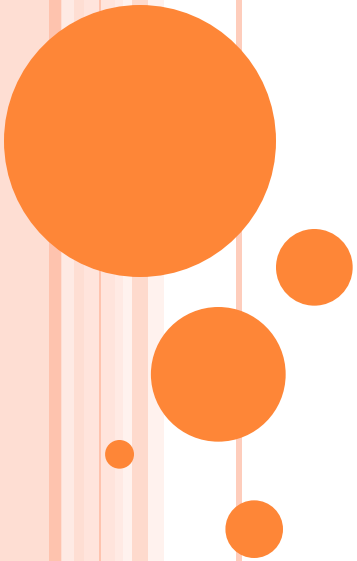


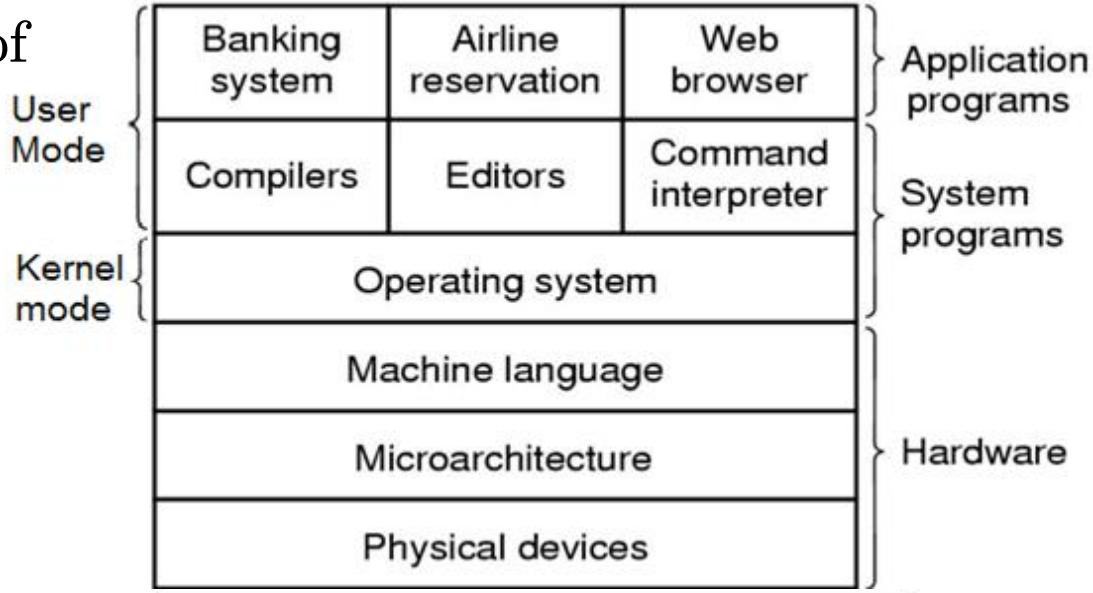
Introduction to OS and Process management

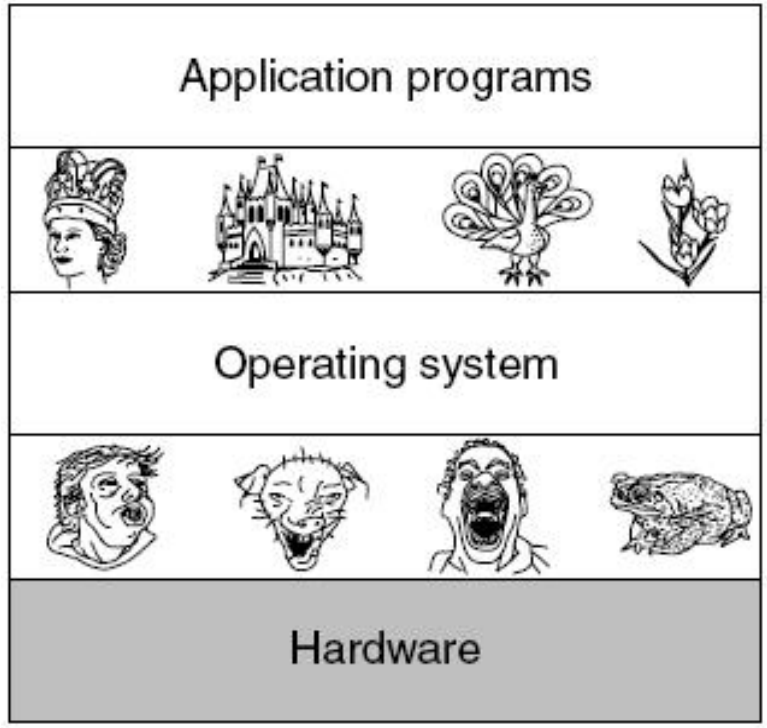
SPOS Unit III

What is an Operating System & it's Functions

- A computer system consists of
  - hardware
  - system programs
  - application programs
- Operating system functions

1. It is an extended[virtual] machine
   - Hides the **messy details** which must be performed
     - Eg writing of floppy
   - Presents user with a virtual machine, easier to use
2. It is a resource manager
   - Each program gets time with the resource
   - Each program gets space on the resource





OPERATING SYSTEM FUNCTION CONTD…

- Computational structures:-
  - Single program
  - Sequence of single program
    - bat file
  - Collection of program
    - cat grep etc
- Concept of resource:- may be H/W or file etc.
- Types:-
  1. System provided resource:- like CPU, Memory
  2. User created resource:-eg file
  3. Preemptable[memory] or non-preemptable resources[cd]

- Sharing of resource
  - If multiple programs tries to print simultaneously.
- 2 ways of preemption:-
  - Sequential sharing:- CPU sharing via Scheduling
  - Concurrent sharing:-like file in read mode
- Keeping track of allocation.
- Require Preemption of resource:- forceful deallocation
- Protection of resource
- Allocation for resource i.e. binding[No. of instances]
  - Based on privileges

Resources

- Allocation Strategies :-
  - Partitioning of resource:-***Static allocation*** approach & hence wastage of resources but easy & widely used for memory.
  - Allocation from pool:- Dynamic allocation approach , Stores entire resources in ***resource table*** & will be allotted *to* ***Privileged*** process if its free like :CPU & memory sharing
- Eg of computer resources:- printers, tape drives, tables
- Processes need access to resources in reasonable order
- Suppose a process holds resource A and requests resource B
  - at same time another process holds B and requests A
  - both are blocked and remain so→deadloack
- Sequence of events required to use a resource
  1. request the resource:- Must wait if request is denied[requesting process may be blocked or may fail with error code]
  2. use the resource
  3. release the resource

Evolution of O. S. Function

- Efficient utilization of available resources
- New feature in computational architecture
- New user requirements
- Different operating system deals these issues in different manner, but most OS contains components with same functionality. Thus need not to study many OS.
- Concept of Job, program, process, batch, coroutines, threads.
  - Job :- sequence of programs in order
  - Program:- sequence of instructions
  - Process :-execution of program or part of program
  - Threads:- will be discussed soon.

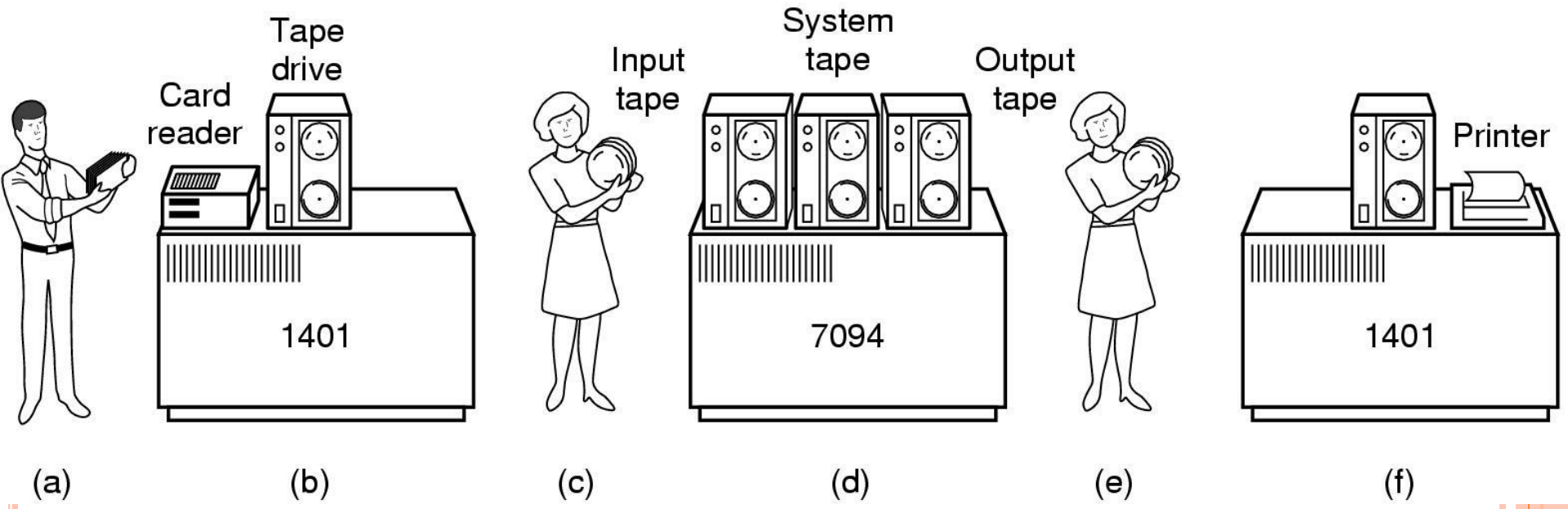
History of Operating Systems

- Zero generation 1700-1900
  - analytical engines by Babbage with ADA for prog.
  - Neumann & Harvard (relay)
- First generation 1945 - 1955
  - vacuum tubes, plug boards
- Second generation 1955 - 1965
  - transistors, batch systems
- Third generation 1965 – 1980
  - ICs and multiprogramming
- Fourth generation 1980 – present
  - personal computers

Various Operating System

- Batch processing system
- Multiprogramming System
- Timesharing system
- Mainframe operating systems
- Server operating systems
- Multiprocessor operating systems
- Personal computer operating systems
- Real-time operating systems
- Embedded operating systems
- Smart card operating systems

BATCH MONITOR FUNCTION



Early batch Processing system:- offilne

- bring cards to 1401
- read cards to tape
- put tape on 7094 which does computing
- put tape on 1401 which prints output

Turn around time:-

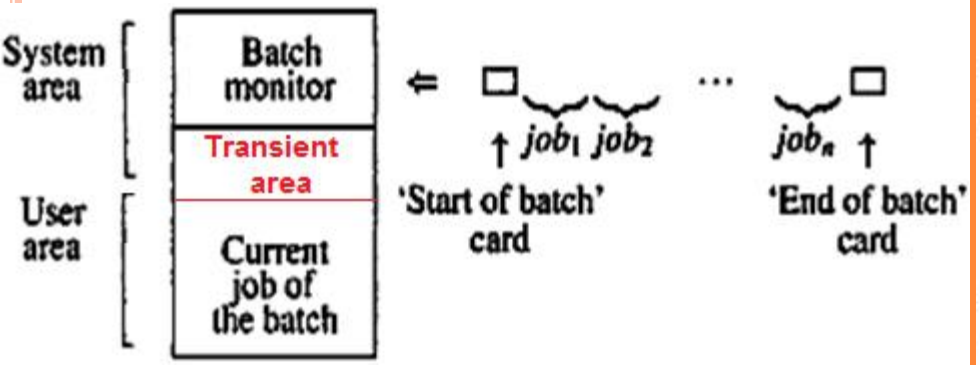
- Time since job submitted to time its result become available.
- Depends on:-batch formation time, execution time, printing time.

