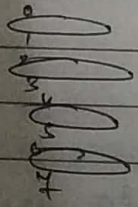
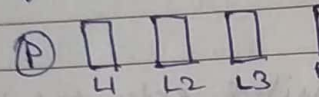
OS Fundamentals

- 4 master partition, unlimited extended partition.
- POST - Power ON self test
- Overclocking - Power $\propto V^2 f C$
  volt ↳ clock freq.
- Remove Ram to clean - Remove charge stored because it forms a capacitor.
- Sound / buggee signals used to identify fauets during booting. Eg - In case of display failure.
- USB - Flash drive - EEPROM - RAM
- Hard Disks - Rotational + sequential - Magnetic - No 1, 0s stored - North/South pole stored in disk.
- After POST → Bootloader to select the disk segment from where to boot.
- OS - Resource Manager.
  - Loads first, stops last.
  - Interface b/w user & computer h/w.
  - Single most complex & essential s/w.
- Hard Disk -  → Multiple disks & surface.
  → To Read data - Sector #, Track #, Head #.

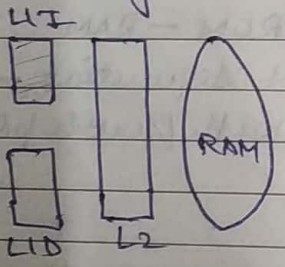
Page No.: study time
Date: / /

- Stored program concept - A prog. will execute only if it is available in the primary memory (RAM)



→ Inclusive - Subset of L3 available in L2 & subset of L2 available in L1 and so on

- Intel Architechture - Harvard Architecture (modified)
- First Architecture - Von ~~Human~~ Neumann - Every thing has to be done using only single memory without cache - Code can be read as data and vice versa
- L1 Cache - split cache - Inst. & Data Cache.
- Code segment - Read Only - For protection
- Data segment - Read & write.



→ To remove the inst from L1I we just discard it but to remove the data from L1D we need to save it in L2 cache first then remove it.

→ ~~Inst.~~ is already present in L2 and it cannot be changed.

- Intel - CISC outside, RISC inside
  - CISC inst. converted to RISC inside µP.
- OS allows the prog to terminate itself to prevent that program from affecting / damaging other programs.
- HW cannot be accessed without the use of OS.
- Floppy Disk - FAT 12
- Passive Resources - Data, message.
- Active Resources - Printer, HW.

6/08/19

• BRK/SBRK – System calls used to allocate memory when malloc/calloc is called.

• Parallel computing – Run multiple programs at the same time.

• Concurrent comp – Run a program for some time then switch to other program so both the programs can be executed together.

• Resident part of OS – Permanantly available in memory
• Non-Resident part of OS – Dynamically loaded in memory as and when req. because they are rarely used.

6/02/19 Min function of OS – load the program in memory, execute correctly, save the result correctly and then unload program safely.
→ Nano kernel – used in sensors and nodes.

• Resident code needed which run by default.
• Hyperthreading – Every core runs more than 1 processes.

* Windows :-
• Xerox first started mouse and windows. in ?
• " " " GUI in 1981 called Xerox Star.
• 1985 – 1990 – Single layer Windows.
User → OS → H/w
• 1993 onwards – New abstract layer.
User → OS → H/w Abstract layer (HAL) → H/w.

• Two types of kernel – WIN x.x (Mostly for desktop) – 1985 – 2000
– Win NT x.x (Mostly for servers) – 1993 onwards.

* History of OS -

- 1957 - BESYS, Bell Labs (batch system).
- 1965 - Multics, Bell Labs, General purpose, MIT (first OS).
- 1969 - UNIX, AT&T's Bell Labs.
- 1971 - First Edition of UNIX, 1972 - Rewritten in C
- 1983 - BSD - First stand packaging of UNIX system called BSD (first UNIX for Intel 386) of 2022.
- 1987 - MINIX - Minimal UNIX-like OS
- 1989 - GPU GPL by Richard Stallman
- 1991 - Linux started by Linus Torvalds.
- 1992 - X windowing system by Open Software Foundation with GUI's.

- Real Time Kernel - Jobs need to be done within a fixed time limit.

- 2001 - 2.4 kernel - Real time kernel
- 2003 - Fedora released
- 2004 - Red Hat Linux version for desktop
  - First version of Ubuntu released.
- 2008 - First version of Android.
- 2009 - Chrome OS by Google
- Open kernel - (GKM) - 2012 - Kernel maintainers

P1 — 5 sec
— I/O - 100 sec
— 15 sec

P2 — 10 sec
— I/O - 100 sec
— 10 sec

P3 — 20 sec
— I/O - 100 sec
— study time

Page No.: 20 sec
Date: / /

* Types of OS -

- Batch System - Executes the program sequentially.
  - After the previous program is executed completely then only next program will start.
  - Only 1 prog. need to be available in the mem. at a time.
  - Shell programming
  - Total time = 5+100+15+10+100 - 120 = 230 sec
- Multi Programming - More than 1 program waiting in the mem.
  - Executes another program whenever the current program moves out of system for I/O.
  - Parallelise CPU execution of 1 prog. with I/O operation of another.
  - Use of DMA and interrupts.
  - ~~Total time = 5+100+15+100+10+100+20~~
    ~~= 350 sec~~
  - Here when P1 goes for I/O P2 & P3 will execute in system seq. and go for their I/O.
  - If P1 I/O is 10 sec, then also when P1 returns prog. in I/O it will resumes execution after P3.
  - The order depends on the priority of the process assigned by the system.
- Multi Tasking - Time division programming.
  - In multi prog. decision point are only prog termination & I/O.

