

----- **Chapter 3 – Operating System Structures** -----

Aspects of operating systems:

1. The services it provides.
2. The interface it makes available to users and programmers.
3. Disassembling the system into its components and their interconnections.

System Components:

O.S. Task: Process Management

A process is a program in execution such as: a batch job, timeshared user program, spooling output to a printer, ---.

A process needs certain resources, including: CPU time, memory, files, and I/O devices to accomplish its task. These resources are either given to the process when it is created, or allocated to it while it is running. For example:

A process to display the status of a file on the screen,

- The file name will be given as an input,
- Execute the appropriate instructions and system calls to obtain the desired information,
- Display it on the terminal.

When the process terminates, the O.S. will reclaim any usable resources. A system is a collection of processes:

- Operating System processes those executes system code
- User processes those executes user code.

Regarding the process management, the O.S. is responsible of:

1. Creation and deletion of user and system processes
2. Suspension and resumption of processes.
3. The provision of mechanisms for process synchronization.
4. The provision of mechanisms for process communication.
5. The provision of mechanisms for deadlock handling (detection, avoidance, and correction).

O.S. Task: Main-Memory Management

- Main Memory is an array of addressable words or bytes that is quickly accessible.
- Main Memory is volatile.
- O.S. is responsible for:
 - Allocate and de-allocate memory to processes.
 - Managing multiple processes within memory – keep track of which parts of memory are used by which processes. Manage the sharing of memory between processes.
 - Determining which processes to load when memory becomes available.

O.S. Task: File System Management

File is a collection of related information defined by creator - represents programs and data.

O.S. is responsible for:

- File creation and deletion
- Directory creation and deletion
- Supporting primitives for file/directory manipulation.
- Mapping files to disks (secondary storage).
- Backup files on archival media (tapes).

O.S. Task: Secondary Storage and Input/Output (I/O) Management

- Since primary storage is expensive and volatile, secondary storage is required for backup.
- Disk is the primary form of secondary storage.
 - O.S. performs storage allocation, free-space management and disk scheduling.
- I/O system in the O.S. consists of
 - Buffer caching and management
 - Device driver interface that abstracts device details
 - Drivers for specific hardware devices

O.S. Task: Networking

- Connecting processors in a distributed system
- Distributed System is a collection of processors that do not share memory, peripheral devices or a clock.
- Processors are connected via a communication network.

- Advantages:
 - Allows users and system to exchange information
 - provide computational speedup
 - increased reliability and availability of information

O.S. Task: Protection and Security

- Protection mechanisms control access of programs and processes to user and system resources (files, memory segments, CPU, etc).
 - Protect user from himself, user from other users, system from users.
- Protection mechanisms must:
 - Distinguish between authorized and unauthorized use.
 - Specify access controls to be imposed on use.
 - Provide mechanisms for enforcement of access control.
 - Security mechanisms provide trust in system and privacy.
- Authentication, certification, encryption etc.

Command-Interpreter System

It is the interface between the user and the O.S.

- Some O.S.s include the command interpreter in the kernel.
- Other O.S.s such as: MS-DOS and UNIX, treat the command interpreter as a special program that is running when a job is initiated, or when a user first logs on (e.g., on time-sharing systems).

Many commands are given to O.S. by control statements. When a new job is started in batch system, or a user logs on to a timeshared system, a program called shell that reads and interprets control statements is executed automatically.

Operating System Services

- One set of operating-system services provides functions that are helpful to the user:
 - User interface - Almost all operating systems have a user interface (UI)
 - Varies between Command-Line (CLI), Graphics User Interface (GUI).
 - Program execution - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
 - I/O operations - A running program may require I/O, which may involve a file or an I/O device.

- o File-system manipulation- The file system is of particular interest. Obviously, programs need to read and write files and directories, create and delete them, search them, list file information, and permission management.
- o Communications – Processes may exchange information, on the same computer or between computers over a network.
 - Communications may be via shared memory or through message passing (packets moved by the O.S.)
- o Error detection – O.S. needs to be constantly aware of possible errors
 - May occur in the CPU and memory hardware, in I/O devices, in user program
 - For each type of error, O.S. should take the appropriate action to ensure correct and consistent computing
 - Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system
- Another set of O.S. functions exists for ensuring the efficient operation of the system itself via resource sharing
 - o **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
 - Many types of resources - Some (such as CPU cycles, main memory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code.
 - o **Accounting** - To keep track of which users use how much and what kinds of computer resources
 - o **Protection and security** - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
 - **Protection** involves ensuring that all access to system resources is controlled
 - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts