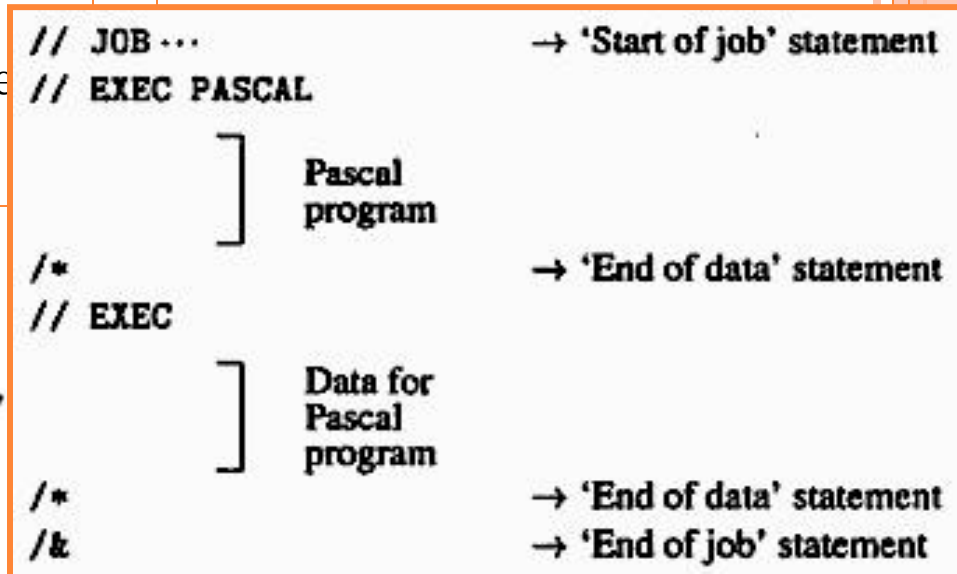
Batch processing does not aim at improving user service but CPU utilization

BATCH MONITOR FUNCTION

- Before initialization:-
  1. Scheduling:-
     - Scheduling is a activity of determining which service request should be handled next by server.
     - Here its FCFS
     - Non-preemptive
     - Its job of user/programmer.
  2. Memory management:-
     - Refer areas in fig. below.
     - Overlay structure used for Transient area(Exclusive functions)
     - But then it needs protection mechanism as memory is share among application & monitor

- During execution:-
  1. Sharing/allocation:- done by user itself at initialization & termination
  2. Protection:- easy as done by user BUT prevention mechanism required if user fails to maintain sequence of cards.

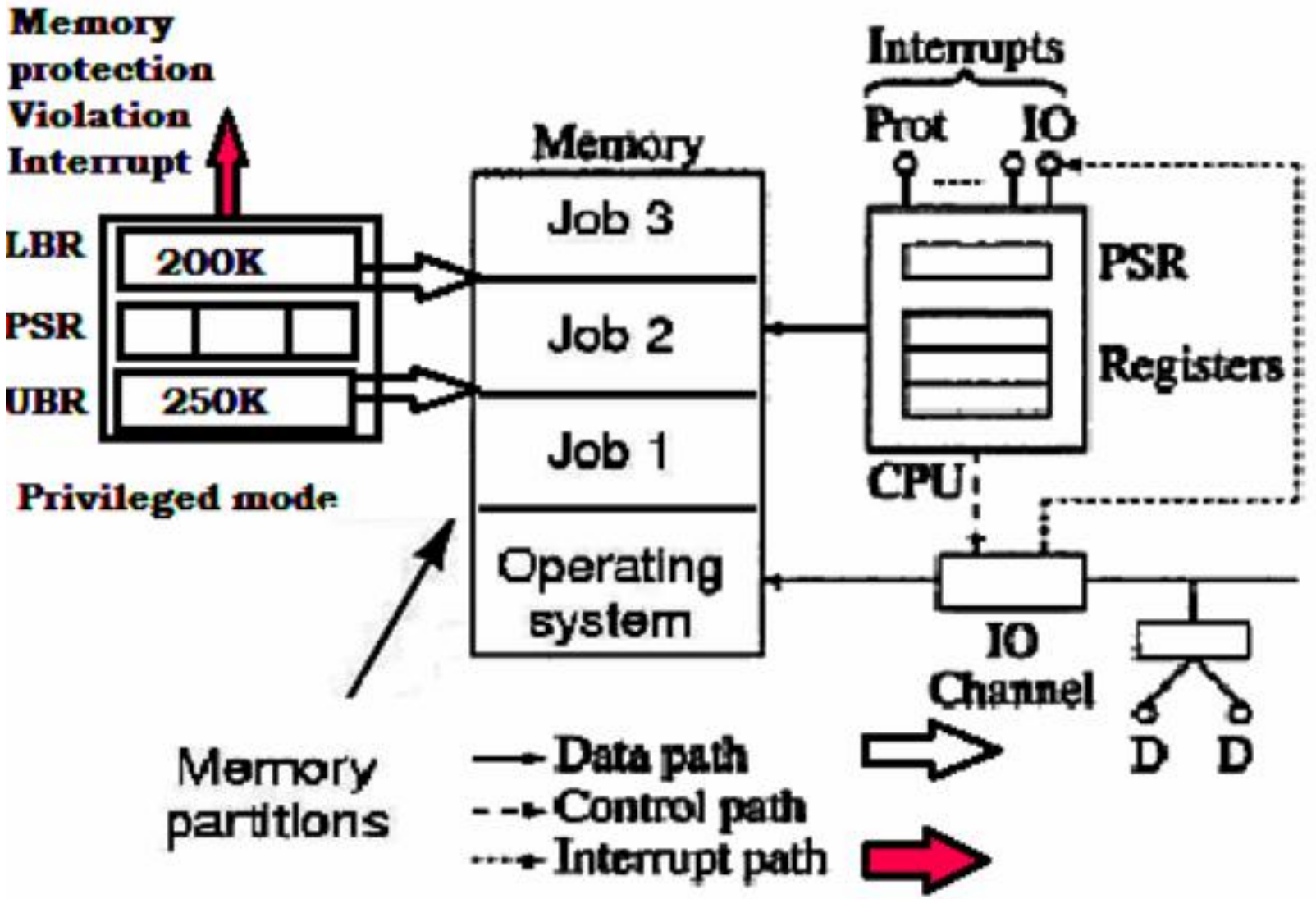




MULTIPROGRAMMING(MP) SYSTEM

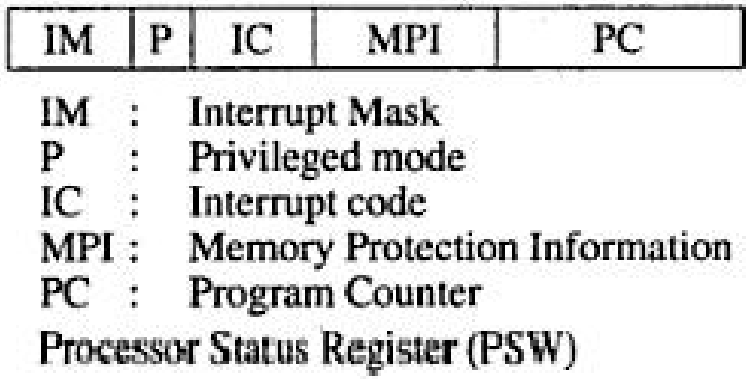
- CPU spends lot of time in WAITING. Eg. Like incase of I/O operation
- New feature permits CPU to delink itself prom I/O operation
- Its **concurrent operation** of 2 programs (I/O & CPU).
- I/O & memory sharing is via partitioning, hence no interference.
- Architectural support:-
  - **IO channel:-**refer diagram
  - **Interrupt hardware:-** refer diagram
  - **Memory protection & synchronization** :- Concept of *fence* i.e. Memory base registers like Lower & upper bound register(LBR & UBR respectively)
  - **Privileged mode of CPU operation**:- required for fence, IO initialization, changing PSW flags, Enable Software Interrupt[SI]
- Multiprogramming system
  - three jobs in memory – 3rd generation
  - Eg:- 360 family
    - OS/360 by IBM
  - Snooping & spooling

MULTIPROGRAMMING(MP) SYSTEM CONTD…



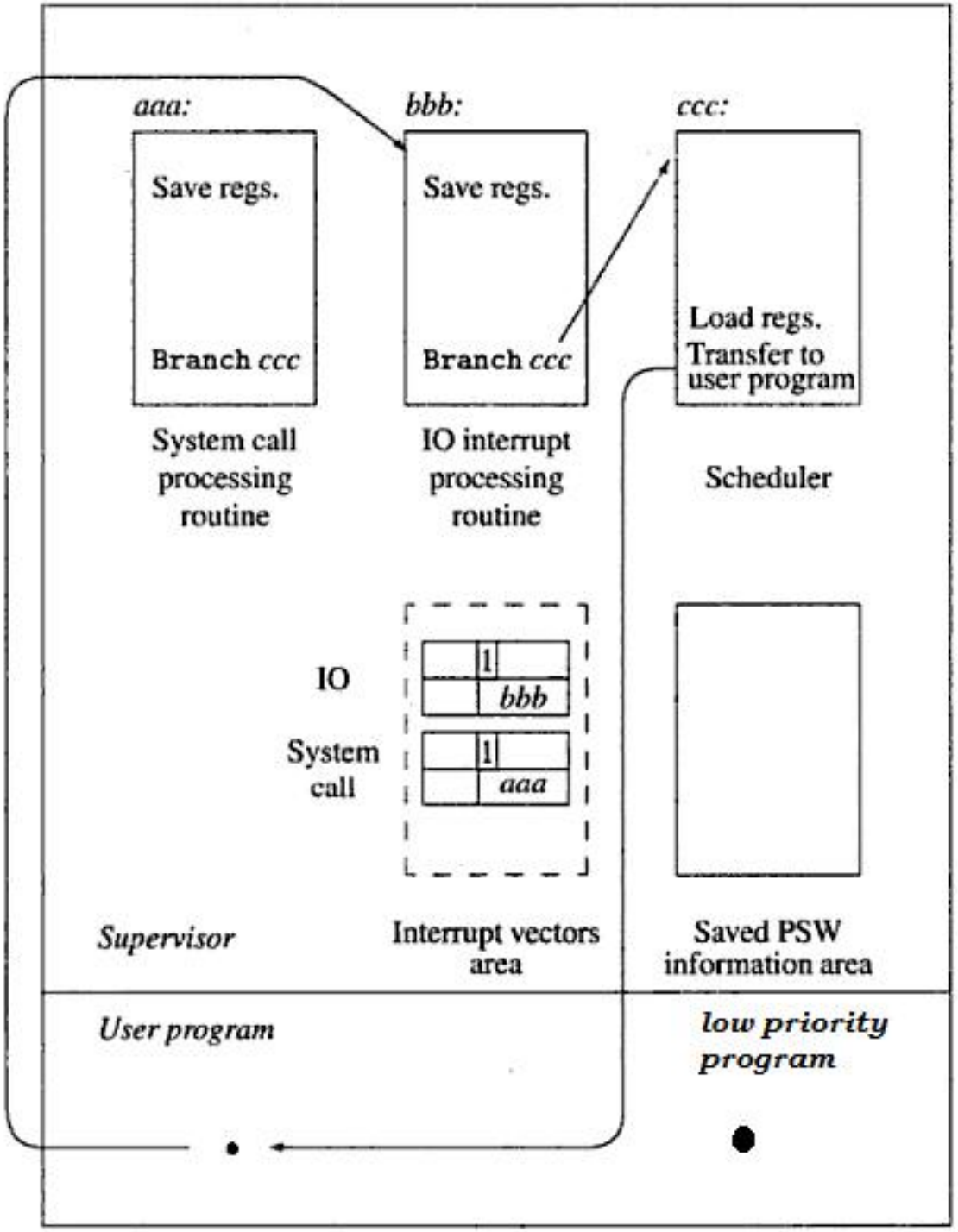
MULTIPROGRAMMING OPERATING SYSTEM CONTD…

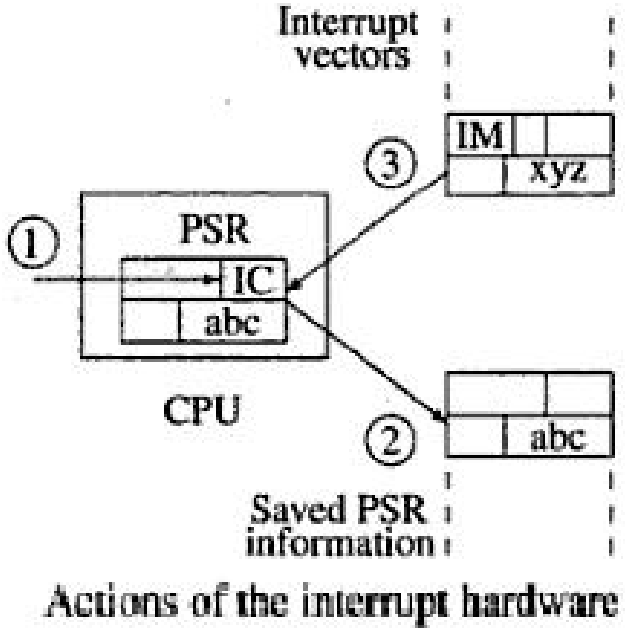
- Interrupt Hardware & processing:-refer fig
  - Process of interrupt generation using flag to acquire attention of processor
  - Contain of Interrupt Vector Table[IVT]:-
    1. Address of routine
    2. Mask status[same copy of PSW register in memory]
    3. Whether CPU should be in privileged mode while processing



Processor Status Register (PSW)

  - Steps[refer next slide for details
    1. House keeping action(save context)
    2. Execute Interrupt [PSW referred for code of interrupt]
    3. RETI:- rather than returning to native routine it transfers to scheduler. if any high priority job exist, it will transfer to that job. Else returns
- **System call**:- A method by which program makes request to OS. These are nothing but software interrupts. Eg:- INT 21H in DOS
- **Can not ensuring user services**:- as no direct relation between turnaround time & execution requirements.