07/08/19

Scanned by CamScanner

- In multi-tasking there is an additional decision point ie time slice expiry.
- After every fixed time the pro current program is stopped and another program starts execution for the same amount of time (or till it's completion).
- This prevents small program from waiting due to larger program.
- Logical extension of multi-programming
- Mainly for interactive I/O system which has many I/O and small CPU.
- Improves response time better than multi-programming.

07/08/19

- NICE value — used to set the time slice value for diff programs.
  - -19 to 20
  - -19 highest priority.
  - 20 lowest priority.
  - usually value is 0.
  - -ve value can only be set by superuser.

- DVFS — Dynamic Voltage & Freq. Scaling.
  - changes the freq. and voltage of CPU depending upon the load dynamically.

- Turbo Boosting — Increasing the freq. in modern intel processors.
  - i7 - 1 level, i9 - 2 levels

- Uni-core – Single core in a processor.
- Multi core – Multiple core in a single processor.
  - L2/L3 cache are shared b/w the cores
  - This makes data transfer b/w the cores very fast.
  - Quickpath – Any core can access the data of another core's L1 cache.
  - Every core has it's own L1 divided cache.

07/08/19 Basic Architectures –

- IF – Inst. Fetch – Mem. read to get the inst. from mem.

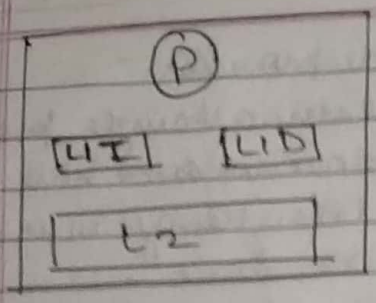
→ So every inst. has atleast 1 mem read for IF.

- ID – Inst. Decode
- Execute
- MEM – MEMORY Read
- WB – Mem. write back.

- In Harvard architecture cache is divided for inst. and data so that two diff. inst. can be executed at a time – One doing IF & other MEM read.

- In Harvard architecture the L2 is also divided so inst. cannot be modified during runtime because inst. cannot be used as data.

- So it was modified to Modified Harvard Architecture in which L1 is divided and L2 is unified so that inst. can be modified during runtime.



→ Modify inst. and ~~save~~ write it back as data from LID to L2. Then read it back from L2 to LII as a inst.

* Types of OS –

- Many Core – When no. of cores become to more so Quickpath is ~~no~~ difficult to execute.
  - Switch is used to connect diff. cores to diff. L1 caches.

- MultiProcessor – No communication ~~b/w~~ on cache levels b/w the cores/processors.
  - Only the external DRAM is shared.
  - Time taken for the data exchange is slightly greater than multi-core machine.

- Multi-Core, Multi Processor – Multiple processor having multiple cores each.
  - CPU and GPU in general PC.

* Multi-Processor – i). Tightly Coupled system
  a). Asymmetric – Processor & co-processor having diff. tasks and purpose.
  b). Symmetric – Similar processor with a shared memory.

→ Multiple processors in a single system

Distributed System —

ii) (Loosely coupled System) — Multiple system/processor not in the same system or connected on a network.
- Can be multiple system on connected on a network.
- DC ++, Client-server, Peer to peer.

* Memory Model —

• UMA (Uniform Memory Access) —
- Only 1 level of mem. & diff. memory of similar kind.
- Time taken by each memory is same.

• NUMA (Non Uniform Mem Acc.) —
- Multiple level of memories with diff. types of memory.
- Time taken to access mem. is diff for diff. levels.

• NORMA (No Remote Mem. Access) —
- No access to the mem. of other system over a network.

* • Client-server — Performance degrades with more no. of users.
• Peer to peer — Performance increases with more no. of users.

• File servers — Distributed file system.
- Same data is available in multiple places.
- Most of the servers — Google, Yahoo.

- Accumulation system - Read more write less systems

* Clustered systems - Loosely Coupled system -

- Symmetric - Multiple servers connected in active mode.
  - Both servers working together.
  - If 1 goes down other will take the full load.

- Asymmetric - Multiple servers with few in active mode and rest in stand-by mode
  - Standby server only works if main server fails.

* Cloud - Logical ext. of virtualisation - Allocate memory only on demand.

- Software as service - Microsoft Office 360.
- Platform as service -

- Infrastructure as service - Storage servers.

* Real-Time Systems - Flights, high end cars, robots (R-OS)
  - Timeliness is imp.
  - Results are to be given within a given deadline.

13/08/19

- Soft Deadline – Deadline is not that strict.
  – with increase in time performance decrease.
  – Simple applications like Media Players
- Hard Deadline – Deadline is strict.
  – After deadline there is no use of the result.
  – Airbags in car.
- Firm Deadline – Mix of both of them.
  – Multiple deadlines in which some deadline miss is allowed but not the last one.

13/08/19 Hardware Protection –

- Dual-Mode operation – Using 1-bit in h/w (mode bit) to switch b/w user mode and supervised mode.
  – User mode – Operation done on behalf of user.
  – Supervisor mode – " " " " " OS. (System mode).
  – INT 80H – Interrupt used in intel microprocessor to switch b/w the modes.
  – Mode bit = 0 – Monitor Mode / Supervised / System.
    Mode bit = 1 – User Mode
  – Switch to monitor mode when int/fault occurs, privileged inst (I/O Operation), work to be done by kernel. (Mem allocation
  – System call changes mode to kernel, return from call resets it to user.

