Process

A. Basic Terminology & Definitions
Task, program, process, thread, job, process state, process state transitions, process scheduling, process context, process mode, event, ready queue, blocked queue, suspended queue, resource, degree of multiprogramming, process priority, I/O scheduling, preemption, preemptive, nonpreemptive, multithreading

B. State Transition Models And Process Life Cycles
User mode, kernel mode, Two state model, five state model, seven state model, eleven state model, Phases in and reasons of Process creation, suspension, termination

C. Process Management
PCB, Process image, mode switch Vs context switch

D. Threads
Thread management, ULT, KLT, process switch Vs thread switch,

E. System Calls For Process Management
Fork, kill, signal, exec, wait, exit, getpid, getppid, nice, brk

F. Process scheduling
Types of schedulers: Mid-term-short term-long term schedulers, response time, waiting time, turnaround time, scheduling policies, uniprocessor scheduling, multiprocessor scheduling, real time scheduling, UNIX scheduling, LINUX scheduling, Comparative assessment of scheduling

G. Processes in Unix, Solaris and windows 2000 thread and SMP Management.

H. Multiple Choice Questions

Basic Terminology & Definitions

Q. 1 Define Task, program, process, thread and job with an example of each.
Ans:

- **Task:** Task is a unit of assigned work. It can also be defined as the unit of programming controlled by OS. Depending on the OS design the task may involve one or more processes.
  Example: Bake a cake

- **Program**: A program is defined as sequence of instruction written to accomplish a task. A program may comprise of one or more processes depending on the statement being executed. Generally referred as a passive entity that does not perform any action.
  Example: A particular recipe given in book.

- **Process**: This is an instance of program in execution. The static statements in the program when executed, take the process form. In contrast to the program, a process is an active entity which needs a set of resources to perform its function. The Linux kernel internally represents processes as tasks. A process elements are: a program, data and process state.
  Example: actually following the steps given the book.

- **Thread**: A thread is a lightweight process. It is also defined as the smallest processing unit that is scheduled by an operating system.
  A thread must be part of a process as it shares the process environment viz, code, data and resources with other threads. Threading concept has increased the computing efficiency to significant extent.

- **Job**: A job is unit of work submitted by user to the system. A job may be interactive or a batch job which may in turn consist of one or more processes.

Q.2 Define process states, process context, process mode, event, degree of multiprogramming, process priority, preemption, preemptive, nonpreemptive, Multithreading.

Ans:

- **Process State:** A process state is a process' internal data maintained by OS for the purpose of supervision and control of the process. It is also called as executional context of the process.

- **Process Context:** whenever a running process is taken away from processor, some of the process state's information needs to be retained. This information called as process context helps the process to resume from the point where it was stopped last time.

The context of a process includes its address space, stack space, virtual address space, register set image i.e. Program Counter, Stack Pointer, Program Status Word, Instruction Register and other general processor registers, accounting information, associated kernel data structures and current state of the process (waiting, ready, etc).

- **Process Mode:** the operational or the privilege mode of process execution is called process mode. A process executes in user mode or a kernel mode. Processor switches the process in between these modes depending on the code the process is running.

- **Event:** An activity that is happens or is expected to happen. Generally this a software message exchanged when the activity occurs. In operating systems, the events may cause the processes to change their state.
  e.g. mouse click, file lock reset, etc.

- **Degree Of Multiprogramming**: with multiprogramming, the CPU can run multiple programs simultaneously or concurrently. The degree of multiprogramming is the maximum number of allowed processes at a time in a system that does not let the CPU performance degrade than a certain threshold. The long term scheduler can limit this number so that all the admitted processes get fair CPU share and they all execute well.

- **Process priority:** In a multiprogramming system, the processes are assigned numerical privileged values indicating their relative importance and/or urgency and/or value.

- **Preemption:** Preemption is the ability of the system to take over a currently executing process by another one (possibly with high privileged one) with an intention to resume the preempted process later on.

- **Preemptive:** Preemptive entities are the one those can be taken over by another similar type of entities.

- **Non-preemptive:** Non-preemptive entities are the one those cannot be taken over by other entities.

- **Multithreading:** Multithreading is defined as the ability of an OS to support multiple, concurrent paths of execution within a single process.
  The thread process relationship is shown in following diagram

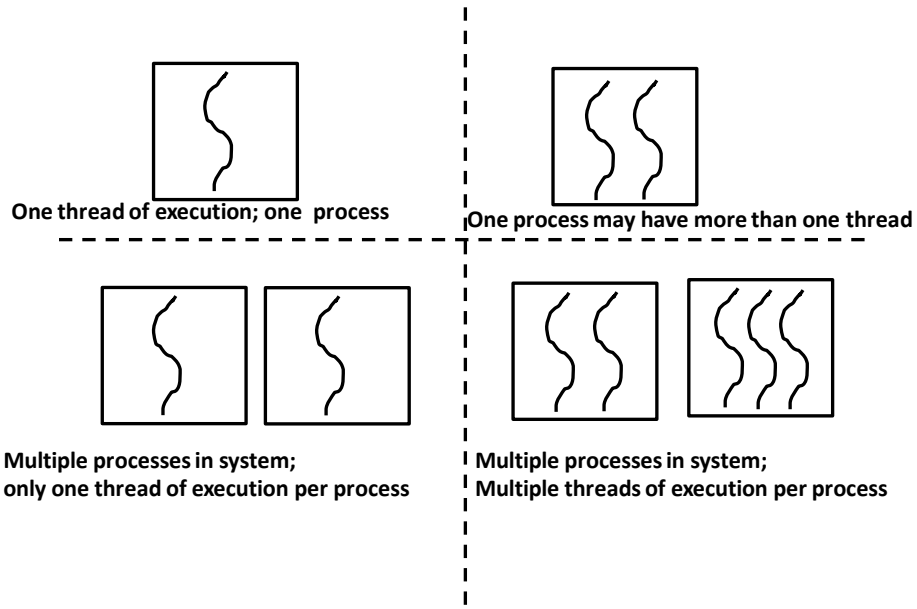


Figure: Threads and process association.

Q.3 Define Process State Transitions, Process Scheduler, I/O Scheduling, Processor Scheduling, Ready Queue, Blocked Queue, Suspended Queue.

- **Process State Transition Diagram:** A process transits among various states from its initiation to termination. The pictorial representation of this changes in states is called as process state transition diagram.

