Dr. SNS RAJALAKSHMI COLLEGE OF ARTS & SCIENCE(Autonomous)

Coimbatore – 49.

DEPARTMENT OF COMPUTER APPLICATIONS

COURSE : OPERATING SYSTEM

CLASS : I BCA 'B'

Unit II

Process Management

- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks
- Process – a program in execution; process execution must progress in sequentialfashion
- A process includes:
  - program counter
  - stack
  - data section

Process State

As a process executes, it changes *state*

- new: The process is being created
- running: Instructions are being executed
- waiting: The process is waiting for some event to occur
- ready: The process is waiting to be assigned to a processor
- terminated: The process has finished execution

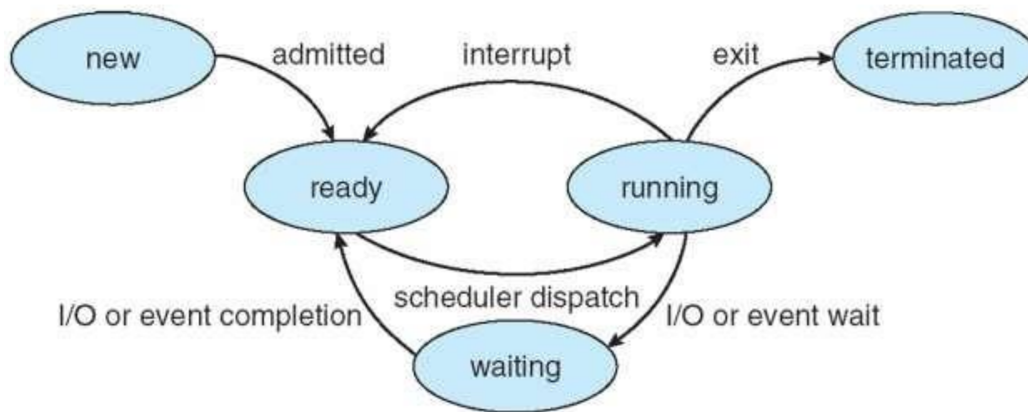


Fig: Process Transition Diagram

PCB: Process Control Block

Information associated with each process

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

Fig: PCB

Context Switching

- When CPU switches to another process, the system must save the state of theold process and load the saved state for the new process via a context switch
- Context of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
- Time dependent on hardware support

Process Scheduling Queues

- Job queue – set of all processes in the system
- Ready queue – set of all processes residing in main memory, ready and waitingto execute
- Device queues – set of processes waiting for an I/O device
- Processes migrate among the various queues

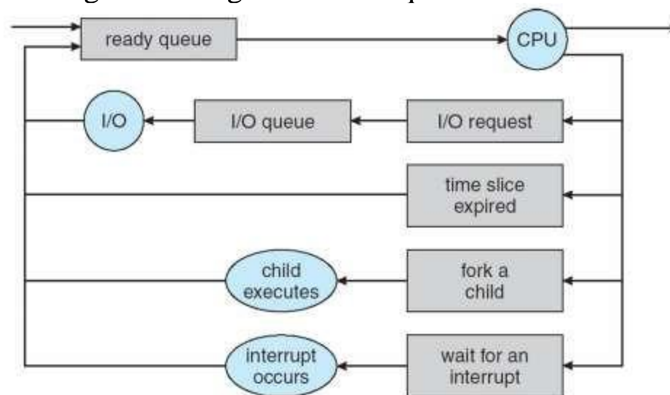


Fig: Process Scheduling

Schedulers

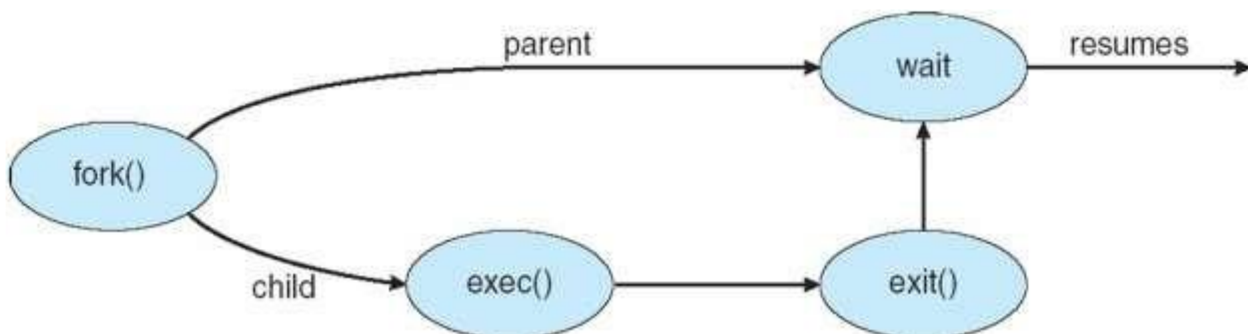
- Long-term scheduler (or job scheduler) – selects which processes should bebrought into the ready queue
- Short-term scheduler (or CPU scheduler) – selects which process should beexecuted next and allocates CPU
- Short-term scheduler is invoked very frequently (milliseconds) (must be fast)
- Long-term scheduler is invoked very infrequently (seconds, minutes) (may beslow)

