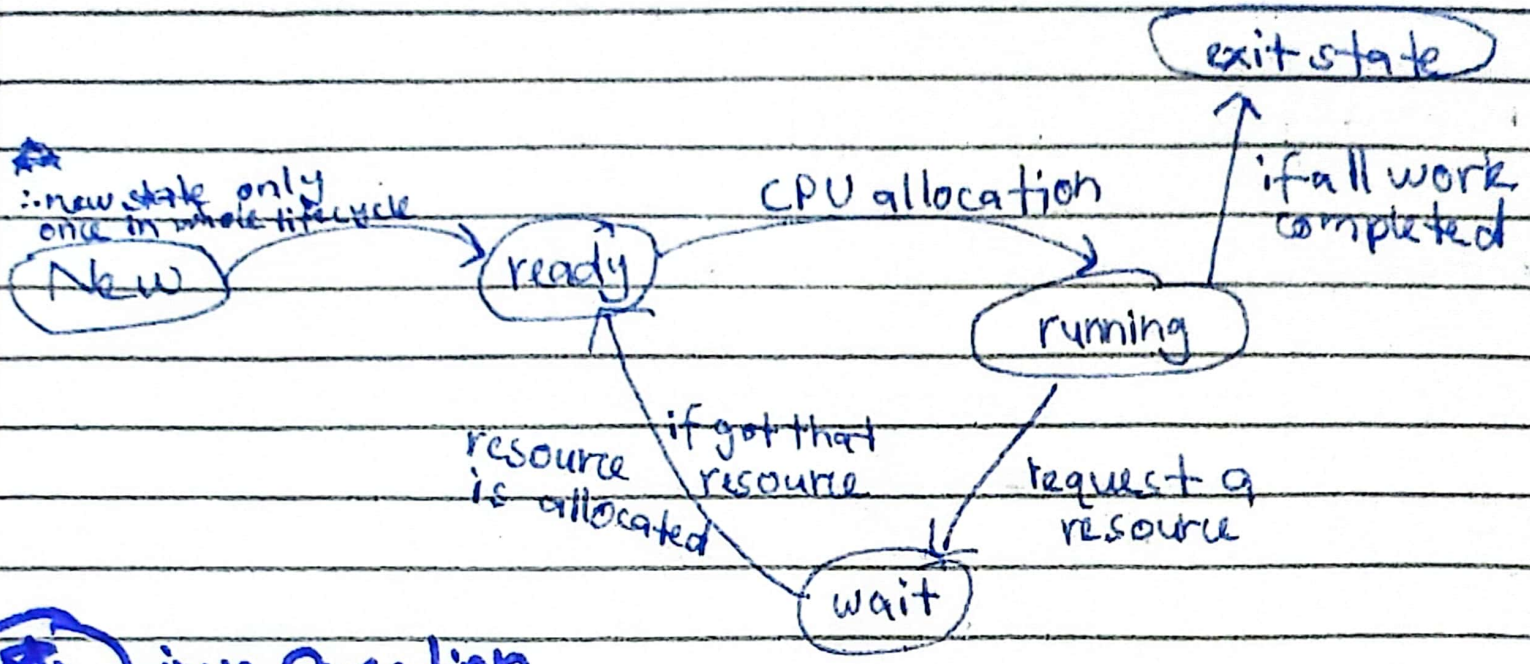
Diff b/w process & program

- process: if exe file runs in RAM
- program: .exe file on H.D.D

A static entity is called program.
Process is dynamic entity.
Program when executed & brought in main memory, it become process.

A process has five states in its life cycle

New state: when process created (got memory)
Ready state: Major part of process life in ready state. ie is ready to run. got resources
Running state: complete its work
wait state: if waiting for ~~a process~~ resources be freed
exit state: when process fully finished.



imp Question

Is this a valid transition?

wait → running

answer : NO

26 September 2024 Thursday

Interrupt Handling:

Two types of interrupt

- How OS handle interrupt

→ Interrupt vector table (data structure)

ID of interrupt	Systemcall

- OS never allow allows software at application level directly access hardware.

  - in DOS(CLI), it was allowed but modern OS don't allow it.
  - in DOS, if program get error, OS also got off.

- OS provide **system call**. They are collection of functions provided by OS to give access of hardware to the application programs.

∴ POSIX ∴system calls collection for POSIX
∴ WIN 32 ∴name of system calls in window

∴System calls are very complex so we use sugar coated.
∴ WIN 32 API

/ Next page

ID	system call Function
0	
1	
2	
⋮	
⋮	
⋮	

Interrupt vector table is a DS maintained by os for handling interrupts

- **Block devices**
- **character devices** (data read/write character by character)
  - ↳ All these devices that read data char by char. eg keyboard, NIC
- **Block devices** read or write data in the form of blocks
  eg usb, HDD, ram

Synchronous vs Asynchronous I/O

Synchronous without thread,

↳ When an I/O operation is performed, the program waits for completion of that I/O operation before moving forward.

- In **asynchoronous I/O**, when an I/O operation is performed, that operation is executed in a thread so the program can continue without waiting for completion of I/O operati

imp

Dual mode of operation:

Two

∴ Mode bit added in hardware

User space — ∴ mode bit = 1 if CPU give process to user
& = 0 means process in not of user space.

kernel space

imp question

Q: Can we sort dual mode of operation is S.W?
How can we implement dual mode of operation without H.W support

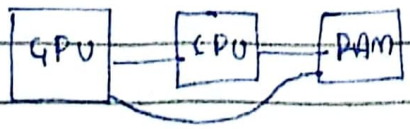
Ans: We can't implement dual mode of operation in S.W & need H.W support like mode bit.

DMA (Direct memory access):

∴ GPU have own CPU / processor (graphics card)

∴ every thing that needs to be processed is need to be brought to ram first.

∴ Devices that bypass CPU & directly access ram & have own CPU.



/Next page

imp

:How DOES use of DMA makes our computer fast?

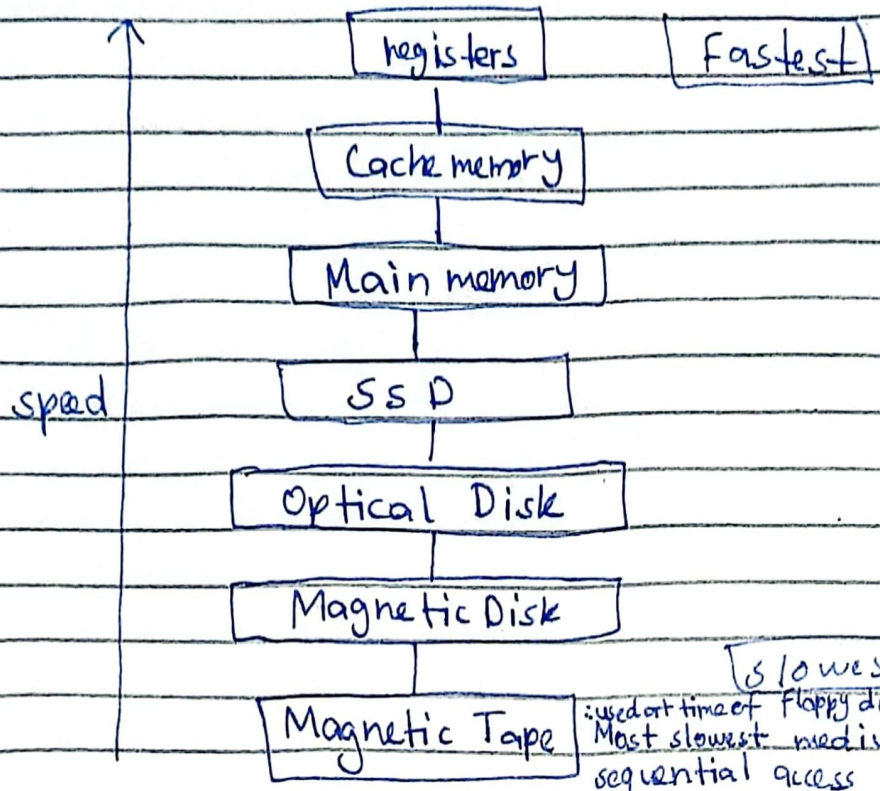
:DMA allows a device that has its own processor to bypass the CPU & directly access/process data in RAM.

∴ DMA is a phoenoma/concept. Not H·W or S.W.

2 October 2024 Wednesday

Storage Structure:

∴means memory hierarchy (speed)



⊙ Magnetic Tape: most slowest with sequential access. Cheap medium. obselete

⊙ Magnetic Disk: Hard disk(magnetic type & not ssd) more speed than magnetic tape. 5200 & 7200 rpm. slow memory.

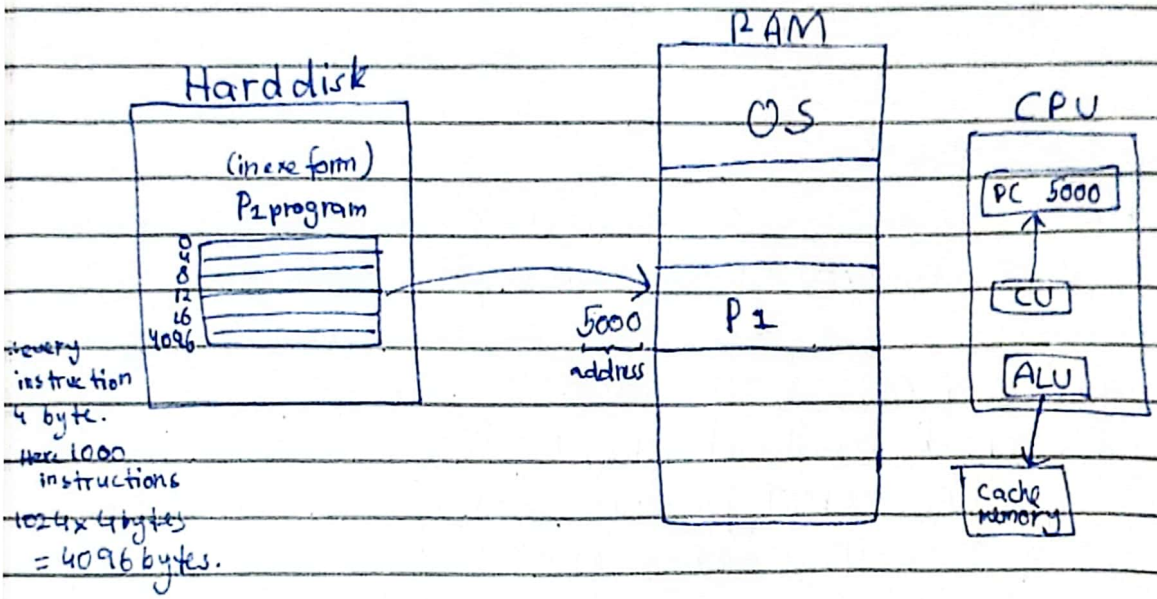
⊙ optical Disk: CD (100mb storage) & DVD (4GB of data storage)

⊙ SSD: All modern use SSD. usb is also a SSD. It is random access. It is very fast

⊙ Main memory: fast than SSD. (RAM)

⊙ cache memory: Present on Motherboard. It never change. It is slow than registers.

⊙ registers: Work at speed of CPU. They are the fastest memory available



∴ when we run program, it is copied in RAM. instarting addresses of RAM, there is always OS present

∴ 32 bit OS = 4 byte each instruction
∴ 64 " " = 8 byte " "

∴ PC (program counter) register has 5000 address of RAM.

