OS-2

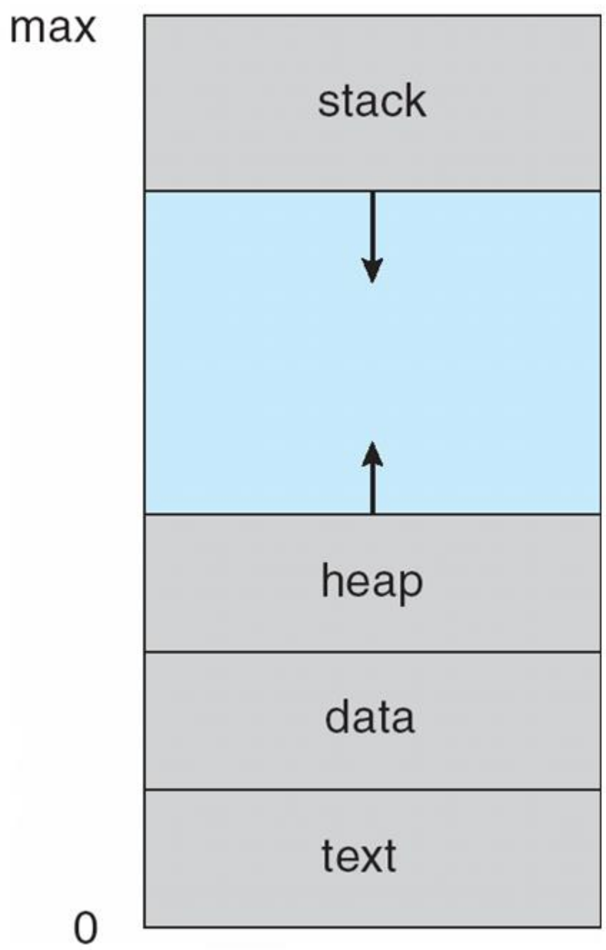
What is Process!

- A process is a program in execution. Process is not as same as program code but a lot more than it.
- A process is an 'active' entity as opposed to program which is considered to be a 'passive' entity.

Process memory is divided into four sections for efficient working :

- The **Text section** is made up of the compiled program code, read in from non-volatile storage when the program is launched.
- The **Data section** is made up the global and static variables, allocated and initialized prior to executing the main.
- The **Heap** is used for the dynamic memory allocation, and is managed via calls to new, delete, malloc, free, etc.
- The **Stack** is used for local variables. Space on the stack is reserved for local variables when they are declared.

