# CSE 4/521 Introduction to Operating Systems

Lecture 4 – Processes

(Process Concept, Process Scheduling, Operations on Processes, Inter-process Communication, Examples of IPC Systems)

Summer 2018

# Overview

Objective:

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

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

# Recap

- Operating-System Services
  - Services such as program execution, accounting, resource allocation, etc.
- User and Operating-System Interface
  - Command Line Interface, GUI, Touchscreen Interface
- System Calls
  - Programming interfaces to the services provided by OS
- Types of System Calls
  - 6 types including – process control, device management, etc.
- System Programs
  - Users' view of OS defined by system programs rather than system calls
- Operating-System Structure
  - Simple, complex, layered, micro-kernel
- Operating-System Debugging
  - Finding and fixing errors (core dump and crash dump), profiling using DTrace

# Questions

1. What is the purpose of system call? (Easy)

2. When would I prefer block or stack over registers for message passing while invoking system calls? (Medium)

3. Is there any other structure for operating systems apart from simple, complex, layered and micro-kernel? (Hard and open-ended)

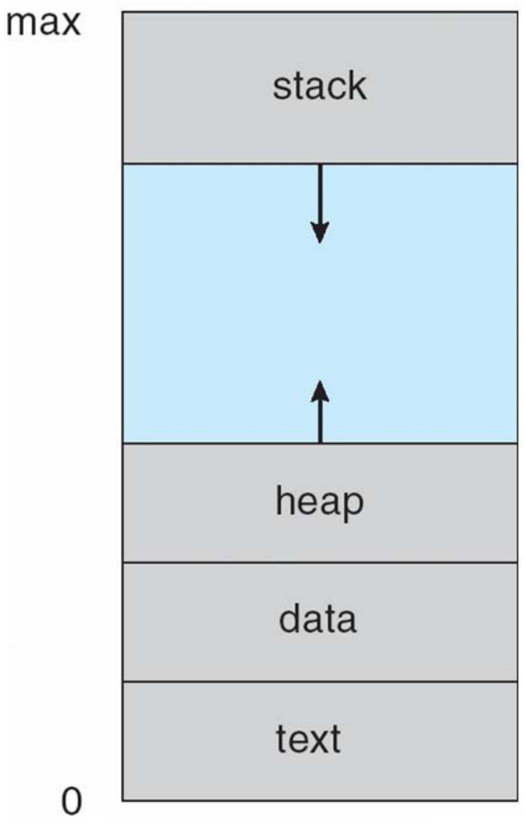
# Table of Content

# Table of Content

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

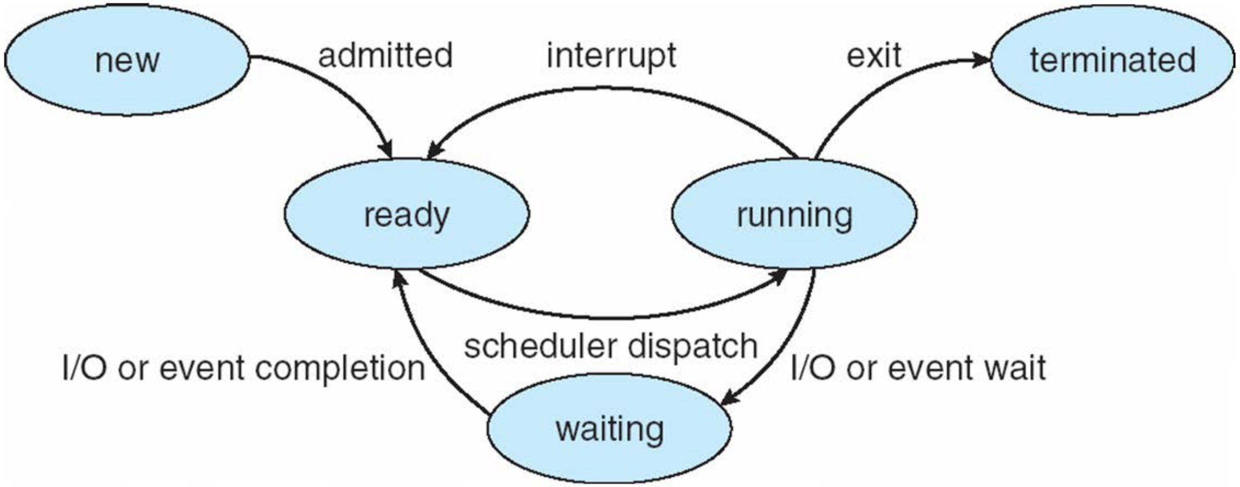
# Process Concept

- Multiple parts of a Process
  - The program code, also called text section
  - Stack containing temporary data
    - Function parameters, return addresses, local variables
  - Data section containing global variables
  - Heap containing memory dynamically allocated during run time