∴ RAM very slow as compared to CPU

∴ When CPU run an instruction, it check the instruction whether it have it or not

CU first cache memory first. If cache memory empty, it goes to RAM & copies all instruction to RAM.

Address instructions

Cache	memory

999 times CU will see cache memory to get data.
~~before~~

imp Question ★★

Q: How use of cache make our computer fast? How cache work? What is cache memory?

A: cache memory works in between RAM & CPU. CPU checks all the instructions before execution in cache memory. If it doesn't found that instruction in cache memory, It brings the complete segment into cache memory. RAM very slow as compared to speed of CPU

∴ cache memory always in mb. It is not in GB as if in GB, then there will be no use of RAM.

∴ Program when load into ram, it become process. CU fetch instruction, ALU decode & then it execute.

∴ CU picks some segment of instructions from RAM & keep in cache

Next page

→ Time sharing
→ multiprogramming

A **multiprogrammed OS** is one that executes more than one program at a time. Multiprogramming is achieved by time sharing.

A **time sharing** means that we are dividing time between multiple processes

∴ can we achieve multiprogramming using single CPU?

OLD DOS OS were not timesharing. They were uni programmed. Now Latest OS are multiprogrammed.

Services of an OS

1) Process management: (things need to performed in a particular order.)

to create, delete of process, synchronization b/w processes, handling priority of processes, allocate resources to processes.

2) Memory mangement: ∴ services of as w.r.t services of OS

allocating & deallocating memory to processes, managing access rights of memory, managing virtual memory

3) Storage management: (Hard disk)

Files creation & deletion, directories creation

& deletion, managing access rights of files & directories, managing partitions of storage (partition & formating are same eg NTFS, FAT32 etc)

4) Protection & security:

Protecting application programs from accessing hardware, managing & securing registry, monitring incoming traffic via ports

connecting point eg TCP IP port etc

3 October 2024 Thursday

Passing Parameters to system calls

2. What are system calls?

- collection of function provided by OS to give access of H.W to application programs.

∴ etc win32 is system call in window, posix in Linux

Application Program

operating system

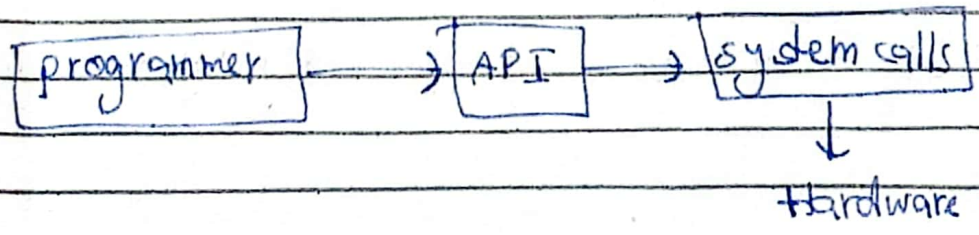
Hardware

Standard header file is provided by language. It is part of language documentation

sugar coated methods

API (application programming interface) written by language developer. At backend they use system calls that access hardware. And programmer use these API.

more than 90% programmer use API.



System calls can also be used by programmers but API used most.

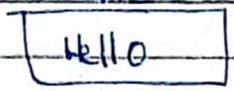
test.c

```
printf ("Hello");
```

∴ when we process this data through system calls; we do

Ways to pass parameter to system calls

(1-) Using registers for passing parameter to system calls.



system call take this data & display on monitors.

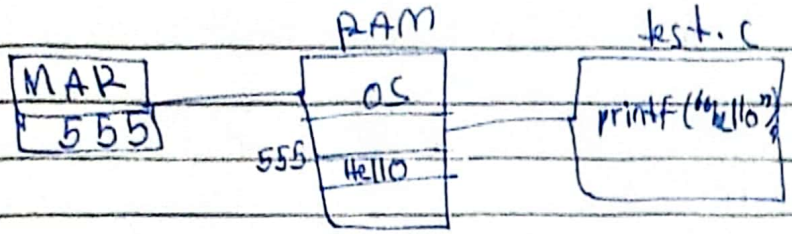
Using registers approach is obselete technique because of size limitation.

(2-) Using block/table in memory

∴ take data from program to give to system call, go to ram, take data address, put address to MAR register & tell system call of MAR. It will go in MAR, get address, go to RAM & get real data.

∴ Linux use this technique for system calls

∴ can use complete ram



3-) Using stack (used in windows)

∴ every process has own stack called call stack
∴ every OS has its own stack. Talking about this here. OS push parameters in stack & pop when needed.

Types of System calls

(1) Process control

∴ type of system calls used for process control

↳ end, abort
- ↓ for normal termination
- ↓ for abnormal termination (eg error in program

↳ load, execute
- ↓ for program to RAM from H.D.D
- ↓ to run program

↳ create process, terminate process
- ↓ to create a new process (create child process)
- ↓ terminate child process

∴ in every OS, process has ability to create multiple **child process** called child process

↳ get process attribute, set process attribute

- every process has unique id in OS
- then priority
- then its child processes
- then its parent process
- no. of file handles (file opened, used by process)
- memory allocated to process

} attributes of process

Kernel process is only process that not have parent

↳ wait for time

↳ wait for event

(2) File management : cursor concept in file

↳ file creation & deletion
↳ open, close
↳ read, write, reposition

when we open a file, a pointer provided to us. ~~of~~ gives functions to reposition cursor ie where to write

File handling — imp question for interview

↳ get/set file attributes
- ↳ name
- ↳ directory
- ↳ permission
- ↳ space
- ↳ extension
- ↳ last modified date

(3) Device management

∴ in linux, every device is connected to system is treated as a file

↳ request device, release device
(attach device)

↳ read, write, reposition

↳ get/set device attributes

↳ logically attach device, logically detatch
(device) (eject usb etc.)

(4) information maintainance

↳ get/set system date/time ∴ cmos battery

↳ get/set system data ∴ every info of system devices etc

↳ get/set file attributes

↳ get/set process attribute
(ctrl+alt+delete)

(5) communication

↳ create/delete communication channel

∴ communication based on 2 things
↳ IP address (of both systems)
↳ port number (this is necessary) (of both systems

↳ send & receive messages

↳ transfer status information

∴ persistent & non-persistent processes

persistent: keep connected until ip change

non-persistent: need to reestablish connection again after g time

↳ attach / Detach remote device

∴ connected ~~connected~~ onserver

9 October 2024 Wednesday

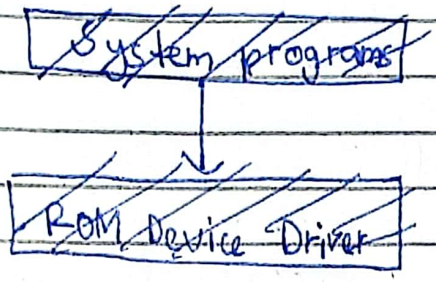
Operating System Structure

∴ How ~~execute~~ organize OS?

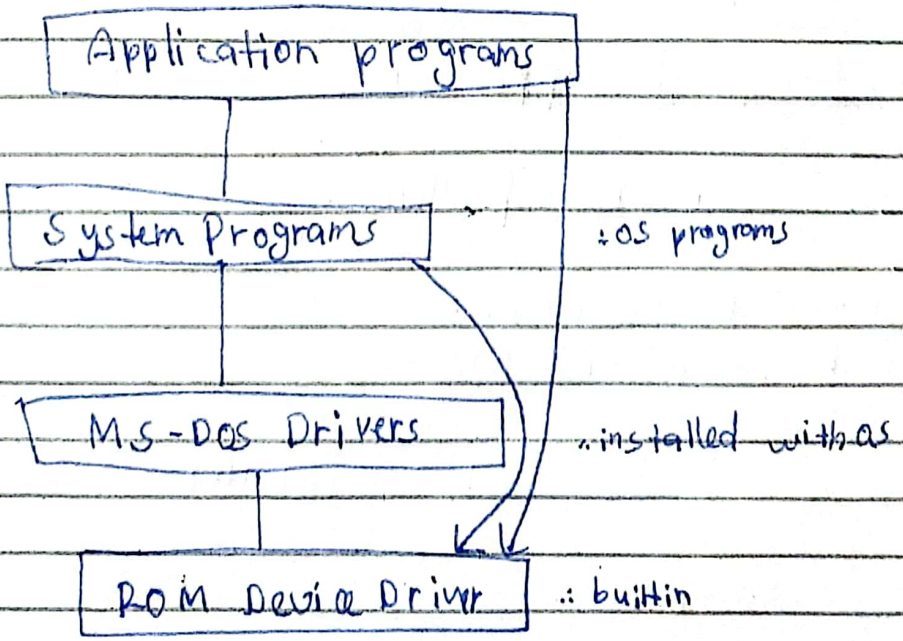
∴ OS has million lines of code so not possible in a single file.

in early ~~days~~ days of OS:

1-) **Simple structure** ∴ obsolete of now



∴ MS-DOS was one of earliest OS. CLI OS. Its structuring was below.



∴ Here, application programs directly accessing hardware & its a problem.
∴ error in application program cause BSOD device driver to corrupt.

Drawbacks

- In simple Structure, if there was a error in a user program, the complete OS used to shutdown.

For eg; runtime error cause program to shutdown in now latest windows but old OS error in program caused OS to shutdown.

- In this version, there was only UNIX programming. If we had to execute a 2nd program, then the first program had to be shut down.

This approach is obselete.

2-) Layered Approach — debugging easy, space issue & Dependency issue

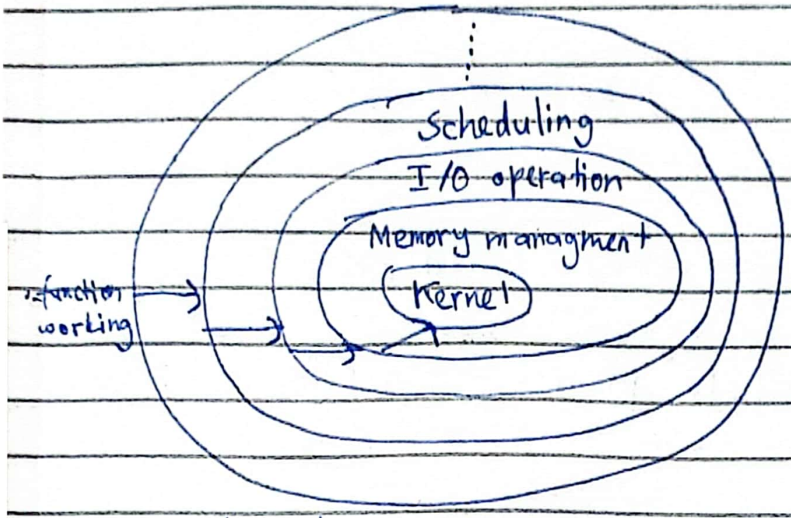
Design pattern

∴ how to organize code?
Most common is MVC

View	Model	controller
⊙ UI related work	⊙ DB related ie backend	⊙ Business logic

- its benifit is that can independly change code/model in long-term.

- In layered approach



ie code divided layer by layer

- The whole purpose of layered approach was to make debugging easy
  (debugging → error.)

∴ It was 2nd era of OS.