- **Process Scheduler:** Process scheduler or the Dispatcher is a software module that works for CPU(processor) scheduling and chooses the next process to run. Its main activities involve switching the process context, switching the execution mode to user. The dispatcher is required to work very fast to improve the efficiency of execution.

- **I/O Scheduling:** I/O scheduling is a process that is involved in making the decision as to which process's pending I/O request should be handled by an available I/O device.

- **Processor Scheduling:** Processor scheduling is a process that makes a decision of which process should get hold of processor next. This process needs the 'dispatcher' – a software module of short term scheduler to make this decision.

- **Ready Queue:** Ready Queue is a data structure that holds the processes which have acquired all required resources for the execution except CPU and are ready to run.

Any process submitted for execution undergoes many states during its lifetime. In certain states, only one process can have that state at any time of instance, while in some states multiple processes are allowed at a time. Such processes are stored in data structure 'Queue'.

- **Blocked Queue:** Blocked queue is a data structure that holds the processes which were executing and now waiting for some event to occur. Unless specified by priority, these processes are removed from this queue on FIFO basis as and when the corresponding events occur.

- **Suspended Queue:** Suspended queue is a data structure that holds the processes which were blocked and are swapped out to secondary memory to make room for processes which are getting blocked newly.

  Unless specified by priority, these processes are removed from this queue on FIFO basis as and when the corresponding events occur. After removal, processes may go to Blocked queue if memory is available but event is still awaited else, they may be shifted to Ready-Suspended state which indicates that event has occurred but memory isn't available.

Q.4 what are the I/O bound and CPU bound processes?
Ans.

I/O bound process: I/O bound processes are the ones those spend more time doing I/O operations than computation, though it may have many short CPU bursts.

CPU bound processes: CPU bound processes spend more time in computations and so they have long CPU bursts. Although they may also involve in very short durations of I/O operations.

Q. 5 Differentiate between parallelism and concurrency.
Ans.

Parallelism: Parallelism is the quality of the processes to execute at the same time. Two processes or events are said to be parallel if they occur at the same time. In parallelism, multiple processes can be active at a time.

Concurrency: Concurrency does not mean parallel. With concurrency, multiple processes are executed one after another by interleaving their execution in such a way that it creates an illusion of parallelism. In concurrency only one process can be active at a time.

Generally the concurrency is achieved by interleaving process execution on (may be single) CPU which creates an illusion that processes run in parallel, while parallelism is obtained by multiple processors operating in parallel at a time. Both the techniques achieve computation speedup, though the inherent mechanism used by both concepts is different.

Q. 6. State advantages of process concurrency.
Ans.

The processes in a multiprogramming environment are a blend of I/O bound and CPU bound processes. If these processes are executed sequentially, they underutilize the CPU and I/O devices both. If executed in interleaved fashion, the system gives better efficiency, relatively lower response, turn around and waiting times.

Example: consider three processes A,B and C with different requirements of resources and execution times. The following diagram shows the effect when they are executed in interleaved and non-interleaved mode.

Figure: Non-interleaved execution of processes

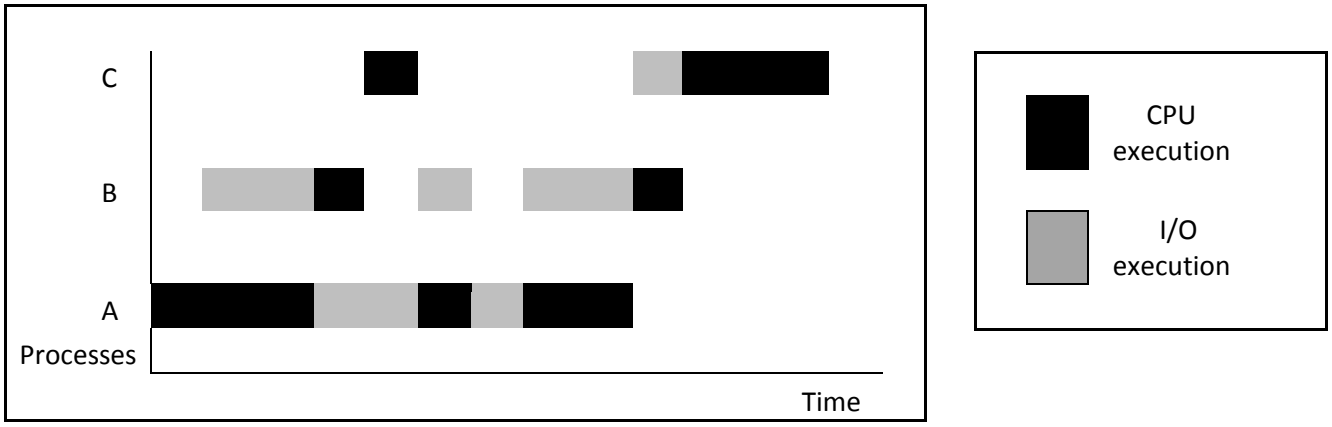


Figure: Interleaved process execution

Q. 7 What are the basic elements of a process?

Ans:

A process is comprised of six components as,

1. Process ID: This is a unique identifier assigned to the process
2. Code: This is program code.
3. Data : These are the data and files required for execution
4. Resources : These are different types of resources allocated by OS
5. Stack: This contains parameters for the functions/procedures called and their return addresses.
6. Process state: This is one of the eleven states process is in.

Q. 8. What are the elements of process environment?

Ans.

The components of process environment are program code and data, memory allocation information, status of the file processing activities, information about the process interaction, information about the resources required for process execution and some miscellaneous information needed by the OS. These elements can be detailed as,

- **Program code and data:** This includes the program code including all the functions and procedures, and the program data, including the stack information.
- **Memory allocation information:** This section has information of the memory areas allocated to process. This proves to be the vital information as it is required to manage the memory accesses by concurrent processes.
- **Status of file processing activities:** This holds the pointers to files opened by process and the current positions in the respective files.
- **Process interaction information:** This holds information necessary to manage interprocess communication through signals and messages, and the IDs of child-parent processes.
- **Resource information:** The resource information of the resources acquired by process is stored.
- **Miscellaneous information :** miscellaneous information needed for a process to communicate with OS is maintained as part of process environment.

