Chapter 1

Processes and Threads

LEARNING OBJECTIVES

- *Basics of operating systems*
- *Services of OS*
- *Evolution of operating systems*
- *Processes*
- *Processes and process control blocks*
- *Process states*
- *Process creation*
- *Suspended processes*
- *OS control structures*
- *Process attributes*
- *Modes of execution*
- *Threads*
- *Thread functionality*
- *Thread synchronization*

Basics of Operating System

An operating system (OS) is a program that controls the execution of application programs and acts as an interface between applications and the computer hardware. The three objectives of an OS are as follows:

1. *Convenience*: An OS makes a computer more convenient to use.
2. *Efficiency*: An OS allows the computer system resources to be used in an efficient manner.
3. *Ability to evolve*: An OS should be constructed in such a way as to permit the effective development, testing and introduction of new system functions without interfering with service.

OS as a User–Computer Interface

Consider the below figure (Figure 1) which shows the hardware and software used in providing applications to a user in a layered fashion.

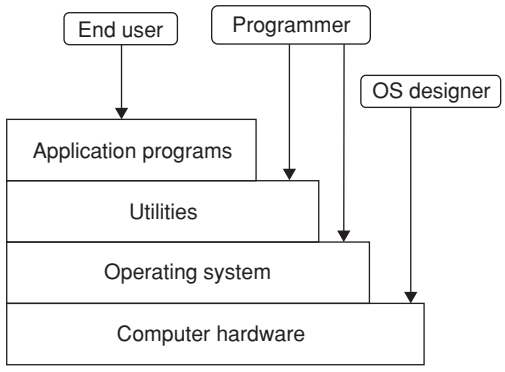


Figure 1 Layers and views of a computer system.

The end user of the application is not concerned with the details of computer hardware. Utilities implement the frequently used functions that assist in program creation, the management of files and control of input/output (I/O) devices. The most important collection of system programs comprises the OS. The OS masks the details of the hardware from the programmer and provides the programmer with a convenient interface for using the system.

Services of an OS

1. *Program development*: An OS provides a variety of facilities and services as editors and debuggers to assist programmers in creating programs.
2. *Program execution*: An OS handles the scheduling duties of program execution for the user.
3. *Access to I/O devices*: An OS provides a uniform interface that hides the details of I/O devices so that the programmers can access the I/O devices using simple reads and writes.
4. *Controlled access to files*: In a system with multiple users, an OS provides protection mechanism to control access to the files.
5. *System access*: For shared or public systems, an OS controls access to the system as a whole and to specific system resources.
6. *Error detection and response*: An OS must provide a response that clears the error condition with the least impact on running applications.
7. *Accounting*: A good OS will collect usage statistics for various resources and monitor performance parameters (viz., response time).

OS as Resource Manager

A computer is a set of resources for the movement, storage and processing of data and for the control of these functions. An OS is responsible for managing these resources.

1. An OS functions in the same way as ordinary computer software, that is, it is a program or suite of programs executed by the processor.
2. An OS frequently relinquishes control and must depend on the processor to allow it to regain control.

The OS directs the processor in the use of the other system resources and in the timing of its execution of other programs.

Figure 2 shows the resources that are managed by an OS.

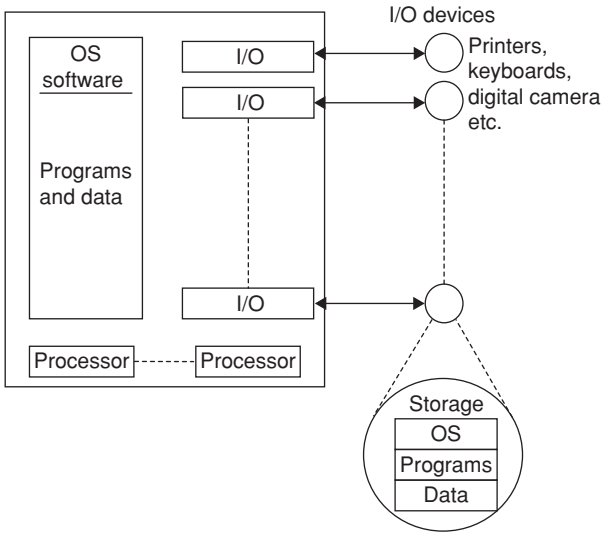


Figure 2 OS as a resource manager.

A portion of the OS lies in the main memory. This includes the kernel or nucleus, which contains the most frequently used functions in the OS and, at a given time, other portions of the OS currently in use. The remainder of main memory contains user programs and data. The allocation of this resource is controlled jointly by the OS and memory management hardware in the processor.

Evolution of OSs

Serial processing

With the earliest computers, the programmer interacted directly with the computer hardware. There was no OS. Programs in machine code were loaded via the input devices. This mode of operation is termed as *serial processing*. Problems with this system are scheduling and setup time.

Simple batch systems

- It requires the grouping up of similar jobs, which consist of programs, data and system commands.
- Users have no control over results of a program.
- Off-line debugging.

Multiprogrammed batch systems

In view of simultaneous execution of multiple programs, it improves system throughput and resource utilization.

Example: Windows XP, 98

- **Multitasking OS:** A running state of a program is called a *process* or a *task*. The concept of managing a multitude of simultaneously active programs, competing with each other for accessing the system resources is called *multitasking*.
- Serial multitasking or context switching is the simplest form of multitasking.

Example: Windows NT, Linux

- **Multiuser OS:** It is defined as multiprogramming OS that supports simultaneous interaction with multiple users.

Example: Linux, Unix, a dedicated transaction processing system (viz., railway reservation system).

- **Multiprocessing OS:** The term *multiprocessing* means multiple CPUs performing more than one job at one time.

The term 'multiprogramming' means situation in which a single CPU divides its time between more than one job.

Time sharing systems

In this kind of OS, the processor time is shared among multiple users. The CPU switches rapidly from one user to another user; each user is given an impression that he/she has his own computer while it is actually one computer shared among many users.

If there are n users actively requesting service at one time, each user will only see on the average $1/n$ of the effective computer capacity, not counting OS overhead.