1- easy debugging of OS code was benifit
2- takes too much time ie slow was drawback
3- Dependency issue ie which layer should depend upon other layer. a drawback. They layer dependency was an issue that which layer should be at which level
∴ layer most near to kernel receive most requests & vice versa

4-

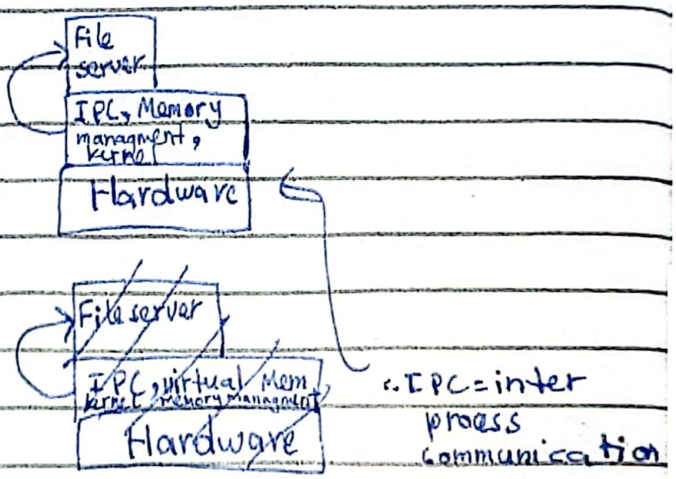
3-) Microkernel Def in Mcqs imp

- Chota kernel
- Non of software or H.W can be run without OS.

Microkernel made for embedded system ie Lift, projector etc.

every device for which we embed software.

- Microkernel is pruned small size operating system



- Necessary components are integrated in the OS. And the rest of the components can be customized.

eg Tiny os

/Next Page

4-) Module/component based

∴ Nothing can be executed if this is not compiled.

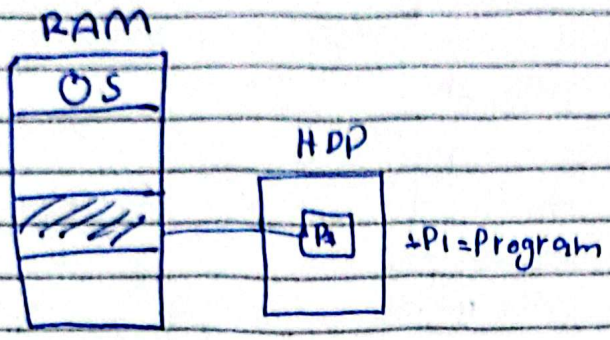
~~This is~~
~~In Module based~~

Module is combined set of classes.

Component / Module is a reusable compiled set of classes. Usually a module / component is identified based upon reusability.

16 October 2024 Wednesday

Segments/Sections of a process :



*program is in ram

1) Text section of a program contains a compiled version of a ~~process~~ program
segment

2) Data section contain static/global variables
retain its life during program running

: two type of variables
↳ local : confined to braces { }
↳ global : declared without scope, out of main()

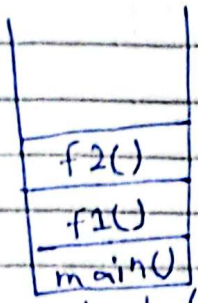
. Data segment of process contain all ~~static~~ /global variables

3) Stack section used to maintain call stack / function stack

```
void main() {
    f1();
}

f1() {
    f2();
}

f2() {
    ---
    ---
    ---
}
```



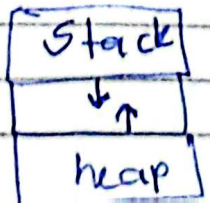
stack (function stack)

∴ The function at top of stack is the one that is actually been executing

∴ function stack = call stack

4) Heap contains dynamically allocated objects ∴ or dynamic allocation of objects

5) Stack & heap grows in opposite directions. Every recursive function increase stack rapidly, if stack become toolarge, it overwrite heap & cause stackoverflow error.



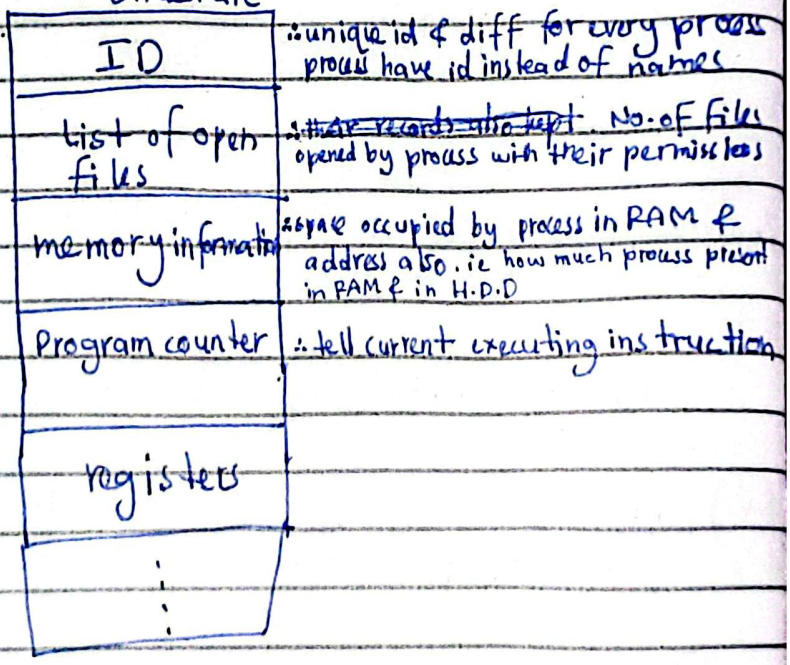
Stack overflow is a situation in which either the stack (long recursion) or heap (large objects creation) increases rapidly & overwrite each other

∴ using loop to create multiple object ie thousands of them cause heap to overwrite stack causing stackoverflow.

Process control block (PCB)

∴ like idcard of a process.

PCB structure



⊙ Registers are physical storage.
⊙ PCB is virtual. It has virtual registers

Q: Why need of registers in PCB, if we already have physical registers in CPU?

Ans: We have multiple processes. They get paused so we use virtual registers to save physical register values in virtual so they value remain safe.

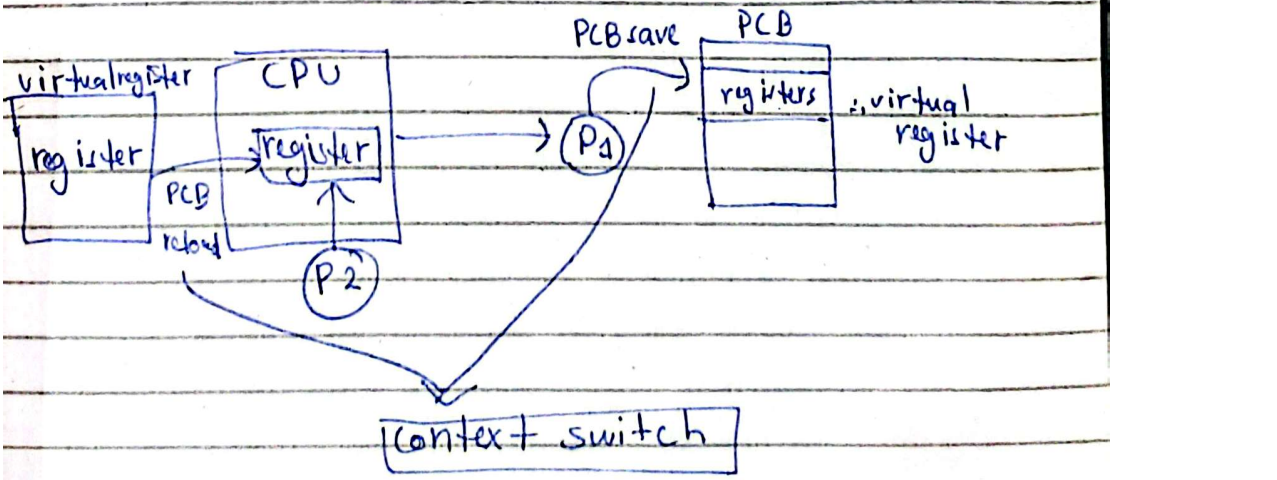
PCB save means that the process that is leaving the CPU all of its register value are copied in to its PCB.

PCB reload means the process that is coming for execution, its registers values are copied from its PCB into the physical registers.

Both of PCB reload & save are context switch.

Context switch is combination of PCB save & PCB reload

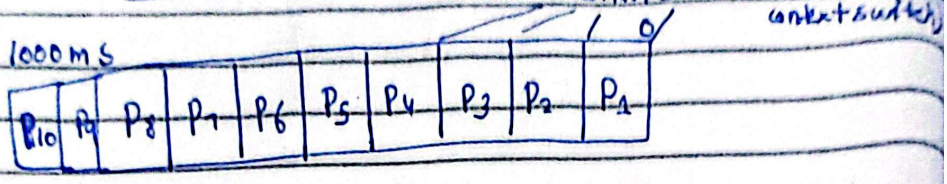
∴ context switch should have min time



Next Page

Q: We have 10 process. each process context switch = 10ms. What will be efficiency?

Ans: 1 sec = 1000 ms Assumption



Gantt chart

Wastage = 100 ms

so efficiency = 900ms = 90%

Context switch is always considered wastage

17 October 2024 Thursday

Schedulars

↳ Type of Schedulars

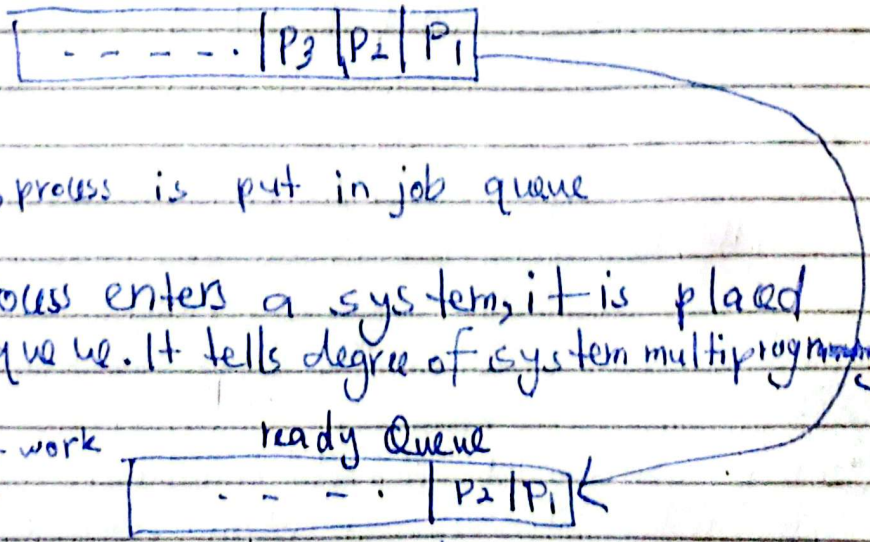
1) Long term schedulars
2) short term schedular
3) Middle / Medium term schedular

- **Schedular** is process of OS that uses a scheduling policy to decide which process will be executed next on cpu

⊙ Dispatcher perform context switch after schedular tell us of next process to be executed.

- **Dispatcher** is a process that basically perform context switch (PCB save & PCB reload)

∴ OS maintains some queues



Every resource will have its own queue called wait queue

Two type of process

1) CPU bound process :. example video processing (:. it keep processing in CPU)
2) I/O bound :. eg SQL s.w / DB s.w

:. Process mostly use CPU during its execution is **CPU bound**. More time spent doing computation.
:. Process that rarely use CPU called **I/O bound**. Any s.w in which bulk amount of data read & write to HDD. More time spent doing I/O operations.