Q. 8 What are the fundamental kernel functions of process control?

Ans

The kernel does three main functions to control the processes in hand as given below,

1. Scheduling: Choose the process as per the scheduling policy to be executed next on the CPU.
2. Dispatching: Set up execution of the chosen process on the CPU.
3. Context save: Save information concerning an executing process when its execution gets suspended.

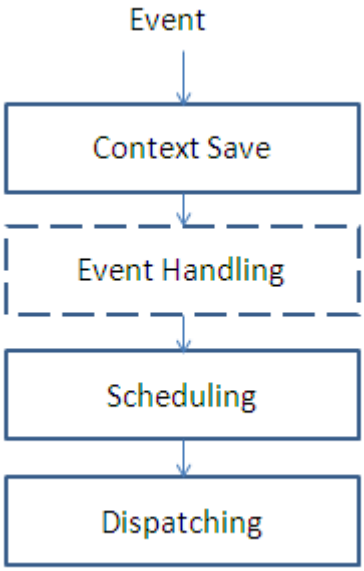


Figure: Fundamental functions to control processes

Here in the diagram, occurrence of the event calls the context save functionality and an appropriate event handling procedure. This event handling may initiate some processes, hence the scheduling function gets invoked to choose the process and in turn, the dispatching function transfers control to the new process.

Q. 9. Define and explain the concept: process state transition.

Ans:

Definition: A state transition for process is defined as a change in its state.

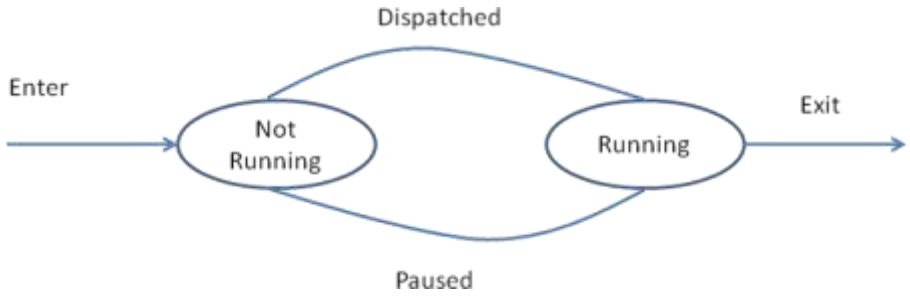
The state transition occurs in response to some event in the computing system. E.g. transition from ready state to running when dispatched by scheduler.

State Transition Models and Process Life Cycles

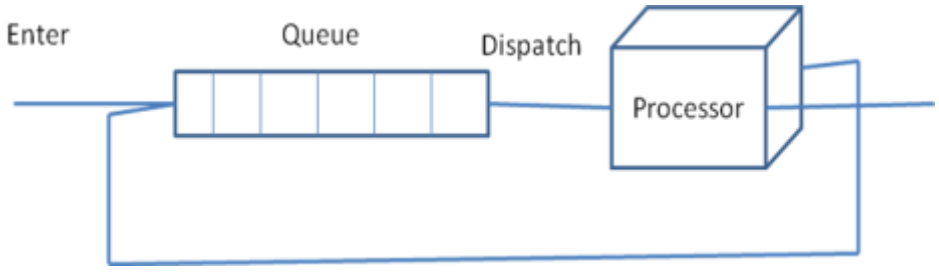
Q. 1. List and explain different states in process state-transition diagram
Ans. The different possible states in a complete state transition diagram are: new, ready, running, suspended, blocked, terminated, etc.

- New: The process is being created
- Ready: the process is ready with all required resources and waiting to be assigned to a processor
- Waiting/Blocked: the process is waiting for some event to occur
- Suspended: the process was waiting for some event to occur, but then is swapped to secondary memory to make room for new processes getting 'blocked'.
- Terminated: the process has finished its execution.

Q. 2. Explain the two-state process model.
Ans:



A



B.

Figure A and B: Two State Model

In two process model, the process is in either of the states, Running or Not Running. A newly created process gets its PCB and enter the system in the Not Running state. It acquires all resources required for its execution and waits for its turn to execute. The dispatcher keeps on interrupting the Running process time to time and selects the other deserving processes to run. Then the Running process is moves to Not Running state and the other selected process transits from Not Running to Running state. The Not Running

processes can be more than one in number and they must be processed in some order. Hence, they are stored in an ordered queue.

Q. 3. Explain the five state process model with the transitions
Ans.

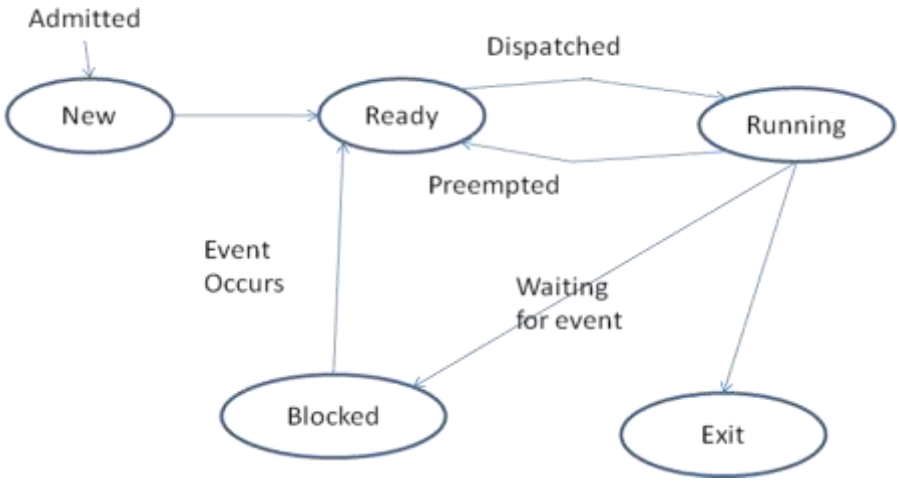


Figure: Five-State state transition diagram

The diagram above depicts the simple life cycle of the process. The state transitions can further be explained as,

State transition	Description
Null→ New	A new process is created to execute a program.
New→ ready	The OS may move a process from New to Ready state depending on the predefined maximum number of processes allowed (Degree of multiprogramming).
Ready→ Running	The process is scheduled by dispatcher. The CPU starts or resumes execution of the instruction codes
Blocked → Ready	The request initiated by process is satisfied or the event on which it is waiting occurs.
Running → Ready	The process is preempted by the OS decides to execute some other process. This transition takes place may be because of expiration of time quantum or arrival of high priority process.
Running → Blocked	The running process makes request for resource(s) or needs some event to occur to proceed further. The process then calls for a system call to indicate its wish to wait till the resource or the event becomes available.