• Every prog maintain a list of file they are opening - Per process table. Of each program.
→ 0, 1, 2 entries are by default filled with input, output and error.
→ If the prog open a file then the file pointer will be added in the table.

• System wide table - If a file is opened by multiple program then that file pointer and the no. of program is kept in that table.
- Every time a prog. uses fclose to close a file then the count is reduced by 1 and when the counter reached 0 then the pointer is removed

→ All these are done by the kernel when the system wants to open or close a file. - System call.
→ EAX forwards the system call from system to kernel when INT command is used.

* Multi-Mode operation - Virtual Machines.

☆ CPU protection - Timer to prevent infinite loop/ process hogging resources.
- Timer is set to int. the program after a certain amount of time and CPU gets back the control.
- Setting counter (loading timer by OS) is a privileged inst.

14/08/

14/08/19

- I/O Protection – All I/O inst are privileged inst
  – Must ensure that a user prog. cannot gain control of the comp in monitor mode.

- Memory Protection – Inter Arch. – 1 Page – 1K or 1M or block

Fragmentation – Dynamic Allocation or variable mem. allocation.

– Max frag – 4K + byte – Internal Fragmentation.

- Base Reg – Stores the starting add of the segment
- Limit Reg – Length of the segment.
- Last Add = Base Reg + Limit Reg.

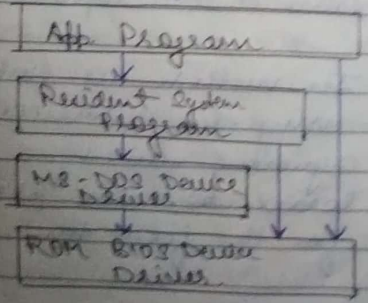
– CPU always generate a logical add.
– If logical add. of a prog. is equal/greater than the value in the Limit Reg then it is segmentation fault.

– Every time a page/block is allocated of size 4K. If prog is smaller than this size then the rest of the block will remain empty – results in fragmentation.
– Mem protection is done using the above 2 Reg.

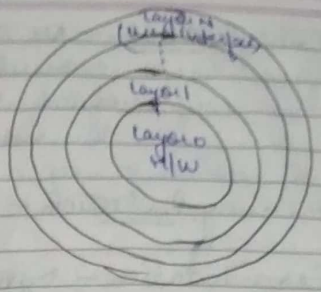
★ OS structure –

- MS-DOS –

– Loosely org layer architecture
– Some layers can access other layers by bypassing layers in btw



* Layered Approach -



- The OS is divided into multiple layers each build on top of other.
- Each layer can use the funct / services of its lower level layer only.
- So user cannot directly use H/w and has to go through the O/S layer.
- This simplifies debugging and system verification.

- Function calls change the locality of reference so it flushes the pipeline because jump is involved so it is not so good. But it also makes debugging easy.

- In linux most of the functions are in-line funct so during compilation the function is copied in the main prog and there is no jump.
- Similar to macros in C/C++.
- In inline funct. the storage will increase because same code is replicated every where.

- In unix we link a shared file libm.so using -lm while compilation dynamically. In dynamic linking a module from libm.so is loaded to main prog only when it is called for the first time.

— Iteration is better than recursion because stack overflow will occure in recursion because same function is being called multiple times filling the stack.

3.* UNIX System Architecture —

— User can only access the kernel by system-call interface/mechanism

• Two types of devices — Character Device
Blocked Device

— Blocked device requires a buffer but char. device doesn't.
— ls-l will give a list in which c-denotes char. & b-denotes block device.

— Sector size of HD - 512 Bytes
— Trasfer Unit - 4K.
So at a time a clusters of 8 sector is transfered from HD to main memory
Now if we need to access 1 byte data again and again then I/O trasf will not take place and mem trasf will happen because 4k data is already present in mem. — memory Mapped I/O.
— Buffer Cache - part of main memory where the block is stored.

→ Inst usually have spatial locality because they are executed sequentially.
→ Data is accessed randomly so no spatial locality

16/08/19

* Locality of Reference —

i). Spatial Loc. — The prob of accessing the next location is very high.
- For array if a[0] is accessed then prob of a[1] is high
- Arrays, structures are allocated mem. together because
ii). Temporal Loc — The prob of accessing the same variable is very high.
- In loops, i variable is accessed many time so temp. loc.

- I/O Memory mapped I/O makes processing faster because of spatial loc. I/O operation is not req. after retrieving the first block and memory operation is done.

- Once a prog goes in I/O op. it gives up the system. Now when it comes back to system then it has to wait till all the prog has finished and then it will execute.
- In mem operation it doesn't go out of system so no waiting for other prog. Makes the prog. efficient.

* Monolithic Kernel — All the functions of the kernel are compiled in a single file.
- To change anything the whole kernel has to be recompiled. System cannot work at that time — Downtime.

- Most UNIX like systems are monolithic kernel.

* Microkernel System Structure -

- Moves as much from the kernel into 'user' space.
- Security - Not too much call for basic funct=.
- Reliability - Any change has to be done only to the small kernel and not big kernels.
- Consists only essential components and very small in size. - Messaging, Mem. Mgmt, CPU Scheduling.

* Nanokernel - Smaller than microkernel.
  - More kernel functionalities are moved to user space than microkernel.
- Usually for very small and less powered devices.
- Easier to switch of functions in user space than in kernel space.
- used in pacemakers, sensors, etc.