1) Long term schedular

In 1990, now obselete.
They were used for batch processing (bulk amount of work/processes bound)

:. if most processes CPU bound, then I/O processes will be very less. So 50 50 ratio maintained.

It is only schedular that work on HDD.

It is an old schedular which is now obselete. It was used in batch processing. It

was slow. It used to work on H.D.D. It used an intelligent algorithm to maintain ideal ratio of CPU bound & IO bound processes (50% 50%)

2) Short term schedular (usually refer as schedular)

work on ram.

Short term schedular is our main schedular that works on ready queue. It works very fast. It cannot use intelligent algorithms to maintain ideal ratio.

∴ ready queue is in ram

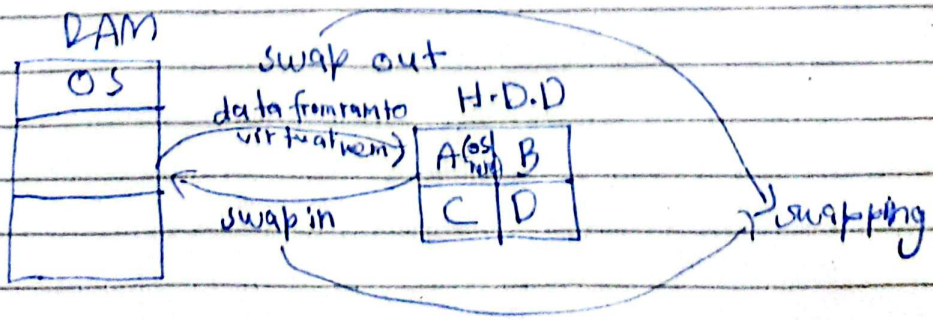
write long, short & middle

Q: What are types of schedular &
Q: What are diff categories of schedular

write categories of short term schedular

3) Medium / Middle term schedular

bridge b/w H.D.D & RAM



~~when format a partition~~

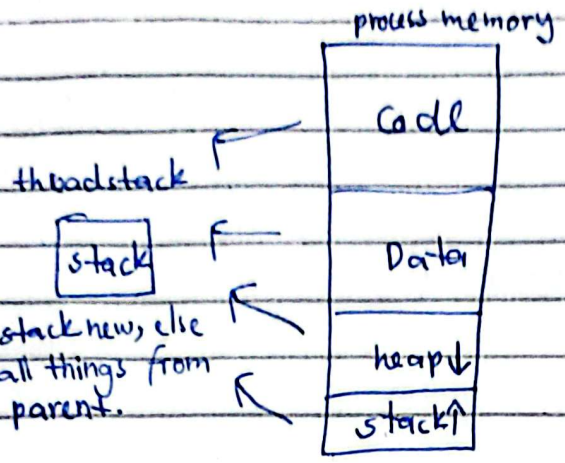
- **Virtual memory** is a memory that is not our main ram. It is a space that is allocated on H.D.D & used as main memory. It has no file filesystem. It has raw drive space
- Virtual memory space is in OS disk. This space has no partition format. Virtual memory used if RAM completely full.

transferring

- Putting data from ram to H.D.D is **swap out**
- Transferring data from H.D.D to ram is called **swap in.**
- **Swapping** is combination of swap in & swap out
- Medium term schedular perform swapping. It is called swapper.

23 October 2024 Wednesday

Thread



A Thread is called a lightweight process. When thread is created, OS share the main process all 4 segments & give a stack to stack. Every thread have own stack. Memory used of its parent.

A **thread** is called lightweight process as it shares memory segment, heap & data segment of its parent process.

∴ Benifit of thread

∴ 1 processor & 2 threads at same time have no benifit.

In latest processor, thread runs at parallel in processor with main ().

↳ responsiveness is benifit

user initiated thing should not get stuck

Benifit of thread

① Threads are created to increase responsiveness of our program. the more thread we create, the more time responsive our program becomes.

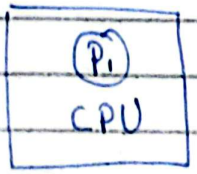
∴ Every Programming lang have thread limit. eg in C, we can't create more than 15 programs threads in a same program.

Q: What is responsiveness?

A: Responsiveness means if we click any button/ labels on an interface, the user should be aware of the system state

② What is premptive vs non-premptive scheduling?

∴ old 1990 OS were non-premptive.



∴ if we take CPU back from P_1 any time, it is premptive. if P_1 give it back on own, then non-premptive

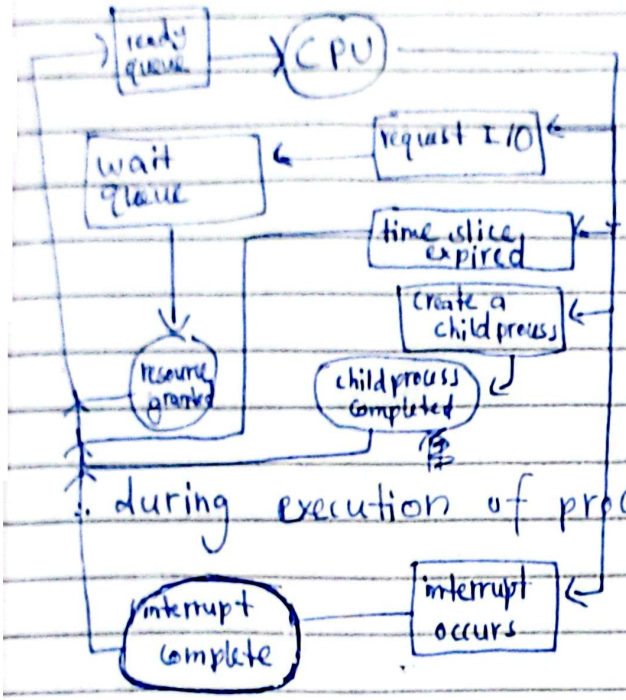
All modern OS are premptive. Every

In premptive scheduling, we can take CPU back ~~of~~ from the process at any time. All modern OS use premptive scheduling

imp ***

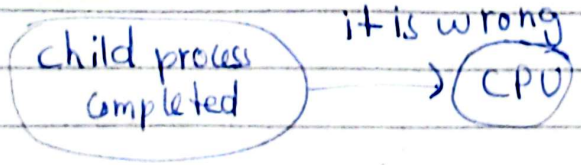
Q: What are diff scenarios in which a process executing on a CPU leave it?

Ans: This diagram
These 4 scenario represent preemptive scheduling



∴ during execution of process all 5 happen.

imp mcq



imp in lab · also 1 Q in lab paper (fork)

⊙ Creating a child process

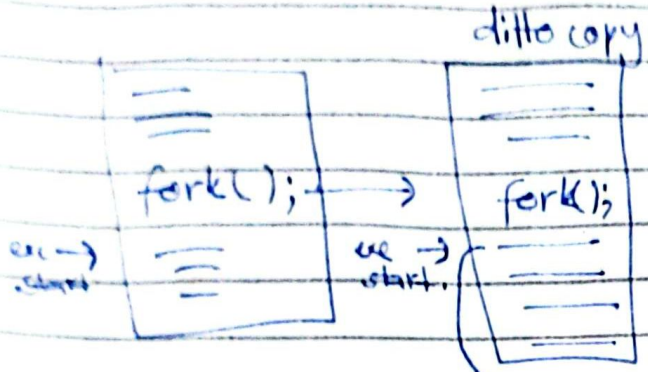
∴ scenario similar to lab

Fork is a systemcall that is used to create a child process. When we create child process, it creates a copy of process. This child process is useless until we use execvp().

fork();

fork can return 3 values

- -ve (child creation was unsuccessful)
- 0 (child process) } sucessfully created
- >0 (parent process) }



∴ program which ret 0 means child process. >0 means 1.

→ execvp() → executable / Binary / compiled file

```
int i = fork();
if (i == 0){
  //child process
    execvp(); //use execvp in child only so use in condition
  }
  else {
    //parent process
  }
```

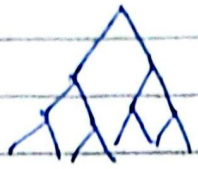
/Next page

total 8 childs

$2^3 = 8$

```
fork();
fork();
fork();
```

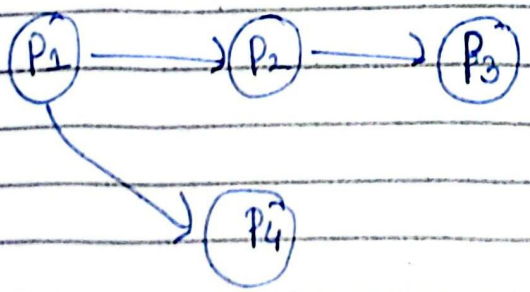
} fork bomb



if we donot put condition on fork, it will increase exponentially

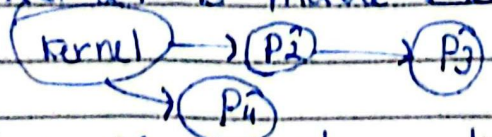
⊙Cascading termination:

∴ cascade (sequence mai khatam kardena)



In old OS, if a process was terminated, its child processes were also terminated. This phenomena is called cascading termination.

A child process was not allowed to exist without its parent. Modern OS donot support cascading termination. in Modern OS, if process is terminated, kernel is made ~~their~~ parent of its child.



∴ Does modern OS support cascading termination?
Imp in lab using code

24 October 2024 Thursday

IPC

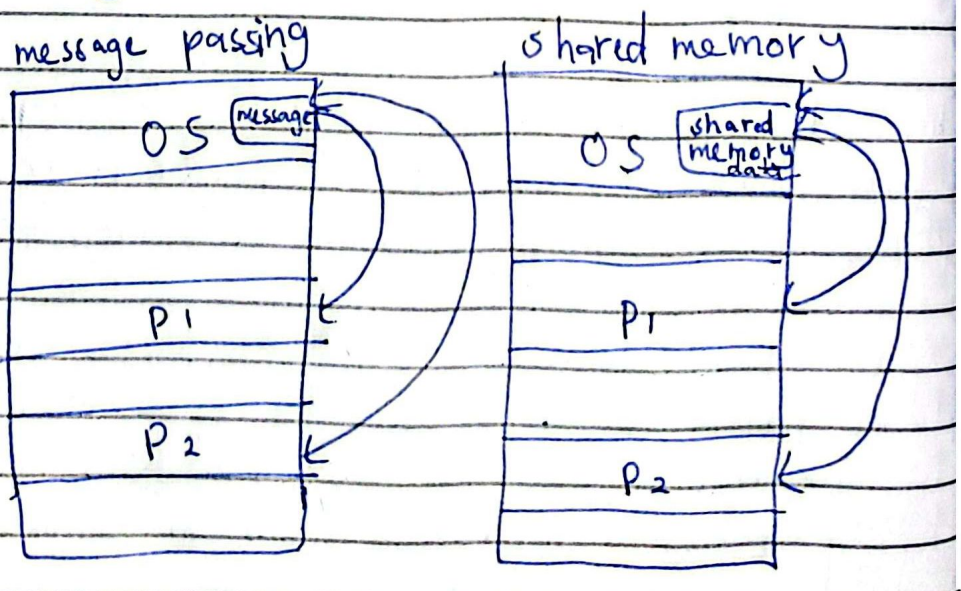
interprocess communication

- Two models in IPC
  - ↳ independent process
  - ↳ cooperating process

~~∴ if process performs~~

- A process, if it communicate with some other process during its life is called a **cooperating process**
- A process that doesn't communicate with other process during its life is called **independent process**

cooperating process have two types of models



∴ normally a process can only access its own memory else OS kill it.

