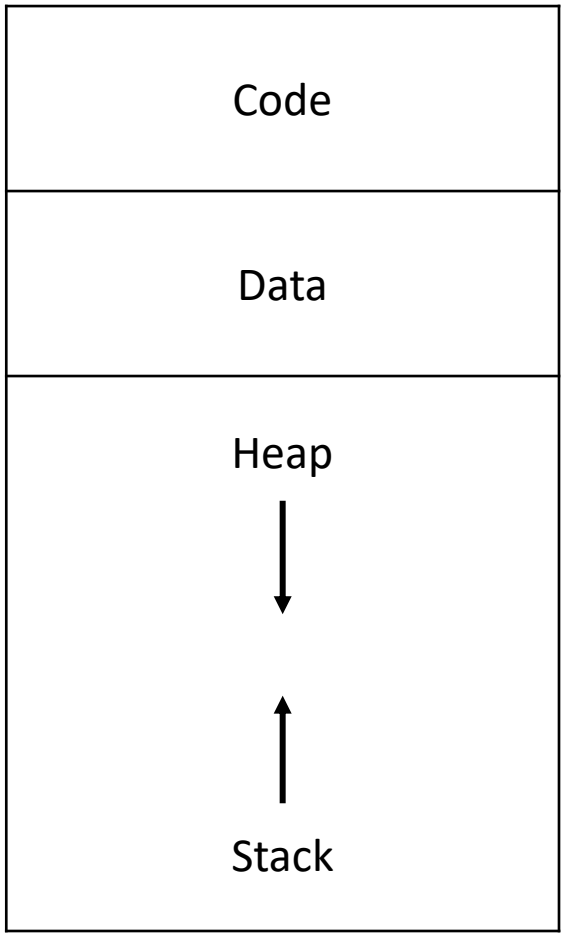
Process, Threads & Inter-process Communication

Saroj Shivagunde

Overview

- Process
  - Concept
  - Memory Allocation for a process
  - Scheduling Queues
  - Life Cycle of a Process
  - Process Control Block
- Inter process communication
  - Shared-Memory Systems
  - Message-Passing Systems
- Threads

