Chapter 3:
Process Concept

COURSE INSTRUCTORS:

DR. MUHAMMAD NASEEM, ENGR. FARHEEN QAZI, MS. FALAK SALEEM

COURSE : OPERATING SYSTEMS (SWE-204)

BATCH : 2019 FALL : 2020

DEPARTMENT OF SOFTWARE ENGINEERING

SIR SYED UNIVERSITY OF ENGINEERING & TECHNOLOGY

1

Today's Agenda

- Process Concept
- Process Scheduling
- Operations on Processes
- Interprocess Communication
- Examples of IPC Systems

2

Objectives

- To introduce the notion of a process -- a program in execution, which forms the basis of all computation
- To describe the various features of processes, including scheduling, creation and termination, and communication
- To explore interprocess communication using shared memory and message passing

3

Process Concept

- An operating system executes a variety of programs:
  - Batch system – **jobs**
  - Time-shared systems – **user programs** or **tasks**
- Textbook uses the terms ***job*** and ***process*** almost interchangeably
- **Process** – a program in execution; process execution must progress in sequential fashion
- Multiple parts
  - The program code, also called **text section**
  - Current activity including **program counter**, processor registers
  - **Stack** containing temporary data
    - Function parameters, return addresses, local variables
  - **Data section** containing global variables
  - **Heap** containing memory dynamically allocated during run time

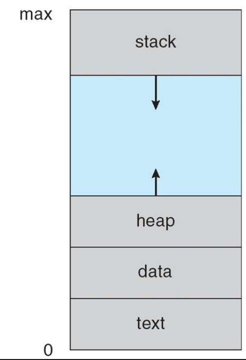
4

Process Concept (Cont.)

- Program is ***passive*** entity stored on disk (**executable file**), process is ***active***
  - Program becomes process when executable file loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc
- One program can be several processes
  - Consider multiple users executing the same program

5

Process in Memory



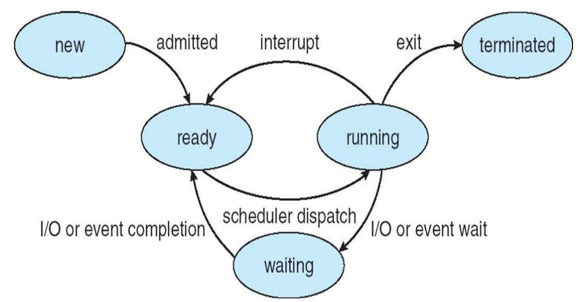
6

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

7

Five State Process Model



8

Process State Transition

- New → Ready: OS ready to schedule the new process.
- Ready → Running: OS Scheduler selects one of the processes in the ready queue to run.
- Running → Exit: Process notifies OS to exit or abort.
- Running → Ready: Processes has reached its quantum and OS uses scheduling algorithm to find the next process in Ready state. Process can also release the processor.
- Running → Blocked: Process issues a request and must wait for the event.
- Blocked → Ready: Event for which the process is waiting has occurred.
- Ready → Exit: Parent process terminates a child process. Parent process terminates and all child processes also terminates.

9

Seven State Model

10

Suspended state and swapping

- **Suspended** : Another process has explicitly told this process to sleep. It will be awakened when a process explicitly awakens it.
- So far, all the processes had to be (at least partly) in main memory
- The OS may need to suspend some processes, i.e: to swap them out to disk. We add 2 new states:
- **Blocked Suspend**: blocked processes which have been swapped out to disk
- **Ready Suspend**: ready processes which have been swapped out to disk

11

Suspended Processes

- Processor is faster than I/O so all processes could be waiting for I/O
  - Swap these processes to disk to free up more memory and use processor on more processes
- Blocked state becomes ***suspend*** state when swapped to disk
- Two new states
  - Blocked/Suspend
  - Ready/Suspend

12

New state transitions (mid-term scheduling)

- Blocked --> Blocked Suspend
  - When all processes are blocked, the OS will make room to bring a ready process in memory
- Blocked Suspend --> Ready Suspend
  - When the event for which it has been waiting occurs (state info is available to OS)
- Ready Suspend --> Ready
  - when no more ready process in main memory or process has higher priority than other Ready processes.
- Ready--> Ready Suspend (unlikely)
  - OS needs to free up Main Memory for current process or next scheduled process.