Actions of the interrupt hardware

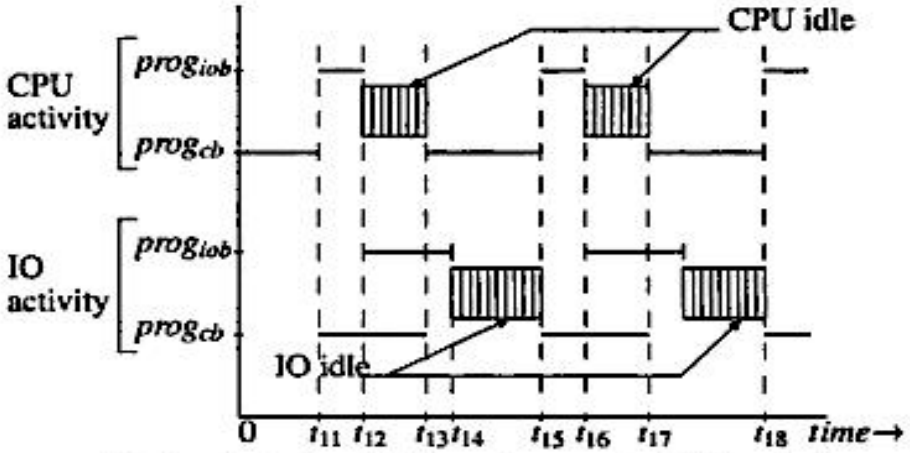
MULTI-PROGRAMMING OPERATING SYSTEM CONTD…

FUNCTION OF MP OS

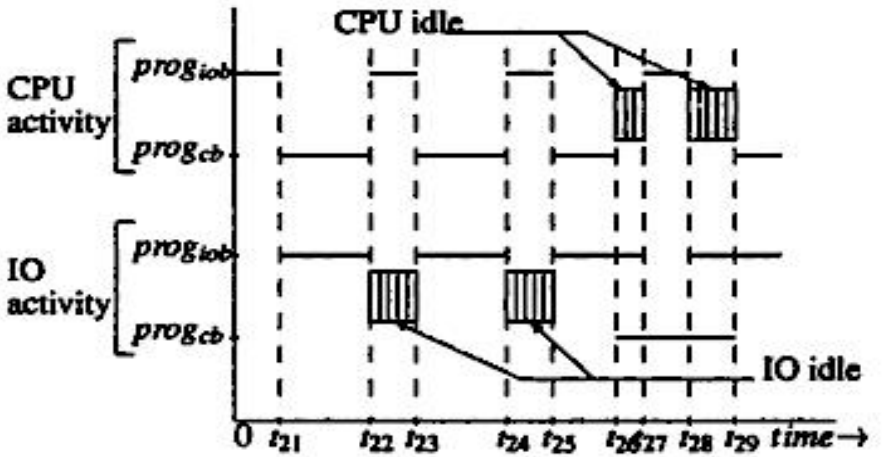
$$\text{Throughput} = \frac{\text{Number of programs completed}}{\text{Total time taken}} = \frac{n}{t_f - t_0}$$

1. Scheduling:-
   - Throughput:- number of programs processed by it per unit time.
   - To optimize throughput
     1. Proper mixing of IO-bound & CPU-bound program required
     2. Preemptive & priority based scheduling
     3. Degree of Multiprogramming:-Tread of in capacity & requirement
   - Priority:-Tiebreaking notion if more than one requests raised

Higher priority to CPU-bound prog.	**Higher priority to IO-bound prog.**
CPU utilization reasonable	CPU utilization reasonable
Poor IO utilization	Reasonable IO utilization
Period of concurrent CPU & IO	Frequent period of concurrent CPU & IO



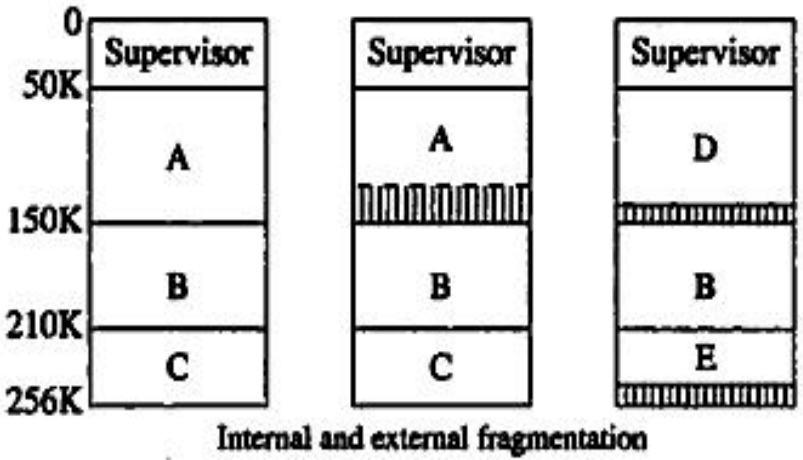
Timing chart when CPU-bound program has higher priority



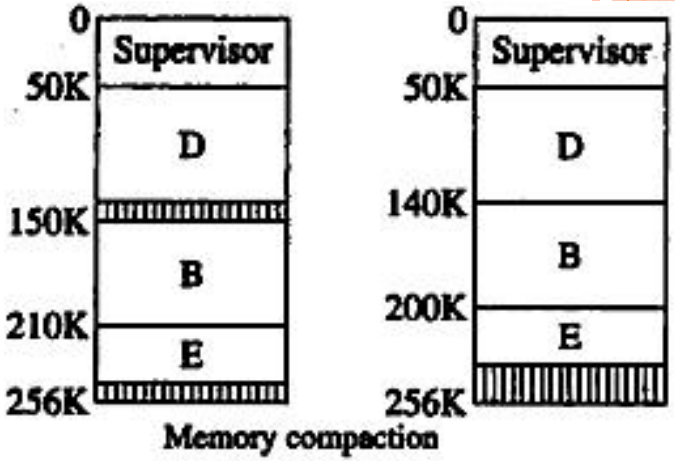
Timing chart when IO-bound program has higher priority

FUNCTION OF MP OS

2. Memory allocation:- static partitioning,
   - Protection:-considering mutual execution only hence no interference. LBR & UBR helps. Nonprivileged mode ensures.
   - Preemption:- forceful deallocation of CPU from program
   - Memory fragmentation:- existence of unusable memory areas in computer system.
     - 2 approaches to overcome this problem:- both r very expensive
     1. Memory compaction:- physical shifting of job. So reallocation
     2. Virtual memory:- require expensive hardware.
3. I/O management:-needs detailed analysis.



Internal and external fragmentation



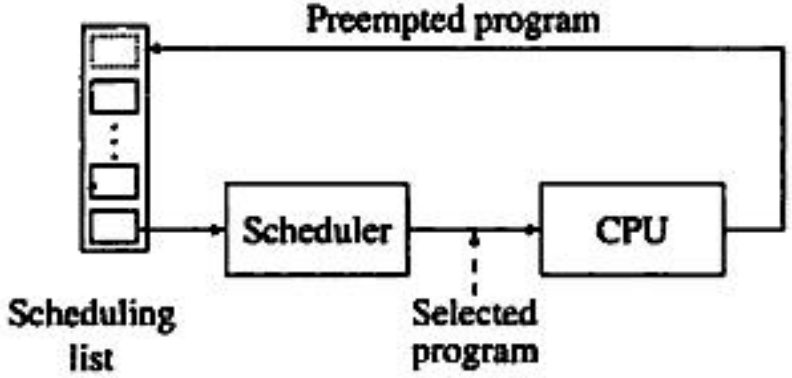
Memory compaction

TIME SHARING SYSTEM:-

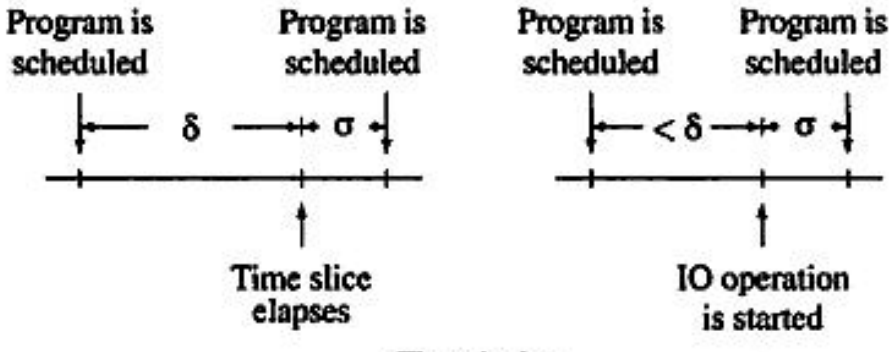
- User started demanding interactive service rather than turnaround time as multiuser OS proposed.
- It creates an illusion that each user had a computer- some what slower actual computer. It improves productivity of user.
- User services:-
  - Response time:-time between submission of request & start reporting of result to user . Hence interactive environment
  - Scheduling:- combination of Round-robin and Time slicing system. Hence required timer & timer interrupt.
    - Process will be preempted if 1) end of Q [fig a]2) IO operation[fig b]

response time $rt = n.(\delta+\sigma)$

utilization efficiency $\eta = \frac{\delta}{\delta+\sigma}$



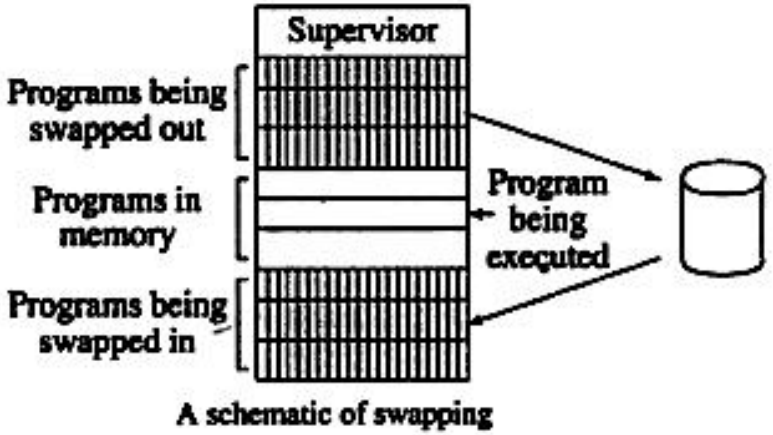
A schematic of round robin scheduling



Time sharing

TIME SHARING SYSTEM:-

- Choice of Q is very important
- Memory management:- policy require computer to posses large memory. Hence swapping[temporarily remove from memory & put data, instructions of active program in disk] required. Tree types are:-
  1. Being swapped out
  2. Being swapped in
  3. Active.
  - Swapping is possible only because we can predict ***when & how***
  - But it improves OS overhead



A schematic of swapping

RTOS:-

- Real time application:- application which require "timely" response to prevent failure. But not quick execution. i.e. **predictable worst-case response time**
- Concept of task proposed. True sence multitasking
- Facilities integrated r:-
  - Multitasking within application
  - Priorities of task
  - Deadline oriented Priority schedul
  - Programmer defined interrupt
- Type:- SOFT & HARD RTOS

Operating System Concepts

- Following are the things to be discussed in details:-
- Processes
- Deadlock
- Memory management
- input/output
- Files
- Security
- shell
- Above point will be discussed in respective chapters

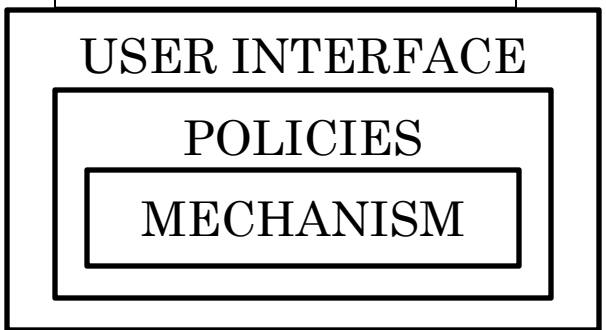
Operating System Structure

- Previously OS faced problems due to **monolithic** structure. As no clear **interface** between architecture dependent & independent. It restricts 1) size 2) sharing ability.
- Strongly depends on two factors:-