- The long-term scheduler controls the *degree of multiprogramming*
- Processes can be described as either:
    - I/O-bound process – spends more time doing I/O than computations,many short CPU bursts
    - CPU-bound process – spends more time doing computations; few verylong CPU bursts

Process Creation

- **Parent** process create **children** processes, which, in turn create other processes, forming a tree of processes
- Generally, process identified and managed via **a process identifier** (**pid**)
- Resource sharing
    - Parent and children share all resources
    - Children share subset of parent's resources
    - Parent and child share no resources
- Execution
    - Parent and children execute concurrently
    - Parent waits until children terminate
- Address space
    - Child duplicate of parent
    - Child has a program loaded into it
- UNIX examples
    - **fork** system call creates new process

exec system call used after a **fork** to replace the process' memory spacewith a new program



Process Termination

- Process executes last statement and asks the operating system to delete it (exit)
  - Output data from child to parent (via wait)
  - Process' resources are deallocated by operating system
- Parent may terminate execution of children processes (abort)
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting
    - Some operating system do not allow child to continue if its parent terminates
      - All children terminated - cascading termination

Inter Process Communication

- Processes within a system may be independent or cooperating
- Cooperating process can affect or be affected by other processes, including sharingdata
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need interprocess communication (IPC)
- Two models of IPC
  - Shared memory
  - Message passing

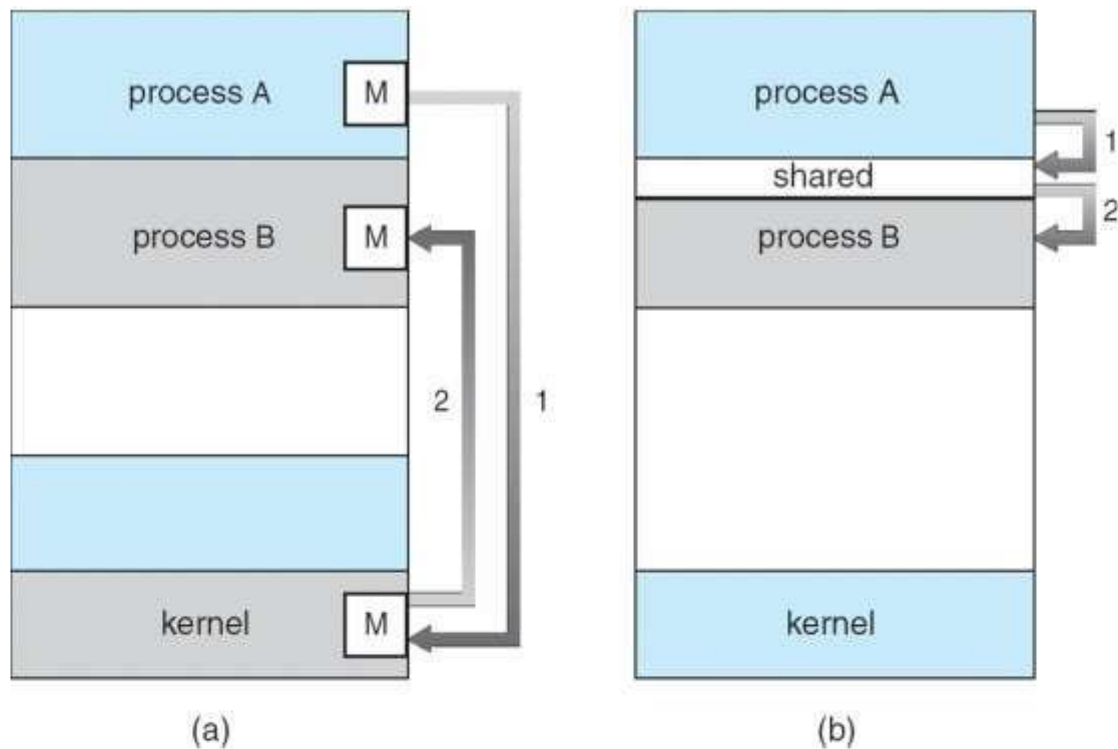


Fig:a- Message Passing, b- Shared Memory

Cooperating Process

- Independent process cannot affect or be affected by the execution of anotherprocess
- Cooperating process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