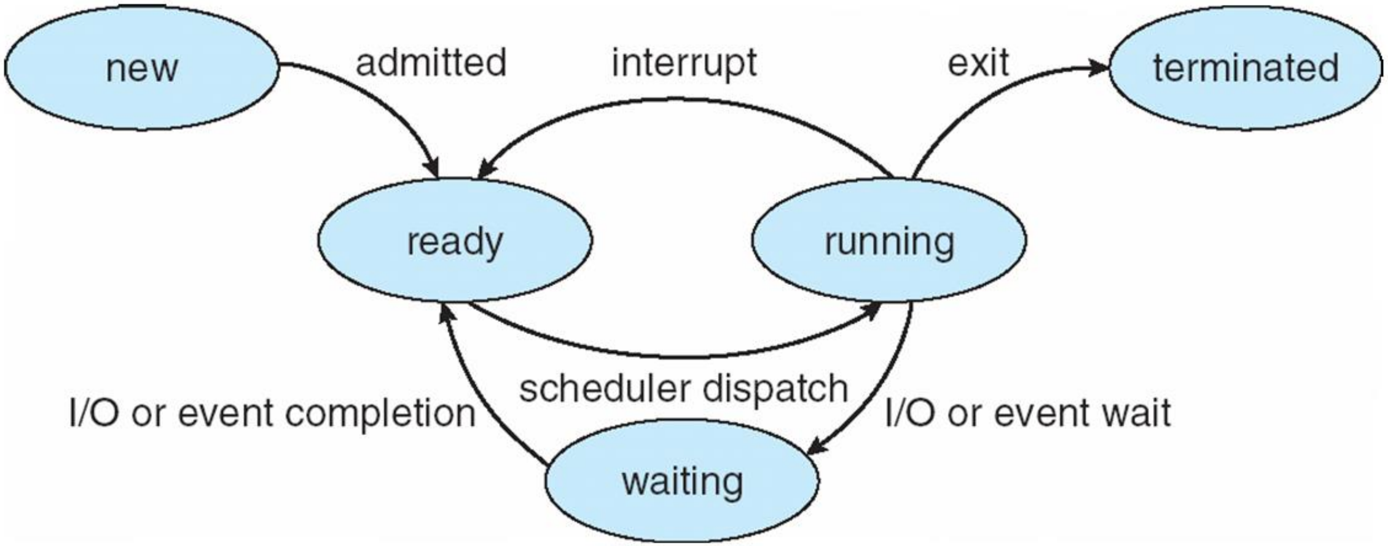
Process in Memory



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(such as an I/O completion or reception of a signal).
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution