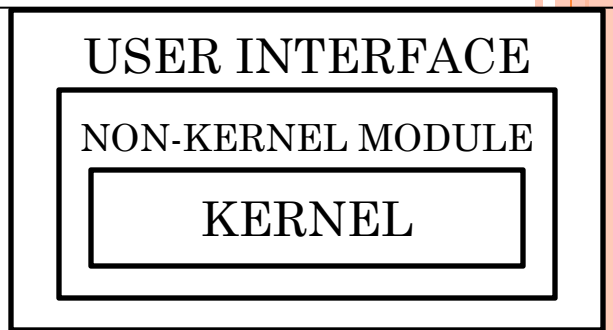
1. Architectural features of computer:-
   - H/W dependant, hence **mechanisms** are depends on this features
2. Features of application domains:-
   - H/W independent, hence **policies** are depends on this features

- Kernel (interrupt driven):- it gets control via **INTERRUPT** when
  1. Exceptional situation arises:-
  2. OS policy module explicitly invokes mechanism
- OS microkernel:-to overcome disadvantage of layered structure
- dependency to service provider
- poor extensibility
- large size of kernel
- hence unreliable

Structure of an OS



OS layers -extended machine



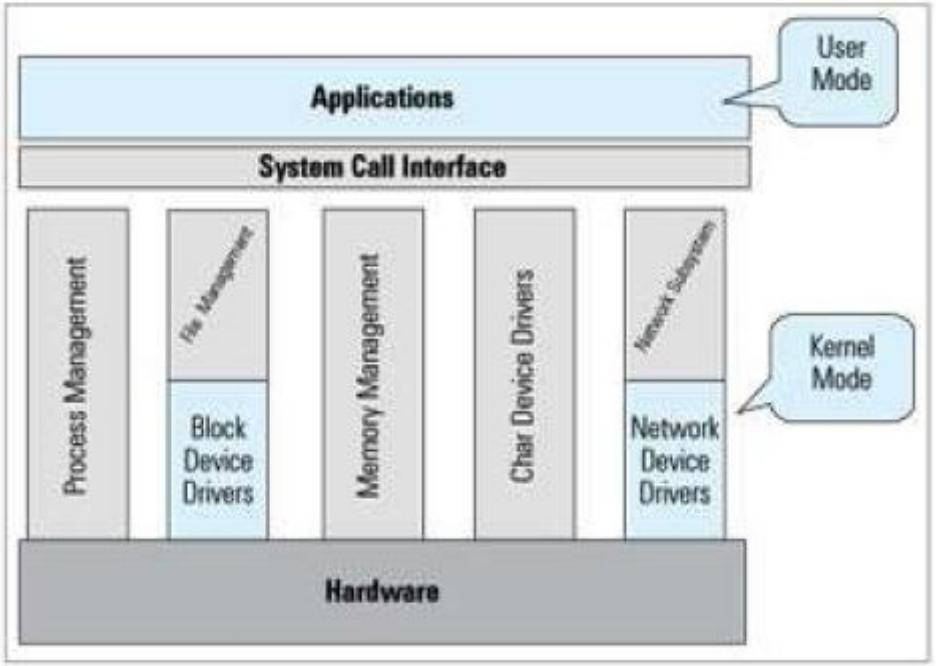


Figure 1: Architecture of a monolithic kernel-based operating system

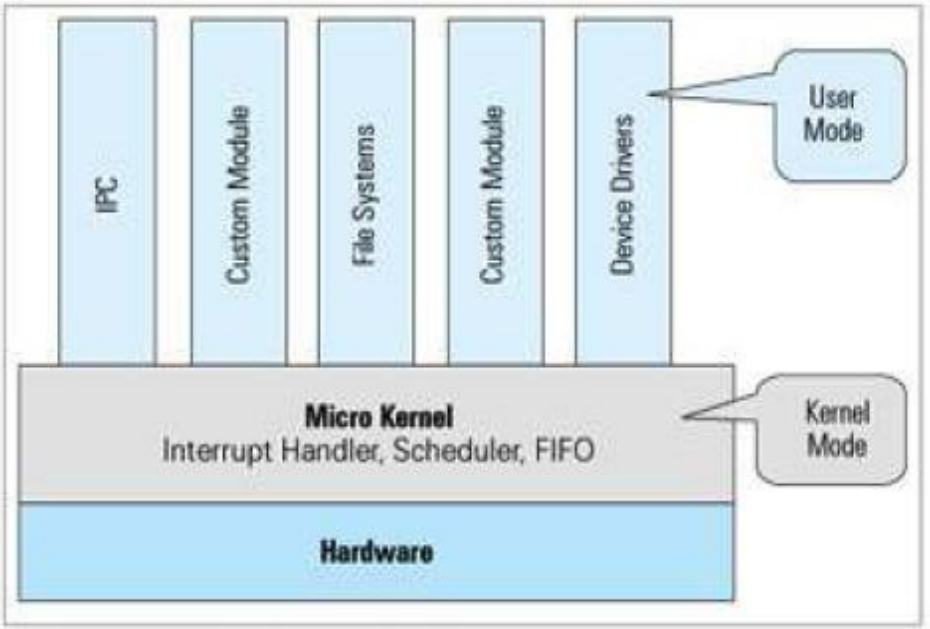


Figure 2: Architecture of a micro kernel

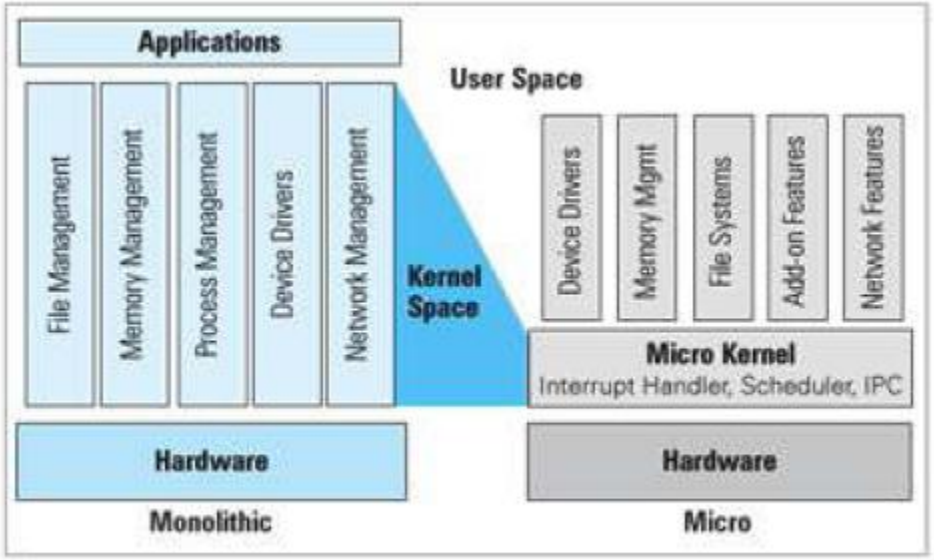


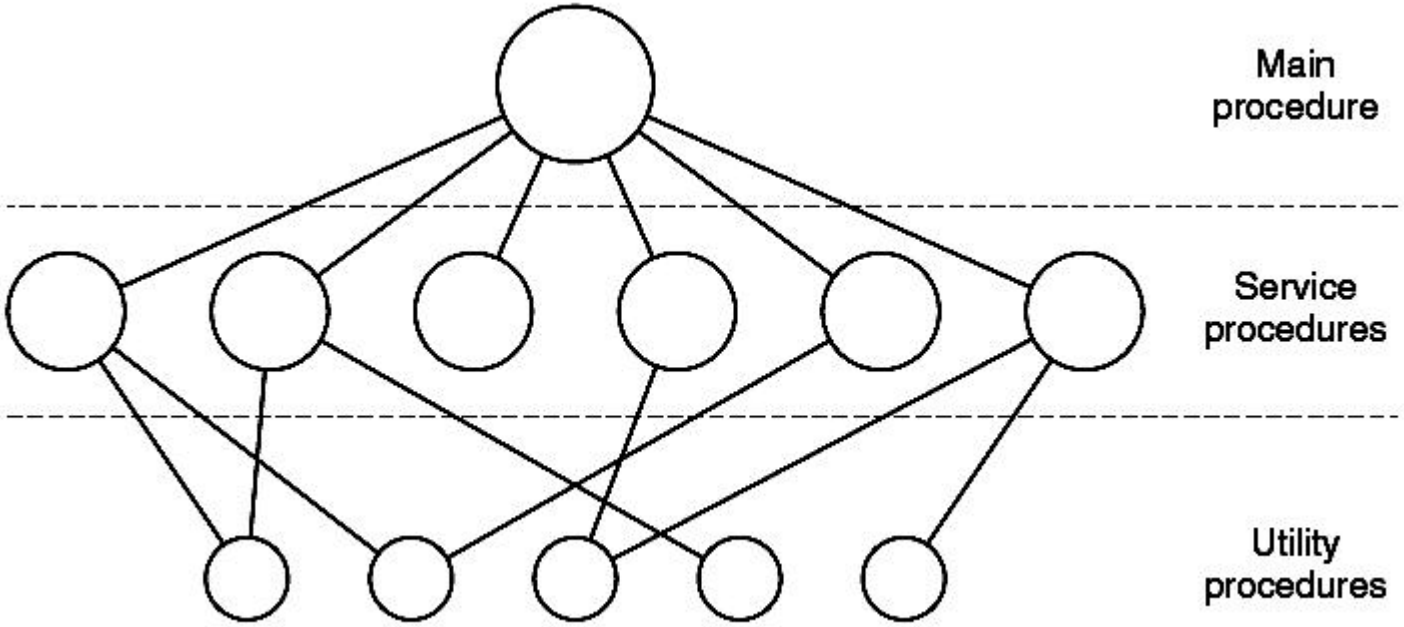
Figure 3: Comparison between monolithic and micro-kernel-based OSs

- ***Kernel mechanisms***:-
- Interrupt processing mechanism
- Scheduling:- dispatch & preemption
- Memory management:-
  - protection, swapping, virtual
- IO management:-
  - Initiation, completion, error, scheduling
- Communication:-
  - IPC & networking

Monolithic Structure

- Basic structure i.e. No structure
- First need to compile all procedures
- Traps with stack are used to switch from user mode to kernel one

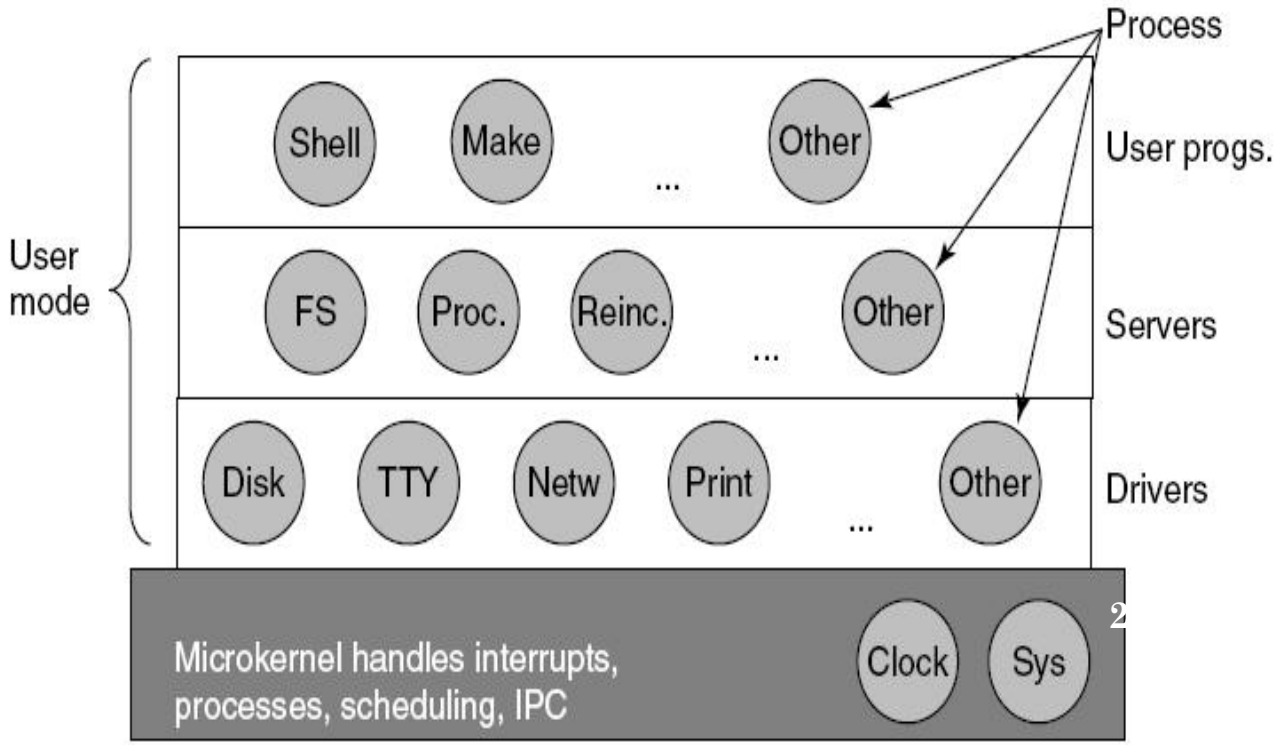
1. Main program that invokes requested service procedure
2. Set of service procedures that carry out system calls
3. Set of utility procedures that help service procedures