# Process Concept – 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



# Process Concept – Process Control Block (PCB)

Information associated with each process

- 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


Structure of PCB

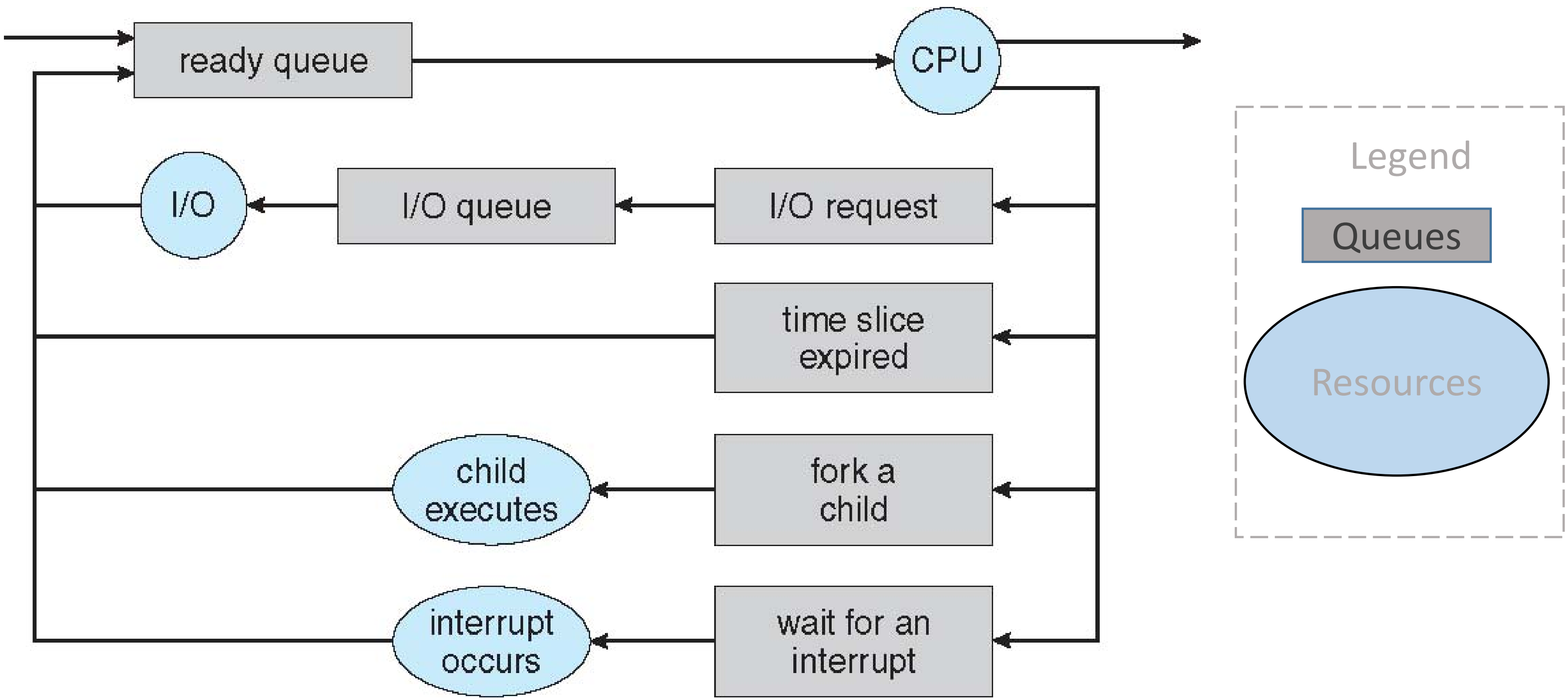