→ Easier to expand microkernel - Easier to modify/add/change the small kernel.
- No need to shut down the system because most funct= are in user space.
- Comm. takes place b/w user module using msg passing via the kernel.
- Main function is to provide communication facility (using msg passing).
- Performance overhead of user space to kernel space communication.

• SSD – JFFS – Journelling flash file sys.

* Modules – Most layered arch. are modules.
  – kernel has internal modules for diff functions/components.
  – Uses Object-oriented prog.
  – Each talk with each other over known interface.
  – Each is loadable as needed within the kernel Dynamic loading
  – Similar to layered arch. with more flexibility. LINUX, Solaris.
  – Microkernel is also modular.
* Hybrid Systems – Windows – Micro kernel + monolithic structure.
  – Solaris and LINUX – Monolithic + Modular.
  – In hybrid systems diff. kernel design/architecture is used for diff function/comp. acc to the need.
  – Improves performance, security, reliability, etc.
  – Apple Mac Os – layered kernel
19/08/19 Android – Based on Linux but modified.
  – Provides process, memory, device-driver management
  – Includes power manag – Not in usual Linux.

• Architecture – ARTVM – And Runtime Virtual Machine – For apps on JVM
  – Bionic – stripped down version of C library designed specifically for portable devices.
  – Stripped down LINUX kernel which removes all unwanted functions of the kernel.

• Database – Each row/record should be a multiple of page of the device to make it more memory efficient.

Page No.: study time
Date : / /

★ OS specific App – gcc – group of C Compiler.
- Apps compiled on one system usually not executable on other OS.
- Each OS provides its own unique system calls, file system, etc.

★ Apps on Multi-OS – Ruby
- Interpretor languages like ~~JAVA~~, Python works on diff. OS ~~because they are executed in virtual machine~~ if interpretor is available on that Os.
- ~~Apps~~ written in languages that includes a VM containing the running app (JAVA).
- Use standard lang. (like C), compile separately on each Os to run on each.

★ System Design – SRS – Software Req. Specification.
- Policies and mechanisms should be isolated with each other so any change in policy can be implemented easily with the same mechanism.

- An OS can be written in higher level lang. also now-a-days and it's easier to port an OS to other h/w if it's in any high level lang.

Scanned by CamScanner

20/08/19

★ System Calls :-

- Single task - Protect OS from incorrect prog.
- Multi task / Multiprog - Protect OS and other prog. from incorrect prog.
- All I/O inst. are privileged.

```
#include <stdio.h>
int main ()
{ char x = getchar();
  return 0;
}
```

→ On compilation we get object file (.o).
→ On compilation the prog. is linked with libc.so

→ The library for getchar() will change the system from user mode to kernel mode for getchar() because it is an privet I/O Op. which is a privileged Op.
→ If the I/P from user is fine for other prog. then the mode changes back to user mode.
→ The lib. does the system call to the kernel.

→ In getchar() the ascii value is returned because it is only 1 value.
→ In scanf() there may be more than 1 I/P so it returns the no. of I/P taken as return by value (pass by value) and gives the pointer to the location as pass by reference.

→ INT x80 is used for system call – vectored int – fixed location.
→ Total 425 system calls in linux which is decided by the value stored in other registers – EAX value. – Based on the no. in EAX the prog. then jumps to the specific location of the purpose.
– NI System Calls – Not implemented system calls.
– Some no. is alloted to a system call by some version of OS.
– The same no. cannot be alloted to another call even by other version of OS to avoid any malfunction or crashing the system.
– So some system calls may not be used by a version – NI system calls and that no. cannot be alloted to other calls.

- scanf(). – To i/p n char. – (n+1) i/p is entered.
– Last one is delimiter – can be space, enter, tab, etc.
– (n+1) read op. are done by the OS-kernel to take i/p from the user.

- ~~fork – Move the~~
- fopen ("filename", "mode") – Reg. are used to pass the parameters to the kernel while system call.

– EBX, ECX, EDX, ESI, EDI – At max. 5 parameters can be passed by user.
– For more than 5 parameters the parameters are stored in user space and the starting add. is passed in a single reg. – Less efficient.
– Min. 1 parameter (for EAX) is passed in a system call.

Page No.: study time
Date : / /

- System call is the only legal way to move from user space to kernel/OS space.
- To pass a value/parameter or return an o/p the prog/os stores it in any reg. and other prog reads it from there.
- 2 types of jump — i). Jump with link.
  ii). Jump without link.

→ With link — The add. of current prog. is stored before jumping to new location.
— All ~~p~~ function calls are jump with link because the prog. has to come back to execute the remaining prog.

→ Without link — The add. of current prog. is not stored before jumping to new location.

- After int K go's to the specific location it checks whether the value of EAX < NR_SYSCALL i.e. ~~to~~ if EAX has the correct value (less than max. no. of system call ~~by~~ the OS).
— Then check if it is 32-bit arch. or 64-bit arch.
For 32-bit — each entry in IVT is 4 bytes.
" 64-bit — " " " " " " 8 bytes.
— Then it will go to that location in ~~the~~ IVT and take the add. to jump.
— ~~EAX has this info~~

- Before going to IVT and after ~~saving~~ checking FAX, it will save the values of all reg. in a memory so that it kernel can use all the reg. and after returning to user mode it will read those back. (Restore All).
- ~~stored in process control block - A block in main memory.~~
- After going to the specific routine the kernel will check whether the prog. is allowed to execute that routine.
  - If not then kernel will not execute it and return a -ve no (for error).
  - perror() will print the error code.

- A prog. can directly make a system call without the need of library in blue.
  - Library is a function call so there will be cache miss.
  - Direct system call will prevent this cache miss.
  - In most of embedded systems direct call is used ~~here~~ to save time and resources.