Layered Structure

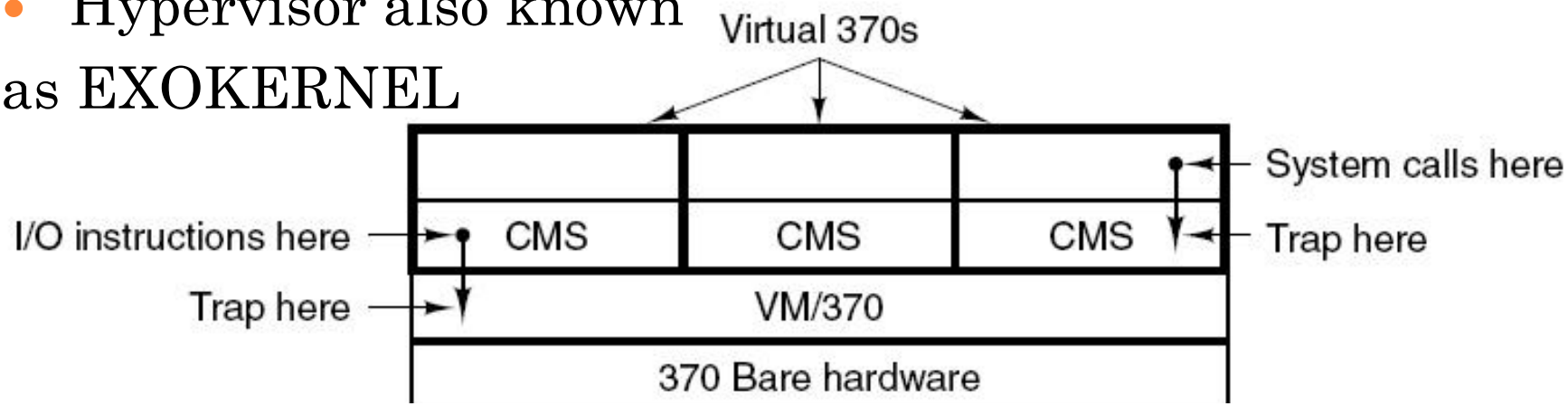
- Eg:- THE system designed by Dijkstra(1968).
- The MULTICS system present idea of concentric rings.
  - Inner ring more **privileged** than outer one.
  - **Traps** used to access services of inner ring.

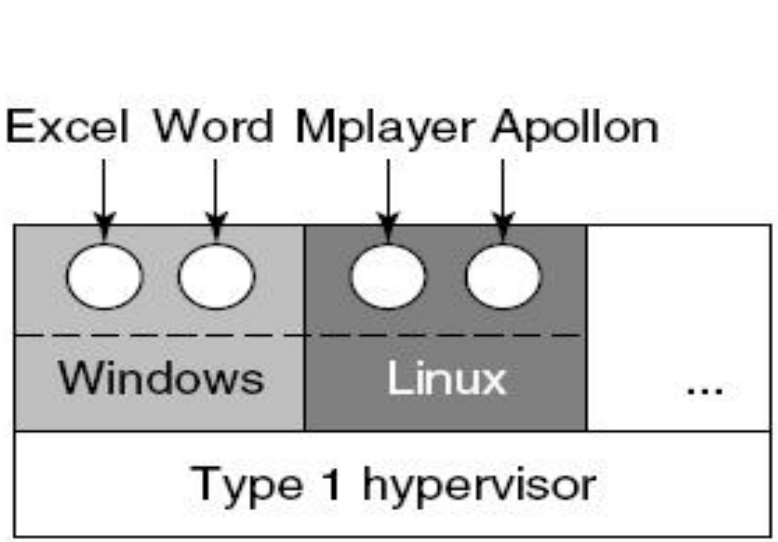
Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

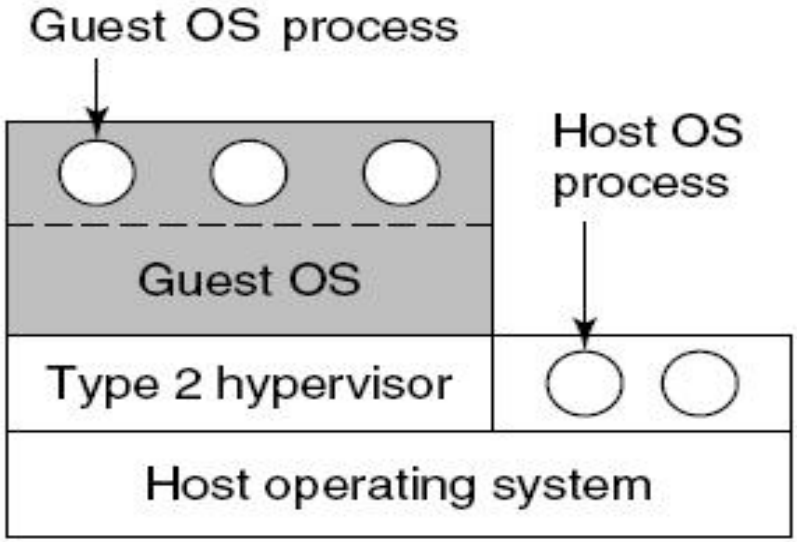


VIRTUAL MACHINE

- In contrast with timesharing system it separates *multiprogramming* from *extended machine*
- *Eg* 8086 mode on 80386 *or JVM with JVM interpreter.*
- Hypervisor also known as EXOKERNEL

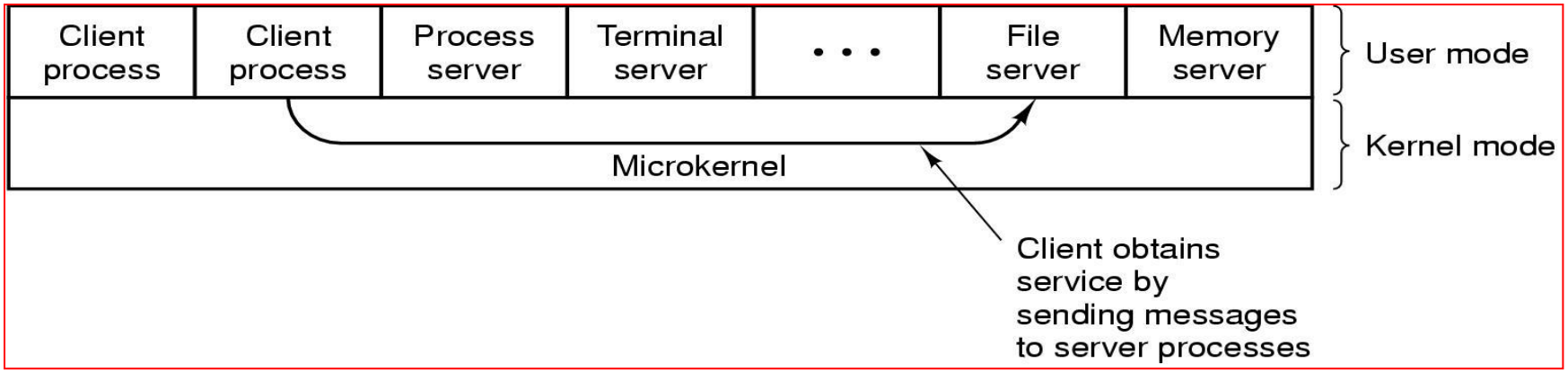


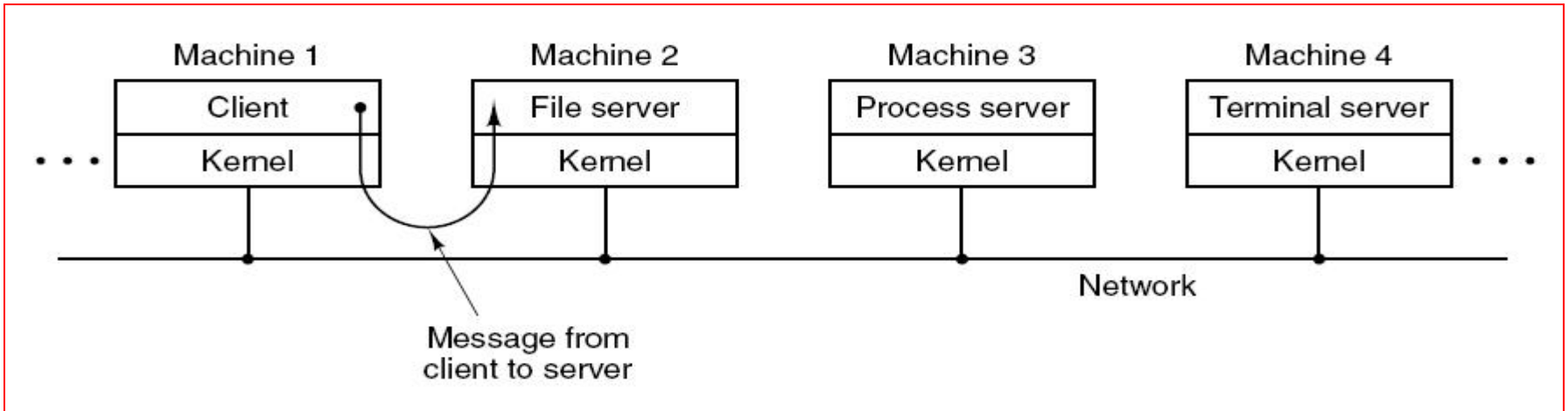




Client server model

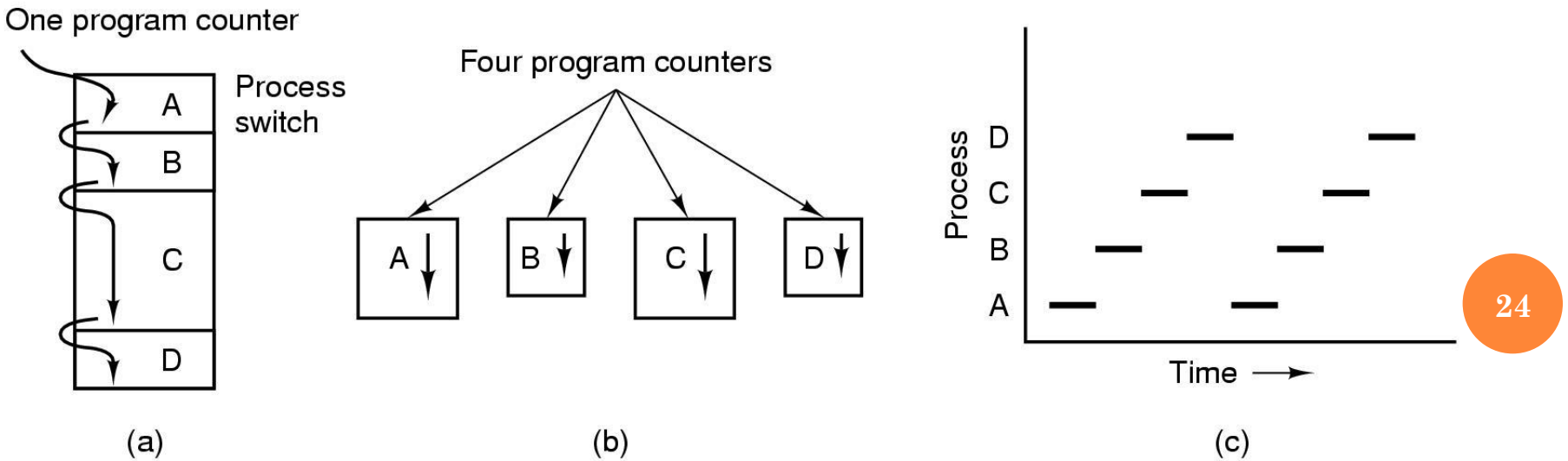
- Usual approach is to implement most of OS in user process than kernel one. Kernel only handles communication between client-server
- Only individual services crash





Processes & Process mode

- Process : abstraction of program running
- In contrast with multiprogramming or multitasking system psuedoparallelism request multiprocessing system.
- Multiprogramming of four programs(a)
- Conceptual model of 4 independent, sequential processes(b)
- Only one program active at any instant©
- Hence process must not be programmed with built-in assumption regarding time.
- Also SCHEDULING, PRIORITY, etc. comes in picture.
- Eg:- computer Scientist baking a cake.