Running → Termination	The program execution is completed or terminated.

Q. 4. What are the causes of process initiation?

Ans:

The main reasons for process creation are: New batch job initiation, Interactive logon, Initiation by OS to provide some service or creation by some other process.

New batch job: while processing the new batch of jobs submitted, the OS creates a process to execute the same.

Interactive logon: when a user logs into the system, a new process is created.

Created by OS to provide some service: The OS initiates a process to perform the service requested by user directly or indirectly, without making the user to wait.

Spawned by an existing process: To support Modularity and/or parallelism, a user program can create some number of new processes.

Q. 5. What are the causes of process blocking?

Ans: The five major reasons of process blocking are:

- Process requests an I/O operation
- Process requests memory or some other resource
- Process wishes to wait for a specific interval of time
- Process waits for message from some other process
- Process wishes to wait for some action to be performed by another process.

Q. 6. What are the causes of process termination?

Ans:

The main reasons of process termination can be listed as,

- **Normal Completion:** The process executes an OS system call to intimate that it has completed its execution.
- **Self termination** (e.g. incorrect file access privileges, inconsistent data)
- **Termination by the parent process:** a parent process calls a system call to kill/terminate its child process when the execution of child process is no longer necessary.
- **Exceeding resource utilization:** An OS may terminate a process if it is holding resources more than it is allowed to. This step can also be taken as part of deadlock recovery procedure.
- **Abnormal conditions during execution:** the OS may terminate a process if an abnormal condition occurs during the program execution. (e.g. memory protection violation, arithmetic overflow etc)
- **Deadlock detection and recovery.**

Q. 7. Draw and explain nine state state-transition diagram.

Ans.

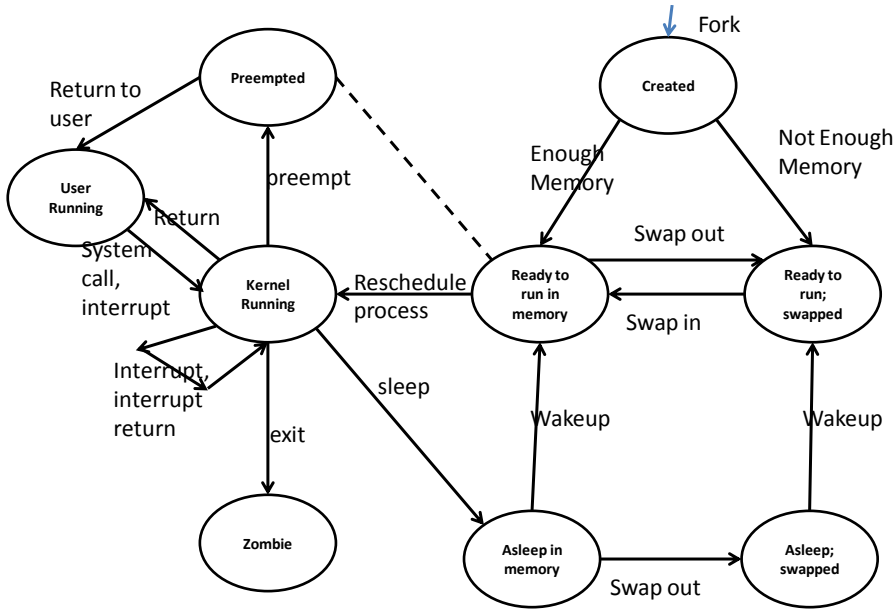


Figure: Nine-state/ UNIX process state transition diagram

UNIX operating system clearly specifies the two modes of execution: user mode, kernel mode. The process gets created on execution for the system call fork(). All the states shown above can be detailed as given in the table below.

Process State	Description
User Running	The process is running in user mode
Kernel Running	The process is running in Kernel mode
Ready to Run, in Memory	The process is ready to run as soon as the kernel dispatches it.
Asleep in Memory	The process is blocked on some event, process is in main memory.
Ready to Run, Swapped	The process is ready to run, but the swapping module must swap it in the main memory, so that kernel can schedule it.
Sleeping, swapped	The process is blocked on some event and it is in secondary memory.
Preempted	The process was returning from kernel mode to user mode, but the kernel preempts it and performs a process switch to dispatch another process.
Created	The process is just created and is not yet ready to run. (may not have all the resources, including memory, which are required to run)
Zombie	The process no longer exists, but it leaves some information (probably accounting information) to its parent process to collect.

Q. 8. What are the events pertaining to a Process?

Ans: The processes wait on some events to execute to complete their execution. Though not complete, some of the events can be listed as follows,

1. Process creation event: a new process gets created
2. Process termination event: a process completes its execution.
3. Timer event: occurrence of timer interrupt
4. Resource request event: a resource request is made by a process
5. Resource release event: process releases a resource and notifies.
6. I/O initiation request event: a process wishes to intiate I/O operation
7. I/O completion event: a process finishes I/O operation
8. Message send event: A message is sent by one process to an another one.
9. Message receive event: a process receives a massage by another one.
10. Signal send event: a signal is sent by a process to an another one
11. Signal receive event: a process receives a signal
12. A program interrupt: An instruction in currently executing process executes some illegal operation and malfunctions.

Q. 9. What are the reasons of process suspension?

Ans:

The basic reasons of process suspension are: swapping, interactive user request, timing, parent process request or some OS reason.

Swapping: The main memory may not be enough to accommodate some ready process. So a currently not running process is shifted from main memory to secondary memory.

Interactive User request: a user may wish to suspend a currently running process for debugging or to manage the use of resources

Timing: A process may be executed on periodic basis and may be suspended while waiting for its next turn of execution.

Parent process request: A parent process may wish to suspend its child process to examine or modify the suspended process or to coordinate the child processes.

Other OS reasons of suspension: The OS may suspend a background or utility process or a process that is causing some problem in normal activities.