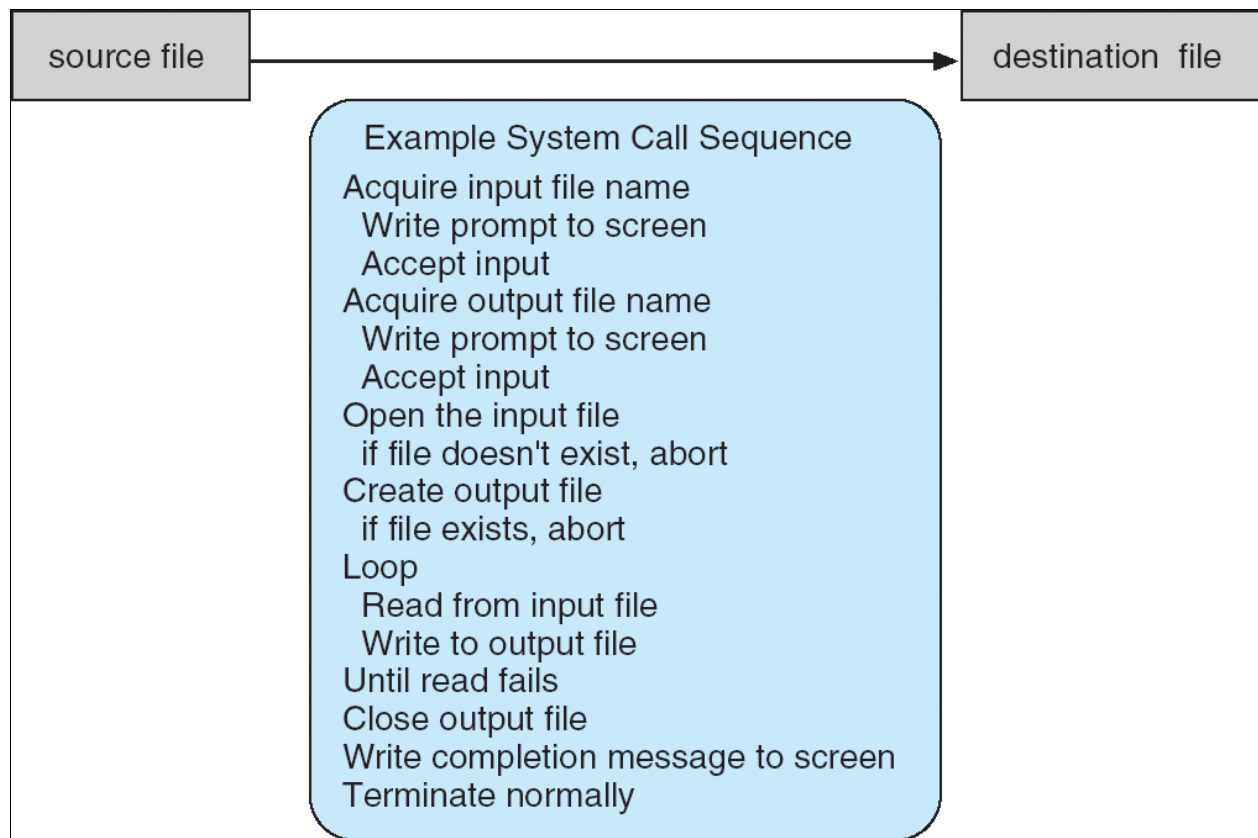
System Calls

- Programming interface to the services provided by the O.S.
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Program Interface (API)** rather than direct system call use

- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)

Example of System Call

System call sequence to copy the contents of one file to another file



----- Chapter 4 : INTERRUPTS AND SPECIAL PURPOSE SYSTEMS -----

Reference: Stallings W., "Operating Systems: Internals and Design Principles", 7th Edition, Pearson Education Limited 2012, ISBN 10:0-273-75150-6.

Process Description and Control

The concept process is fundamental to the structure of modern computer O.Ss in analyzing problems of:

- Synchronization
- Deadlock
- Scheduling in O.Ss.

Most requirements that OS must meet can be expressed with reference to processes:

- The OS must interleave the execution of multiple processes to maximize processor utilization while providing reasonable response time.
- The OS must allocate resources to processes in conformance with a specific policy (e.g., certain functions or applications are of higher priority) while at the same time avoiding deadlock, deadlock occurs if two processes need the same two resources to continue and each has ownership of one. Unless some action is taken, each process will wait indefinitely for the missing resource.
- The OS may be required to support interprocess communication and user creation of processes, both of which may aid in the structuring of applications.