Memory Allocation for a Process



Contains program code. Also called as text part

Contains global variables

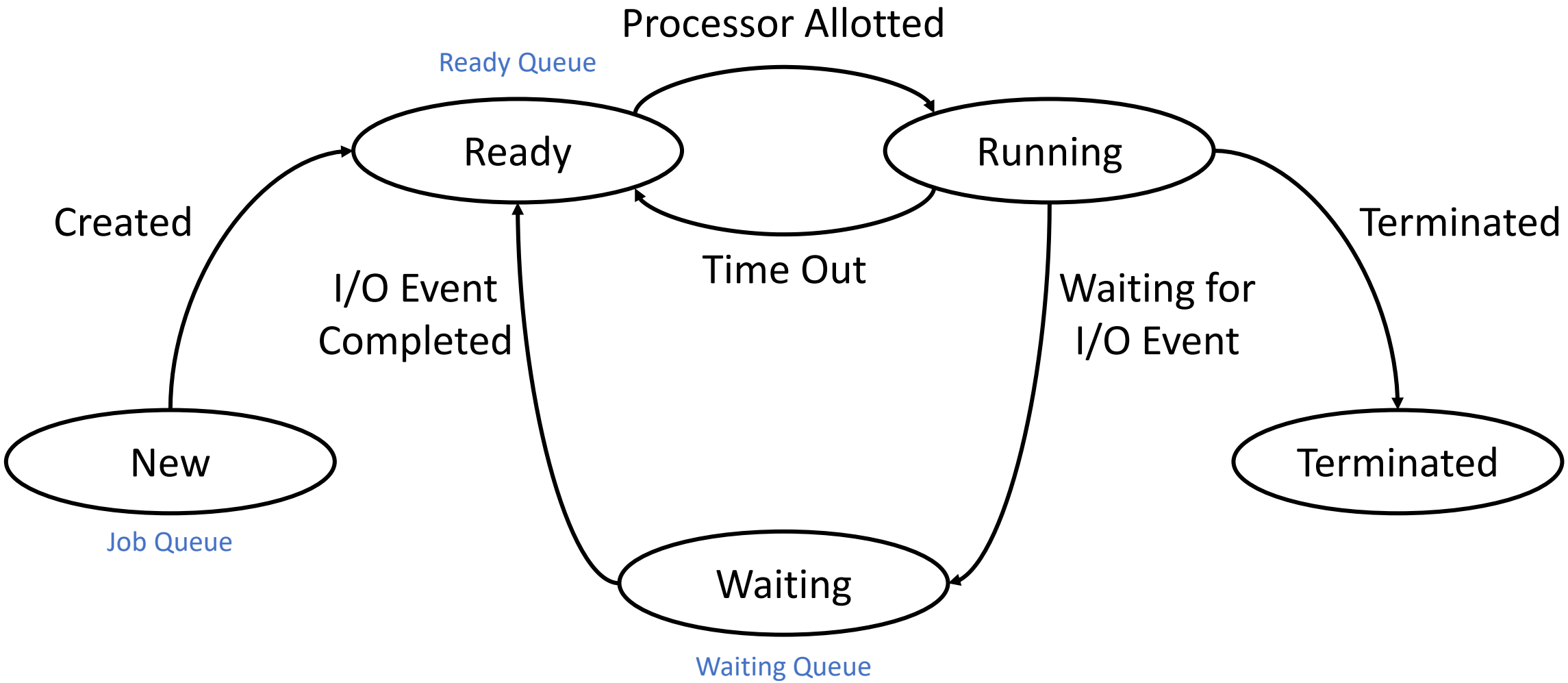
For dynamic allocation during runtime

Contains temporary data such as function parameters, return addresses, and local variables

Scheduling Queues

- Job Queue
  - A newly created process joins the job queue
- Ready Queue
  - Ready to run processes are kept in the ready queue
  - Scheduler dispatches processes from this to the CPU in accordance with the scheduling algorithm
- Device Queue
  - Each I/O device has its own queue. Processes must wait for access if another process is using the device

Life Cycle of a Process



Process Control Block

Process State	Current state of the process
Process Privileges	Access privileges
Process ID	ID of the process
Program Counter	Address of the next instruction to be executed
CPU Registers	Values of the CPU registers
Scheduling Information	Priority value, pointer to the scheduling queue
Memory Management Information	Values of base & limit registers, paging/segmentation Information
Accounting Information	CPU or real time, time limits, job/process numbers
I/O Information	list of allocated I/O devices, list of open files
⋮	Any other information

Inter-Process Communication

- Independent Processes
  - Processes that do not depend on other processes for completion of their tasks
- Cooperative Processes
  - Processes that depend on other processes for completion of their tasks
  - Such processes share data with other processes
- Reasons for Process Cooperation
  - Information Sharing
  - Computation Speedup
  - Modularity
  - Convenience

Inter-Process Communication

- Shared Memory
  - A region of memory is allocated as the shared region between the cooperating processes
  - Each process can access and modify the data in this region
  - This communication type is used for sharing large amount of data
  - Faster as system calls are required only while creation of the shared region
  - Performs poorly on multi-core systems due to cache coherency issue
- Message Passing
  - Communication between the processes happens via messages
  - Each process can send a message containing data to another process
  - This communication type is used for sharing small amount of data
  - Slower as each message requires invocation of system calls
  - Is better than shared memory on multi-core systems
  - Useful for distributed systems

Shared Memory Systems

- Shared memory systems can be thought of as producer-consumer systems
- A producer process produces/provides data or files and the consumer process consumes that data
- Shared memory region in this system is treated as a buffer
  - **Bounded Buffers**- have bounds on the size of the buffer. A producer must wait if the buffer is full, and the consumer may have to wait if the buffer is empty
  - **Unbounded Buffers**- have no bound on the size of the buffer. A producer can keep on producing new items, but a consumer may have to wait for a new item
- Useful for the systems that reside on the same CPU

Message-Passing Systems

- Message passing systems provide at least two functionalities- Send Message and Receive Message
- Messages can be fixed or variable in size
  - Fixed size messages are easier for OS to implement, but make programming difficult
  - Variable size messages are complex for OS to implement, but make programming easier
- Communication Link Types for Message Passing
  - Direct (symmetric/asymmetric) or indirect communication (Process mailbox/OS mailbox)
  - Synchronous or asynchronous communication (blocking/non-blocking send/receive)
  - Buffering (Zero/Bounded/Unbounded)

Threads

- A thread is a subset of a process that has an ID, program counter, registers and a stack, and shares other resources (code, data, files, etc.) with the parent process
- Thread is created with `fork()` system call
- Benefits of Using Threads-
  - Responsiveness
  - Resource sharing
  - Economy in Creation and Context-switch
  - Scalability
- A thread can be a user thread, or a kernel thread. Their relationships are-
  - Many-to-One (Many user threads mapped to one kernel thread)
  - One-to-One (One user thread mapped to one kernel thread)
  - Many-to-Many (Many user threads mapped to many kernel threads)

End