Process Management

Q. 1. How Operating System executes a program?

Ans. Every program that is loaded for execution contains declarations of variables and functions if any, statements which may or may not contain data explicitly. During program execution, the instructions use values stored in data area and the stack to perform the intended computation. Any process' address space is formed by its instructions, data and program stack. The OS first needs to allocate some memory to accommodate this address space to realize process execution.

In simple programs the control flows between the main program and the functions according to program logic. The OS treats it as a single program as it is not aware of existence of functions. In such cases, the program execution constitutes a single process (one-to-one relationship).

If the program is coded in a language that contains special features for concurrent programming (e.g. JAVA), then during execution, the OS is informed to execute some portions of the program concurrently. In such a case, one program may have multiple processes (one-to-many relationship) or threads running on its behalf.

Q. 2. What are the child processes? State the advantages of having parent-child process relationships.

Ans. The Operating System creates a process when a program is submitted to it for execution. Depending on the coding style used for writing the program this process may create other processes while in execution. Such processes are called child processes and the processes generating them are called the parent ones. These child processes can also become parents and give birth to new level of child processes.

Advantages of parent-child process relationship:

Sr No.	**Advantage**	**Description**

Q. 3. What are the different mechanism processes use to interact with each other?

Ans. In a typical multiprogramming environment, the processes are required to interact with each other. The basic mechanism can be listed as follows,

Sr No	Interaction Mechanism	Description
1	Data Sharing	The processes interact with each other by altering data values. If more than one processes update the data the same time, they may leave

		the shared in inconsistent state. So, shared data items are protected against simultaneous access to avoid such situation.
2	Message Passing	In this mechanism, the processes exchange information by sending messages to each other.
3	Synchronization	In certain computing environments, the processes are required to execute their actions in some particular order. To help this happen, the processes synchronize with each other to maintain their relative timings and execute in the desired sequence.
4	Signals	The processes may wait for events to occur. It can be intimated to processes through the signaling mechanism.

Q. 4. What are the components associated with process switching?

Ans: Broadly, process switching has two components pertaining to overhead: execution related overhead, resource use related overhead.

Execution related overhead: During process switching the context of running process is saved and to that of the new process needs to be loaded. This overhead is unavoidable in process switching.

Resource use related overhead: A process holds many resources required for execution during its runtime. This resource information leads to large size of process state information, which adds to the process switching overhead.

Q. 5. What are the different control structures maintained by OS to manage processes?

Ans:

The operating system needs the universal information about the current status of each process and resources in system. To have this information handy, OS maintains tables of information about it each entity that system is managing.

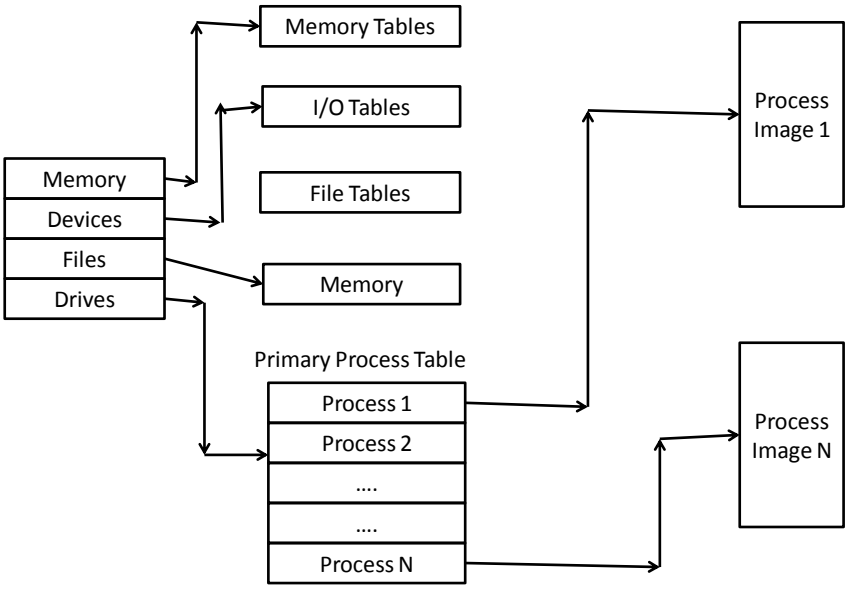


Figure: Operating system Control Structures

Memory Tables: Memory tables keep track of both main and secondary memory. Active processes are stored in main memory and when required, they are moved to secondary memory through the mechanism called 'swapping'. The memory tables maintain the following information:

- The main memory allocation to all processes in system
- The secondary memory allocation to all processes in system
- Shared memory regions in main and virtual memory and their attributes
- Miscellaneous information required to manage virtual memory.

I/O Tables: I/O tables keep track of I/O devices and channels in the computing system. The I/O devices are also resources required by processes. So at any given instance, I/O devices may be available or allocated to a particular process.

File Tables: File tables keeps track of all files, their locations on secondary memory, their current statuses and other attributes such security, sharing, etc. Most of the operating systems, this information is maintained by a module called File Management System.

Process table: Process tables manage processes. They maintain information of processes, their child process references, statuses, allocated resources, process contexts, information required for process synchronization and so on. These pieces of information are stored in process images.

Q. 1. What are the typical fields of a PCB?

Ans:

The Process Control Block contains all information pertaining to a process that is used in controlling the process operation, resource information and information needed for inter-process communication.

Sr No	PCB Field	Contents
	Identifiers	
1	Process ID	This is unique process ID assigned to it at the time of creation
2	Child and Parent IDs	The child and parent process IDs are required for process synchronization
3	User Identifier	The unique identifier associated with the user in multiuser systems.
	Processor State Information	
3	User Accessible Registers	Every processor offers some general purpose registers those are accessible to user. The number of available registers differs with every processor type.
4	Control and Status registers	This field is also called PSW(Program Status Word) which typically contains **Program counter** (Contains the address of the next instruction to be fetched), **Condition codes** (Result of the most recent arithmetic or logical operation), **Status information**(Includes interrupt enabled/disabled flags, execution mode)
5	Stack Pointers	Each process needs one or more LIFO stacks to store parameters and return addresses in case of procedure or system calls. The stack pointer points to stack top.
	Process Control Information	
6	Scheduling and state information	This is information about process state, priority, scheduling related information and event information in case the concerned process is waiting on that!
7	Data Structuring	A process may linked to other processes in queue, ring or some other structure. This information is stored in data structuring field.
8	Interprocess communication	The processes need various flags, messages and signals be exchanged to interact with each other. Some or all this information in stored in PCB.
9	Process Priority	The priority is a numeric value, which may be assigned to a process at the time of its creation. Some priorities can be altered by user and some change with process age, too.
10	Memory Management	This field holds the pointer to segments or to the page tables which describe portions of memory assigned to the process.
11	Resource ownership and Utilization	All the resources or the number of instances of resources assigned to process are described in this field.