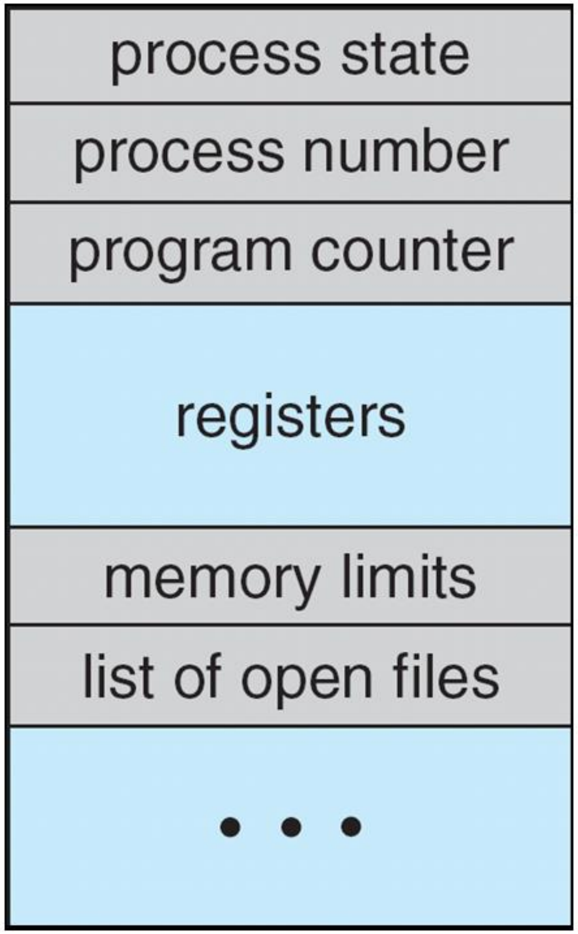
Diagram of Process State



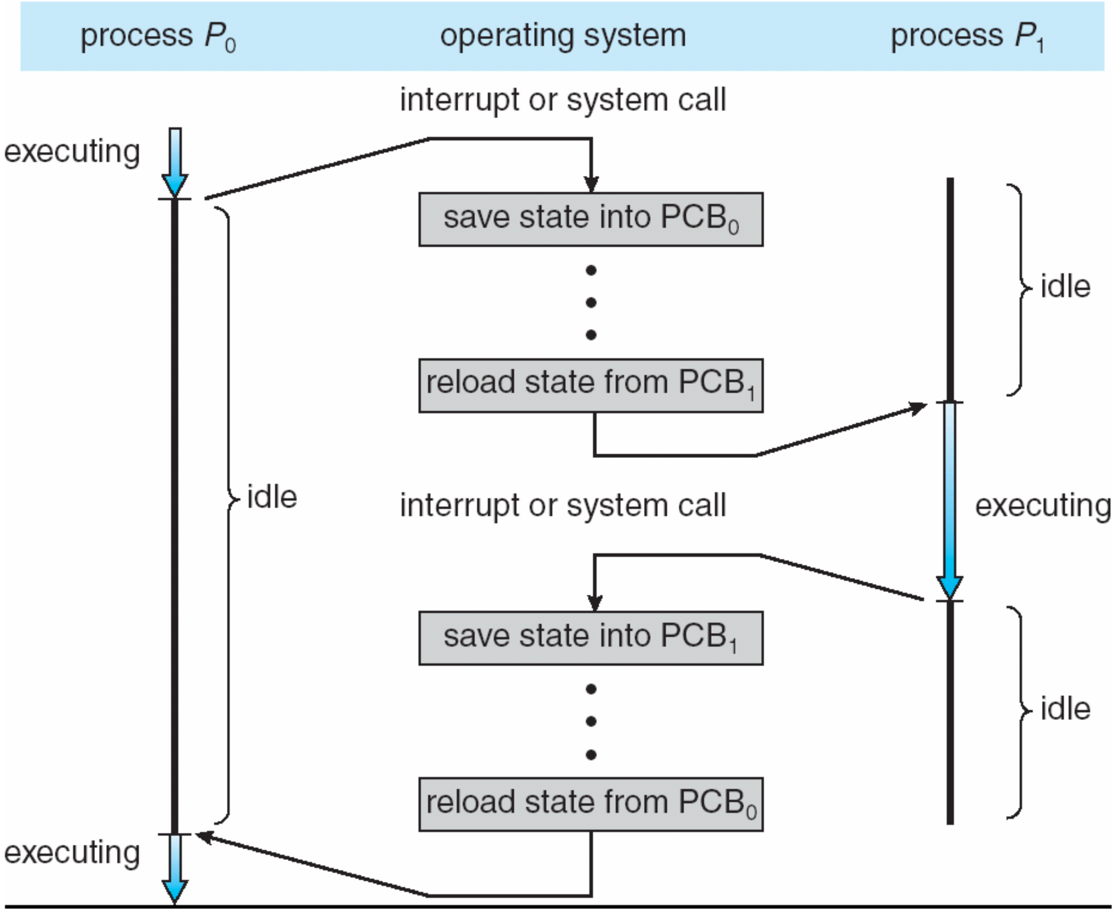
Process Control Block (PCB)

There is a Process Control Block for each process, enclosing all the information about the process. It is a data structure, which contains the following

- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files



CPU Switch From Process to Process



What is Process Scheduling?

The act of determining which process is in the **ready** state, and should be moved to the **running** state is known as **Process Scheduling**.

Scheduling fell into one of the two general categories:

- **Non Pre-emptive Scheduling:** When the currently executing process gives up the CPU voluntarily.
- **Pre-emptive Scheduling:** When the operating system decides to favour another process, pre-empting the currently executing process.

Process Scheduling

- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler** selects among available processes for next execution on CPU
- Maintains **scheduling queues** of processes
  - **Job queue** – set of all processes in the system
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - **Device queues** – set of processes waiting for an I/O device
  - Processes migrate among the various queues