- Three methods to pass parameter -

  i). Store in reg. - for max 5.
  ii). Store in user space and pass the first location in 1 reg - for more than 5.
  iii). Push all parameter in stack by prog. and pop all par. by OS/kernel.

  - Performance degrades in (ii) & (iii) because OS has to access memory which takes time.
  - very few system calls with more than 5 parameter - IO control.

- saveall() - ~~the~~ reg. are saved in a specific location.
  - The reg. does not loose it's value, it just makes a backup of it in ~~memory~~ memory.

• write op to hd/memory is faster than read op.
→ write – non blocking op – Asynchronous. Prog. will not wait for any data to return.
→ read – Blocking op – Prog. will wait for the data to return.
→ Read is given priority over write because read makes the prog. wait and write does not.
– write is given priority only in case of synchronous write.

21/08/19 • Allocation of mem. to a process is done by kernel but accessing done by the ~~user~~ user for the allocated space.

• Before call/jump we store the location of next inst. so that when returning it goes and ex the next inst. and not the previous inst again.

• X = X+5; – Non atomic – can start the process and pause in b/w without completion.

```
load x EAX
EAX ← EAX +5
store EAX, 5
```

Atomic Instructions.
– Each inst is atomic inst i.e. if they start they will complete it and then stop or move to other inst.

• A prog can only start/stop by kernel because it needs resource management which is done by OS

- System calls includes so many jumps so there will be cache miss before each jump.
  - So avoid system calls as they are bad and degrade performance.

- for (i=0, i<1024, i++)
  {
  ;
  }.

- In assembly code there will be a backward unconditional jump at the end for the loop and a forward condition jump for break (when i ≥ 1024).

```
r0 ← i
r0 ← 0.
L0: beq r0, 1024, L1   — conditional jump - Jump out of the loop if r0=1024.
    ;                    Always predicted not taken
    ;                    Accuracy ≈ 99.9%
    ;
    add r0, r0, 1      — i++
    jump L0            — Jump back unconditional.
L1: ;                    Always predicted taken.
    ;                    Accuracy ≈ 99.9%.
```

- Pipeline has prefetched inst according the probability/prediction.
  - If the inst is not in pipeline then flush it which results in cache miss.
  - Jump results in flushing because it moves to other inst which is not prefetched.

- System call is most expensive in terms of processing time and resources as it includes context switching (switching the mode).
  - So function calls are better than system call.

- Context switching – changing from user to kernel mode & vice versa.
  - Reg. values are stored in memory and restored back from mem. on returning.
  - ~~Only preserved reg. are stored and unpreserved not~~

- In function calls only preserved reg. are stored and not the unpreserved reg.

23/08/19 Assignment – float a, float b → res. (add)

- Check if both nos. are +ve.
- Return -ve nos. will show an error because -ve nos. are for system error.
- Floating pt. no. cannot be returned by system call so the no. has to be converted to int. representation, returned as that int rep. and converted back to floating rep. by the wrapper function.

★ New System Calls –

- Passing arguments as reg. than storing it in mem. and passing the ptr. because in latter case the ptr. has to be checked for multiple

factors before the values are read from these which takes so much of time.

- After system call return it is possible that the calling process doesn't get chance immediately and has to wait for some time.

* Types of System Call —

- Process control — A prog. in execution is called a process.
  - Prog - Passive
  - Process - Active, have process count, stack, resources, etc.
  - Creating and terminating a process is system call.
  - Even if the user terminates / abort the process there is a system call because os has to manage all the resources.
  - Allocating and access mem. ∫
  - fork (), wait (), exit (), kill (), alarm (),
- File Management — Open (), close (), read (), write (), lseek (), create ().
  - fopen ("a1.txt", "w") — If file doesn't exists it will create one file.
  - If it does exist then it will open write mode — It has to delete all the file content and then open it has a new file.
  - More operation if the file exists.

- Device Management –
- Information Maintenance –
- Communication –
- Protection –