Figure: Fields of Process Control Block

Q. 6. What the typical contents of a process image?

Ans:

The collection of program, stack, data and process attributes are called process image. The process image is generally maintained as a continuous or contiguous block of memory which resides in secondary memory. To execute a process, its process image is loaded in main memory or virtual memory.

Typical elements of process image

Sr No	Process image element	Description
1	User Stack	This is part of user space that contains program data, a user stack area, and modifiable/editable programs.
2	User Program	The program under execution.
3	Stack	Each process needs one or more LIFO stacks to store parameters and return addresses in case of procedure or system calls.
4	Process Control Block	Process control block(PCB) contains the data which is used by OS to control and manage the process.

Figure: Typical Elements of Process Image

Q. 7. Write the step by step sequence of actions taken up by OS when a process is created?

Ans:

1. Create a process
2. Assign a unique process ID to newly created process
3. Allocate the memory for the process and create its process image
4. Initialize process control block
5. Set the appropriate linkages to the different data structures such as ready queue etc.
6. Create or expand the other data structures if required.

Q. 8. Differentiate between process context switch and mode switch.

Ans:

Context Switch: Execution of a process may be stopped to respond an interrupt. This exercise needs to save Process Image be saved of one process and load process image of the new process loaded.

Here the processes are switched and processes keep on changing their status as Running and Not running.

Mode switch: every process may switch in between a low privileged user mode and high privileged kernel mode in its lifetime.

Here, the process remains the same. Its status always remains as 'Running' and only the mode of execution keeps on changing. Process switch requires more effort than a mode switch.

Q. 9. Detail the process switching operation.

Ans:

If the process switch changes its state, process switch or context switch occurs. The steps involved in this exercise are as follows:

1. Process context, including program counter and other registers is saved.
2. Update the PCB of the process that was currently running state. The new state assigned may be one of the other states (Ready, Blocked, Ready/Suspend or Exit).
3. Move this updated PCB to appropriate queue (Ready; Blocked on Event i; Ready/Suspend).
4. Select another deserving process for execution.
5. Update the PCB of chosen process and change the process state as Running.
6. Update the memory management data structures.
7. Restore the context of the chosen process so that it can resume from the point if it was interrupted last time, or can start its execution if it was loaded for the first time.

Threads

Q. 1. What are the advantages of threads?

Ans:

The primary advantages of threads include low computation overhead, speed-up and efficient communication.

Low switching overhead: A process thread state does not contain resource allocation state and communication state, instead it contains only computation state. This makes thread switch operation far lighter than process switch.

Speed-up: A process can be broken into multiple threads, which can run concurrently. This mechanism can speed-up the execution speed of an application on uniprocessors and multiprocessors, both.

Efficient Communication: Threads of a process can communicate with each other using shared data space, thus this avoids the necessity of communication system calls.

Q. 2. What are different types of thread?

Ans.

Threads are implemented in different ways. The main difference is on the basis of kernel's knowledge of thread existence. The methods of thread implementation are **Kernel Level Threads(KLTs), User Level Threads(ULTs), hybrid threads.**

Out of these three types, the kernel level threads are most expensive to manage and the user level threads are the cheapest ones. The third type tries to strike the balance between the rest two types. Experiments have proved that switching between kernel level threads is 10 times faster than process switching, and switching user level threads is over 100 times faster than process switching. Though so fast, ULTs cannot support pure multiprocessing and though slow, KLTs provide better parallelism and speed-up in multiprocessor systems.

Q. 3. Explain the working of Kernel Level Threads.

Ans:

The threads require some mechanism to create them, manage them in their lifetime and then destroy them. In Kernel Level Threads (KLTs), kernel provides library to create, schedule, manage and destroy the threads. as these library functions are executed by kernel, it always invokes system calls for every activity involving threads.

When a process calls upon create_thread system call, the kernel assigns a thread ID to it and allocates a thread control block(TCB). This TCB has a pointer to the PCB of its parent process. In case of thread switch, the system saves context of the interrupted thread in its TCB. Then the dispatcher considers all the TCBs in hand, and selects a ready thread. If this new thread belongs to the same process whose thread was earlier interrupted, new process context is not required to be loaded. Otherwise, process switch occurs along with thread switch and the execution resumes. Basically, if both

the threads belong to the same process, then actions to save and load process context are unnecessary. This concept reduces the switching overhead to great context. Moreover, in case of multiprocessor systems, the kernel can schedule multiple ready threads belonging to same process on multiple processors, thus can exploit parallelism to the fullest!

Q. 4. State advantages and disadvantages of Kernel Level Threads.

Ans:

Advantages:

1. KLT offers all advantages that a process has, and still it proves better than a process being 'lightweight'.
2. KLTs belonging to same process can be scheduled in parallel on multiple processes, thus exploiting the parallelism in true sense

Disadvantages:

1. KLTs are solely handled by Kernel and so thread switching is also carried out by kernel in case of event handling. This proves to be very expensive operation even if the interrupted and next scheduled threads both belong to the same process.

Q. 5. What are User Level Threads?

Ans:

The User Level Threads(ULTs) are created and implemented by thread library that is provided by the programming language in which the threads are coded. The thread code is linked with the process. This management does not involve kernel, i.e. in other words, the kernel has no knowledge of ULTs existence. The system schedules the process, and the process threads are identified by the thread library as ready, and internal scheduling is again done by the thread library.

ULT creation and operation:

1. A process calls upon a library function *create_thread* for creation of the new thread.
2. The library creates a TCB(Thread Control Block) for the newly created thread and starts considering if this thread is worth 'scheduling'.
3. When the thread in *running* state invokes the library function to synchronize with other thread, library function checks the new *runnable* thread and switches to that thread of process. This all happens without any intervention of kernel.
4. If the thread library cannot find any runnable thread, the process invokes a '*block me*' system call. This process gets unblocked when some event activates on its threads.
5. The thread library code is part of each process. It performs 'scheduling' to select a thread and organizes its execution. The information stored in TCB is used by thread library while selecting the next *runnable* thread.
6. When a particular thread is scheduled, its CPU state becomes the process' CPU state, and the process stack pointer points the thread stack.