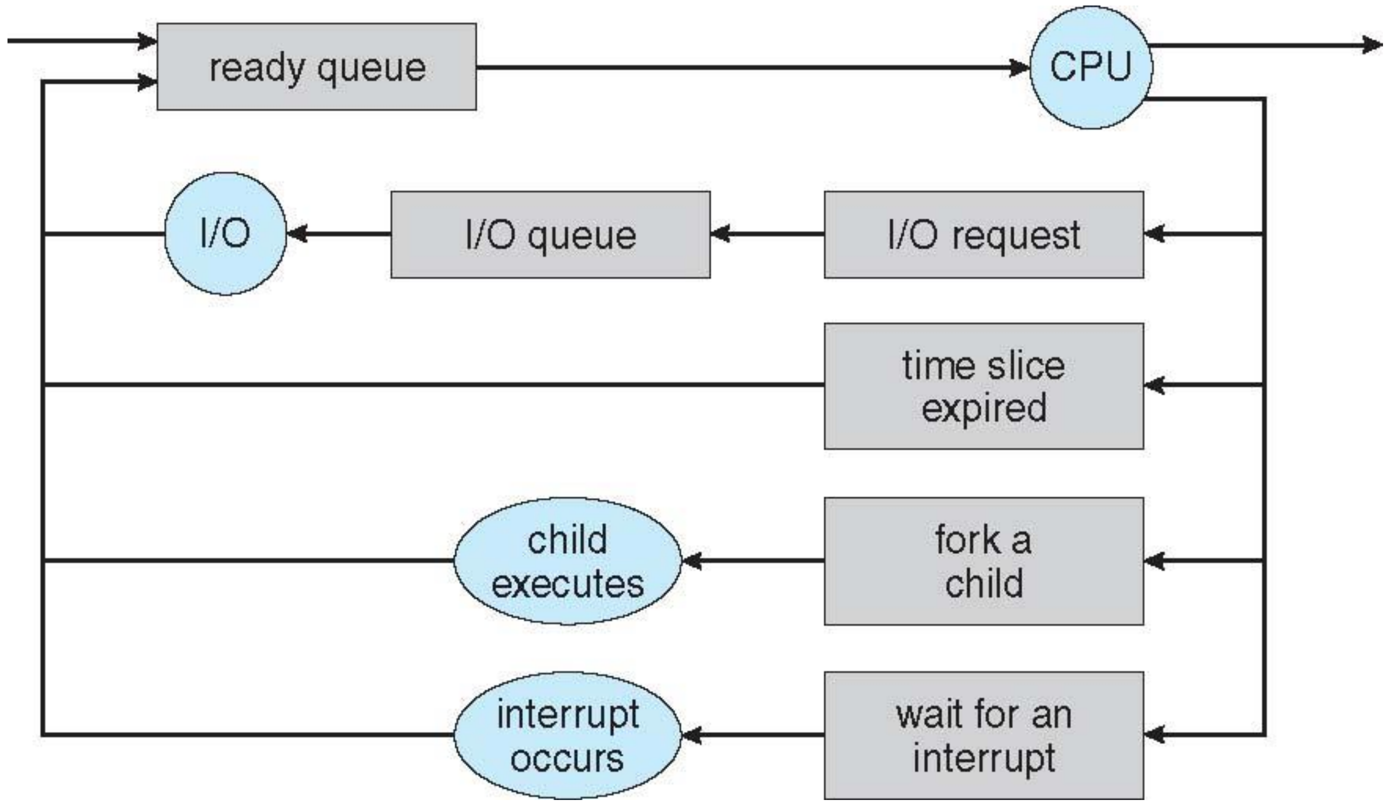
Process Scheduling

- A new process is initially put in the **Ready queue**. It waits in the ready queue until it is selected for execution(or dispatched). Once the process is assigned to the CPU and is executing, one of the following several events can occur:
- The process could issue an I/O request, and then be placed in the **I/O queue**.
- The process could create a new subprocess and wait for its termination.
- The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.

In the first two cases, the process eventually switches from the waiting state to the ready state, and is then put back in the ready queue. A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources deallocated.