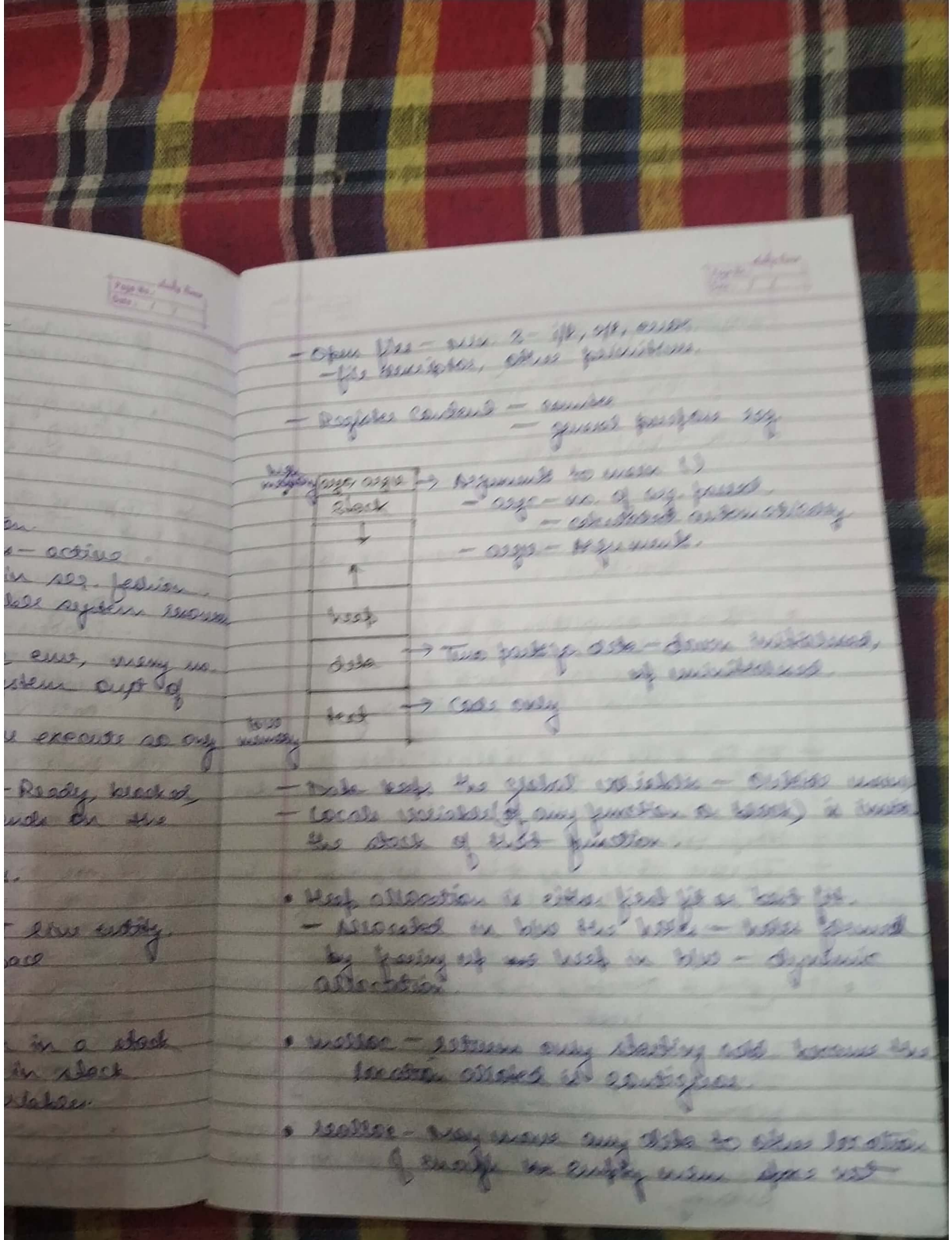
26/08/19 Process and Thread –

+ Process – A prog in execution.
  - Prog – passive, process – active
  - Process execution in seq. fashion.
  - Process uses available system resources.

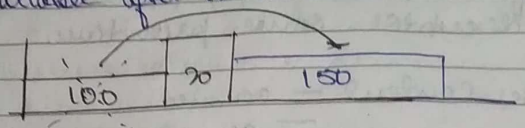
– In multiprog./multitasking env, many no. of process can reside in system out of which 1 will execute.
– In batch env. only 1 will execute as only 1 is usually there in mem.
– Process can have diff state – Ready, blocked, running etc – and it depends on the condition.
– OS manages all the processes.

- Program – just code → NOT live entity.
- Process with data, work space
  - stack, heap.

– Initialised variable are down in a stack and uninitialised are above in stack.
– .bss – store uninitialised variables.



available after it to add.



- brk - heap pointer.
  - used by malloc to allocate mem. in heap.
- esp - stack pointer
  - used to allocate mem. in stack during run-time.

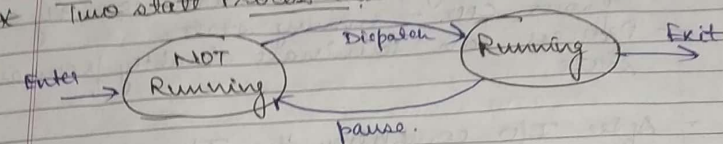
- heap - for completion of process
  - scope - whole process.

  Stack - for specific function
  - scope - that function only.

- ~~If~~ OS stores any variables in stack for any function and the end removes the stack after the completion of that funct^n.
- ~~Heap is not~~
- If user stores any data in heap then user has to remove that data at the end of the process.

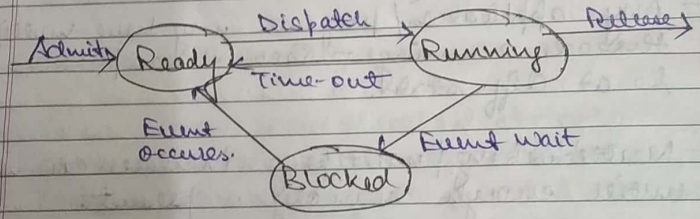
- Dispatcher - Decides which process has to execute next.
  - It is called every time there is time expiry, I/O of a process or completion of the process.
  - May select the same process or any process to run depending upon the schedule and priority.

* Two state Process Model -



- Not Running - Process may be blocked, went for I/O, ready but didn't get chance to execute, etc..

* Three state Model -



- Change in Process/Decision Pt. - Departure/Exit
  - I/O
  - Time slice Expiry.

- In time out the process moves back to ready and not blocked because it was ready.

- If process goes for I/O or is waiting for an event to occure then it goes to blocked. Now block has to go to ready and then go to running, it cannot go to running directly from blocked.