PROCESS MANAGEMENT SYSTEM:-PROCESS CREATION

Principal events that cause process creation

1. ***System initialization***(foreground & background [daemons])→ps or task-manager
2. ***Execution of a process creation system***(Copy-paste)
3. ***User request*** to create a new process(double click opens program in new windows)
4. ***Initiation of a batch job***(in batch OS only)

- Following **system calls** are used **by a process** to create another process
  - Eg:- in UNIX ***fork*** to create clone of running process. Then child need to execute ***execve*** to change its memory image.
  - Eg:-in Windows ***CreateProcess*** for creating a process & loading proper image in. It uses **10 Parameters** like name, function code, priority, stack, handler, pvpara.
  - Even if its not exact replica of parent it can share some of the resources from parent process like file opened. Etc.

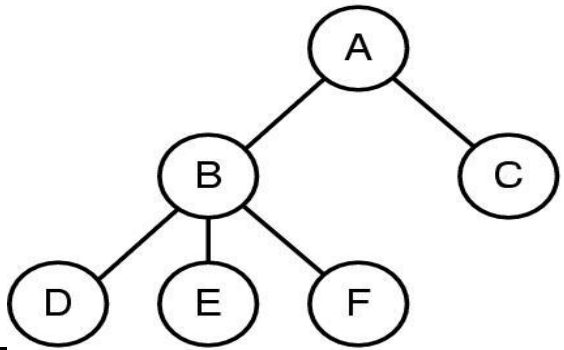
Process Termination

Sooner or later process needs to be terminated:-

1. ***Normal exit*** → means done the job, ***exit*** [UNIX] or ***ExitProcess*** [windows], button
2. ***Error exit*** → eg accessing illegal memory location, or /0
3. ***Fatal error*** → bad or missing parameter
4. ***Killed by another process*** → Killer has to be authorized eg ***kill***[unix] or ***TerminateProcess*** [Windows]

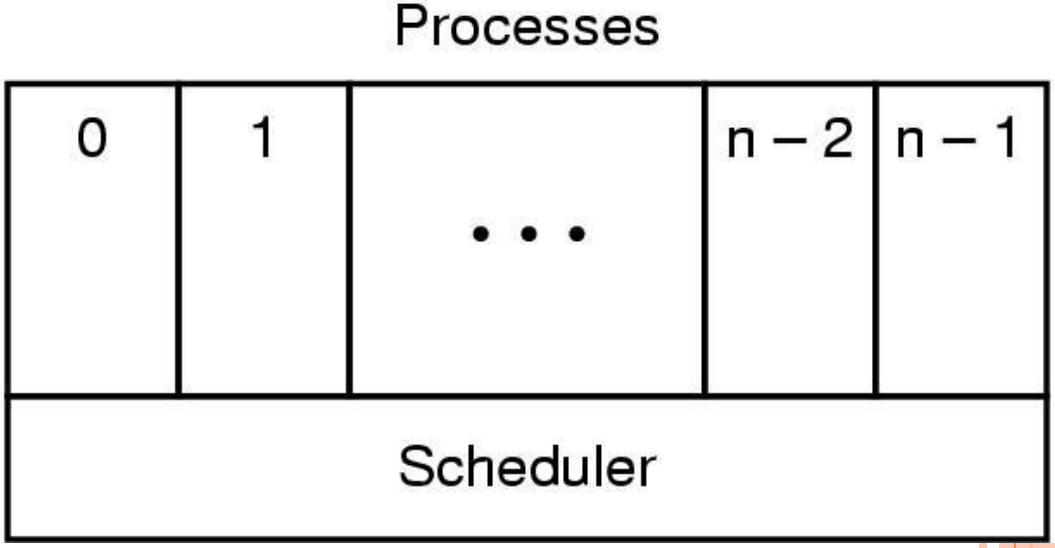
Process Hierarchies

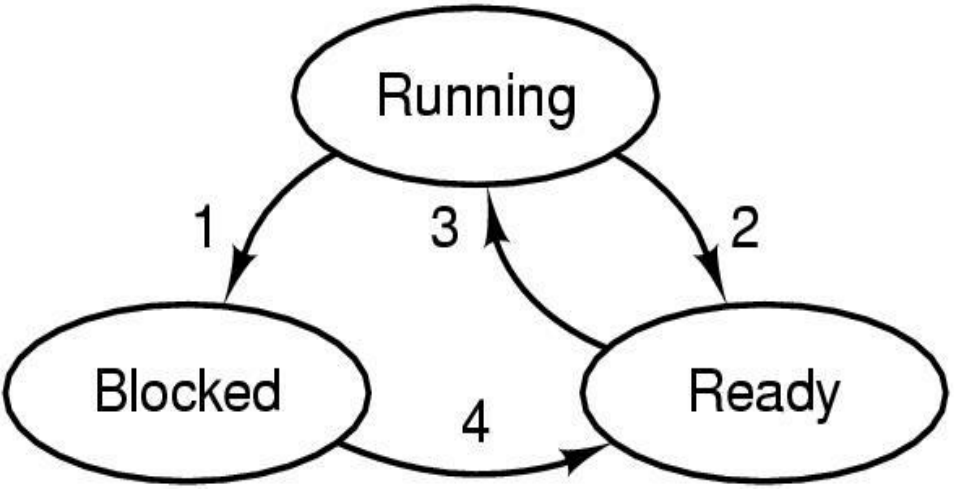
- Parent creates a child process, child processes can create its own process. & there association
- Grouping:- for accessing with single instruction/keystroke
- Unix:- initialization of UNIX from ***init***
- Windows:- no concept of hierarchies but a transferable token[Handler] is with parent



Process states

- Lowest layer of process-structured OS
  - handles interrupts, scheduling
- Above that layer are sequential processes
- Waiting time should be specified prior to going into blocked state.
- **Alarm signal**





1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Process implementation

- **Process table:-** Holds process entry
- **Process control Block:-** process entry from table [fig 1]
- Location associated with each IO interrupt is **IVT**
- Process & steps of interrupt execution shown in [fig 2]

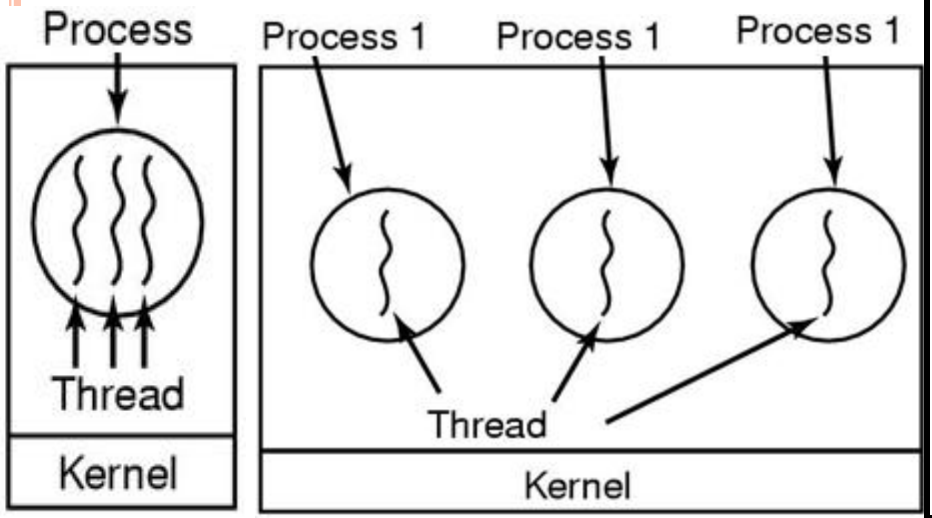
Process management	**Memory management**	**File management**
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID 0 for root
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

1. Hardware stacks program counter, etc.
2. Hardware loads new program counter from interrupt vector.
3. Assembly language procedure saves registers.
4. Assembly language procedure sets up new stack.
5. C interrupt service runs (typically reads and buffers input).
6. Scheduler decides which process is to run next.
7. C procedure returns to the assembly code.
8. Assembly language procedure starts up new current process.

THREADS & THREAD USAGE

- Process= resource grouping + execution **i.e. thread**
- Multithreading:-**Quasi-parallel** execution but in **same working environment**. Here switching is easy But **risky**.
- **No interthread protection**:- (1) its impossible (2) its not required.
- Threads can in either in blocked, running, ready or **terminated** state.
- **Complications** faced by a thread:- (1) if parent has multiple threads should the child also have them? Like in fork. (2) if thread from parent deleted or unblocked, should child too? (3) interthread communication is essential

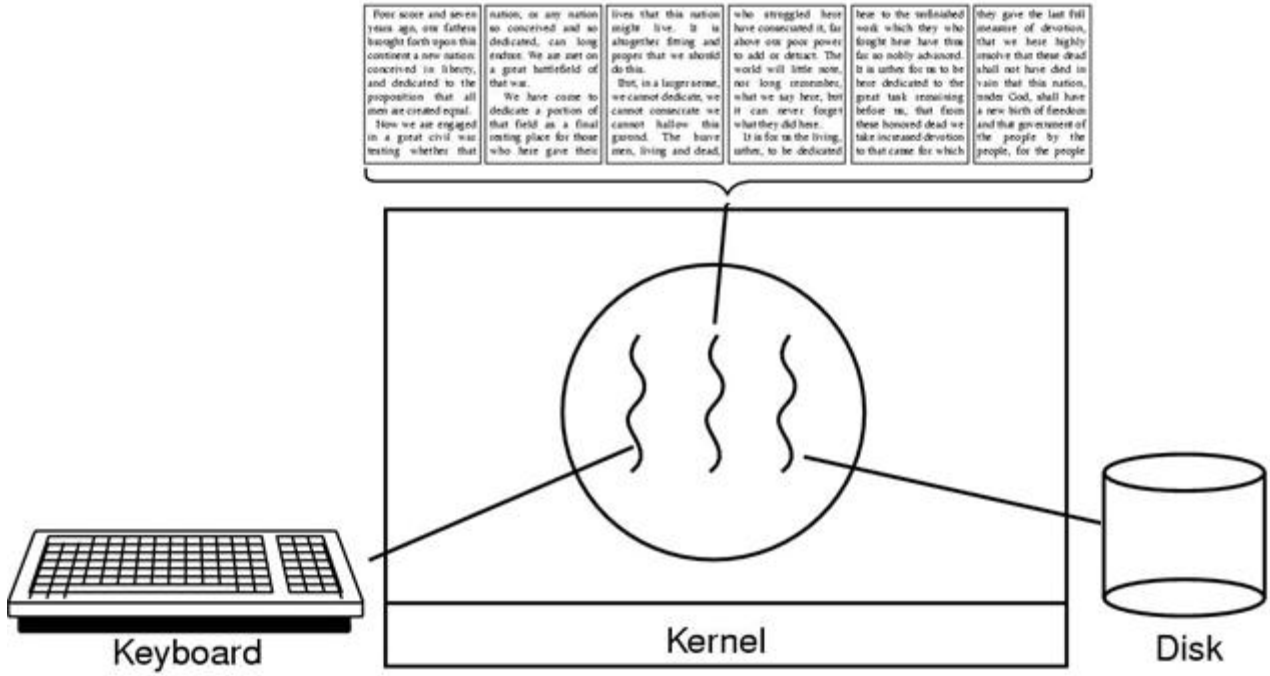
THREAD USAGE→



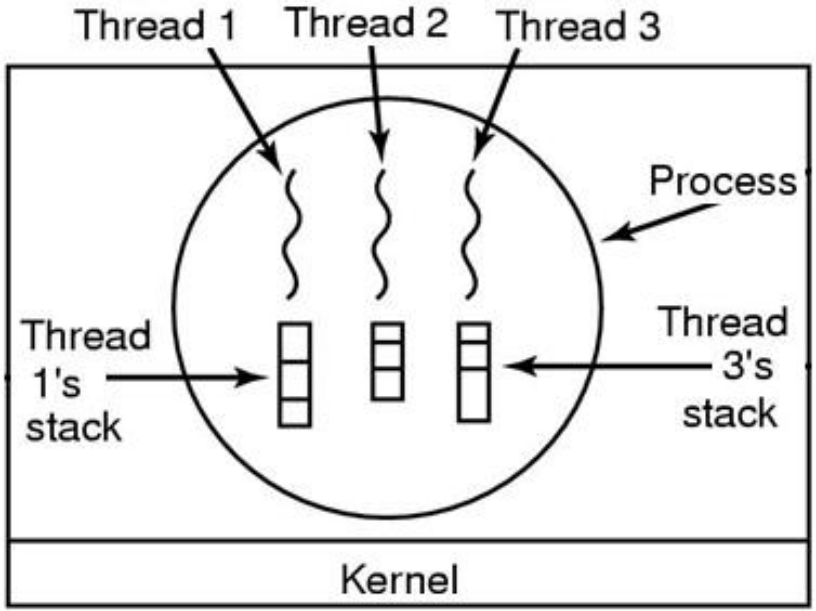
1. Multiple activities can execute even if one among them blocked.
2. May requested by application (shared address space required.)
3. Easy to create, destroy & switch. 100 times faster
4. Speedup application as CPU & I/O bound part can be overlapped.
5. Useful on multi-CPU platform.