Representation of Process Scheduling

- **Queueing diagram** represents queues, resources, flows



Schedulers

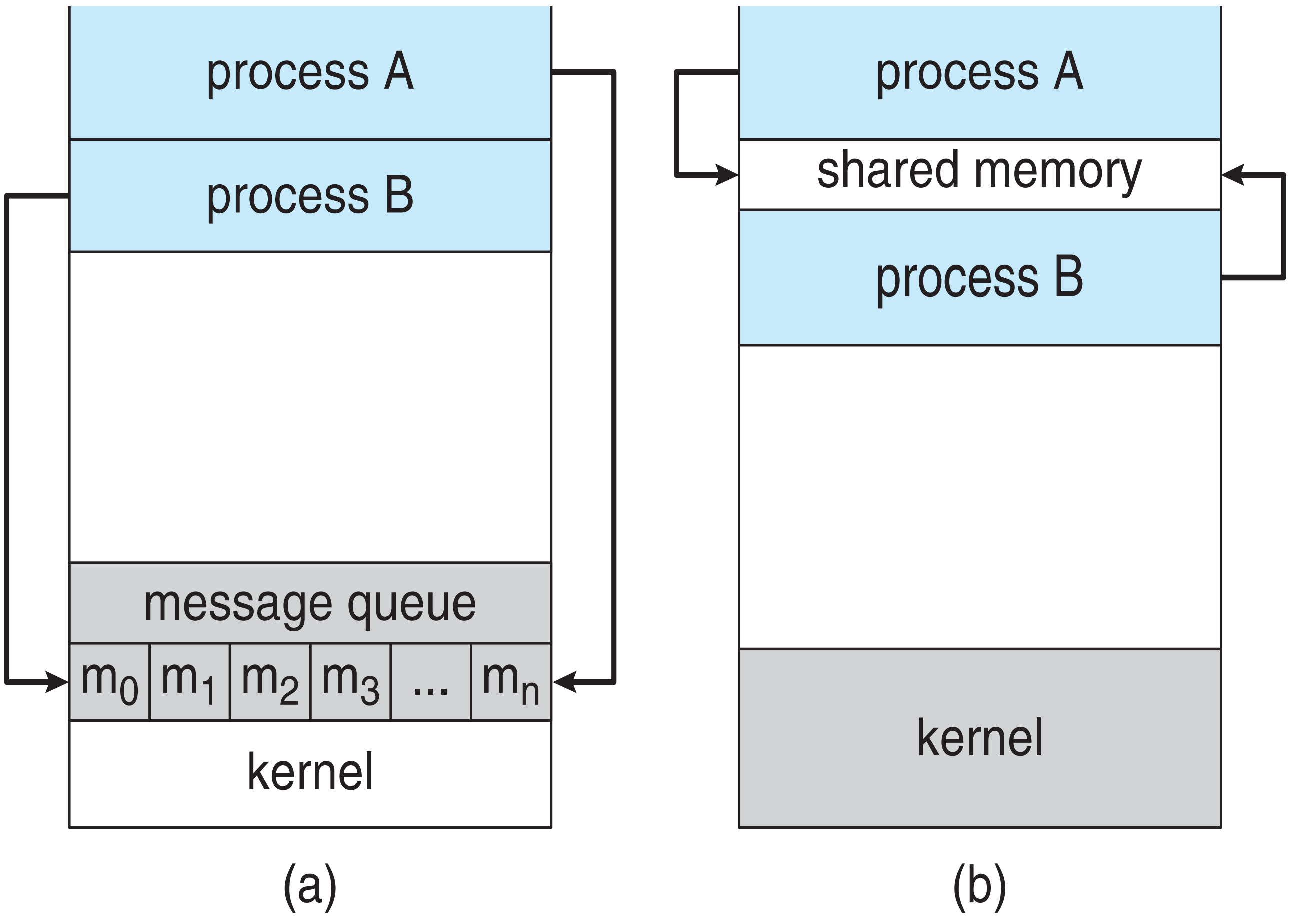
- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked frequently (milliseconds) $\Rightarrow$ (must be fast)
- **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into the ready queue
  - Long-term scheduler is invoked infrequently (seconds, minutes) $\Rightarrow$ (may be slow)
  - 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 very long CPU bursts
- Long-term scheduler strives for good ***process mix***

Interprocess Communication

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

Communications Models

(a) Message passing. (b) shared memory.