How the O.S. can manage the execution of applications so that:

- Resources are made available to multiple applications.
- The physical processor is switched among multiple applications so all will appear to be progressing.
- The processor and I/O devices can be used efficiently.

Processes and Process Control Blocks

There are several definitions of process:

- A program in execution
- An instance of a program running on a computer
- The entity that can be assigned to and executed on a processor
- A unit of activity characterized by the execution of a sequence of instructions, a current state, and an associated set of system resources
- An entity that consists of a number of elements (program code and a set of data associated with that code)

At any given point in time, while the program is executing, this process can be uniquely characterized by a number of element including those shown in figure-1:

Identifier: to distinguish a process from other processes.
State: if the process is executing, it is in the running state.
Priority: relative to other processes.
Program counter: the address of the next instruction in the program to be executed.
Memory pointers: includes pointers to the program code + data associated with this process + any memory blocks shared with other processes.
Context data: data those are present in registers in the processor while the process is executing.
I/O status information: outstanding I/O requests + I/O devices (e.g., disk drives) assigned to this process + a list of files in use by the process +
Accounting information: the amount of processor time + clock time used + time limit + account numbers +
<ul style="list-style-type: none"> • • • •

Figure-1: Simplified Process Control Block

The information in figure is stored in a data structure, typically called a Process Control Block (PCB) that is created and managed by the O.S. Thus, a process consists of program code and associated data plus a process control block.

Process (task) States

Consider a very simple example. Figure-2 shows a memory layout of 3-processes assuming no use of virtual memory. In addition, there is a small **dispatcher** program that switches the processor from on process to another.