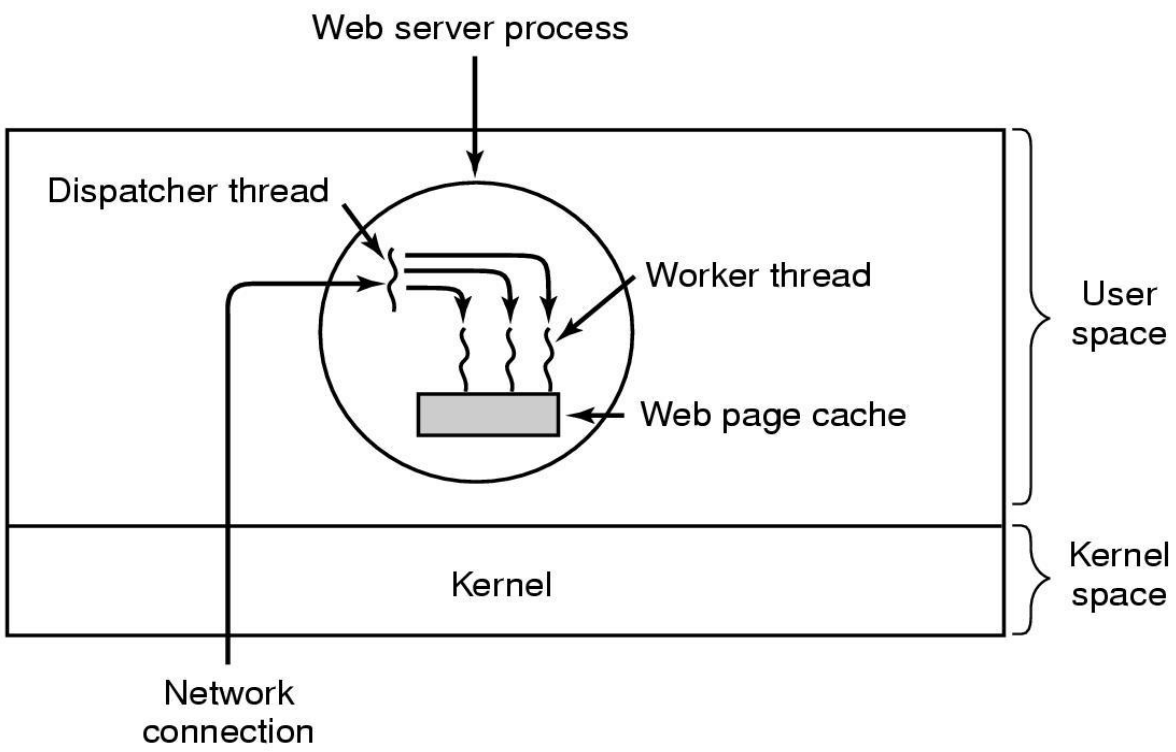
Threads Contd…

- Items shared by all threads in a process
- Items private to each thread
- Eg:- Word processor with format[1st line], edit[800th line] operation.



Per process items	Per thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	





Model	Characteristics
Threads	Parallelism, blocking system calls
Single-threaded process	No parallelism, blocking system calls
Finite-state machine	Parallelism, nonblocking system calls, interrupts

```
while (TRUE) {
  get_next_request(&buf);
  handoff_work(&buf);
}
```

```
while (TRUE) {
  wait_for_work(&buf)
  look_for_page_in_cache(&buf, &page);
  if (page_not_in_cache(&page)
      read_page_from_disk(&buf, &page);
  return_page(&page);
}
```

- Thread functions:- thread_create, thread_exit,thread_wait, thread_yield.
- Multithreaded web server
- Rough outline of code for previous slide

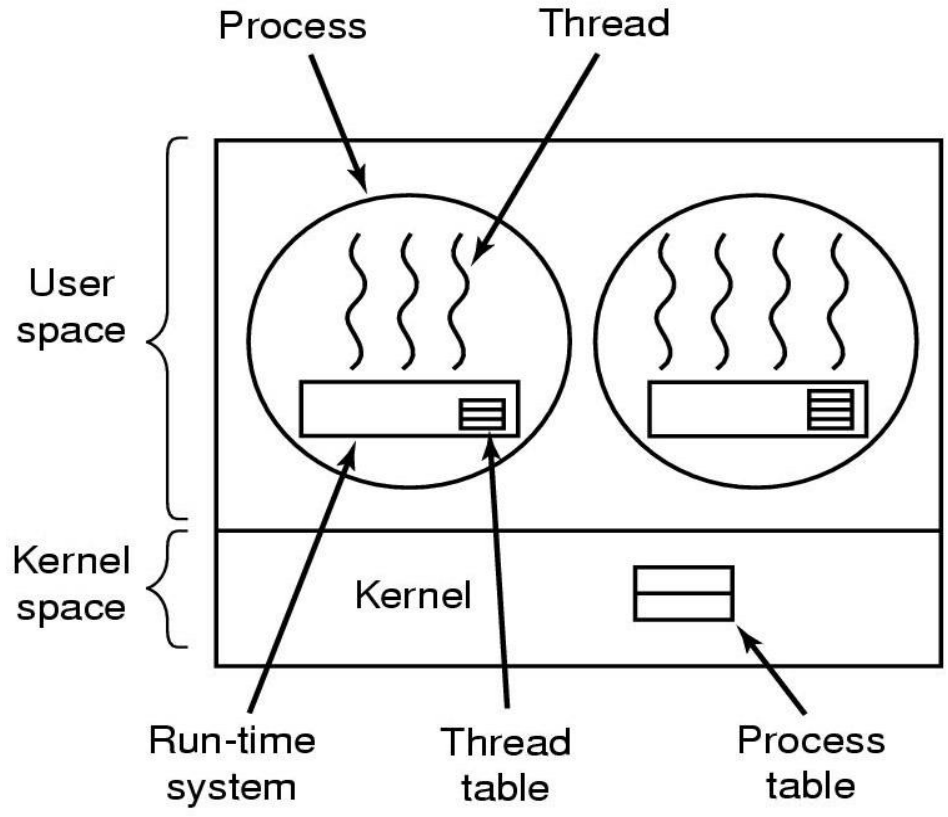
(a) Dispatcher thread

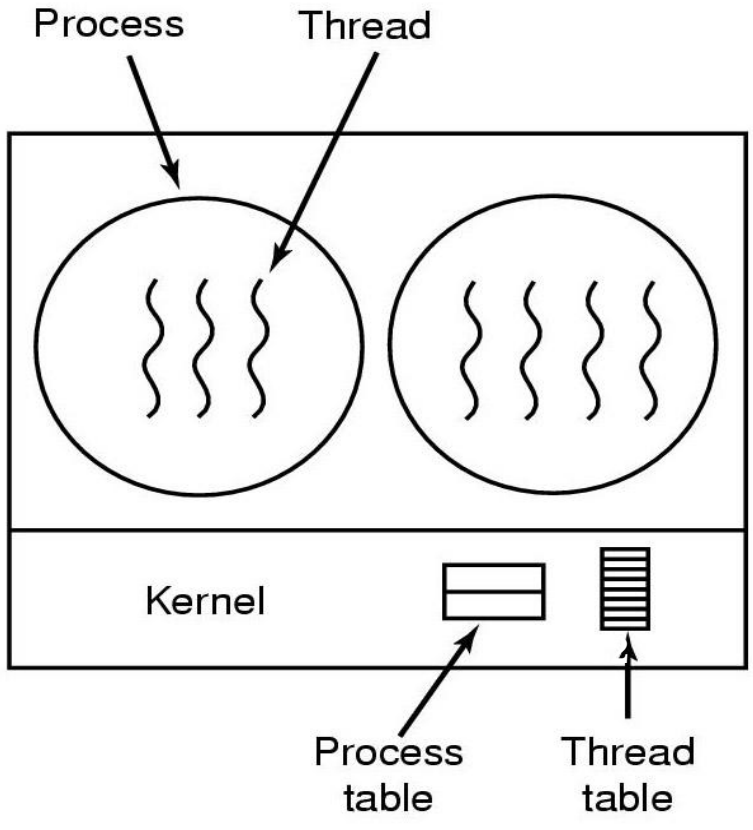
(b) Worker thread

- Three ways to construct a server

THREAD IMPLEMENTATION

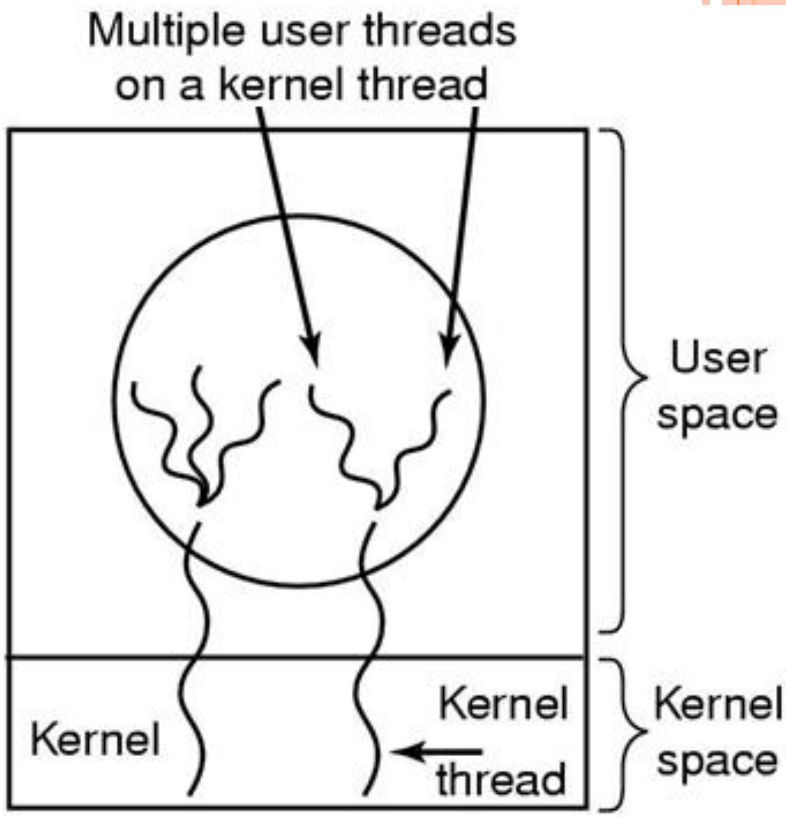
- 3 ways:- (1) In user space (2) In kernel space (3) Hybrid
- User space adv:- (1) switching easy & hence can be done in user space (2) no trap required. (3) choice of separate algorithm for each process. (4) can handle large no of threads, as only local threads are active.
- Disadv:- (1) how to implement blocking system calls, e.g.:- getch() block all threads(process). (2) may require changes in OS. Or JACKET OR WRAPPER (3)**no clock interrupt & hence wait for give-up**. (4) Programmers don't want to design program as threads.