Bootstrap Bootstrap is an initial program which runs, when a computer is powered up (or) restarted. The task is to initialize system aspects (CPU registers to device controllers to memory contents). It is stored within the computer hardware known as *firm ware* (EEPROM).

PROCESSES

Processes and Process Control Blocks

Process A process is an instance of a program in execution. Two essential elements of a process are as follows:

1. Program code
2. Set of data

At any given point in time, while the program is executing, the process can be uniquely characterized by a number of elements, including the following:

1. Identifier
2. State
3. Priority

4. Program counter
5. Memory pointers
6. Context data
7. I/O status information
8. Accounting information

This information is stored in a data structure, typically called a *process control block*, that is created and managed by the OS.

Process control block (PCB) It contains sufficient information so that it is possible to interrupt a running process and later resume execution as if the interruption has not occurred.

Process States

The behaviour of an individual process can be characterized by listing the sequence of instructions that executed for that process. This listing is referred to as a *trace* of the process. Also the behaviour of a processor is shown by listing the traces of the various processes that are interleaved.

Dispatcher A dispatcher is a small program that switches the processor from one process to another.

Two-state process model In the simplest possible process model (Figure 3), at any time, a process is either being executed by a processor or not, that is, a processor may be in one of two states: *running* or *not running*.

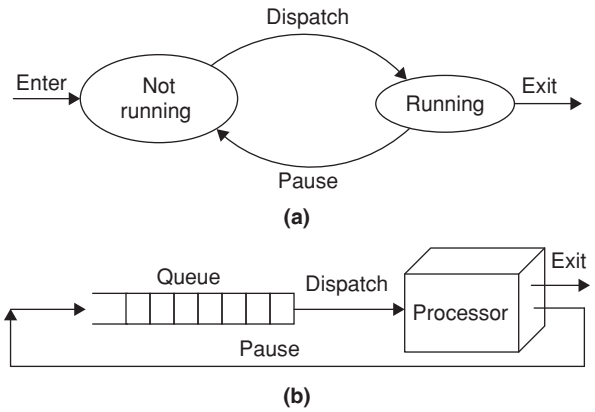


Figure 3 Two-state process model. (a) State transition diagram, (b) Queuing diagram

When the OS creates a new process, it creates the PCB for the process and enters that process into the system in the *not running* state. The process exists, is known to the OS, and is waiting for an opportunity to execute. From time to time, the currently running process will be interrupted and the dispatcher portion of the OS will select some other process to run. The former process moves from the *running* state to the *not running* state and one of the other processes moves to the *running* state.

Processes that are not running must be kept in some sort of queue, waiting their turn to execute. Figure 3(b) shows the structure. There is a single *queue* in which each entry is a pointer to the PCB of a particular process.

Creation and Termination of Processes

Process creation

When a new process is to be added to those currently being managed, the OS builds the data structures that are used to manage the process and allocates address space in main memory to the process. The common events which lead to process creation are as follows:

1. New batch job
2. Interactive logon
3. Created by OS to provide a service
4. Spawned by existing process

When the OS creates a process at the explicit request of another process, the action is referred to as *process spawning*. When one process spawns another, the former is referred to as the *parent process* and the spawned process is referred to as the *child process*.

Process termination

The following are the reasons for process termination:

1. *Normal completion*: Process executes an OS service call to indicate its completion.
2. *Time limit exceeded*: The process has run longer than the specified total time limit.
3. *Memory unavailable*: The process requires more memory than the system can provide.
4. *Bounds violation*: The process tries to access a memory location that it is not allowed to access.
5. *Protection error*: The process attempts to use a resource that is not allowed to access.
6. *Arithmetic error*: The process tries a prohibited computation.
7. *Time overrun*: The process has waited longer than a specified maximum for a certain event to occur.
8. *I/O failure:* Error occurs during input or output.
9. *Invalid instruction*: The process attempts to execute a non-existent instruction.
10. *Privileged instruction*: The process attempts to use an instruction reserved for OS.
11. *Data misuse*: A piece of data is of the wrong type or is not initialized.
12. *Operator or OS intervention.*
13. *Parent termination.*
14. *Parent request.*

Five-state model

The five states in Figure 4 are as follows:

1. *New*: The process is created but not admitted to the pool of executable processes.
2. *Running*: Process in execution, that is, it is using CPU.
3. *Blocked*: Waiting for some event to occur (i.e., I/O) before it can continue execution.

4. *Ready*: Process is ready for execution. Just it is waiting.
5. *Exit*: The process has been aborted by parent process or has finished its execution.

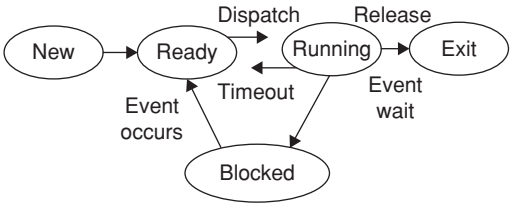


Figure 4 Process states.

Figure 4 indicates the types of events that lead to each state transition for a process. The possible transitions are as follows:

1. NULL → New: A new process is created to execute a program.
2. New → Ready: The OS will move a process from the New state to the Ready state when it is prepared to take on an additional process.
3. Ready → Running: When it is time to select a process to run, the OS chooses one of the processes in the Ready state.
4. Running → Exit: The currently running process is terminated by the OS if the process indicates that it has completed or if it aborts.
5. Running → Ready: The reasons for this transition are
   - Running process has reached the maximum allowable time for uninterrupted execution.
   - As the OS assigns different levels of priority to different processes, there will be pre-emption.
   - A process may voluntarily release control of the processor.
6. Running → Blocked: A process is put in the blocked state if it requests something for which it must wait.
7. Blocked → Ready: This transition occurs when the event for which the process has been waiting occurs.
8. Ready → Exit: A parent may terminate a child process at any time.
9. Blocked → Exit: Parent may terminate any blocked process.