IPC-Message Passing

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting toshared variables
- IPC facility provides two operations:
  - send(*message*) – message size fixed or variable

- receive(*message*)

- If *P* and *Q* wish to communicate, they need to:
  - establish a *communication link* between them
  - exchange messages via send/receive
- Implementation of communication link
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)

Direct Communication

- Processes must name each other explicitly:
  - **send** (*P, message*) – send a message to process P
  - **receive**(*Q, message*) – receive a message from process Q
- Properties of communication link
  - Links are established automatically
  - A link is associated with exactly one pair of communicating processes
  - Between each pair there exists exactly one link
- The link may be unidirectional, but is usually bi-directional

Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports)
  - Each mailbox has a unique id
  - Processes can communicate only if they share a mailbox
- Properties of communication link
  - Link established only if processes share a common mailbox
  - A link may be associated with many processes
  - Each pair of processes may share several communication links
  - Link may be unidirectional or bi-directional
  - Operations
    - create a new mailbox
    - send and receive messages through mailbox

        - destroy a mailbox

    - Primitives are defined as:
        - **send**(*A, message*) – send a message to mailbox A
        - **receive**(*A, message*) – receive a message from mailbox A
    - Allow a link to be associated with at most two processes
    - Allow only one process at a time to execute a receive operation
    - Allow the system to select arbitrarily the receiver. Sender is notified whothe receiver was.

Synchronisation

- Message passing may be either blocking or non-blocking
- **Blocking** is considered **synchronous**
    - **Blocking send** has the sender block until the message is received
    - **Blocking receive** has the receiver block until a message is available
- **Non-blocking** is considered **asynchronous**
    - **Non-blocking** send has the sender send the message and continue
    - **Non-blocking** receive has the receiver receive a valid message or null

Buffering

Queue of messages attached to the link; implemented in one of three ways

1. Zero capacity – 0 messages
   Sender must wait for receiver (rendezvous)
2. Bounded capacity – finite length of *n* messages
   Sender must wait if link full
3. Unbounded capacity – infinite length
   Sender never waits

Thread

- A thread is a flow of execution through the process code, with its own program counter, system registers and stack.
- A thread is also called a light weight process. Threads provide a way to improve application performance through parallelism.

- Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

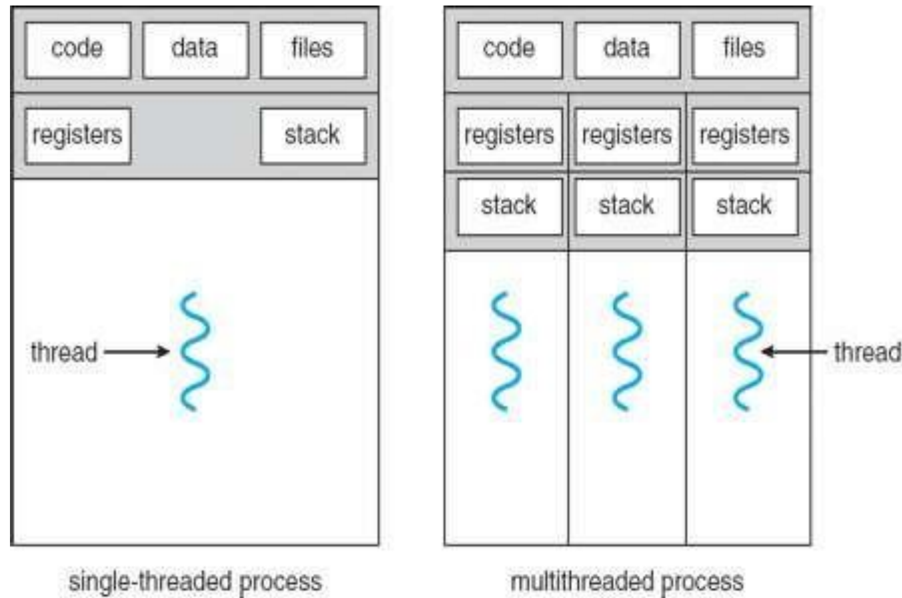


Fig: Single threaded vs multithreaded process

Benefits

- Responsiveness
- Resource Sharing
- Economy
- Scalability

User Threads

- Thread management done by user-level threads library
- Three primary thread libraries:
    - POSIX Pthreads
    - Win32 threads
    - Java threads