∴ in shared memory, P_1 store its data in shared memory in OS & then P_2 reads its. This memory shared in both processes.

Shared memory is fast. It is usually used for large amount of data communication. Shared memory have overhead. The overhead is that allocation & deallocation of memory of the process is responsibility.

- In **message passing** every message is transfered through OS. It makes it's slow.

∴ OS not involved in shared memory. OS involved in

- For distributed processes, we have to use message passing.
  eg facebook chat
  ∴ in distributed system, only message passing can be used
- For small amount of data, we prefer to use message passing. For large amount of data, we prefer to use shared memory.

classical problem

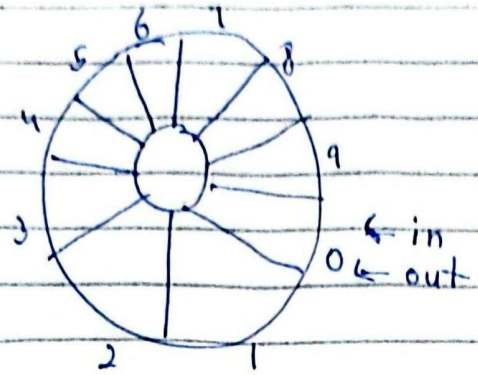
Producer consumer problem

∴ classical problem
old example

∴ two process
- ↳ producer
- ↳ consumer

buffer is a shared memory

Circular buffer of size 10



∴ producer put things in buffer, consumer use it

∴ two pointer
- ↳ in (used by producer) ∴ point to next empty space
- ↳ out (used by consumer)

shared memory

```
int in=0
int out=0
xx define buffer_size =10 //constant
int buffer[buffer_size]
```

It is discourage to directly we value so use constant.

Q. Condition to check buffer empty or full?

Ans:

Empty

in == out //(if buffer is empty)

Full

//use mod operator when in going to 0.

(in +1) % buffer - size = out

Now, code

a consumer process will not consume anything ~~until~~ if buffer is empty

Producer

```
while((in+1)% buffer_size=out)
   ; //empty statement
//if buffer not full so,

{Buffer[in] =item;
 in = (in+1)% Buffer size;

//while loop keep running if buffer
full & empty statement
keep runing
↳ These run if buffer empty
```

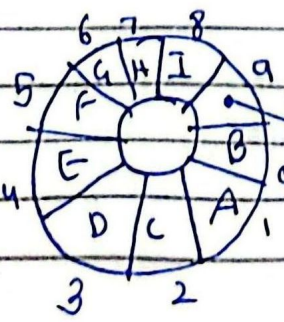
```
while (in = = out)
   ;
item = Buffer [out];
out = (out +1) % Buffer-size
```

① ISSUE in these codes

↳ end index will ~~on~~ always be 1 ahead wasted. if in = 9, then

(9+1) % 10 = = 0

0 = = 0 (here our last index always empty)



↳ This, wasted space & buffer considered empty.

```
int in = 0;
int out=0;
#define Buffer-size = 10
int Buffer [Buffer-size]
int count=0; //This added now
```

producer

```
while (count == Buffer-size)
    ;
Buffer [in] = item;
in = (in+1) % Buffer-size;
count = count + 1;
```

consumer

```
while (count == 0)
    ;
item = Buffer [out];
out = (out+1) % Buffer-out;
count = count - 1;
```

⊙ now this code will fully utilize complete space in Buffer.

30 October 2024 Wednesday

:preemptive scheduling : chaltay huy process say cpu wapis lena

Dispatcher:

① Dispatcher is a process that perform context switching.

Dispatcher have Dispatch Latency

Time taken by dispatcher to perform context switch

① Time taken by the dispatcher to perform context switch is called **dispatch latency**

How to say which scheduler is better in early days?

→ CPU utilization
→ response time
→ thlough put
→ Turn dround time
→ waiting time

CPU utilization means how much CPU was used in % during the process life.

eg how much a time process we cpy in 60 sec. It should be as high as possible.

② The time when a process was submitted for execution till the time it was first executed. We want **response time** to be as low as possible.

eg; how much wait for first time to run

⊙ The no. of task completed per unit of time is **through put**. We want through put to be as high as possible.

eg, 10 students
1 student viva = 6 min

Scheme 1	Scheme 2
every student give viva of 6 min	every student give viva of 3+3 min
⊙ After 30 min, what will be through put of this system? = 5	→ here = 0 in first 30 min.
response time = 54 for last student	response time = 27 for last student.

⊙ The sum of all the wait times is **waiting time**.

∴ every process has own waiting time.

⊙ The time when a process was submitted for execution till its completion is called **turnaround**

∴ shuru se akhir

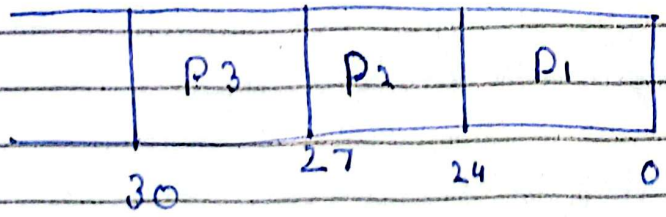
Numericals | 1 Q in paper

First Come First serve (FCFS)

↳ it is non-premptive scheme

process	Burst time (how long will process run)
P_1	24
P_2	3
P_3	3
	30

Gantt chart



now finding wait time;

$P_1 = 0$
$P_2 = 24$
$P_3 = 27$

Now their Avg $= \frac{51}{3} = 17$

∴ Due to first process (time), rest have to wait.

convoy effect

When a process having small burst time has to wait because of a process

having large burst time, then this problem is called **Convoy effect.**

so we do;

Process	burst time
P_2	3
P_3	3
P_1	27
	30

so wait time;

$P_2 = 0$
$P_3 = 3$
$P_1 = 6$

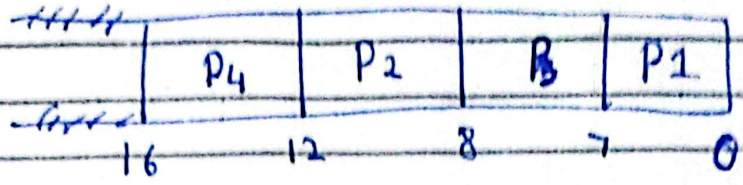
$\text{avg} = \frac{9}{3} = 3$

Shortest job first (SJF)

↳ Non premptive

	arrival time	burst time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

∴ in shortest job first, run process with shortest burst time



wait time

P1 = 0
P2 = 8 - 2 = 6 ∴ arrival - starting time
P3 = 3
P4 = 7

avg = 16/4 = 4

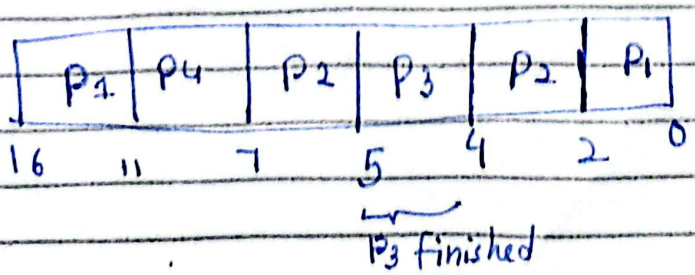
** impf difficult.

↳ Premptive (one process multiple time in Gantt chart)

∴ if naya burst time chota, run raya otherwise putane

	Arrival time	Burst time
P1	0	5
P2	2	2
P3	4	1
P4	5	4

∴ dotted line



Now
wait time

P1 11-2=9

P2 = 5-4=1

P3 = 0

P4 = 7-5=2

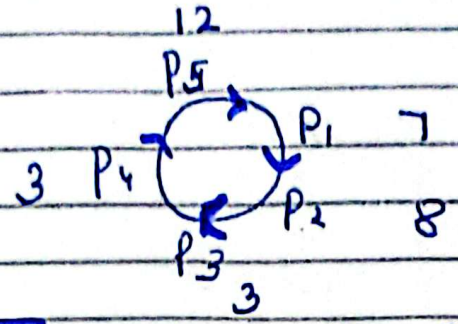
avg = 3

SJF give most less avg less time. it is most optimal scheduling scheme

13 November 2024 Wednesday

Round Robin (RR)

∴ execute processes in circular manner



- **Quantum** is the unit for which we max run a process in a cycle.

∴ every process given max time of quantum in a cycle.

eg; Quantum = 5

	Burst time
~~P1~~	~~7~~ 2
~~P2~~	~~8~~ 3
~~P3~~	3
~~P4~~	3
P5	~~12~~ ~~7~~ 2

P5	P5	P2	P1	P5	P4	P3	P2	P1

33 31 26 23 21 16 13 10 5 0

wait time

P1 = 21 - 5 = 16

P4 = 5 + 13 = 18

P3 = 10

P4 = 10

P5 = 16 + 5 = 21

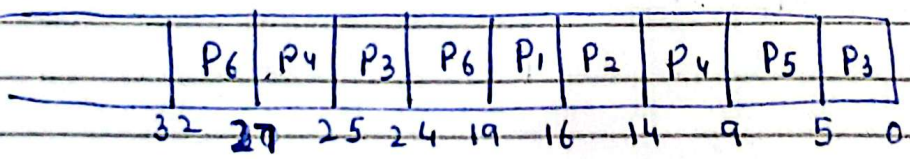
∴ in paper, data will be same as midterm question

Quantum = 5

Process	Arrival time	Burst time
P_1	4	3
P_2	2	2
P_3	0	1
P_4	1	2
P_5	0	4
P_6	5	5

: Here P_3 at first in RR as index short.

P_3
P_5
P_4
P_2
P_1
P_6



$P_1 = 12$
$P_2 = 12$
$P_3 = 19$
$P_4 = 19$
$P_5 = 5$
$P_6 = 19$

- in RR, Quantum duration get changed by OS by passage of time.
- If Quantum made too much high, it get start behaving like FCFS.
- If Quantum very small (eg; Q=1 & context switch = 1ms), then system efficiency become 50%.

imp

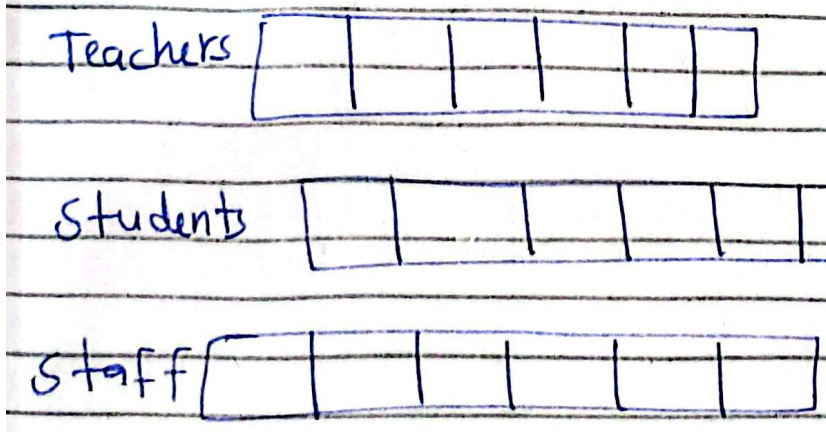
Q: What is impact of increasing / decreasing Quantum in RR?