Queuing model for five-state model

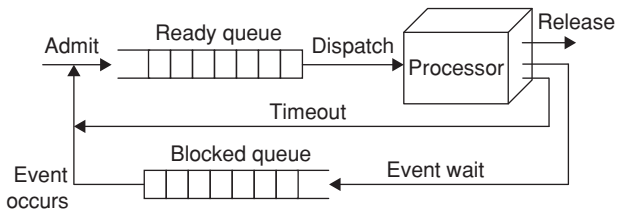


Figure 5 Single blocked queue.

Suspended Processes

Need for swapping In five-state process model using multiple blocked queues, the memory holds multiple processes. Moreover, the processor can move to another process when one process is blocked. But the processor is so much faster than I/O that it will be common for all of the processes in memory to be waiting for I/O. Thus, even with multiprogramming, a processor could be idle most of the time.

Then we can extend the main memory to accommodate more processes, but it is not an efficient solution. Another solution to this problem is swapping. Swapping involves moving part or all of a process from main memory to disk. When none of the processes in main memory is in the ready state, the OS swaps one of the blocked processes out onto disk into a suspend queue. The OS then brings in another process from the suspend queue or it honours a new process request. Then execution continues with the newly arrived process. With the use of swapping, another state is added to the process in the behaviour model.

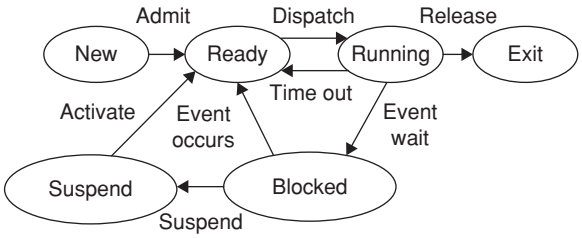


Figure 6 Process state-transition diagram with suspend state.

The four distinguishable states in this process model are as follows:

1. *Ready*: The process is in main memory and is available for execution.
2. *Blocked*: The process is in main memory and awaiting an event.
3. *Blocked/suspend*: The process is in secondary memory and awaiting an event.
4. *Ready/suspend*: The process is in secondary memory but is available for execution as soon as it is loaded into main memory.

Figure 7 shows the process state model with two suspend states:

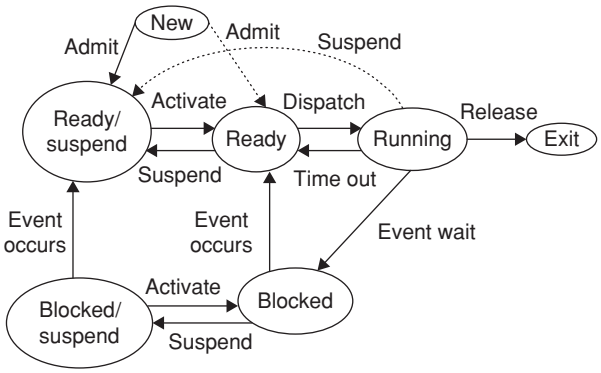


Figure 7 Process state transition diagram with suspend state.

Uses of Suspension

Characteristics of suspended process are as follows:

1. The process is not immediately available for execution.
2. The process may or may not wait on an event.
3. The process was placed in a suspend state by either itself, a parent or the OS.

Reasons for process suspension

- *Swapping*: To Release sufficient main memory.
- *Other OS reason*: OS may suspend a background process.
- *Interactive User Request*: A user may wish to suspend execution of a program.
- *Timing*: A process may be executed periodically and may be suspended.
- *Parent Process request*: A parent process may wish to suspend execution of a descendent.

OS Control Structures

If the OS is to manage processes and resources, it must have information about the current status of each process and resources. The OS constructs and maintains tables of information about each entity that it is managing. Figure 8 shows the general structure of OS control tables:

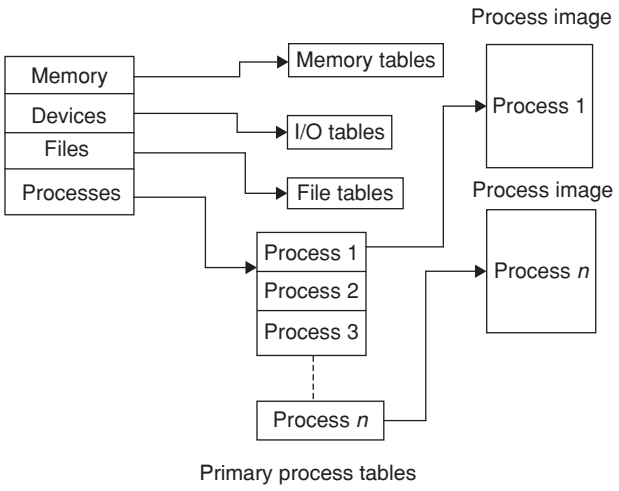


Figure 8 OS control tables.

Different tables maintained by the OS are

1. Memory
2. I/O
3. File
4. Process

Memory tables: These tables are used to keep track of both main and secondary memory.

I/O tables: These are used by the OS to manage the I/O devices and channels of the computer system.

File tables: These tables provide information about the existence of files, their location on secondary memory, their current status and other attributes.

Process tables: An OS must maintain process tables to manage processes.

Process Control Structures

The OS must know about

1. Process location
2. Process attributes

Process Location

The collection of program, data, stack and attributes is referred as process image.

The location of a process image will depend on the memory management scheme being used. The process image is maintained as a contiguous or continuous block of memory. This block is maintained in secondary memory, usually disk, so that the OS can manage the process, at least a small portion of its image must be maintained in main memory. To execute the process, the entire process image must be loaded into main memory or at least virtual memory. Thus the OS needs to know the location of each process on disk and for each such process that is in the main memory, the location of that process is in main memory.

For this, the OS maintains process tables. There is a primary process table with one entry for each process. Each entry contains, at least, a pointer to a process image.