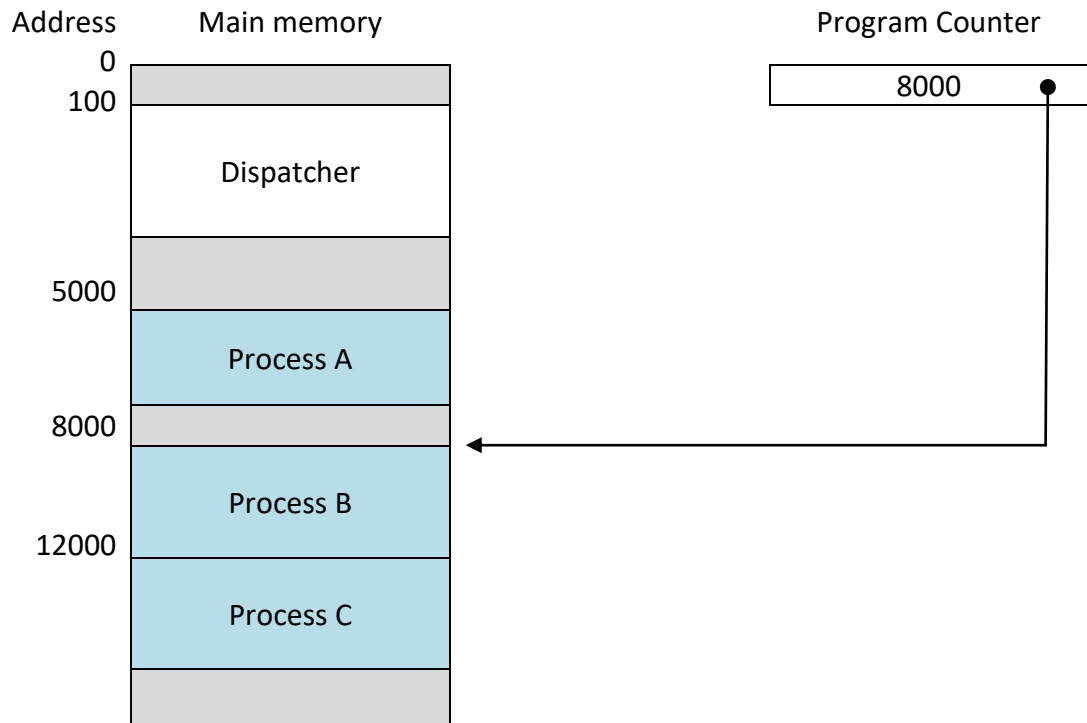


Figure-2: Snapshot of example execution (Figure 4) at Instruction Cycle 13

5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

(a) Trace of process A (b) Trace of process B (c) Trace of process C

5000 = Starting address of program of process A

8000 = Starting address of program of process B

12000 = Starting address of program of process C

Figure-3: Trace of Process of Figure 2

1	5000		27	12004	
2	5001		28	12005	
3	5002		-----	-----	Time-out
4	5003		29	100	
5	5004		30	101	
6	5005		31	102	
-----	-----	Time-out	32	103	
7	100		33	104	
8	101		34	105	
9	102		35	5006	
10	103		36	5007	
11	104		37	5008	
12	105		38	5009	
13	8000		39	5010	
14	8001		40	5011	
15	8002		-----	-----	Time-out
16	8003		41	100	
-----	-----	I/O request	42	101	
17	100		43	102	
18	101		44	103	
19	102		45	104	
20	103		46	105	
21	104		47	12006	
22	105		48	12007	
23	12000		49	12008	
24	12001		50	12009	
25	12002		51	12010	
26	12003		52	12011	
			-----	-----	Time-out

100 = Starting address of dispatcher program

Shaded areas indicate execution of dispatcher process;

First and third columns count instruction cycles;

Second and fourth columns show address of instruction being executed;

Figure 4: Combined Trace of Processes of Figure 2

Figure 4 shows the interleaved traces resulting from the first 52 instruction cycles. The shaded areas represent code executed by the dispatcher. The same sequence of instructions is executed by the dispatcher in each instance because the same functionality of the dispatcher is allows a process to continue execution for a maximum of six instruction cycles, after which it is interrupted;

The Two-State Process Model

The OS's principal responsibility is the execution of processes; this includes:

- Determining the interleaving pattern for execution
- Allocating resources to process

The first step in designing an OS to control processes is to describe the behavior that we would like the processes to exhibit.

We can construct the simplest possible model by observing that, at any time, a process is either being executed by a processor or not. In this model, a process may be in one of two states: Running or Not Running, as shown in Figure 5(a). When the OS creates a new process, it creates

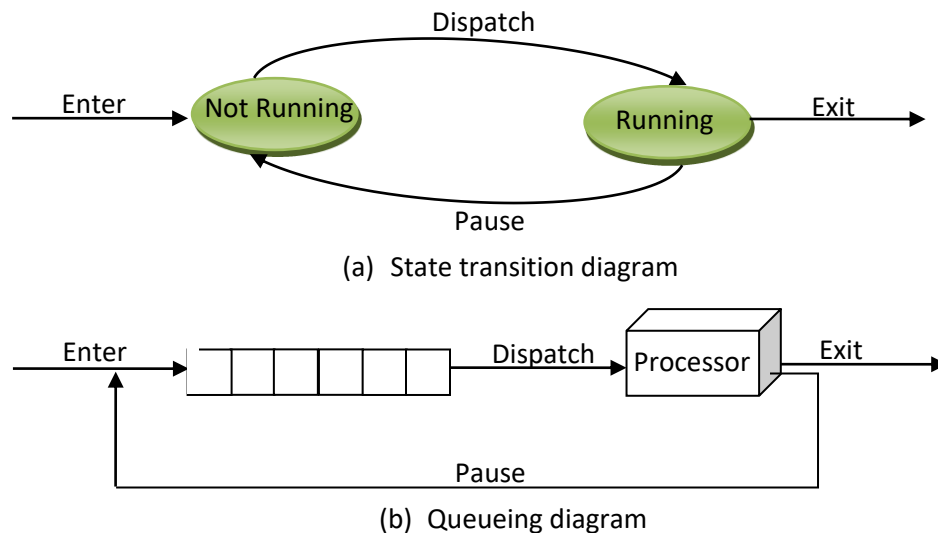


Figure 5: Two-State Process Model

a process control block 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.

Figure 5(b) describes the behavior of the dispatcher in terms of this queueing diagram. A process that is interrupted is transferred to the queue of waiting processes. Alternatively, if the process has completed or aborted, it is discarded (exits the system). In either case, the dispatcher takes another process from the queue to execute.

The Creation and Termination of Processes

The life of a process is bounded by its creation and termination.

Table-1: Reasons for Process Creation

New batch job	The OS is provided with batch job control stream, usually on tape or disk. When the OS is prepared to take on new work, it will read the next sequence of job control commands.
Interactive log-on	A user at a terminal logs on to the system.
Created by OS to provide a service	The OS can create a process to perform a function on behalf of a user program, without the user having to wait (e.g., a process to control printing).
Spawned (تنتج) by existing process	For purposes of modularity or to exploit parallelism, a user program can dictate the creation of a number of processes. 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.