Ans
- if we increase quantum too much, it start behaving like FCFS.
- if we decrease quantum too small (eg 1 ms), then CPU efficiency decrease a lot.

Multi level Queue

(Theoretical in paper. No numerical)

∴ combination of prev schemes.



Here to implement scheduling, we have;
1- absolute priority
2- Proportional allocation

2nd ✓

∴ in absolute priority, 3rd queue may never run if first 2 queue keep get processes & not get empty.

⊙ **Starvation** is a scenario in which a process never gets a chance to being executed

∴ process never get CPU

∴ in proportional allocation, we divide total time b/w all queues.

eg total = 1000 ms

teachers - 500ms

students = 300ms

staff = 200ms

∴ in proportional allocation, there is no starvation.

imp

Q: Theoretical question:

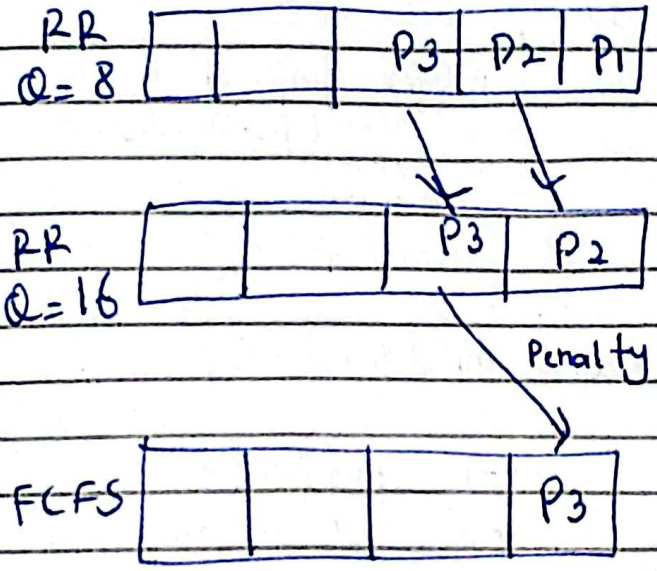
Ans: total no. of users = queues made

14- November 2024 Thursday

Multi level feedback queue scheduling

(No numerical. Theory only).

∴ Here first decide no. of queues regardless of no. of users



∴ every process at first go in queue 1. if not completed, go to 2 then 3.

eg;

	Burst time		
P3	= 30	~~22~~	6
P2	= ~~12~~	-4-	
~~P1~~	= 5-		

Queue 1 Queue 2 Queue 3

given

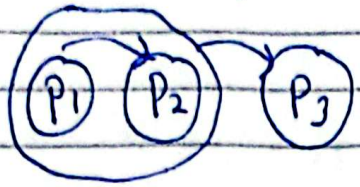
∴ Here processes' favour whose § burst time is less.

• A process is allowed to change its queue so it is called **feedback queue**

Process Synchronization

① If a process execution is dependent upon output of some other process, then they should be executed in a particular order.

eg;




```
#define Buffer-size 10;
int Buffer ( Buffer-size);
int count=0;
int in,out=0
```

Now;

Producer	consumer
while(count = Buffer_size) ;	while(count = 0) ;
Buffer[in] = item;	item = Buffer[out];
count = count +1;	out = (out+1) % Buffer-size;
in = (in+1) % Buffer-size	count = count +1
→ critical section	

∴ assume count = 4

C	P	} running order of producer & consumer.
P	C	
4	4	: count value

∴ performing +1 - in register

② Assembly code

∴ register

S1 R1 = count;
S2 R1 = R1 + 1;
S3 count = R1;

count = count + 1

T1 R2 = count
T2 R2 = R2 - 1
T3 count = R2

count = 4 initially.

S1	S1
S2	S2
T1	T1
T2	T2
S3	T3
T3	S3

Tell count value ...

3 5 = count

- When we create multithreading application, we need to make sure if there is shared data, we should use Locks
- When two or more than two process access & manipulating shared data & the output becomes unpredictable, then this problem is called **Race condition**

∴ Critical section is race condition solution.

- The part of code where shared data is being manipulated is called **critical section**

This line — entry section
critical section
This line — exit section
}; — remainder section

- The code just before the critical section is entry section. Just after critical is exit section. Code after exit section called remainder section

3 things in sol for critical section:

~~Citeria for critical section problems solutions:~~ only two in write

1- ~~Bounded wait~~
2- ~~progress~~
3-
5-

1- Mutual exclusion
2- Progress
3- Bounded wait

P1	P2	P3
C.S	C.S	C.S

∴ in all P1 P2 P3, C.S pointing to ag same variable counter etc.

- **Mutual exclusion**: only process can be in its critical section at any given time
- **Progress**: There should be no deadlock in system / system should not halt.
- **Bounded wait**: There should be a limit on how many times a process is allowed to enter in its critical section.

@ synchronized {~} //in java

it make sure no race condition happen.

20 November 2024 Wednesday

Peterson solution for critical section problem: (code related Q in paper)

∴ Peterson was 1st person to propose critical section problem.

Q: It was limited to only two processors.
↳ it was a S.W based solution.
(used variables only.)

∴ Peterson solution don't work in reality. We study it only as it was its sol.

Q Does peterson sol worked.

Q: Why it never worked

Ans: ↳ it was a s.w based solution so it itself ran into race sol. (its solution itself ran to race condition)

Shared memory

```
int turn;
Bool flag[2]
```

: 0 mean P0 turn & vice versa ready
// flag[0] = true means P0 is ready & vice versa

P0	P1
do {	do {
flag[0] = true;	flag[1] = true;
turn = 1;	turn = 0;
while (turn == 1 && flag[1])	while (turn == 0 && flag[0])
;	;
Critical section	C.S

//Bari dusry process ki thi aur wb khali bhi tha
//exit section of c.s
flag [0] = false
} while (1)
	flag [1] = false;
	}
	while (1)

now; let initial conditions be;

```
int turn=0;
flag [0] = true;
flag [1] = true
```

now;

```
turn=0;
flag[0] = false;
flag [1] = false;
```

Q: How does peterson sol satisfy critical section sol condition?

Ans: We have a variable of int type called turn. It can have one value either 0 or 1. When it have value 0, P0 will go into its critical section & when have val 1, P1 go to its critical section. This way it satisfy mutual exclusion

Q: A process wait get how many times after its turn?

Ans: it has bounded wait of 1

A process in its exit section always allow the other process to enter its critical section. It is its progress. Hence there is no deadlock

∴ Peterson sol don't work in reality due to its while statements.

It is a S.W based sol.

Test and set condition :

(code in paper. maybe dry run or functions may be changed etc.)

∴ Was a first H.W based sol.
∴ Test and set is a instruction/function.

↳ It is a atomic solution

∴ atom is smallest particle.

① An instruction that takes one clock cycle to be executed is called **Atomic Instruction**

∴ normal instruction take 3–6 clock cycle.

```
Bool Testandset(Bool * target){
    Bool * rv = *Target;
    *Target = True;   //pointer always passed by refrence
    return *rv;
    }
```

, it takes only 1 clock cycle

// now let;

```
Bool * Test = False;
Testandset(*Test);
```

true/false

// jo bhej ty hyn, whi return hoga par orignal variable true hojaega.

Shared memory

Bool * Test = false

P0	P1
while (Test and Set (test)) { ; }	while (Test and set (test)) { ; }
C.S	C.S
* test = False ;	* test = false ;

P2

while (Test and set (test)) { ; }

C.S

* test = false;

⊙ it will work for more than 2 processes

∴ It works at it has atomic instruction.

☹ ∴ (whenever a process enters its C.S, it sets target = true for other processes so they keep stuck in loop)

28 November 2024 Thursday

No. of common OS
all OS total users

Title 1pg 2pg report 1 refrences (link)

Quiz 1

04-12-24

Everything covered

- Process synchronization
  - ↳ Race condition
  - ↳ Critical section
    - ↳ Sol for C.S
      - ↳ Peterson sol
        - ↳ Test and Set

- **Semaphores** are variable that are used for process synchronization. They have 2 operation. wait & signal. Both are atomic instruction.

```
eg; int a;
    a+10;
    a-10;
    a*10;
    a/10;
```

Semaphore S; (only 2 function for it)

This code imp

```
wait(s);
signal(s)
```

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

```
signal(s){
    s++;
}
```

// wait check semaphore value.
// both function are built-in.

example:

Semaphore s = 1;

P1	P2	P3
wait(s);	wait(s);	wait(s);
[C.S]	[C.S]	[C.S]
signal(s);	signal(s);	signal(s);

// wait & signal are atomic functions.

∴ if Semaphore.S = 2; two process go in their C.S.
∴ if S = 3, then 3 process in C.S.
∴ it violate mutual exclusion.

↳ counting:
↳ Binary:

⊙ Two type of semaphore:
↳ Binary: always use Binary semaphore for critical section problem. It has only two values. 0 and 1.

↳ counting: can be any value

** imp in MCQ & short

⊙ Spinlock: a semaphore that spin while checking for condition

⊙ **Spinlock** is a situation in which a semaphore continuously check while condition & waste cpu time/cycle

```
while(S<=0)
    ;
```
it is spinlock in wait()

⊙ Solution is we make processes wait by putting in sleep

/Next page

```
typedef struct {
    int value;
    int process list;
} Semaphore
```

// now updated code (not imp for paper)

```
wait(S){
S -> value--;
if (S->value <0){
    Add the process
    to list
    block();
}}
```

```
Signal(S){
S -> value++;
if (S-> value <=0){
    Remove process from
    Queue
    wakeup(p);}
}
```

// S -> value means S.value in java OOP

Here even S = 2, then two processes will again go in (CS). So always start from S=1

// jo process arha hy, usay que ye may lag kar block kar do

// here ~~-ve~~ minus value tell how many process waiting for go to critical section

Sol for Spinlock process

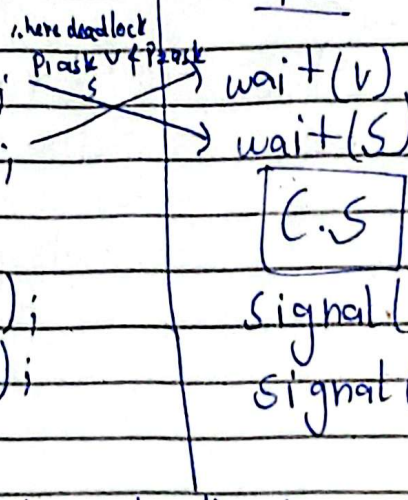
We put every waiting, in a queue & sleep/block it. Then when it's turn come, we remove from queue & goto C.S

imp subjective

Let we have 2 semaphores;

```
Semaphore S;
Semaphore V;
```

P1	P2
wait(S);	wait(V);
wait(V);	wait(S);
C.S	C.S
signal(S);	signal(V);
signal(V);	signal(S);



Here will be deadlock. both will be waiting

imp Real world example of deadlock using semaphore:

Student 1	Student 2
pencil	rubber

```
if
wait(S);
wait(V);
```

```
wait(S)
wait(V)
```

① It will not be a deadlock.

(*) from slides imp

Q: if we have 3 printers & two scanners, we want to allocate it among 5 employes such that each employe get a maximum of 1 scanner & 1 printer. Give its semaphore based solution.