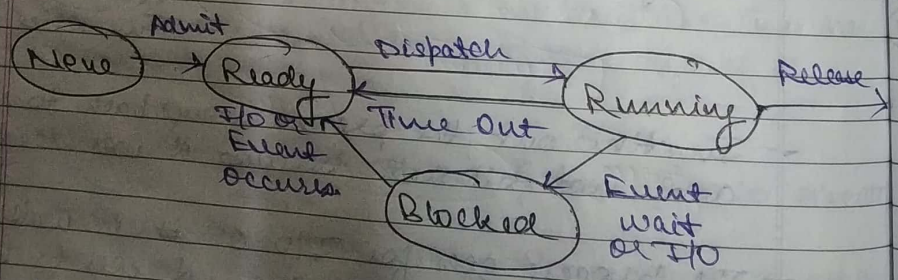
- Process can go out terminate correctly only from CPU. - running state.
- Process can terminate abruptly from any state

- After I/O completion the I/O device sends an int. to indicate completion. until then the process is in blocked state. When kernel gets a chance it will acknowledge the interrupt and move the prog. from blocked to ready. Then it will be moved to running acc. to priority.

- Any application/program may have multiple threads running - each thread/process may be at diff. state.

- Multitasking/multiprog. increases CPU utilization until enough memory is present.

* Four State Model -



- A process is moved from new to ready only if there are enough resources to accomodate it.

- If we used 3-state model then after the resources are exhausted adding new process will degrade the performance — In 4 state model the performance does not degrade because process are kept at hold until resources are available to use.

- Resource will be released whenever a process is terminated/completed.

- Once admitted the process cannot go back. The process will remain in other 3 state till completion.

* Multiple Blocked Queue —

- Each process goes to blocked state for diff reasons.
- so each process has a blocked queue if it has to wait for any event to complete or multiple events to complete.
- After the all the blocked events are completed for process then it will move to ready state and the blocked queue for that process is removed.

- Only in case of time slice expiry the process doesn't go to blocked and goes directly to ready state without any blocked queue.

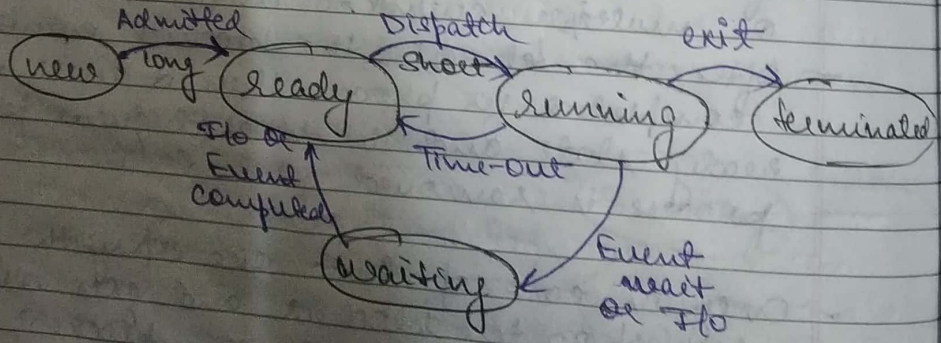
- To execute process efficiently, system may put a max. limit on processes in ready queue.

27/08/19

Five state model

```
int main ()
{
    ;
    return
}
```

- Process are maintained as a tree. So any process calling another process is called the parent process.
- Return value is for the parent process to know that the child has executed properly.
- But parent process may not be ready to execute so the child process has to be in an ~~terminal~~ hold state called terminated state until the parent process is ready to collect the return value and execute.



- Short Term Scheduler - Ready ↔ Running
  - Also called dispatcher.
  - The scheduler that move process from ready to running or running to ready
  - Highest freq.
- Long Term Scheduler - New → Ready
  - Scheduler that decides which process to move from new to ready
- Mid-Term / Scheduler -
  - Scheduler that suspends any process if the CPU is not able to handle many processes admitted
  - Until the completion of a program it is impossible to predict the time taken and memory requirement of the prog.

+ Process Scheduling -

- Whenever a child process is completed then first child process goes to ready state and then the parent process, i.e. if both the process has same priority then child is executed first and then the parent process.
- Whenever a parent process creates a child process the remaining quantum time of the parent is divided into two - one is given to the parent and other to the child.

- ~~The data of the both~~

Page No.: study time
Date : / /

both the

- From next cycle onwards ~~each~~ processes will be given time slices independently.

- If parent has higher priority than child then parent will execute before child.

★ Six state Model -

★ Suspended Process -

- Easy to suspend a ready process.
  will create (if sent to suspend state)
- Blocked process has a dangling ptr i.e. the pointer that ~~is~~ points to the location where the process will come back ~~to~~ from blocked state. - Difficult to remove.

- The data of suspended process is moved to swap space so that other process can ~~you~~ use the space freed by it.
  suspended
- Ready & ~~Swapped~~ state - suspended state when process ~~was~~ is suspended by from ready state.

23/02/19

- If enough resources are not available for ~~the~~ running processes then they will take somebody else's resources. ~~or~~ Now when it goes to I/O someone else will try to take it's I/O. so every process will take other's resources and no productive work will take place so processes has to be moved to suspend ~~state~~ to ~~pro~~ increase productivity.

★ cooked
Raw P

both the
each processes will
dependently.

ity than child then
child.

process.
end state)
ying ptr i.e. the
the location where
ck from blocked
ue.

ess is moved to
process can
it.

ked state when
s is suspended
ady state.

aiting for
e take somebody
it goes to Flo
its Flo. so
resources and
place as
suspend state

* Cooked Partition – with a file system
Raw Partition – without a file system
– swap partition.

- Dangling ptr. – If a blocked process is moved to suspend state then it will create dangling ptr.

– Dangling ptr is created when the process has a location ptr to go back after the blocked state but os has assigned that location to other proces because first process was suspended.