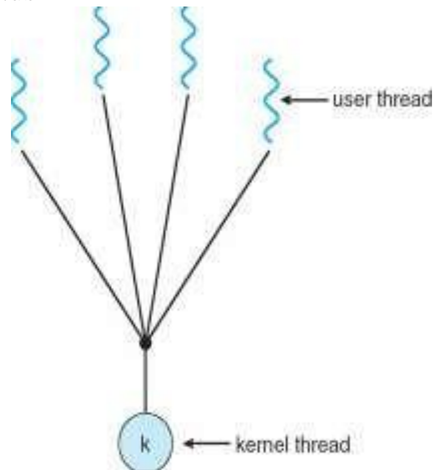
Kernel Thread

- Supported by the Kernel
- Examples
    - Windows XP/2000

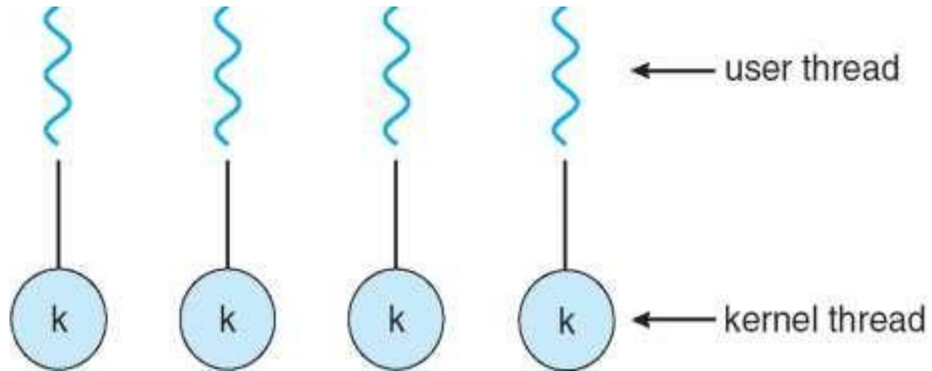
- Solaris
- Linux
- Tru64 UNIX
- Mac OS X

Multithreading Models

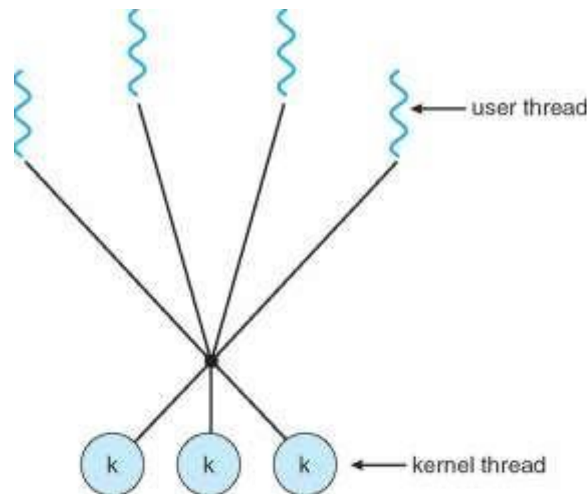
- Many-to-One
  - Many user-level threads mapped to single kernel thread
  - Examples:
  - Solaris Green Threads
  - GNU Portable Threads



- One-to-One
  - Each user-level thread maps to kernel thread
  - Examples
  - Windows NT/XP/2000
  - Linux
  - Solaris 9 and later



- Many-to-Many
    - Allows many user level threads to be mapped to many kernel threads
    - Allows the operating system to create a sufficient number of kernel threads
    - Solaris prior to version 9
    - Windows NT/2000 with the *ThreadFiber* package



Threading Issues

- Semantics of fork() and exec() system calls
- Thread cancellation of target thread
    - o Asynchronous or deferred
- Signal handling
- Thread pools
- Thread-specific data
- Scheduler activations

Thread Cancellation

- Terminating a thread before it has finished
- Two general approaches:
    - o Asynchronous cancellation terminates the target thread immediately
    - o Deferred cancellation allows the target thread to periodically check if it should be cancelled