Table-2: Reasons for Process Termination

Normal completion	The process executes an OS service call to indicate that it has completed running.
Time limit exceeded	Include total elapsed time ("wall clock time"), amount of time spent executing, and, in the case of an interactive process, the amount of time since the user last provided any input.
Memory unavailable	Process requires more memory than the system can provide.
Bounds violation	The process tries to access a memory location that it is not allowed to access.
Protection error	e.g., use a resource such as a file that it is not allowed to use, or writing to a read only file.
Arithmetic error	Such as: Division by zero or tries to store numbers larger than the H/W can accommodate.
Time overrun	The process has waited longer than a specified maximum for a certain event to occur.
I/O failure	Such as: inability to find a file, failure to read or write after a specified maximum number of tries, or invalid operation (such as reading from the line printer).
Invalid instruction	The process attempts to execute a nonexistent instruction.
Privileged instruction	Attempts to use an instruction reserved for the OS.
Data misuse	A piece of data is of the wrong type or is not initialized.
Operator or OS intervention	E.g., if a deadlock exists.
Parent termination	When a parent terminates, the OS may automatically terminate all of the offspring of that parent.
Parent request	A parent process typically has the authority to terminate any of its offspring.

A five-State Model

The queue in the two state figure 5 is a FIFO list and the processor operates in **round-robin** fashion on the available processes (each process in the queue is given a certain amount of time returned to the queue, unless blocked waiting for an I/O operation to complete). Thus using a single queue, the dispatcher could not just select the process at the oldest end of the queue.

Rather the dispatcher would have to scan the list looking for the process that is not blocked and that has been in the queue the longest.

- To handle this situation is to split the Not Running state into two states: Ready and Blocked as shown in figure 6.

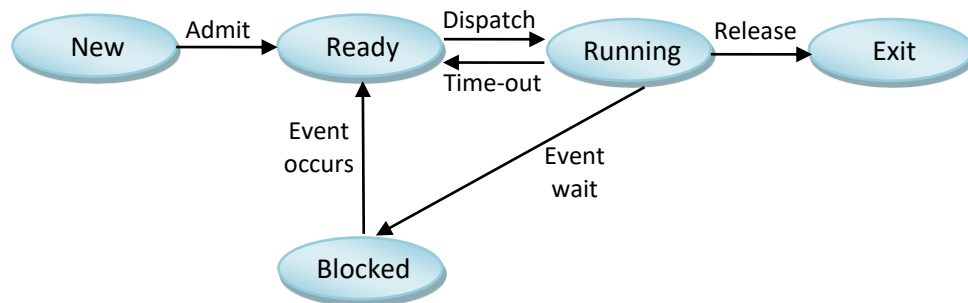


Figure 6: Five-State Process Model

The five states in this new diagram are:

- **Running:** the process that is currently being executed. Assuming a computer with a single processor, so at most one process at a time can be in this state.
- **Ready:** a process that is prepared to execute when given the opportunity.
- **Blocked/Waiting:** a process that cannot execute until some event occurs, such as the completion of an I/O operation.
- **New:** A process that has just been created but has not yet been admitted to the pool of executable processes by the OS. Typically, a new process has not yet been loaded into main memory, although its process control block has been created.
- **Exit:** a process that has been released from the pool of executable processes by the OS, either because it halted or because it aborted for some reason.

The possible transitions are as follows:

- Null → New
- New → Ready
- Ready → Running
- Running → Exit
- Running → Ready
- Running → Blocked
- Blocked → Ready
- Ready → Exit
- Blocked → Exit

Figure 7 shows the transition of each process among the states.