# Table of Content

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

# 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



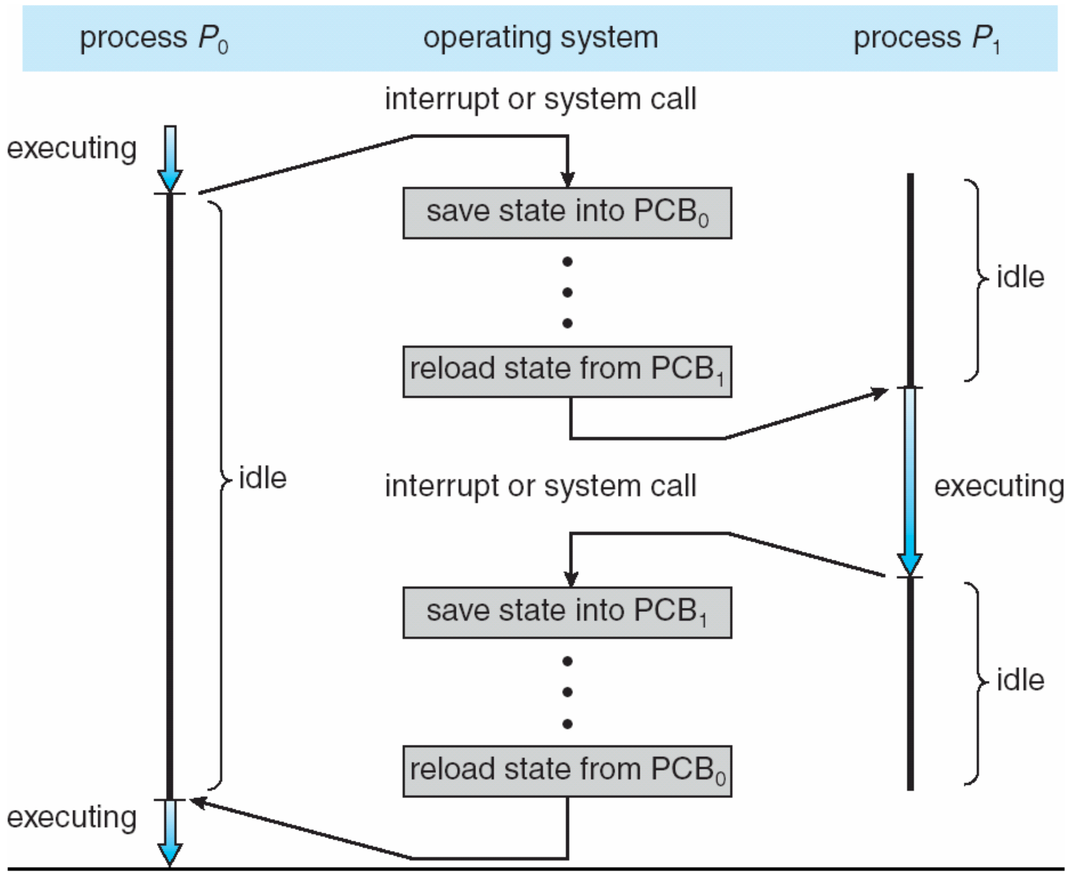
# Process Scheduling - Schedulers

- 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
- 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