Process Attributes

The typical information required by the OS for each process is as follows:

1. Process identification
2. Process state information
3. Process control information

Process identification Each process is assigned a unique numeric identifier, which may simply be an index into the primary process table. The identifier for a PCB includes the following:

1. Identifier of the process
2. Identifier of the process that created current process
3. User identifier

Process state information It consists of the contents of processor registers. It includes details of

1. User-visible register
2. Control and status registers
3. Stack pointers

Process control information It consists of the additional information needed by the OS to control and coordinate the various active processes. It includes the following:

1. Scheduling and state information
2. Data structuring
3. Interprocess communication

4. Process privileges
5. Memory management
6. Resource ownership and utilization

Process Control

Modes of Execution

Most processors support at least two modes of execution as follows:

1. More-privileged mode
2. Less-privileged mode

Two modes are required to protect the OS and key OS tables from interference by user programs.

1. ***More-privileged mode:*** This is also referred as *system mode*, *control mode* or *kernel mode*. Certain instructions can only be executed in Kernel mode (e.g., reading or altering a control register, viz., PSW, primitive I/O instructions, etc.).

   The Kernel of the OS is a portion of the OS and encompasses the important system functions.

   The functions of an OS kernel are as follows:
   - Process management
   - Memory management
   - I/O Management
   - Support functions
2. ***Less-privileged mode:*** This is also referred as *user mode*, because user programs typically would execute in this mode.

   In this mode, the software has complete control of the processor and all its instructions, registers and memory.

Process creation If the OS decides to create a table, it has to proceed as follows:

1. Assign a unique process identifier to the new process.
2. Allocate space for the process.
3. Initialize the PCB
4. Set the appropriate linkages.
5. Create or expand other data structures.

Process switching In process switching, a running process is interrupted and the OS assigns another process to the running state and turns control over to that process. The design issues are as follows:

1. When to switch processes
2. Mode switching
3. Change of process state

When to switch processes A process switch may occur anytime that the OS has gained control from the currently running process. The mechanisms for interrupting the execution of a process are as follows:

1. Interrupt
2. Trap
3. Supervisor call

Interrupt When an interrupt occurs, the control is first transferred to an interrupt handler, which does some basic housekeeping and then branches to an OS routine that is concerned with the particular type of interrupt that has occurred (e.g., clock interrupt, I/O interrupt, memory fault, etc.).

Trap Trap related to an error or exception condition gets generated within the currently running process. If the error is fatal, the currently running process is moved to exit state and a process switch occurs, otherwise the action of the OS will depend on the nature of the error and design of the OS.

Supervisor call The OS may be activated by a supervisor call from the program being executed. The case of system call may place the user process in blocked state.

Mode switching If the processor identifies that any interrupt is pending, then

1. it sets the PC to the starting address of an interrupt handler program.
2. it switches from user mode to Kernel mode so that the interrupt processing code may include privileged instructions.

During this process, the context of the process, that has been interrupted, is saved into that PCB of the interrupted program. The context of a program includes PC, other processor registers and stack information.

The occurrence of an interrupt does not necessarily mean a process switch.

Change of process state The mode switch is a concept distinct from that of the process switch.

A mode switch may occur without changing the state of the process that is currently in Running state. In that case, the context saving and subsequent restoral involve little overhead. However, if the currently running process is to be moved to another state then the OS must make substantial changes in its environment. Thus, the process switch, which involves a state change, requires more effort than a mode switch.

System call In order to access the OS services, an interface is required which is provided by the system call.

All the system call routines are executed in Kernal mode. Whenever the system call is invoked, the process status word is changed from user mode to Kernal mode ($0 \rightarrow 1$).

System calls are of six types as follows:

1. File system
2. Process
3. Scheduling
4. Interprocess communications
5. Socket
6. Miscellaneous

Execution of the OS

There are three possibilities to consider about OS execution:

1. Separate Kernel (Figure 9)
2. OS functions execute within user processes (Figure 10)
3. OS functions execute as separate processes (Figure 11)

Separate Kernel

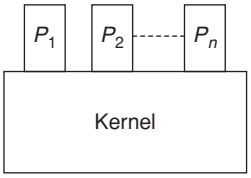


Figure 9 OS as separate Kernel.

Here the Kernel of the OS is executed outside of any process. When currently running process is interrupted or issues a supervisor call, the mode context of this process is saved and control is passed to the Kernel.

Execution within user process

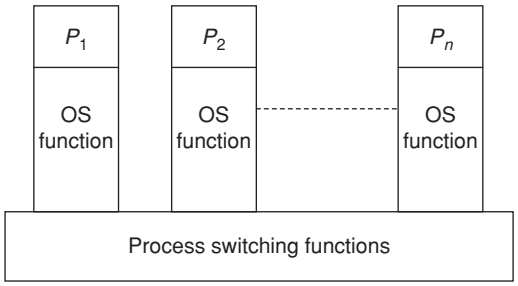


Figure 10 OS functions execute within user processes.

We execute virtually all OS software in the context of user process. To pass control from a user program to the OS, the mode context is saved and a mode switch takes place to an OS routine, that is, a process switch is not performed, just a mode switch within the same process.

Process-based OS

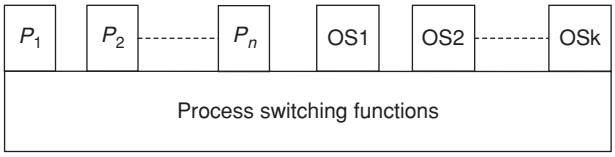


Figure 11 OS functions execute as separate process.

Here the OS is a collection of system processes. This approach encourages the use of modular OS with minimal, clean interface between the modules.

THREADS

A *thread* is a basic unit of CPU utilization. It comprises a thread ID, a program counter, a register set and a stack.

Multithreading

It refers to the ability of an OS to support multiple, concurrent paths of execution within a single process.

The threads which belong to same process can share their

1. Code section
2. Data section
3. Other OS resources

If a process has multiple threads of control, it can perform more than one task at a time. Figure 12 shows single threaded and multithreaded process models:

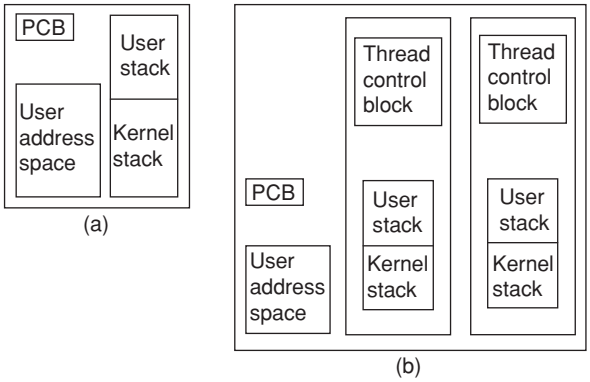


Figure 12 Process models. (a) Single-thread process model (b) Multithreaded process model.