Q. 6. What are the advantages and disadvantages of User level Threads?

Ans:

Advantages:

1. As the thread synchronization and management is handled by thread library, this avoids the overhead of invoking the system calls.
2. The thread switching is simpler and cheaper than in KLTs.
3. This kernel independence enables each process to have its own scheduling policies that suits its own nature.

Disadvantages:

1. The kernel isn't aware of ULT existence; hence, it cannot distinguish between a thread and a process.
2. Blocking of a thread blocks the entire parent process.
3. Since the kernel schedules a process and then thread library schedules a process within a process, at most only one thread can be in operation at any time.
4. ULTs does not support pure multiprocessing as only one thread in one process is active at a time.

Solutions:

1. The program can be written to create multiple processes than multiple threads. this overcomes all the disadvantages given above. But this solution cannot have advantages of thread switch and has to incur overhead of process switch.
2. Blocking of threads mentioned in disadvantage number 2 cannot be avoided with the technique of jacketing. In this, instead of executing a blocking call to system calls, the thread invokes an application level I/O jacket routine which in turn does the thread's calling job.

Q. 7. What is a hybrid thread model?

Ans:

A hybrid thread model has both user-level and kernel-level threads and a method of associating ULTs with KLTs. These different methods of association provide various combinations of the low switching overhead of ULTs and the high concurrency and parallelism of KLTs.

These association methods can be summarized as: **Many-TO-One, One-to-One and Many-to-Many.**

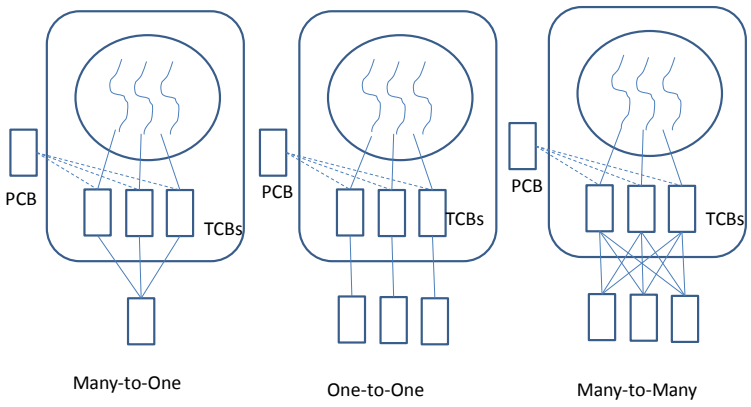


Figure: Association of KLTs with ULTs

The kernel creates KLTs in a process and associates a Kernel Thread Control Block with each corresponding KLT.

In **Many-to-One** association method, a single KLT is created in each process by the kernel and all ULTs created in a process by thread library associated with this KLT. This approach gives an effect similar to more ULTs.

Advantages:

1. ULTs can be concurrent without being parallel.
2. Thread switching is cheap

Disadvantage: Blocking of an ULT in a process blocks all threads in the process.

In the **One-to-One** association, each ULT is mapped permanently into a KLT. This provides an effect very similar to KLTs.

Advantages:

1. This association gives pure multiprocessing on multiprocessor systems
2. Blocking of one thread in a process does not block rest of the threads in the same process.

Disadvantage: switching between threads is done at kernel level and thus incurs high overhead.

The **Many-to-Many** association produces an effect which is very similar to one ULT being mapped into any KLT. This achieves parallelism between ULTs by mapping them into different KLTs.

Advantages:

1. Achieves high parallelism
2. Low thread switching overhead
3. Blocking of one thread does not block rest of the threads in the process.

Disadvantages: system is quite complex to implement.

Q. 8. What is a multithreaded process model?

Ans:

In a multithreaded environment, a thread is called lightweight process while, a process is simply a unit of resource allocation and unit of protection.

The process maintains only:

- A virtual address space to store the process image
- Protected access to processors and with rest of the processes for interprocess communication, with files and with I/O devices.

The thread contains:

- A thread execution state
- A thread context, generally saved when process isn't in *Running* state.
- A thread execution stack
- Thread storage area to store local and temporary data
- A shared access to resources of its process.

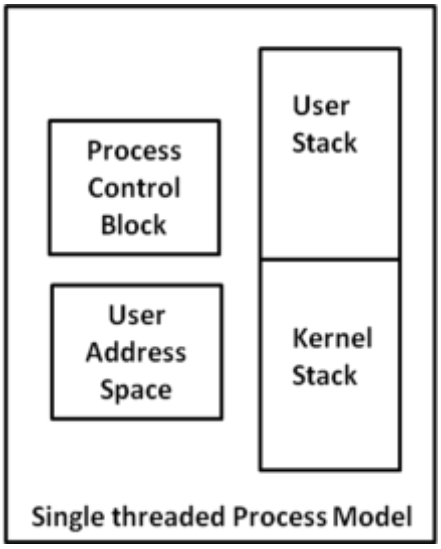


Figure: Single threaded Process Model

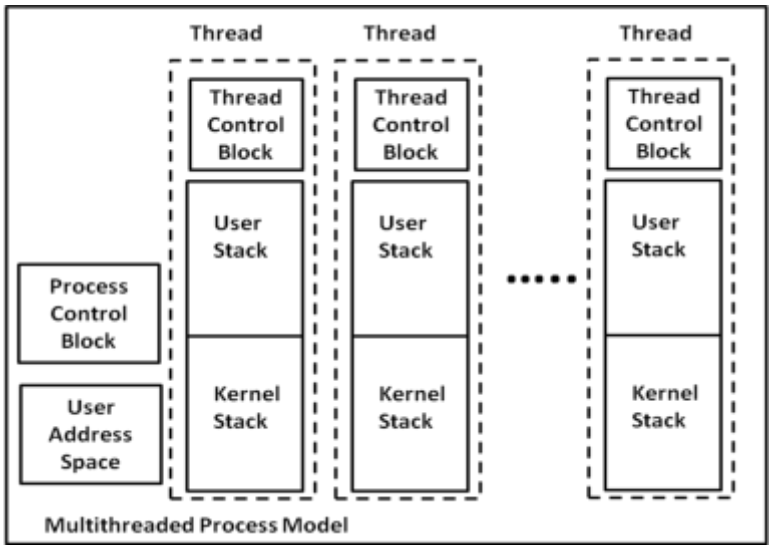


Figure: Multithreaded Process Model

As shown in above figure, all the threads of the same process share process control block and user address space, i.e. they share the execution state and the resources assigned to the same process.