Cooperating Processes

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

Interprocess Communication – Shared Memory

- An area of memory shared among the processes that wish to communicate
- The communication is under the control of the users processes not the operating system.
- Major issues is to provide mechanism that will allow the user processes to synchronize their actions when they access shared memory.
- Synchronization is discussed in great details in Chapter 5.

Interprocess Communication – Message Passing

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

Message Passing (Cont.)

- If processes *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:
    - Shared memory
    - Hardware bus
    - Network
  - Logical:
    - Direct or indirect
    - Synchronous or asynchronous
    - Automatic or explicit buffering

Classical IPC Problems

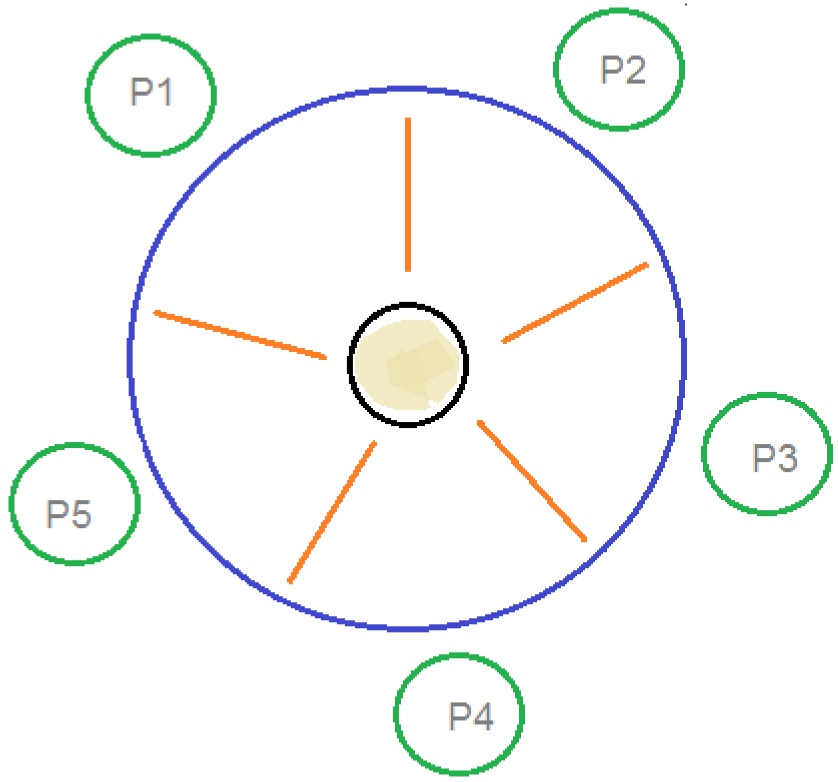
Dining Philosophers Problem

Total 5 philosophers are seated in a round table. On left and right to each philosophers is one fork/chopstick placed.

- When a philosopher thinks, he does not interact with his others.
- From time to time, a philosopher gets hungry and tries to pick up the two forks that are closest to him (the forks that are between him and his left and right neighbors).
- A philosopher may pick up only one fork at a time. Obviously, he cannot pick up a fork that is already in the hand of a neighbor.
- When a hungry philosopher has both his forks at the same time, he eats without releasing his forks.
- When he is finished eating, he puts down both of his forks and starts thinking again.

The problem is to design a protocol to satisfy the liveness condition: *any philosopher who tries to EAT, eventually does.*

Cont…



Cont…

- **Readers Writer Problem**

Readers writer problem is another example of a classic synchronization problem.

<u>The Problem Statement</u>

There is a shared resource which should be accessed by multiple processes. There are two types of processes in this context. They are **reader** and **writer**. Any number of **readers** can read from the shared resource simultaneously, but only one **writer** can write to the shared resource. When a **writer** is writing data to the resource, no other process can access the resource. A **writer** cannot write to the resource if there are non zero number of readers accessing the resource at that time.