As OS based on its design will be in one of the following manners (Figure 13):

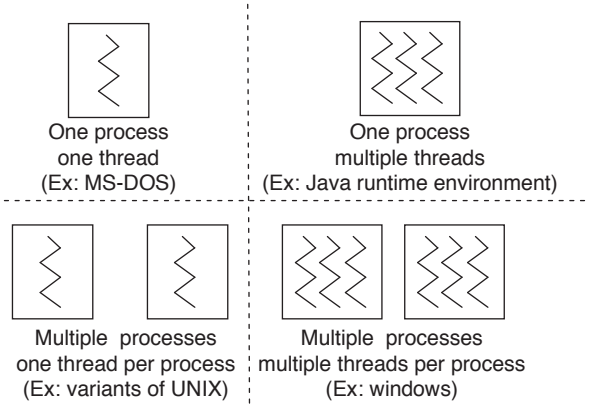


Figure 13 Threads and processes.

The threads of a process consist of the following:

1. Thread execution state.
2. Saved thread context when not running
3. An execution stack
4. Some pre-thread static storage for local variables.
5. Access to the memory and resources of its process, shared with all other threads in that process.

All the threads of a process share the state and resources of that process.

Benefits of multithreaded programming

1. *Responsiveness*: Multithreading an interactive application may allow program to continue running even if

part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user.
2. *Resource sharing*: Threads share the memory and the resources of the process to which they belong by default.
3. *Economy*: It is economical to create a new thread in an existing process than to create a brand-new process. It takes less time to context switch between two threads of same process than to switch between processes. Also the time to terminate a thread is less than process termination.
4. *Scalability*: Multithreading on a multi-CPU machine increases parallelism.

Applications that benefit from thread As the threads take advantages of multiple processors, image processing which can be done in parallel, will execute in threads.

Animation rendering is another thread application, where each frame can be rendered in parallel, as each one is independent of other GUI programming will execute at least two threads when it is processing large number of files.

Applications that cannot benefit from thread The main drawbacks of threads is if kernel is single threaded, system call of one thread will block the whole process, in which CPU will be idle during the blocking period.

The other major drawback is security as it is possible that a thread can overwrite the stack as the other thread, as they were meant to cooperate on a single task. Applications that are developed using PHP does not support multithreading at the server side.

Thread Functionality

The key states for threads are as follows:

1. Running
2. Ready
3. Blocked

There are four basic thread operations associated with a change in thread state:

1. *Spawn*: When a new process is spawned, a thread for that process is also spawned.

   Also, a thread within a process may spawn another thread within the same process.
2. *Block*: When a thread needs to wait for an event, it will block. Then the processor may turn to the execution of another ready thread in the same or different process.
3. *Unblock*: When an event for which a thread is blocked occurs, the thread is moved to Ready queue.
4. *Finish*: When a thread completes, its register context and stacks are deallocated.

Multithreading on a Uni-processor

On a uni-processor, multiprogramming enables the interleaving of multiple threads within multiple processes. For example, consider the execution of three threads *A*, *B*, *C* in two processes on a single processor which are interleaved (Figure 14).

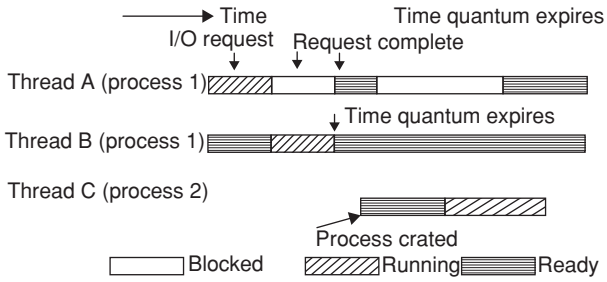


Figure 14 Multithreading on a uni-processor.

Execution passes from one thread to another, either when the currently running thread is blocked or its time slice is exhausted.

Resources used in thread creation and process creations

As process has heavy weight, when it is created, new address space is required, which includes stack, heap and data section, etc. If a process shares the memory, then the IPC is expensive.

The thread is a light-weight process, if it doesn't require any new resources, as it will share the process resources to which it belongs. The major benefit of this is, several threads belong to same activity and can run under same address space.

Thread Synchronization

All of the threads of a process share the same address space and other resources, such as open files. Any alteration of a resource by one thread affects the environment of the other threads in the same process. So, synchronization mechanism is required to coordinate the activities of all the threads within a process.

The techniques used for thread synchronization is the same as process synchronization techniques, which has been discussed later in this book.

Types of Threads

User-Level Threads

All thread management is done by the application. The Kernel is not aware of the existence of threads. Thread creation and scheduling are done in user space. User-level threads (Figure 15) are fast to create and manage. User-level library provides support for creating, managing and scheduling threads. In single-threaded Kernel, blocking system call from user level thread will block the entire process, even if other threads are ready to run.

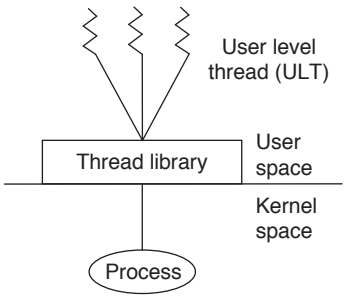


Figure 15 Pure user-level threads.

Advantages of ULTs

1. ULT creation does not require Kernel mode privileges.
2. ULT scheduling can be application specific.
3. ULTs can run on any OS.

Disadvantages of ULTs

1. When a ULT executes a system call, not only is that thread blocked, but also all of the threads within the process are blocked.
2. In a pure ULT strategy, a multithreaded application cannot take advantage of multiprocessing.