Thread Pools

- Create a number of threads in a pool where they await work
- Advantages:
    - o Usually slightly faster to service a request with an existing thread thancreate a new thread
    - o Allows the number of threads in the application(s) to be bound to the sizeof the

pool

Thread Scheduling

- Distinction between user-level and kernel-level threads
- Many-to-one and many-to-many models, thread library schedules user-levelthreads to run on LWP
  - Known as **process-contention scope (PCS)** since scheduling competition is within the process
- Kernel thread scheduled onto available CPU is **system-contention scope(SCS)** – competition among all threads in system

Difference between Process and Thread

Process	**Thread**
Process is heavy weight or resource intensive.	Thread is light weight taking lesser resources than a process.
Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
In multiple processing environments each process executes the same code but has itsown memory and file resources.	All threads can share same set ofopen files, child processes.
If one process is blocked then no other process can execute until the first process isunblocked.	While one thread is blocked and waiting, second thread in the sametask can run.
Multiple processes without using threads usemore resources.	Multiple threaded processes usefewer resources.
In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

Process Scheduling

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a *cycle* of CPU executionand I/O wait

- **CPU burst** distribution

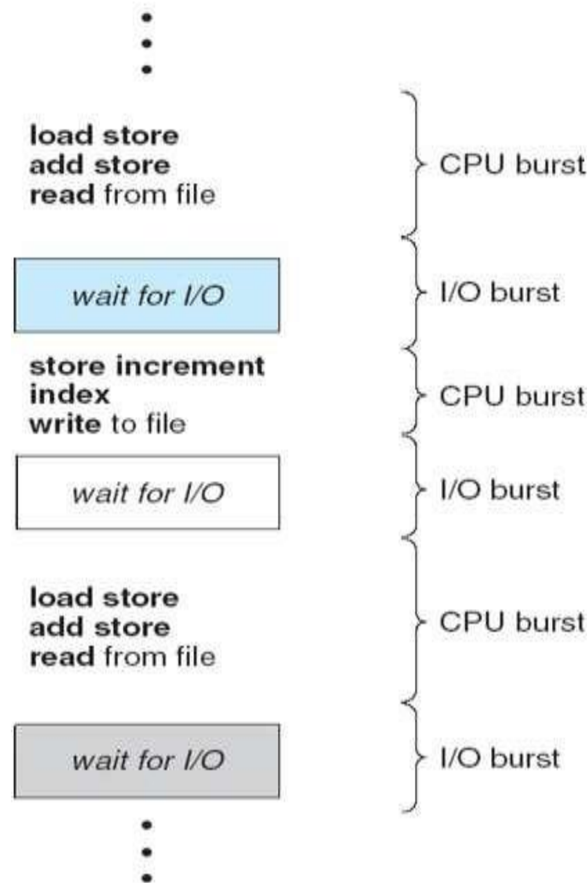


Fig: CPU burst and I/O burst

CPU Scheduler

- Selects from among the processes in memory that are ready to execute, andallocates the CPU to one of them
- CPU scheduling decisions may take place when a process:

1. Switches from running to waiting state
2. Switches from running to ready state
3. Switches from waiting to ready
4. Terminates

- Scheduling under 1 and 4 is **nonpreemptive**
- All other scheduling is **preemptive**

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- Dispatch latency – time it takes for the dispatcher to stop one process and startanother running

CPU Scheduling Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

CPU Scheduling Algorithms

A. First Come First Serve Scheduling

- Schedule the task first which arrives first
- Non preemptive In nature

B. Shortest Job First Scheduling

- Associate with each process the length of its next CPU burst. Use theselengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes

o The difficulty is knowing the length of the next CPU request

Priority Scheduling

- A priority number (integer) is associated with each process

Thread Scheduling:

Scheduling of threads involves two boundary scheduling,

- Scheduling of user level threads (ULT) to kernel level threads (KLT) vialightweight process (LWP) by the application developer.

- Scheduling of kernel level threads by the system scheduler to performdifferent unique os functions.

Lightweight Process (LWP) :

Light-weight process are threads in the user space that acts as an interfacefor the ULT to access the physical CPU resources. Thread library schedules which thread of a process to run on which LWP and how long.

The number of LWP created by the thread library depends on the type of application. In the case of an I/O bound application, the number of LWP dependson the number of user-level threads.

This is because when an LWP is blocked on an I/O operation, then to invoke the other ULT the thread library needs to create and schedule another LWP. Thus, in an I/O bound application, the number of LWP is equal to the number of the ULT. In the case of a CPU bound application, it depends only onthe application. Each LWP is attached to a separate kernel-level thread.