# Process Scheduling – 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

# Process Scheduling – Context Switch



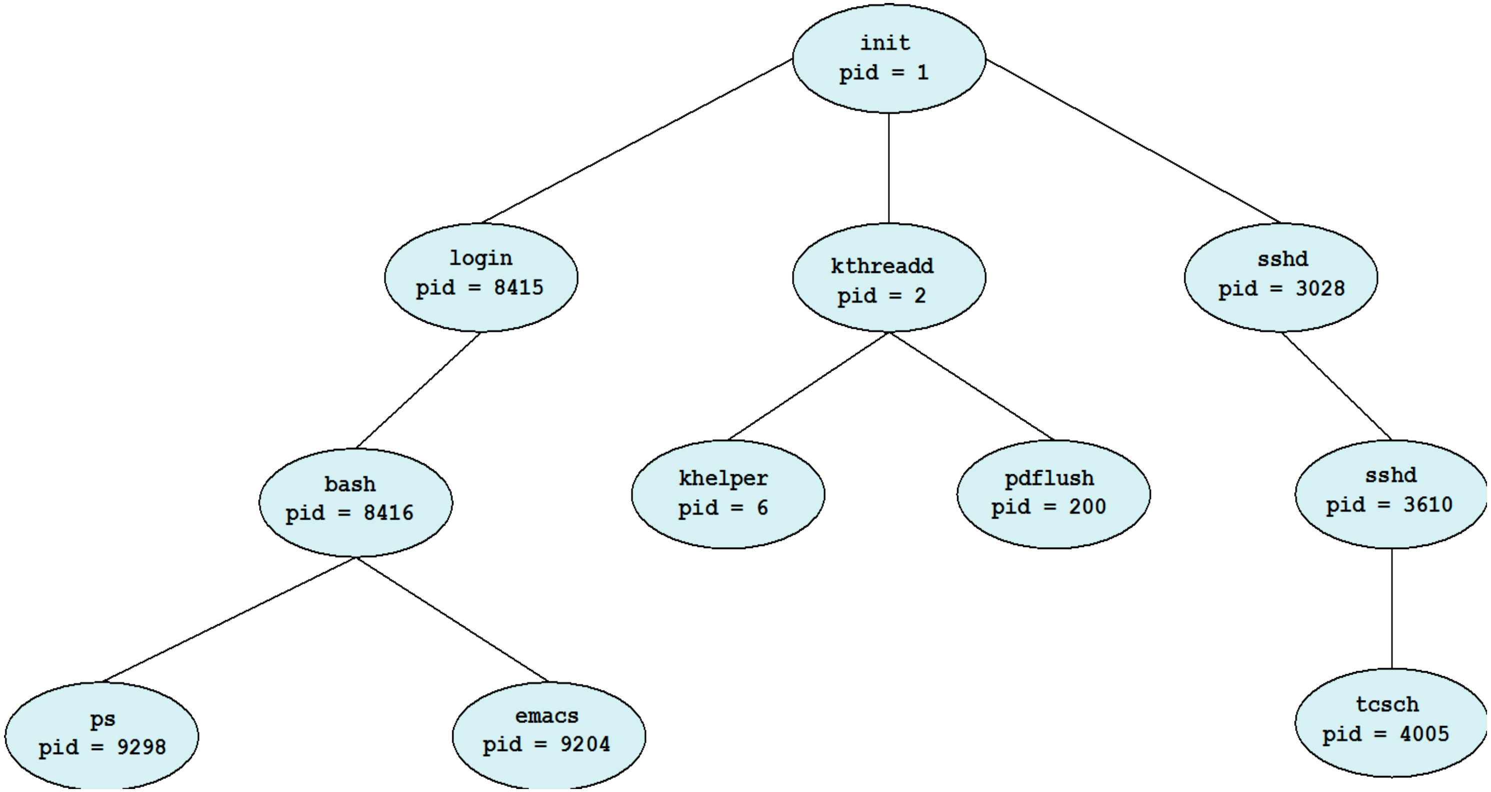
# Table of Content

- Process Concept
- Process Scheduling
- Operations on Processes – Process Creation and Process Termination
- Interprocess Communication
- Examples of IPC Systems