Kernel-level Threads

Kernel-level threads (KLTs) are supported directly by the OS. The creation, scheduling, management are done by kernel in kernel space. They are slower to create and manage. In a multiprocessor, Kernel can schedule threads on different processors.

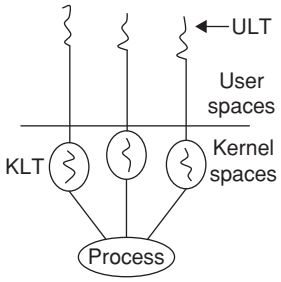


Figure 16 Pure kernel-level threads.

Advantages of KLTs

1. Kernel can simultaneously schedule multiple threads from the same process on multiple processors.
2. If one thread in a process is blocked, the kernel can schedule another thread of the same process.
3. Kernel routines themselves multithreaded.

Disadvantages of KLTs

The transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

Combined Approach

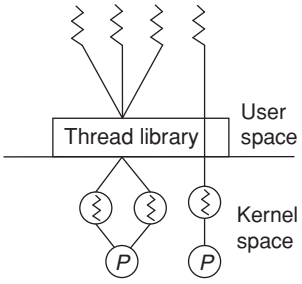


Figure 17 Combined approach.

In combined approach (Figure 17), multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process.

Relationship between the threads and processes

1. *One-to-one relationship*: Each thread of execution is a unique process with its own address and resources.
   Example: Traditional UNIX.
2. *Many-to-one relationship*: A process defines an address space and dynamic resources ownership. Multiple threads may be created and executed within that process.
   Example: Windows NT, Solaris, Linux.
3. *One-to-many relationship*: A thread may migrate from one process environment to another. This allows a thread to be easily moved among distinct systems.
   Example: Emerald
4. *Many-to-many relationship*: It combines the attributes of M:1 and 1:M cases.
   Example: TRIX

Threading Issues

fork( ) and exec( ) System Calls

A fork( ) system call is used to create a separate, duplicate process. In UNIX, each process is identified by its process identifier, which is a unique integer. A new process is created by fork( ) system call.

The new process consists of a copy of the address space of the original process. This mechanism allows the parent process to communicate easily with its child process. Both processes continue execution at the instruction after the fork( ), with one difference: the return code for the fork( ) is zero for the new process, whereas the process identifier of the child is returned to the parent. The exec( ) system call is used after a fork( ) system call by one of the two processes to replace the processes memory space with a new program.

If one thread in a program calls fork( ), then UNIX chooses two alternatives as follows:

1. Duplicates all the threads
2. Duplicates only the thread that invoked the fork( ) system call.

If a thread invokes the exec( ) system call, the program specified in the parameter to exec( ) will replace the entire process including all threads.

Cancellation

Thread cancellation is the task of terminating a thread before it has completed. A thread that is to be cancelled is often referred to as the *target thread*. Cancellation of a thread may occur in two different scenarios as follows:

1. *Asynchronous cancellation*: One thread immediately terminates the target thread.
2. *Deferred cancellation*: The target thread periodically checks whether it should terminate, allowing it an opportunity to terminate itself in an orderly fashion.

Microkernels

Microkernel (Figure 18) is a small OS core that provides the foundation for modular extensions.

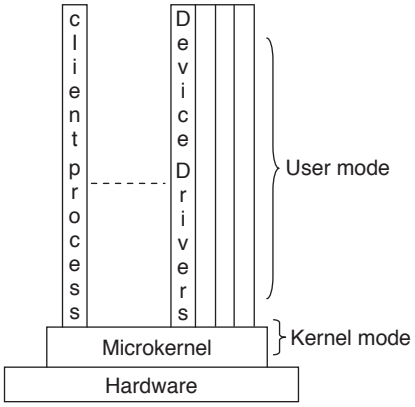


Figure 18 Microkernel architecture.

Advantages of Microkernel Organization

- Uniform interfaces
- Extensibility
- Flexibility
- Portability
- Reliability
- Distributed system support
- Support for object oriented OS

Exercises

Practice Problems I

Directions for questions 1 to 15: Select the correct alternative from the given choices.

1. In fork( ) system call, the return value to the parent process and the child process are respectively
(A) PID of child process, 1
(B) PID of child process, 0
(C) PID of child, PID of parent process
(D) PID of parent process, PID of child

2. Which of the following is not an advantage of thread?
(A) Inter process communication
(B) Less memory space occupied by thread
(C) Less time to create and terminate than a process
(D) Context switching is faster

3. A process executes the following segment of code

for ($i = 1$; $i < 10$; i ++) fork( );

The number of new processes created is
(A) 1024 (B) 1023
(C) 1025 (D) 1028

4. For each of the following transitions, between process states, which transition is not possible?
(A) Running → Ready
(B) Blocked → Suspend
(C) Ready → Ready/Suspend
(D) Blocked → Running

5. An operating system can be mapped to a five-state process model. A new event has been designated as capable to pre-empt the existing processes in order to trigger a new process to complete. Select the correct statement from below:
(A) A new state need to be added to the existing transition model to accommodate the changes.
(B) The existing model still holds good.
(C) Both the states and transitions of the existing model have to be changed.
(D) Only the transitions need to be modified.

6. The advantage of having multiple threads over multiple processes is
(i) Less time for creation
(ii) Less time for termination
(iii) Less time for switching
(iv) Kernel not involved in communication among threads
(A) (i), (ii), (iii) (B) (i), (ii), (iv)
(C) (ii), (iii), (iv) (D) (i), (ii), (iii), (iv)

7. Select the correct sequence of steps taken by the processor when an interrupt occurs
(i) Switch from user mode to kernel mode.
(ii) Set the program counter to the first instruction of the interrupt handling routine.
(iii) Save the current context.