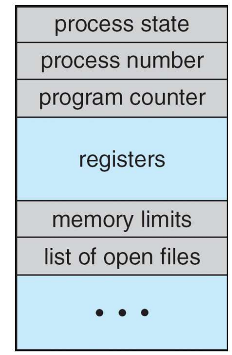
13

Process Control Block (PCB)

Information associated with each process

(also called **task control block**)

- 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



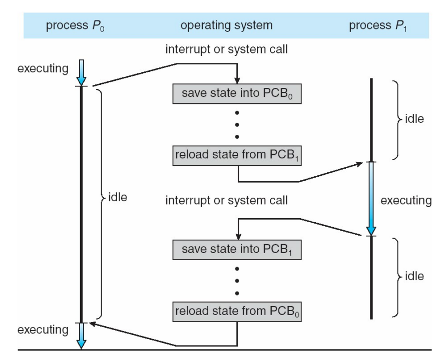
14

Context Switch

- When CPU switches to another process, the system must **save the state** of the old 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
  - The more complex the OS and the PCB ➔ the longer the context switch
- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU ➔ multiple contexts loaded at once

15

CPU Switch From Process to Process



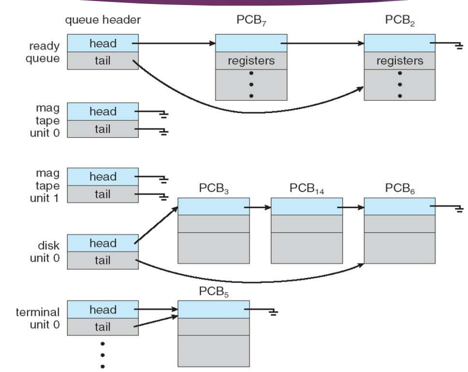
16

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

17

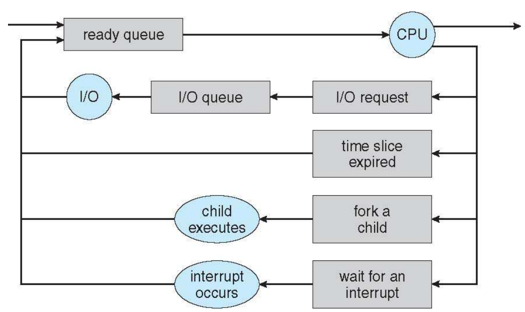
Ready Queue And Various I/O Device Queues



18

Representation of Process Scheduling

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



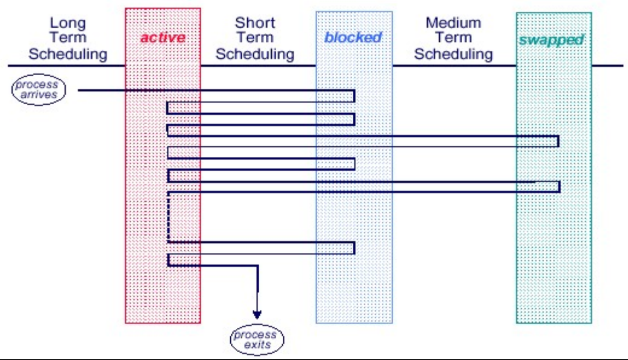
19

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) ⇒ (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) ⇒ (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***

20

Life cycle of a typical process



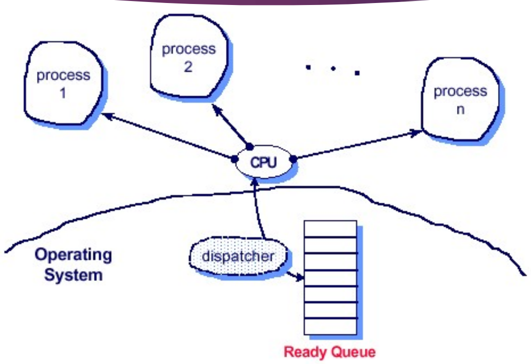
21

Dispatcher (short-term scheduler)

- Swaps processes out to secondary storage.
- It prevents a single process from monopolizing processor time.
- It decides who goes next according to a scheduling algorithm. (chapter 6)
- The CPU will always execute instructions from the dispatcher while switching from process A to process B.