– ~~If the os can take care~~

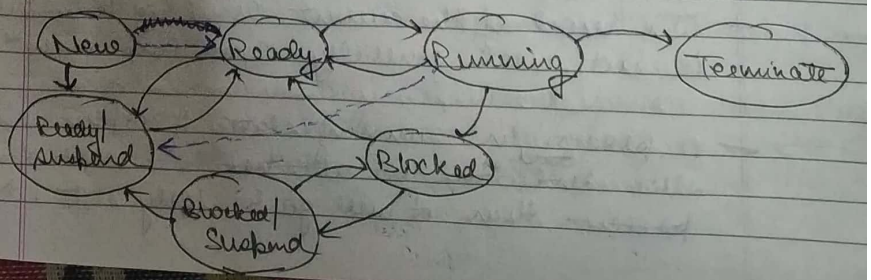
– Blocked process only saves offset value and os manages the base value. If os can take care of the base value while suspending a blocked process then Blocked/suspend state is also possible where the process which are suspended from blocked state are present.

– Os has to change the base value for any blocked/suspended state so that when it comes back it ptr. to the correct location because it only has the offset value.

– In six state model there is only ready suspend state.

~~If~~

★ Seven state Model –



- The data of both the suspend states are in the secondary partition of the secondary mem. (HDD).
- The data of other states are in the primary memory (RAM).

- Blocked suspend can go to ready suspend if I/O it was waiting for completion in the background.

- If OS is capable of managing changing pts. then 7 state model is preferred which has both the suspended states.

- Blocked process can never go to ready suspend state.

- Blue line is a virtual line - The change does not happen directly - it will go through the intermediate states only.

- New process is admitted to ready suspend state first and then moved to ready state when all the resources are available. ★

- New state exists because when creating a new process some data structures have to be created for the first time and some valid identifiers also. So when process has all those available but not in execution cycle then it is in new state.

- In most of the current OS long term scheduler is not used generally - least sup. scheduler.

- A process in running state if expires it's time slice and the OS also has to suspend a process then it will go to ready state. Now

...und states are in the ...em. (MSD)
...are in the ...

...to ready suspend if ...completed in the ...

...of changing ptr. then ...ed worked has both ...

...go to ready suspend ...

...time - The change ... - it will go through ...mediate states only ...

...ready suspend ... to ready state ...e available.

...n creating a new ...ane to be created ...ueid identifies ... all those ... ...ion cycle then ...

...ng term scheduler ...up. scheduler ...

...expires it's time ... ... state. Now ...

If OS has to suspend a process then it's a good idea to suspend the recently expired process and move it to ready suspended state from ready state. So there is a ~~it~~ internal line from running to ready suspended to convey this idea. - The process has to go through ready state and doesn't move directly from running to blocked suspended.

- suspension is very clear when it becomes clear necessary to suspend a process to move forward. The most higher priority processes are rarely suspended, ~~only~~ lower priority processes are usually suspended.

★ Mine - State Model — Unix System.

Created — New
Ready to run in memory — Ready
Ready to run swapped — Ready suspended
Asleep in mem — Blocked
Sleep, swapped — Blocked suspended.

- Two diff states for user process and kernel process.
- Only kernel can perform I/O operation ~~so only~~ or start a new process so ~~only kernel can~~ any process can be blocked from kernel running state only.
- A ~~task~~ successfully terminated process can only happen from kernel and it will go to zombie state. - User cannot terminate the process.

30/08/19

- No path from blocked suspended to blocked because when we suspend a process we want it to be for as long as possible so a process need to go to ready suspended from blocked suspended and from there it will be moved to ready and then running.

- Scheduler is running in kernel space so any new process will be added by the kernel ~~space~~ running only from ready state and not by user running.

+ Preempted - while returning from the system call, kernel will first check to whom the control of CPU is to be given i.e. if any other process of higher priority is waiting.

- If the calling process was the highest priority then it will move from kernel running to user running directly.

- If a higher priority ~~of~~ process is available then the system call calling process will be moved to preempted state and all it's data will be backed up. The higher priority process will then be moved from ready to kernel ~~run~~ running.

- Time slice expiry is also a system call so when a user prog has expired time it will have a system call. Now if a higher priority process is in ready then the previous process will be

moved to preempted after taking backup.
- From preempted the process is moved to ready state.
- Preempted is a temporary intermediat state.

- Preempted — Ready to Run is bidirectional. If the process is changed then a process from ready needs to be moved from user running via the preempted state.

- If a prog. has time slice 50 units and it called a system call at 20 units then it will be moved to kernel running now while returning it may be moved to preempted state dep. on the priority even though it ~~may~~ have time left to execute ~~that~~ — That's why system call is bad because the calling process may not get a chance to execute again immediety.

- Context Saving — If a process is changed then the previously running process has to save all it's data in reg., process control block and stack. ~~and the new process also has~~

- The new process also has to load all it's saved previous data to the correspond reg, stack and process control block.

- User running and kernel running do not run ~~simult~~ simultaneously, only one of these process run at a time — CPU can execute only 1 process at a time.

- Preemption require two process — the preempted process will move from kernel running to preempted to ready and new will

★ LINUX System Model —

Running State — Two states
- Executing state for running process.
- Ready state for ready process.

- Blocking state is divided in 2 —
Uninterruptable — Moves to ready only when the event is finished.
- Any other signal cannot ~~have~~ move to ~~any~~ ~~this~~ ready.

Interruptable — Moves to ready ~~with~~ when either the event has occured or there is a signal to move it to ready.