Ans:

At any given time, max two employees go into C.S

Semaphore P = 3

Semaphore S = 2

S_0

P_1	P_2	P_3	P_4	P_5
wait(P)	. . .	. . . .	. . .	. .
wait(S)	. . .	. . . .	. . .	. .
C.S	. .	. . . .	. . .	. .
signal(P)	. . .	. . . .	. . .	. .
signal(S)	. .	. . . .	. . .	. .

11 December 2024 11th Wednesday

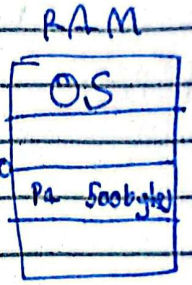
Memory management

↳ Base register - 1000
Limit register
↳ 500 bytes

∴ OS use two register to limit a process to its memory. Base register has process starting ~~size~~ add. Limit has ~~total~~ size

Every address that P1 is accessing is greater than base register & less than Base + Limit..

Base register < P1 < Base + Limit



Q: How does an OS prevents a process from accessing illegal memory

Base & Limit register are used for this purpose. base register ontain starting add & Limit contain its size. So range will be.
~~total~~.

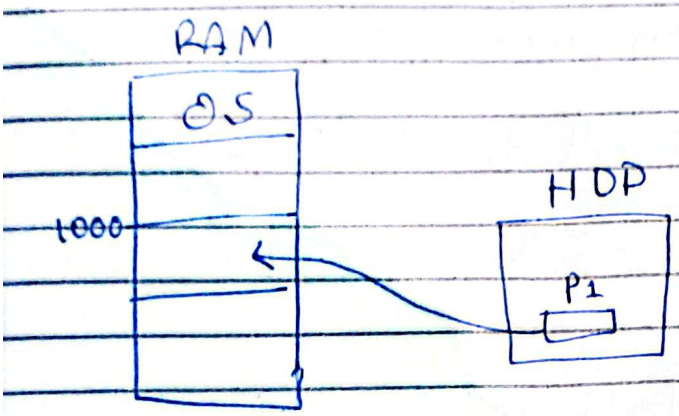
Q: What is purpose of base & limit register?
Ans: Same ans

One Question

Address binding:

Def

Assigning addresses to program in ram.



① compile time technique.

∴ A program has two phases
↳ compilation : means made its exe
↳ loading : means bring exe to ram

∴ in 1980, compile time technique used. Programmer had access of complete ram

∴ Jab compile hua, OS ny ram mai check kiya ye pura kahan aega aik sath.

Q:

Def: In **compile time address binding**, the OS checks the ram that if there is enough space in which that process can be loaded and assigns the ram address to that process in HDD.

⊙ Program portability:

∴ compile time didn't gave compile portability

in compile time which is obsolete, there was no program portability

∴ compile time had all disadvantages

even on power off, there was error

⊙ Load time technique

Every program in load time, its address will start from 0. In Loading time we made things easier.

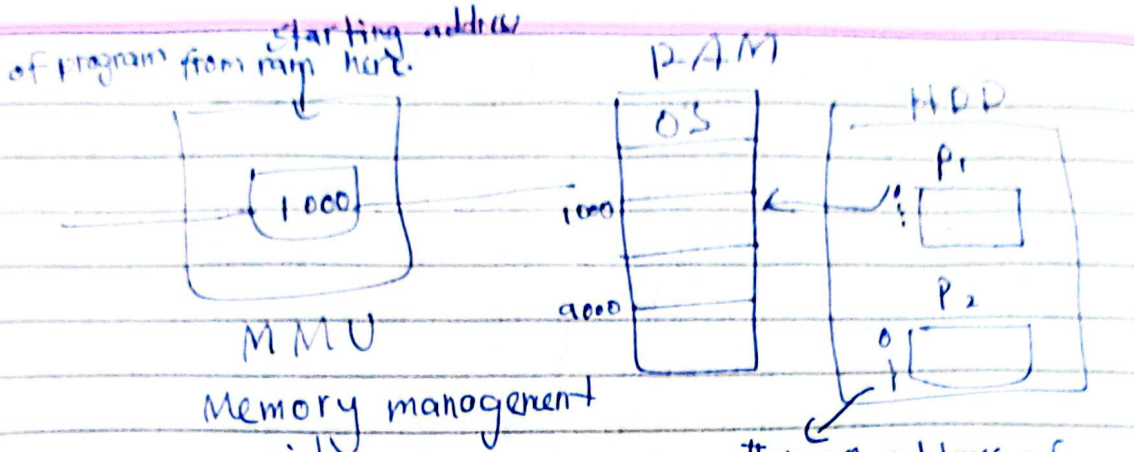
∴ Programs are portable

∴ Modern OS have virtual memory. There was no provision of virtual memory in this load time. Once program loaded, there was no option to bring back to HDD.

⊙ Execution time address binding

∴ use virtual memory

Next page



Programs were able to use virtual memory.

In **execution time address binding** we use a H-W MMU to translate the addresses. All the program when compiled, have their address starting from 0 on HDD but when a ~~process~~ program is loaded into RAM, its starting address is pasted into MMU. Because of all these changings, we are able to used virtual memory.

slides picture in paper in objective

Paging imp

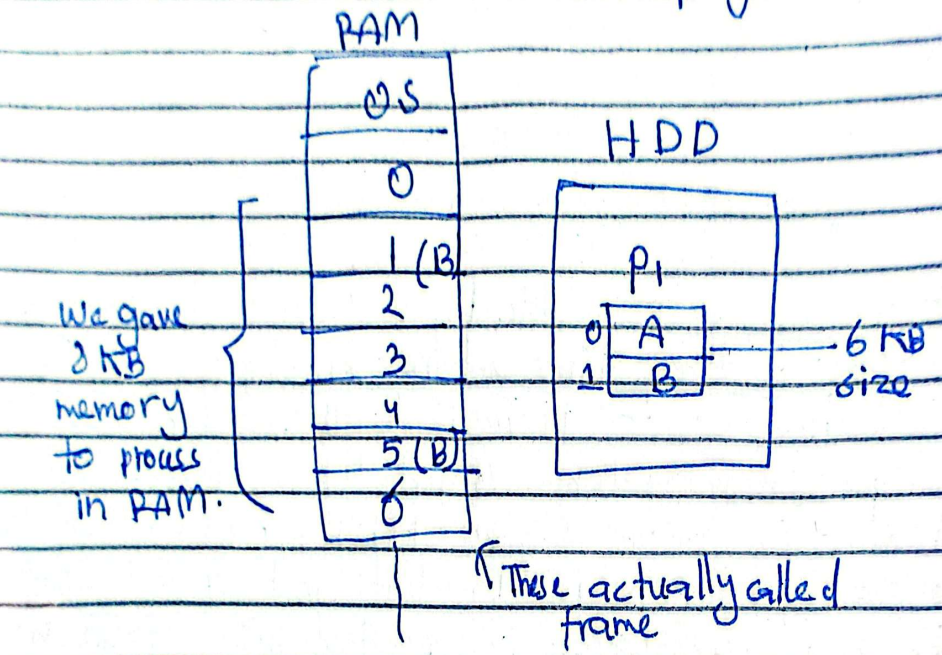
In paging

- it is memory management technique.

In **paging**, we divide our ram in equal size chunks called ~~chunk~~ page/Frame. page/frame size = 4kB to 8kB

for 32 bit OS = 4B each instruction.

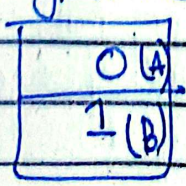
here $\frac{4096B}{4B}$ = 1024 instructions in a page



∴ Before paging, we can't run a process whose size was greater than ram. but how possible due to chunks.

⊙ two memory in paging:
↳ physical memory: RAM
↳ logical memory: same size as program. (fake memory)

∴ logical memory has page also



⊙ OS use page table.

leftside always grow sequentially

Page table

Page No	Frame No
0	5
1	1

← This from RAM

Logical view | physical view

∴ Logical memory grow sequentially

∴ Format of instruction here;

Page No.	offset
0	100

0 → 1023 = offset

← every instruction format like

offset means instruction number. for address, do x4.

we page number from here & goto that in ram. so we got

5	100

Paging is a memory management technique in which we divide ram in equal size chunks called frame/page. Page size is 4kb to 8kb.

/ Next page

Q: Khali page table. Logical & physical
ey page table fill karo.

RAM

OS
0
1 (B)
2 (C)
3
4 (B)
5
6 (A)

HD P2

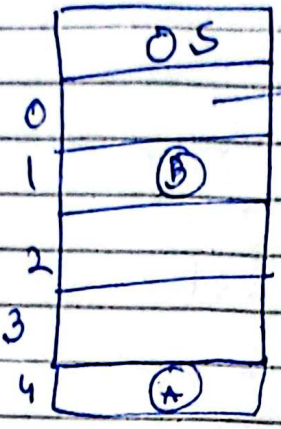
0	A
1	B
2	C
3	D

Page table

Page No.	Frame No.
0	6
1	4
2	2
3	1

12 December 2024 Thursday

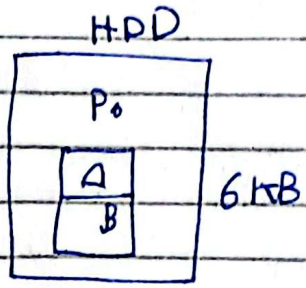
- Paging is memory management technique



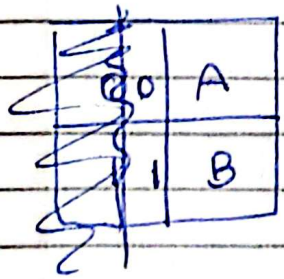
Frame/page

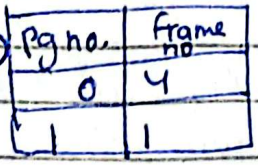
Pagesize = 4kB

$\frac{4096}{4}$ = 1024 instruction is a single page



logical mem





4|100

0	100

Internal fragmentation

memory wastage

∴ in upper example, B use 2kB only so 2kB is wasted.

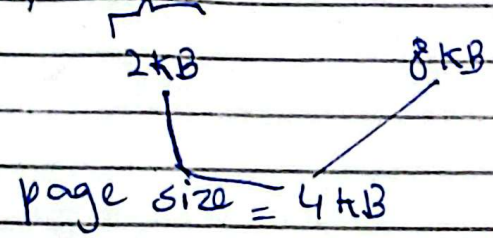
⊙ When the allocated memory is more than requested memory, it is called **internal fragmentation**.

Q: How can we eliminate I.F?

Ans: It can never be eliminated. It can only be reduced.

continued

now; 2048 instruction



∴ increase page size = internal fragmentation increase & vice versa

↓

∴ last frame only got 1 instruction

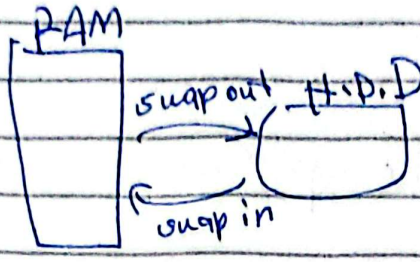
→ By using a smaller page size

Swapping:

∴ inpaging, we use virtual mem.

⊙ When we use a persistent storage like H.D.D as a memory, it is called **virtual memory**.

∴ When ram fill, remove some pages from ram to H.D.D. It is called swap out.