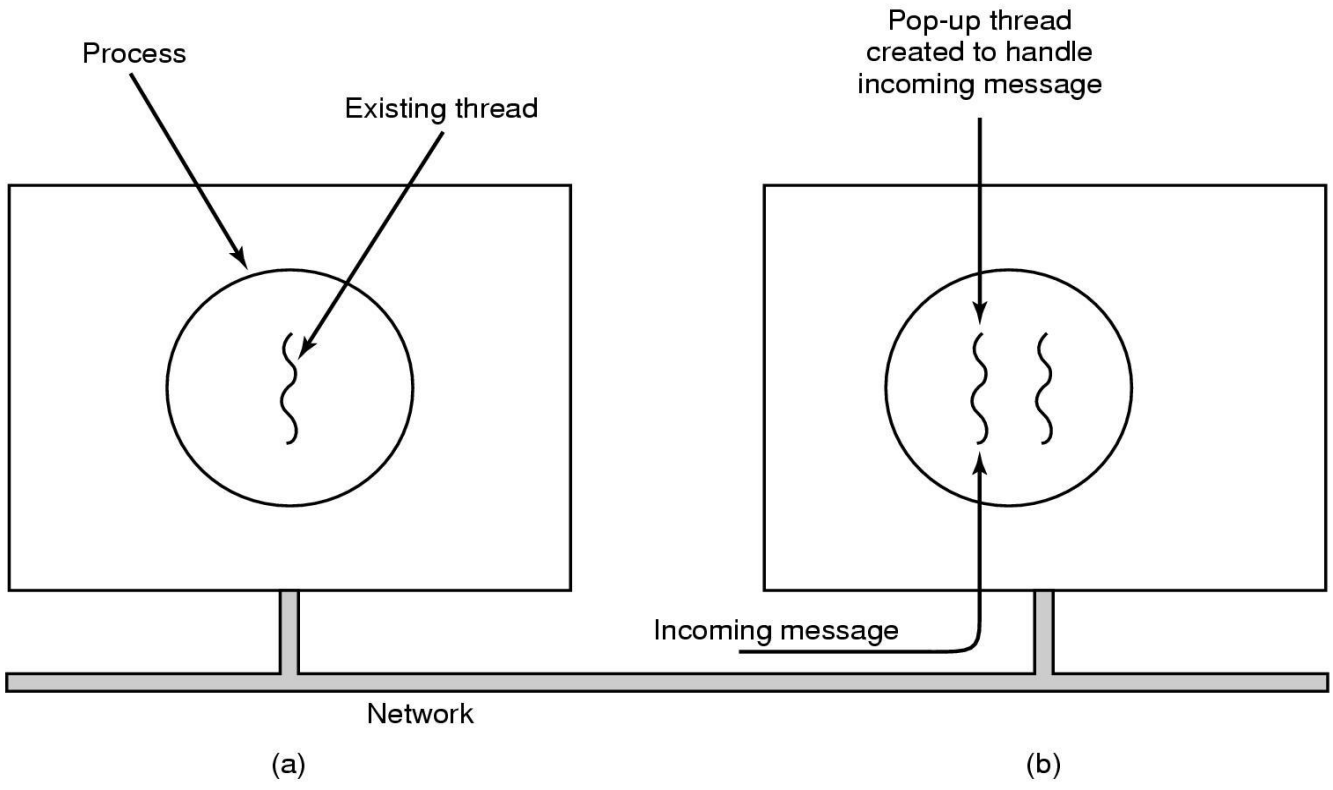
THREAD IMPLEMENTATION IN KERNEL & HYBRID

- Scheduling will be done by kernel & hence **next thread can be from any other process** also.
- Everything need to be implemented as **system call & hence costlier**.
- Termination costlier & hence better option is to **recycle** threads.
- Hybrid:- GOAL:-Combination is to mimic functionality of kernel thread in user space. Gain performance of user space threads. Heavy but best.
- Eg:-**scheduler activation:-** But user thread **should not make nonblocking system calls or check in advanced is it safe or not.**
  - Adv:-efficient:- avoiding unnecessary transaction in modes
  - Run-time-system decides blocking in coordination with Kernel which informs RTS using **UPCALLS**
  - Virtual processors assigned by kernel helps in multiprocessing environment.
  - Disadv:-Layer inversion.



POP-UP THREADS

- In distributed system:-**Traditional** mechanism uses thread in polling
- Pop-up threads are **like interrupt**:- new thread generated for new incoming message. Reduces **Latency** & response time very short.
- Brand new threads do not have any **history**. So easy to handle.
- But in which process does thread runs should be decided in advanced.
- Kernel pop-up threads are faster & easier but its buggy nature may **hamper** entire system performance.
- Creation of a new thread when message arrives
  (a) before message arrives
  (b) after message arrives



Making Single-Threaded Code Multithreaded

- Many pitfalls & should be answered:-

1. Shared data problem: eg:- *errno. variable.* → semaphore, jacket, context switching, banning sharing
2. Many library procedures are not reentrance: eg:-network → jacket, rewrite libraries.
3. Signals: as kernel doesn't know threads where to forward it.→ no sol
4. Stack management:- as kernel not involved, who will protect.

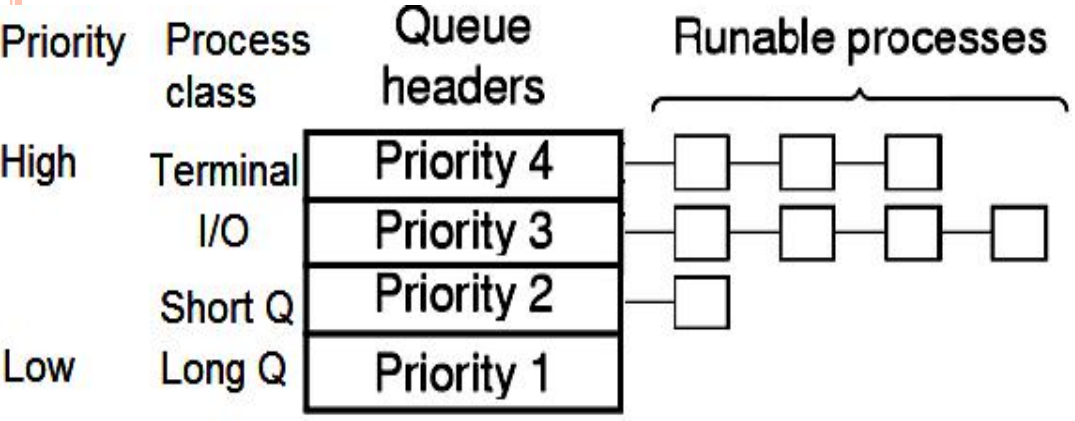
- CONCLUSION:- Semantic of system calls should be redefined, rewritten, but should be backward compatible.

InterThread Communication & Scheduling

- Passing info easy & practical.
- Protection not required & impossible
- Sequencing uses same policies as that of IPC.
- Refer scheduling slides for details

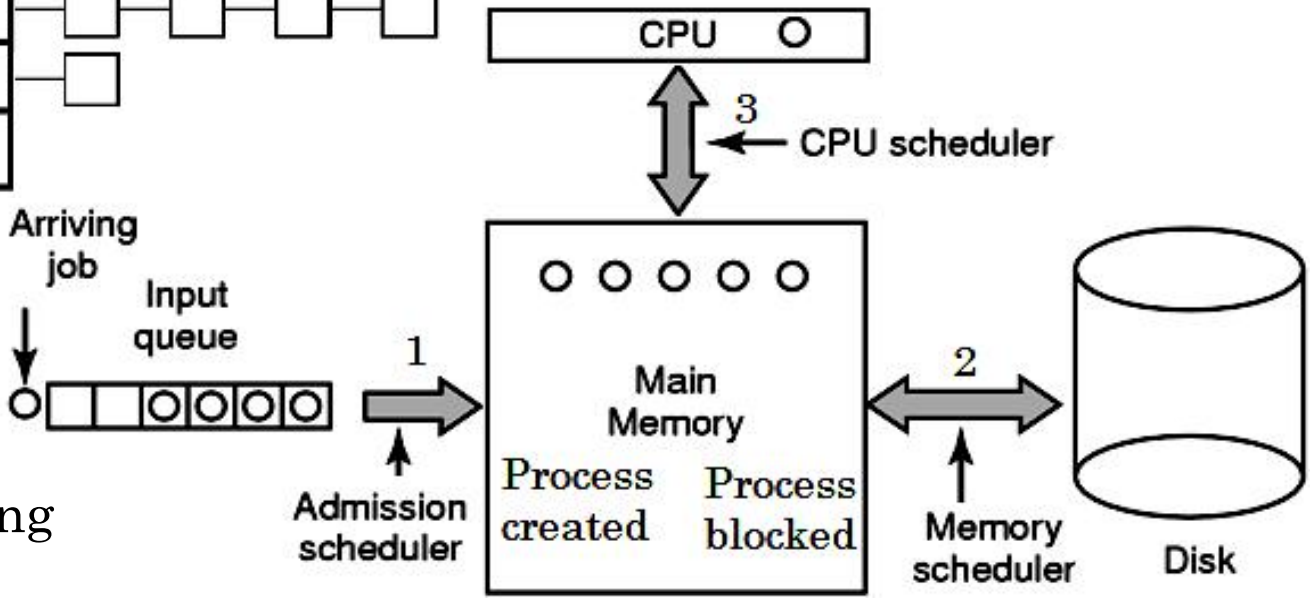
SCHEDULING

- Refer Expt 5 for details of following points:- definition, necessity, type as preemptive, non-preemptive, scheduling criteria's, Shortest Job/process First, Shortest Job/Process Next, Priority Scheduling, Round Robin, Time sharing, FIFO, real time guaranteed scheduling
- When to schedule:- (1) new process created, (2) process exits, (3)process blocks on I/O, (4) I/O interrupt occurs (5)Clock interrupt.
- Combination of scheduling algorithms used to improve performance



e.g:-multiple queue

guaranteed scheduling :- ratio bust/ waiting time



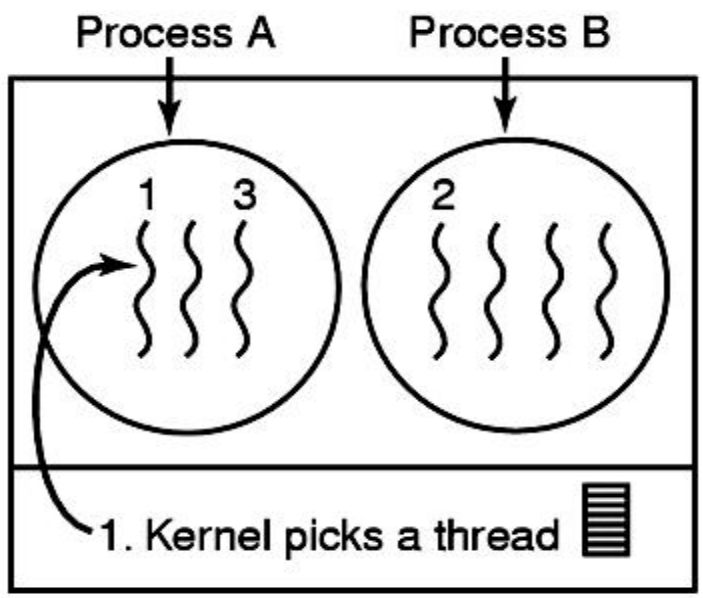
- Scheduling may be of type (1)static (2) dynamic
- Three level scheduling

Few more about scheduling

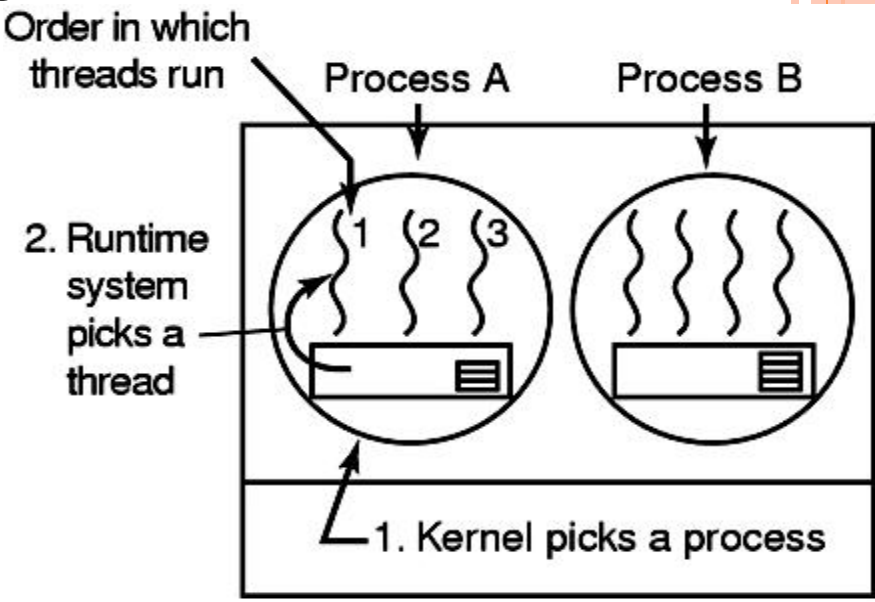
- Lottery scheduling:- random ticket to all processes. So the winner.
  - Bumper prizes for high priority processes or
  - Cooperative processes may exchange tickets if they wish.
- Fair-share scheduling:- type of RR. Integrated concept of priority.
  - AEBECEDE V/S ABCDE
- Policy V/s Mechanism:- like in PS, Kernel knows mechanism but user/ process should mention policy.

Thread Scheduling:-Possible scheduling of kernel & user-level threads

- 50-msec process quantum
- Threads run 5msec per CPU burst



Possible: A1, A2, A3, A1, A2, A3
Also possible: A1, B1, A2, B2, A3, B3



Possible: A1, A2, A3, A1, A2, A3
Not possible: A1, B1, A2, B2, A3, B3

Scheduling algorithm goals

All systems

- Fairness - giving each process a fair share of the CPU
- Policy enforcement - seeing that stated policy is carried out
- Balance - keeping all parts of the system busy

Batch systems

- Throughput - maximize jobs per hour
- Turnaround time - minimize time between submission and termination
- CPU utilization - keep the CPU busy all the time

Interactive systems

- Response time - respond to requests quickly
- Proportionality - meet users' expectations

Real-time systems

- Meeting deadlines - avoid losing data
- Predictability - avoid quality degradation in multimedia systems

Thank you