# Operations on Processes – Process Creation

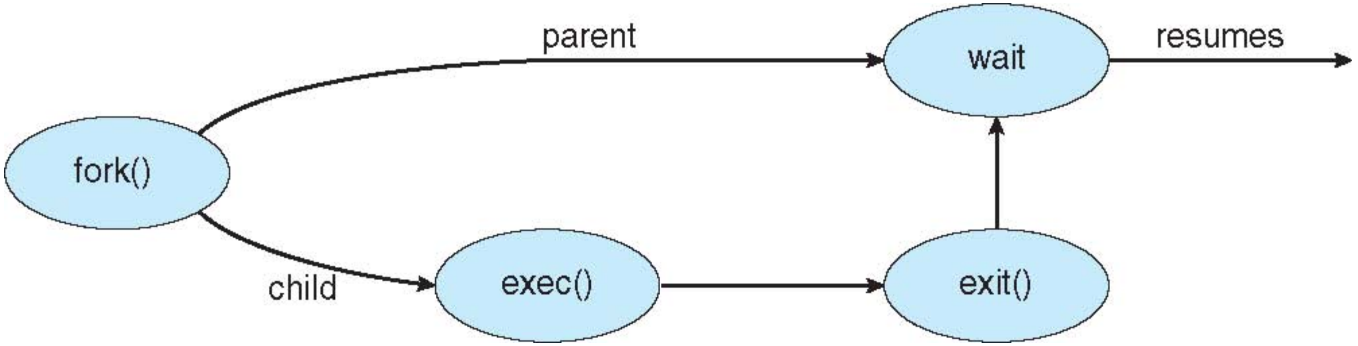
- Parent process create children processes, which, in turn create other processes, forming a tree of processes
- Generally, process identified and managed via a process identifier (pid)
- Resource sharing options
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution options
  - Parent and children execute concurrently
  - Parent waits until children terminate

# Operations on Processes – Process Creation



# Operations on Processes – Process Creation

- 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



# Operations on Processes – Process Creation (Example)

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
      fprintf(stderr, "Fork Failed");
      return 1;
    }
    else if (pid == 0) { /* child process */
      execlp("/bin/ls","ls",NULL);
    }
    else { /* parent process */
      /* parent will wait for the child to complete */
      wait(NULL);
      printf("Child Complete");
    }

    return 0;
}
```

# Operations on Processes – 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

# Operations on Processes – Process Termination

- 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

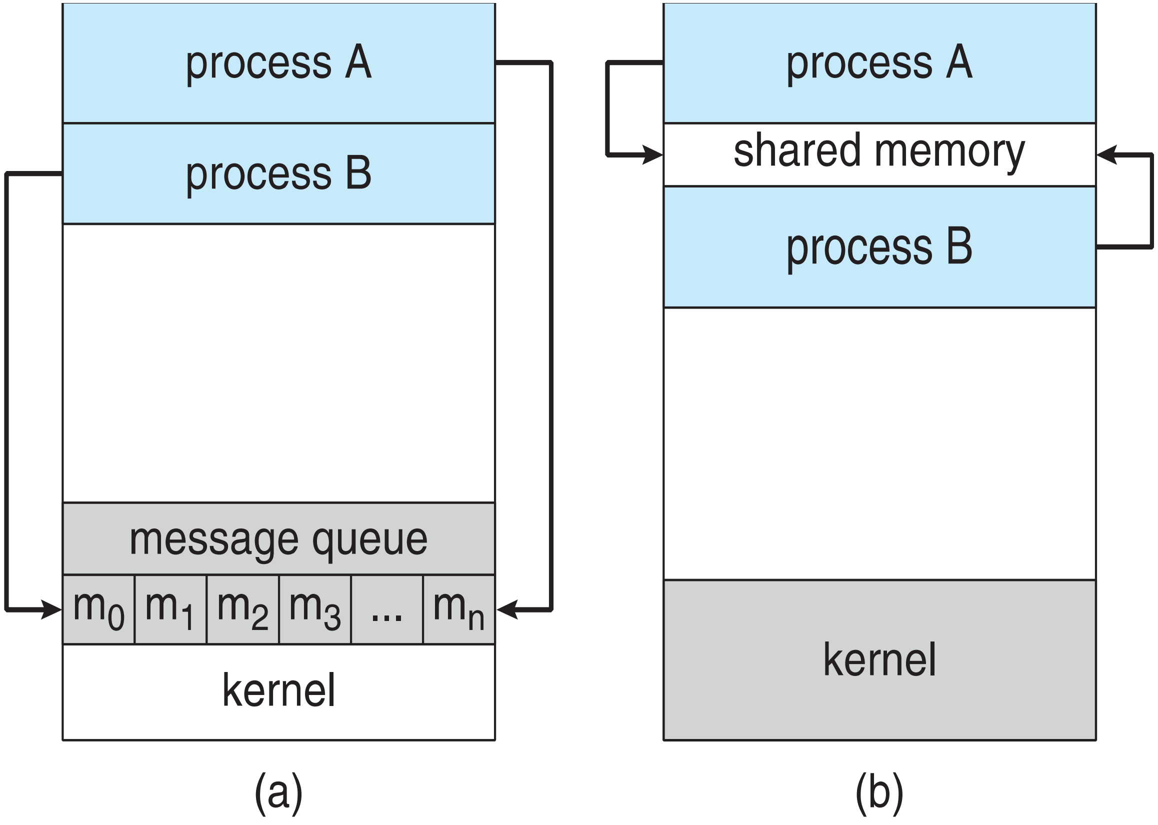
# Table of Content

# 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

# Interprocess Communication

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



# Interprocess Communication – Producer-Consumer Problem

- Paradigm for cooperating processes, producer process produces information that is consumed by a consumer process
  - unbounded-buffer places no practical limit on the size of the buffer
  - bounded-buffer assumes that there is a fixed buffer size

# Interprocess Communication – Bounded Buffer (Producer-Consumer)

Producer