Q. 9. List the operations associated with process threads?

Ans:

There are four basic thread operations associated with a change in thread state viz: spawn, block, unblock, finish.

- **Spawn:** Typically, when a new process is spawned, a thread for that process is also spawned. Also, a thread can also spawn another threads in the same process. The new thread also gets its own registers, context and the stack space.
- Block: when a thread waits on some event, it blocks itself and the thread context is saved.
- **Unblock:** when the event on which the process is waiting occurs, the thread gets shifted to ready queue.

- **Finish:** when a thread finishes its execution, its register and stack contexts are deallocated.

Q. 10. Explain the different process-thread relationships.

Ans.

The process and its thread share either of four relationships amongst them as: One-to-One, Many-to-Open

Q. 11. List some typical uses of threads in single user environment.

Ans:

Though not a complete list, some applications of threads in single user environment are:

- **Foreground and background work:** multiple threads of a single program can execute some processes in background and foreground independent of each other. E.g. the HTML pages when load, different processes load text, images, videos and rest of the graphics independent of each other.
- **Asynchronous processing:** asynchronous elements of program are implemented as threads.
- **Execution speed:** as threads can execute independent of each other, one batch of jobs can be processed by some threads while the other thread may read next batches of job.
- **Modular program development :** projects or programs those accept inputs from various sources and pass on information to various modules of processing, are better be implemented using threads.

System Calls for Process Management

Q. 1. List the different system calls used for process management

Ans:

The process management related system calls are listed as follows,

Sr No	System Call	Description
1	fork()	This system calls creates a new process.
2	exec()	This call is used to execute a new program on a process.
3	wait()	This call makes a process wait until some event occurs.
4	exit()	This call makes a process to terminate
5	getpid()	This system call helps to get the identifier associated with the process.
6	getppid()	This system call helps to get the identifier associated with the parent process.
7	nice()	The current process priority can be changed with execution of this system call.
8	brk()	This call helps to increase or decrease the data segment size of the process.
9	Kill()	The forced termination of any process can be executed with this system call.
10	Signal()	This system call is invoked for sending and receiving software interrupts

Q. 2. Explain working of fork() system call

Ans:

The **fork()** system call is used to create a new process. When the system executes this system call in response to process creation request, following steps are carried out.

1. The operating system allocates a slot in the process table for the newly created process.

2. This new process, i.e. child process is then assigned a unique ID.
3. System then creates a logical copy of the parent process context.
4. The file and inode table counters associated with parent process are incremented as this area may be shared between parent process and the child process.
5. The child process ID is returned to the process and ‘0’ value is assigned to the child process.
6. All the above steps are carried out in kernel of the parent process.
7. The control of execution may remain with the parent process, may be transferred to child process or handed over to a third process leaving the parent and child processes in ‘Ready-to-Run’ state.

Q. 3. What happens when the system terminates a process with exit() system call?

Ans: The exit() system call is executed to terminate a process. This call is generally executed with some status code which is passed as an argument to the call. This status_code indicates the termination status of the corresponding process. The sequence of actions taken by OS in response to exit() is as follows,

1. Kernel sends a close call to all open files of the process.
2. Kernel also releases the memory assigned to the pr0ocess for execution.
3. The User area of the process is destroyed by the system.
4. The process structure is not destroyed though, and is retained till the parent process destroys the same.
5. The process though terminated, still exists as a dead process and is generally called a Zombie process.
6. The exit call also sends signal to the parent process indicating the termination of its child process.

Processes in Unix, Solaris and windows 2000 thread and SMP Management.

Q. 1 what are the Windows process attributes?

Ans:

Windows supports concurrent processing and multithreading. Windows process object attributes can be listed as,

Attribute	Description
Process ID	Unique process identifier
Security Descriptor	Describes process owner, process access rights for users, process access permissions for the users of the system.
Base priority	A baseline execution priority for the process and its threads
Default processor affinity	In a multiprocessor environment, the set of processors on which processes or threads can run.
Quota limits	The maximum amount of paged and nonpaged system memory, paging file space, and processor time a user's processes can use.
Execution time	The total amount of time all the threads of the process have spent with processor.
I/O counters	The variables those keep record of the number and types of I/O operations that the process's threads have performed.
VM operation counters	The variables those keep record of the number and types of virtual memory operations that the process's threads have performed.
Exception/debugging ports	These are interprocess communication channels to which the process manager can communicate in case the process causes any kind of exception.
Exit status	The reason of process termination.

Q. 2 what are the windows thread attributes?

Ans:

The windows thread attributes can be listed as given in the following table:

Attributes	Description
Thread ID	A unique ID assigned to thread
Thread context	This consists of register values and other data that defines the executions state of the thread
Dynamic priority	The execution priority of the thread at any given moment
Base priority	The basic priority assigned to the thread. This is also the lowest priority level the thread can have.
Thread processor affinity	In a multiprocessor environment, the set of processors on which processes or threads can run.
Thread execution time	The total time a thread time has spent with processor; which includes user mode and kernel mode processing, both.
Alert status	This is a flag that denotes in case a waiting thread can execute an asynchronous procedure call.
Suspension count	This is count of the number of times the thread's execution gets suspended without being resumed..
Impersonation token	This is a temporary access token that permits a thread to execute operations on behalf of another process.
Termination port	These are interprocess communication channels to which the process manager can communicate when the thread terminates.
Thread exit status	This field describes the reason for the thread's termination.

Q. 3 what are the different types of processes in UNIX?

Ans:

The Unix OS has three types of processes as: **user process, kernel process and daemon process.**

User process: The low privileged processes created on behalf of user are called user processes.

Kernel process: The high privileged processes initiated by operating system in response to calls made by user processes are called kernel processes. The system calls are the kernel processes.

Daemon process: these processes perform the functions in a system wide basis. The functions are generally of auxiliary kind, but they are very important from the point of controlling of the computing environment of the system. These processes once created, exist throughout the life time of the process. E.g. print spooling, network management.