22

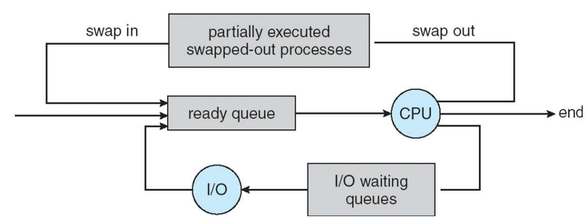
Dispatcher at Work



23

Addition of Medium Term Scheduling

- **Medium-term scheduler** can be added if degree of multiple programming needs to decrease
  - Remove process from memory, store on disk, bring back in from disk to continue execution: **swapping**



24

Operations on Processes

- System must provide mechanisms for:
  - process creation,
  - process termination,
  - and so on as detailed next

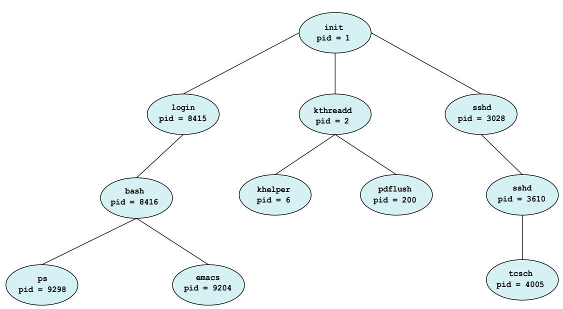
25

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 options
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution options
  - Parent and children execute concurrently
  - Parent waits until children terminate

26

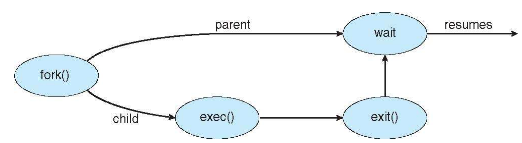
A Tree of Processes in Linux



27

Process Creation (Cont.)

- 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 space with a new program



28

Process Termination

- Process executes last statement and then asks the operating system to delete it using the `exit()` system call.
  - Returns status data from child to parent (via `wait()`)
  - Process' resources are deallocated by operating system
- Parent may terminate the execution of children processes using the `abort()` system call. Some reasons for doing so:
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - The parent is exiting and the operating systems does not allow a child to continue if its parent terminates

29

Process Termination

- Some operating systems do not allow child to exists if its parent has terminated. If a process terminates, then all its children must also be terminated.
  - **cascading termination.** All children, grandchildren, etc. are terminated.
  - The termination is initiated by the operating system.
- The parent process may wait for termination of a child process by using the `wait()` system call. The call returns status information and the pid of the terminated process

```
pid = wait(&status);
```

- If no parent waiting (did not invoke `wait()`) process is a **zombie**
- If parent terminated without invoking `wait`, process is an **orphan**

30

Interprocess Communication

- **Independent** process cannot affect or be affected by the execution of another process
- **Cooperating** process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
  - ***Information sharing*** (ex.: shared file)
  - ***Computation speedup*** (break up process into sub tasks to run faster and can be achieved only if the computer has multiple processing elements – CPUs or I/O channels)
  - ***Modularity*** (dividing system functions into separate processes or threads)
  - ***Convenience*** (individual user may work on many tasks at the same time could be editing, printing, and compiling in parallel)

31

Interprocess Communication

- Mechanism for processes to communicate and to synchronize their actions
- Two models of IPC:

1) shared memory

cooperating processes exchange information by reading and writing data to a shared region of memory.

* allows maximum speed and convenience of communication.
* faster than message passing (system calls only to establish the region. All accesses are routine memory accesses, no assistance from the kernel).

32

Contd…

2) message passing

messages are exchanged between the cooperating processes

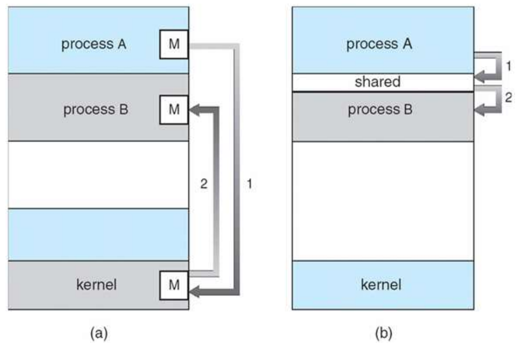
→useful for exchanging smaller amounts of data.