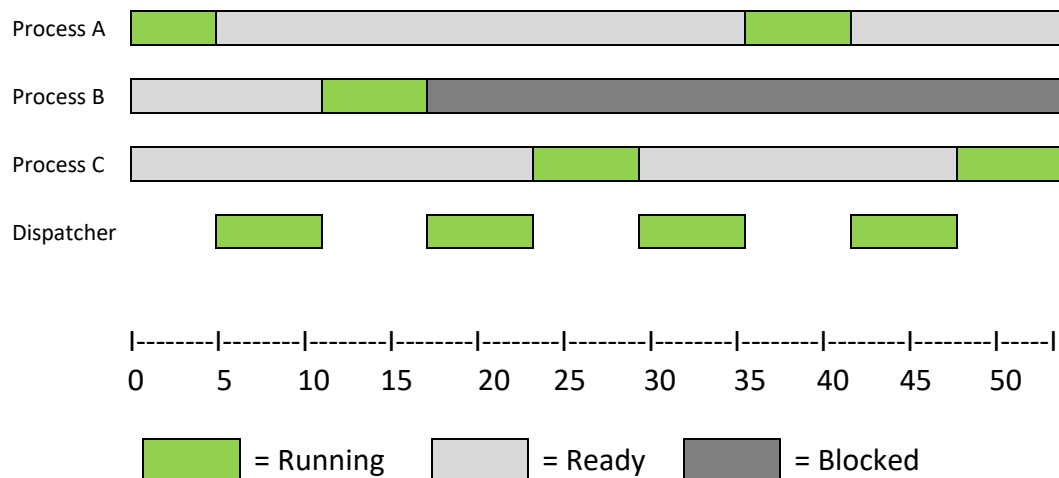


Figure 7: Process States for the Trace of Figure-4

Figure 8 suggests the way in which a queuing discipline might be implemented with two queues: a Ready queue and a Blocked queue. As each process is admitted to the system, it is placed in the Ready queue. When it is time for the OS to choose another process to run, it selects one from the Ready queue. In the absence of any priority scheme, this can be a simple FIFO queue. When a running process is removed from execution, it is either terminated or placed in the Ready or Blocked queue, depending on the circumstances. Finally, when an event occurs, any process in the Blocked queue that has been waiting on that event only is removed to the Ready queue.

This latter arrangement means that, when an event occurs, the OS must scan the entire blocked queue, searching for those processes waiting on that event. In a large OS, there could be hundreds or even thousands of processes in that queue. Therefore, it would be more efficient to have a number of queues, one for each event. Then, when the event occurs, the entire list of processes in the appropriate queue can be moved to the Ready state (Figure 8b).if the dispatching of processes is dictated by a priority scheme, and then it would be convenient to have a number of Ready queues, one for each priority level. The OS could then readily determine which is the highest-priority ready process that has been waiting the longest.

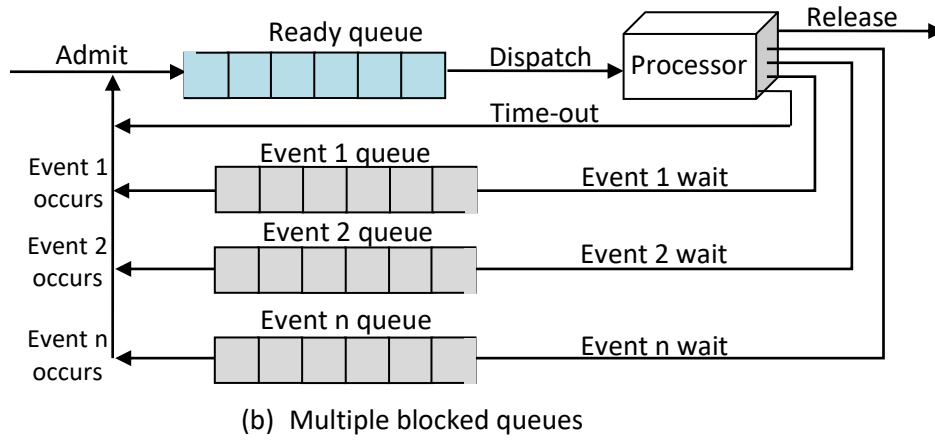
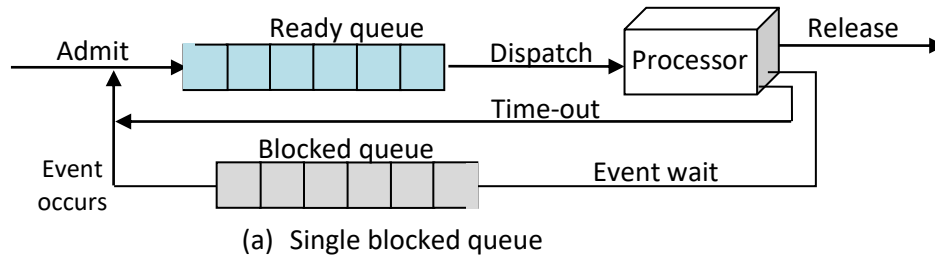


Figure 8: Queueing Model for Figure 6