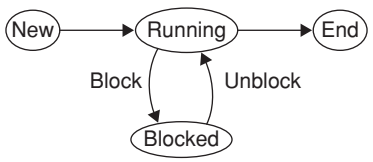
(A) (i), (ii), (iii)
(B) (i), (iii), (ii)
(C) (iii), (ii), (i)
(D) (ii), (i), (iii)

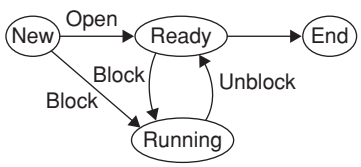
8. What are the necessary steps for a new process creation?
(i) Assign an identifier to the new process.
(ii) Suspend all other processes.
(iii) Allocate space for the process.
(iv) Initialize process control block.
(v) Update process-related data structures.
(vi) Update process state information wherever necessary.
(vii) Set the process to user mode
(viii) Notify all the machines in the network about the new process.
(ix) Set the state of the new process as suspended
(A) All except (ii), (v), (vi), (viii)
(B) All except (ii), (vii), (viii), (ix)
(C) All except (iv), (v), (vii), (ix)
(D) All except (iii), (iv), (vii), (ix)

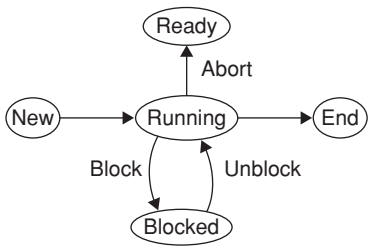
9. A processor, while executing the instruction sequence of user mode process, received *n* interrupts. If no other activity is reported to processor during the execution of the instruction sequence, what is the number of mode switches and process switches experienced?
(A) $2n$, $2n$ (B) n, n
(C) $2n$, 0 (D) n, 0

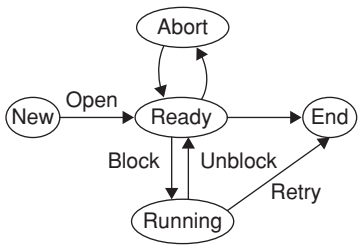
10. Assume a program needs to implement threads, what are the resources that need to be encompassed in critical section?
(i) Global variables
(ii) Local variables
(iii) Static variables
(iv) Function parameters that are passed as reference pointers
(v) Global constants
(A) (i), (ii), (iii), (v)
(B) (i), (iv), (v)
(C) (i), (iii), (v)
(D) (i), (iii), (iv)

11. Which of the following is an appropriate four-state model for a process?

(A)


(B)


(C)


(D)


12. Which of the following are considered as disadvantage of user level thread (ULT)?
(i) Calls made from system will block all threads in a process
(ii) When scheduled in a multiprocessing environment only one thread per process can be executed at a time.
(iii) ULTs cannot communicate with each other in a process.
(iv) The cost of creating a thread is high
(A) (i), (ii), (iii) (B) (i), (iii), (iv)
(C) (i), (ii) only (D) (ii), (iii), (iv)

13. Choose from below, advantages of kernel-level threads:
(i) Kernel can simultaneously schedule multiple threads from the same process on multiple processors.
(ii) Kernel routines can be multithreaded.
(iii) If one thread of a process is blocked then kernel can schedule another thread from the same process.
(A) (i) only (B) (i), (ii)
(C) (i), (ii), (iii) (D) (i), (iii)

14. Assume that part of a program takes long time to execute. Select an option from below that can enhance performance:
(i) Implement the part that takes long time as a separate process and use the results as needed from the main program.
(ii) Implement both the parts as two different threads in the same process.
(A) (i) only
(B) (ii) only
(C) Both (i) and (ii)
(D) Neither (i) nor (ii)

15. What is a kernel-level thread?
(i) Threads that are spawned by OS Kernel
(ii) Threads that are launched by user by directly accessing the kernel
(A) (i) only (B) (ii) only
(C) Both (i), (ii) (D) Neither (i) nor (ii)

Practice Problems 2

Directions for questions 1 to 14: Select the correct alternative from the given choices.

1. Many-to-many multithreading model is used in which of the following operating system?
(A) Windows NT/2000 with Thread Fibre
(B) Windows 95
(C) Windows 98
(D) Solaris Green Threads

2. Which of the following does not interrupt a running process?
(A) Device (B) Timer
(C) Scheduler (D) Power failure

3. Which of the following need not be saved on a context switch between processes?
(A) General purpose registers
(B) Translation look aside buffer
(C) Program counter
(D) All of the above

4. Which of the following actions is/are typically not performed by the OS when switching context from process A to process B?
(A) Saving current register values and restoring the register values for process B
(B) Changing address translation tables
(C) Swapping out the memory image of process A to the disk
(D) Both (B) and (C)

5. For each thread in a multithreaded process, there is a separate
(A) Process control block
(B) User address space
(C) User and kernel stack
(D) Kernel space only

6. When a supervisor call is received
(A) Mode switch happens
(B) Process switch happens
(C) Both (A) and (B)
(D) Neither (A) nor (B)

7. What is the purpose of jacketing?
(A) Convert non-blocking system call to blocking system call
(B) Convert blocking system call to non-blocking system call
(C) Convert blocking system call into a new thread
(D) Convert non-blocking system call into a new thread

8. Which of the following statements is/are always true?
(i) Time taken for mode switch is always greater than process switch.
(ii) Time taken for mode switch is always less than process switch.
(iii) Time taken for mode switch is always equal to process switch.
(A) (i) and (iii) (B) (ii) and (iii)
(C) Only (i) (D) Only (ii)

9. Which of the following is the property of time sharing systems?
(i) Multiple user access
(ii) Multiprogramming
(A) (i) only (B) (ii) only
(C) Both (i) and (ii) (D) Neither (i) nor (ii)

10. Which of the following is/are not a valid reason for process creation?
(i) Created by OS
(ii) Interactive logon
(iii) Privileged instruction
(A) (i), (ii) (B) (ii), (iii)
(C) (i), (iii) (D) (iii) only

11. Which of the following is/are reason(s) for blocking a running process?
(i) A call from the running program to a procedure that is a part of OS code.
(ii) A running process may initiate an I/O operation.
(iii) A user may block a running process.
(A) (i), (ii) only (B) (ii), (iii)only
(C) (i), (iii) only (D) (iii) only

12. If the OS is pre-empting a running process because a higher priority process on blocked/suspend queue has just become unblocked, then the running process moved to _____ queue.
(A) Suspend (B) Ready/suspend
(C) Blocked (D) Blocked/suspend

13. Which of the following is used to call an OS function?
(A) Interrupt (B) Trap
(C) Supervisor call (D) All of these

14. Which of the following is a general component of a thread?
(i) Thread ID
(ii) Register set
(iii) User stack
(iv) Kernel stack
(A) (i), (iii), (iv) (B) (i), (ii), (iv)
(C) (i), (ii), (iii) (D) (i), (ii), (iii), (iv)

Previous Years' Questions

1. Consider the following statements with respect to User-level threads and Kernel-supported threads. **[2004]**

(1) Context switch is faster with Kernel-supported threads
(2) For user-level threads, a system call can block the entire process
(3) Kernel supported threads can be scheduled independently
(4) User level threads are transparent to the Kernel

Which of the above statements are true?