→easier to implement than is shared memory for intercomputer communications.

→implemented using system calls (more time, kernel intervention).

33

Communications Models



34

1. Shared memory system

- IPC using shared memory requires communicating processes to establish region of shared memory.
- The region resides in the address space of the process creating the shared memory segment.
- Shared memory requires that two or more processes agree to remove the restriction.
- Exchange information by reading and writing data in the shared area.
- The processes are responsible for ensuring that they are not writing to the same location simultaneously.

35

Example: Producer-Consumer Problem

- A common Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process.
- Ex. The assembler produce object modules, which are consumed by the loader.
- Ex. Web server produces(provides) HTML files and images, which are consumed (read) by the web browser requesting the resource.
- One solution to the producer-consumer problem uses shared memory.
- To Allow the P and C processes to run concurrently, we must have a buffer that can be filled by the P and emptied by the C.
- This buffer reside in a region of memory that is shared by the P and C processes.

36

Contd....

- *The P and C must be synchronized, so that the C does not try to consume an item that has not yet been produced.*
- *Two types of buffers:*
  - *unbounded-buffer* places no practical limit on the size of the buffer (the C may wait, the P can always produce new items)
  - *bounded-buffer* assumes that there is a fixed buffer size (the C must wait if buffer empty, the P must wait if the buffer is full).

37

2. Message Passing

- Provides a Mechanism for processes to communicate and to synchronize their actions.
- Message system – processes communicate with each other without resorting to shared variables.
- IPC message-passing facility provides two operations:
  - **send**(*message*) – message size fixed or variable
  - **receive**(*message*)
- 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 (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)

38

Contd…

- Several methods for logically implementing a link, and the send()/receive() operations :
  - Direct or indirect communication.
  - Synchronous or asynchronous communication.
  - Automatic or explicit buffering.

39

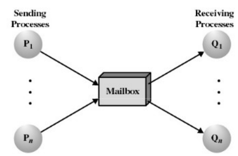
(i). Direct Communication

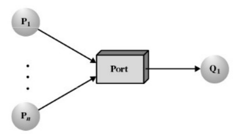
- Processes must name each other explicitly:
  - **send** (*P, message*) – send a message to process P
  - **receive**(*Q, message*) – receive a message from process Q
- The Properties of the communication link:
  - Links are established automatically between every pair of processes that want to communicate. The processes need to know only each other's identity to communicate.
  - 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

40

(ii). Indirect Communication

- Messages are directed and received from shared 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, with each link corresponding to one mailbox.
  - Link may be unidirectional or bi-directional





41

Contd…

- A mailbox may be owned either by a process (part of its address space) or by the OS.
- When a process that owns a mailbox terminates, the mailbox disappears.
- OS allows the process the 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

42

Contd…

- Mailbox sharing
  - P_1, P_2, and P_3 share mailbox A.
  - P_1, sends a message to A; P_2 and P_3 receive from A.
  - Who gets the message?
- Solutions
  - 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 (P_2 or P_3 not both) Sender is notified who the receiver was.

43

(iii). Synchronization

- Message passing may be either blocking or non-blocking
- **Blocking** is considered **synchronous**
  - **Blocking send** has the sender block until the message is received by the receiving process or by the mailbox.
  - **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 operation.
  - **Non-blocking receive** has the receiver receive either a valid message or null
- When Both send() and receive() are blocking, we have a
- **rendezvous** between the sender and the receiver.

44

(iv). Buffering

- Messages exchanged by processes reside in a temporary queue.
- Queue of messages attached to the link; implemented in one of three ways
  1. Zero capacity – max length of 0 messages (cannot have any messages waiting in it).
- Sender must wait(blocked) for receiver (rendezvous)
  2. Bounded capacity – finite length of *n* messages Sender must wait (block) if link is full
  3. Unbounded capacity – infinite length Sender never waits.

45

End of Chapter 3

Thank you

46