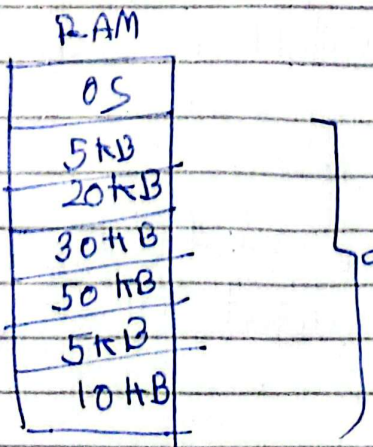
⊙ **Swapping** is combination of swap in & swap out.

⊙ **Swapspace / virtual space** is name of space of virtual mem in H.D.D. swapspace has no partition. It has raw format.

⊙ **Segmentation** ∴ imp

2nd mem management technique now obsolete

⊙ **Segmentation** is a mem mangement technique.



⊙ in segmentation, we can never execute a process whose size is more than ram

Now, let a S.T.B. process

① I.F in paging

② E.F in segmentation

i. Jab mem pari bhi ho par akhti na ho

① When total mem does exist to satisfy a process req but it is not contingous, then this is called E.F

external fragmentation

Q: How can we eliminate E.F?

Ans: Yes, it can be eliminated by using a technique compaction.

shift ram contents upward.

② It is a sol of external .F in which we shift contents of the ram to make all the space available in a contigous manner.

Now:

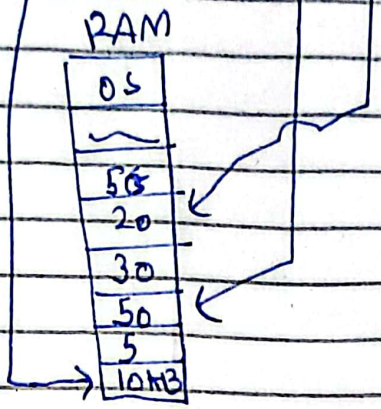
imp by HOD in exam

3 techniue of segmentation

Memory Allocation

i) Best fit
ii) worst fit
iii) First fit.

for 7MB process



∴ Best fit say give process the most chota block in ram. scan the ram & then allocate

⊙ We will allocate minimum memory.

∴ in worst fit, we give the maximum size. It is because there is a chance a new process can use it later.

∴ Both Best & worst are equally useable.

∴ Best & worst scan complete ram. it is their disadvantage. time waste

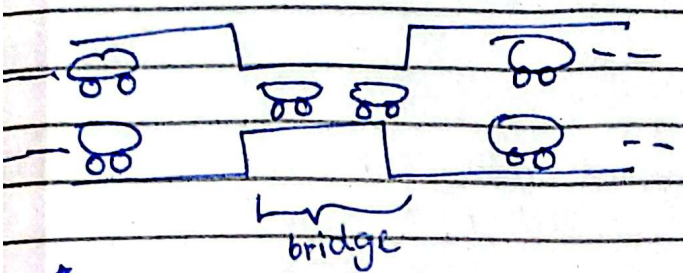
⊙ in First fit, give the 1st space available. We don't want to waste time on scanning the ram.

9 December 2024 Wednesday

Deadlock:

Def: "A situation in which there is no process being made in system"

eg: Bridge crossing info



∴ computer science / OS have no reversability concept

Q: What is deadlock & give real life example?

(give (no) process or thread example)

Imp in paper 1

Condition for deadlock:

There are 4 conditions & all must be true for deadlock to occur.

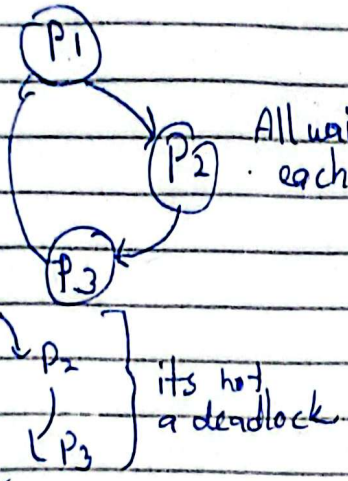
i) mutual exclusion
ii) Hold & wait
iii) No preemption
iv) Circular wait

i) mutual exclusion states the resource can not be shared among the entities.
∴ in upper example, bridge is our resource

ii) Every process should have occupied a resource & it should be waiting for a resource.

iii) No prempt tion: No process can snatch a resource eg car can't push other.

iv) for circular wait; Every process should be dependent on other process.



imp in paper

Bankers Algo:

It is a deadlock avoidance algo. OS uses this algo before allocating a resource to a process.

Now;

Process in system=5

P0 → P4

Resources

A = 10
B = 5
C = 7

} each ki resource have

An allocation matrix

	A	B	C
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2

∴ represent which process have how many resources

An Max matrix (it represent

	A	B	C
P0	7	5	3
P1	3	2	2
P2	9	0	2
P3	2	2	2
P4	4	3	3

every process tell at start that in lifetime, how many instance i ask of a resource.

) Now a matrix we will make ourself
) Need matrix
∴ max - allocated = need

	A	B	C
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

now find remaining resources

A	B	C	
7	2	5	(allocated)

Remaining : <3 3 2>

total - allocated

imp

∴ Some times total not given. remaining given so do allocated + remaining = total

∴ Bank algo says if we can run all processes in a sequence, we are in safe state.

∴ in paper; p_1 has highest priority
p_4 has 2nd highest.

∴ here 5! = 120 sequences possible.

∴ in paper: 6 combinations (3×2) (3!)
<p0 p2 ---->

⊙ Remaining question here

<p1 p3 p4 p2 p0>

tell for this sequence is system in safe state or not

P_1 Remaining - p1(need)
<3 3 2> - <1 2 2>
<2 1 0> } rest remaining resources

∴ if any value negative while solving, stop algo & write system is in unsafe state

at the end, A=10, B=5 & C=7

Now P_1 has gone from system, it return all resources.

so do remaining + p1 (max)

$\langle 2\ 1\ 0 \rangle + \langle 3\ 2\ 2 \rangle$

$\langle 5\ 3\ 2 \rangle$

~~P3~~ ~~$\langle 5\ 3\ 2 \rangle - \langle$~~

~~$\langle 5\ 3\ 2 \rangle - \langle 2\ 2\ 2 \rangle$~~

P3 $\langle 5\ 3\ 2 \rangle - \langle 0\ 1\ 1 \rangle$

$= \langle 5, 2, 1 \rangle$

Σ max (P3) + rem

remaining $\langle 2\ 2\ 2 \rangle + \langle 5\ 2\ 1 \rangle$

$= \langle 7\ 4\ 3 \rangle$

P4 $\langle 7\ 4\ 3 \rangle - \langle 4\ 3\ 1 \rangle$

$= \langle 3\ 1\ 2 \rangle$

remaining $\langle 3\ 1\ 2 \rangle + \langle 4\ 3\ 3 \rangle$

$= \langle 7\ 4\ 5 \rangle$

P2 $\langle 7\ 4\ 5 \rangle - \langle 6\ 0\ 0 \rangle$

$= \langle 1\ 4\ 5 \rangle$

remaining $\langle 1\ 4\ 5 \rangle + \langle 9\ 0\ 2 \rangle$

$= \langle 10\ 4\ 7 \rangle$

P0 ⟨10 4 7⟩ - ⟨7 4 3⟩

= ⟨3 0 4⟩

remaining ⟨3 0 4⟩ + ⟨7 5 3⟩

= ⟨10 5 7⟩

A B C

Proved

how for ⟨P4 P3 P2 P1 P0⟩

- if in max matrix, there is a value larger than total resources, then don't solve it.

Resource Allocation Graph:

method of algo for deadlock detection.

(P1) : process in circle

Resource : R1

} dot means instance of each resource

R2

P1 — allocation edge

request edge

Q: Sometime graph given.
Q: some time data given. make graph & then tell if deadlock or not.

Let process P

$P = \{P_1, P_2, P_3\}$

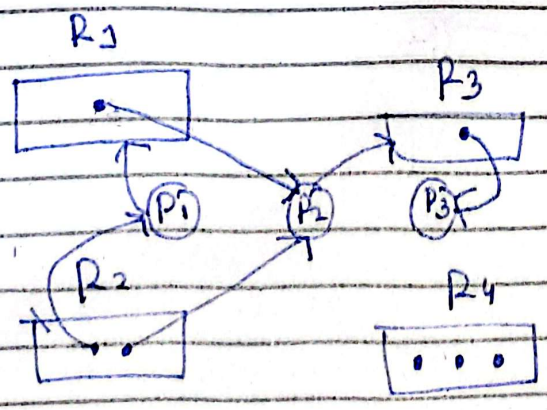
$R = \{R_1, R_2, R_3, R_4\}$

(1) (2) (1) (3) } instances

process : Resource request

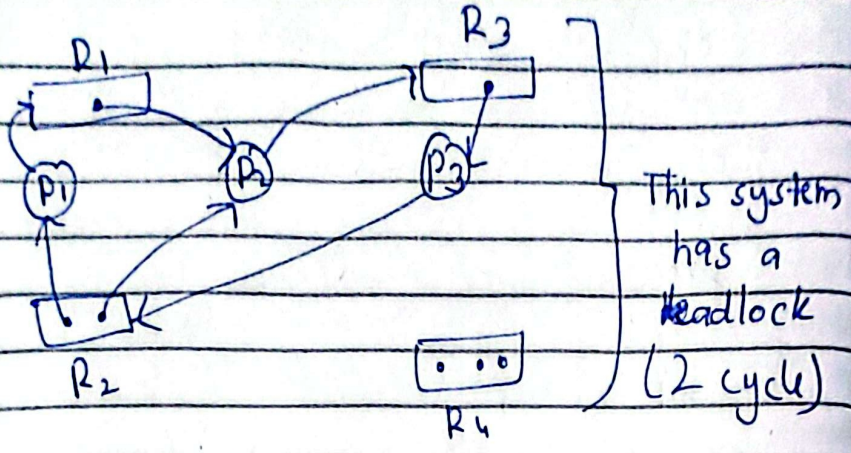
then E set for edges:

$E = \{ P_1 \to R_1, P_2 \to R_3, R_1 \to P_2, R_2 \to P_1, R_2 \to P_2, R_3 \to P_3 \}$



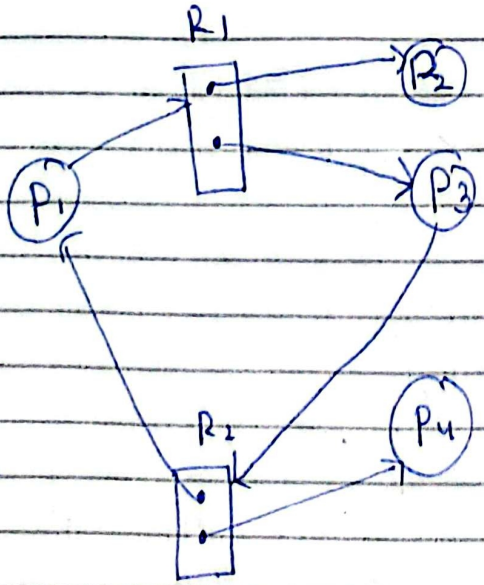
∴ Find if there is a cycle in graph.
No cycle = no deadlock

Next Pg



∴ if there is a cycle, check each process one by one if it can execute.

Example:
A graph in which there is cycle but no deadlock.



P1 can execute.
∴ Find those processes which can execute Independ

10 objective (picture, T/F / blanks)

5 out of 4

don't attempt extra question.

To the point

- do numerical at end.