(A) 2, 3 and 4 only (B) 2 and 3 only
(C) 1 and 3 only (D) 1 and 2 only

2. Which one of the following is true for a CPU having a single interrupt request line and a single interrupt grant line? **[2005]**

(A) Neither vectored interrupt nor multiple interrupting devices are possible
(B) Vectored interrupts are not possible but multiple interrupting devices are possible
(C) Vectored interrupts and multiple interrupting devices are both possible
(D) Vectored interrupt is possible but multiple interrupting devices are not possible

3. Normally user programs are prevented from handling I/O directly by I/O instructions in them. For CPUs having explicit I/O instructions, such I/O protection is ensured by having the I/O instructions privileged. In a CPU with memory mapped I/O, there is no explicit I/O instruction. Which one of the following is true for a CPU with memory mapped I/O? **[2005]**

(A) I/O protection is ensured by operating system routine(s)
(B) I/O protection is ensured by a hardware trap
(C) I/O protection is ensured during system configuration
(D) I/O protection is not possible

4. Consider the following code fragment: if (fork ()==0) **[2005]**

```
{ a = a + 5; printf("%d,%d\n",a,&a);}
else{a=a-5;printf("%d,%d\n",a,&a);}
```

Let u, v be the values printed by the parent process, and x, y be the values printed by the child process. Which one of the following is *true*?

(A) $u = x + 10$ and $v = y$
(B) $u = x + 10$ and $v \neq y$
(C) $u + 10 = x$ and $v = y$
(D) $u + 10 = x$ and $v \neq y$

5. Consider the following statements about user-level threads and Kernel-level threads. Which one of the following statements is *false*? **[2007]**

(A) Context switch time is longer for Kernel-level threads than for user level threads.
(B) User level threads do not need any hardware support.
(C) Related Kernel-level threads can be scheduled on different processors in a multi-processor system.
(D) Blocking one Kernel-level thread blocks all related threads.

6. Which of the following statements about synchronous and asynchronous I/O is NOT *true*? **[2008]**

(A) An ISR is invoked on completion of I/O in synchronous I/O but not in asynchronous I/O
(B) In both synchronous and asynchronous I/O, an ISR (Interrupt Service Routine) is invoked after completion of the I/O
(C) A process making a synchronous I/O call waits until I/O is complete, but a process making an asynchronous I/O call does not wait for completion of the I/O
(D) In the case of synchronous I/O, the process waiting for the completion of I/O is woken up by the ISR that is invoked after the completion of I/O

7. A process executes the following code

for (i=0; i<n; i++) fork( );

The total number of child processes created is **[2008]**

(A) n (B) $2^n - 1$
(C) 2^n (D) $2^{n+1} - 1$

8. A CPU generally handles an interrupt by executing an interrupt service routine **[2009]**

(A) As soon as an interrupt is raised.
(B) By checking the interrupt register at the end of fetch cycle.
(C) By checking the interrupt register after finishing the execution of the current instruction.
(D) By checking the interrupt register at fixed time intervals.

9. A thread is usually defined as 'light weight process' because an operating system (OS) maintains smaller data structures for a thread than for a process. In relation to this, which of the following is true? **[2011]**

(A) On per-thread basis, the OS maintains only CPU register state
(B) The OS does not maintain a separate stack for each thread
(C) On per thread basis, the OS does not maintain virtual memory state.
(D) On per thread basis, the OS maintains only scheduling and accounting information.

10. Let the time taken to switch between user and kernel modes of execution be t_1 while the time taken to switch between two processes be t_2. Which of the following is *true*? **[2011]**

(A) $t_1 > t_2$
(B) $t_1 = t_2$
(C) $t_1 < t_2$
(D) Nothing can be said about the relation between t_1 and t_2

11. A process executes the code
fork();
fork();
fork();
The total number of **child** processes created is **[2012]**
(A) 3 (B) 4
(C) 7 (D) 8

12. Which one of the following is **FALSE**? **[2014]**
(A) User level threads are not scheduled by the Kernel.
(B) When a user level thread is blocked, all other threads of its process are blocked.
(C) Context switching between user level threads is faster than context switching between Kernel-level threads.
(D) Kernel-level threads cannot share the code segment.

13. Threads of a process share **[2017]**
(A) global variables but not heap.
(B) heap but not global variables.
(C) neither global variables nor heap.
(D) both heap and global variables.

14. Which of the following is/are shared by all the threads in a process? **[2017]**
I. Program counter
II. Stack
III. Address space
IV. Registers
(A) I and II only (B) III only
(C) IV only (D) III and IV only

Answer Keys

Exercises

Practice Problems 1

1. B	**2.** A	**3.** B	**4.** D	**5.** D	**6.** D	**7.** C	**8.** B	**9.** C	**10.** D
11. A	**12.** C	**13.** C	**14.** C	**15.** A					

Practice Problems 2

1. A	**2.** C	**3.** B	**4.** C	**5.** C	**6.** A	**7.** B	**8.** D	**9.** C	**10.** D
11. A	**12.** B	**13.** C	**14.** D						

Previous Years' Questions

1. A	**2.** C	**3.** A	**4.** D	**5.** D	**6.** A	**7.** B	**8.** C	**9.** C	**10.** C
11. C	**12.** D	**13.** D	**14.** B						