```
item next_produced;
while (true) {
        /* produce an item in
next produced */
        while (((in + 1) %
BUFFER_SIZE) == out)
                ; /* do
nothing */
        buffer[in] =
next_produced;
        in = (in + 1) %
BUFFER_SIZE;
}
```

Consumer

```
item next_consumed;
while (true) {
        while (in == out)
                ; /* do nothing */
        next_consumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;

        /* consume the item in next
consumed */
}
```

# 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.

# Interprocess Communication – Message Passing

- IPC facility provides two operations:
  - **`send`**(*message*)
  - **`receive`**(*message*)

- The message size is either fixed or variable
- Concerns regarding implementation of communication link
  - Direct or indirect
    - Direct – Processes must name each other.
    - Indirect – Messages are directed and received from mailboxes
  - Synchronous or asynchronous
    - Synchronous – Blocking send and blocking receive
    - Asynchronous – Non-blocking send and non-blocking receive
  - Automatic or explicit buffering
    - Implemented either as zero capacity, bounded capacity and unbounded capacity

# Table of Content

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

# Examples of IPC Systems

Interprocess Communication through:

1. Shared memory (POSIX)
2. Message passing (Mach)
3. Hybrid : Shared memory + Message passing (Windows)

# Examples of IPC Systems - POSIX

- Process first creates shared memory segment

```
shm_fd = shm_open(name, O CREAT | O RDWR, 0666);
```

- Set the size of the object

```
ftruncate(shm fd, 4096);
```

- Process writes to the shared memory

```
sprintf(shared memory, "Writing to shared memory");
```

# Examples of IPC Systems – POSIX (Producer-Consumer)

Producer

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main()
{
/* the size (in bytes) of shared memory object */
const int SIZE = 4096;
/* name of the shared memory object */
const char *name = "OS";
/* strings written to shared memory */
const char *message_0 = "Hello";
const char *message_1 = "World!";

/* shared memory file descriptor */
int shm_fd;
/* pointer to shared memory obect */
void *ptr;

   /* create the shared memory object */
   shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);

   /* configure the size of the shared memory object */
   ftruncate(shm_fd, SIZE);

   /* memory map the shared memory object */
   ptr = mmap(0, SIZE, PROT_WRITE, MAP_SHARED, shm_fd, 0);

   /* write to the shared memory object */
   sprintf(ptr,"%s",message_0);
   ptr += strlen(message_0);
   sprintf(ptr,"%s",message_1);
   ptr += strlen(message_1);

   return 0;
}
```

Consumer

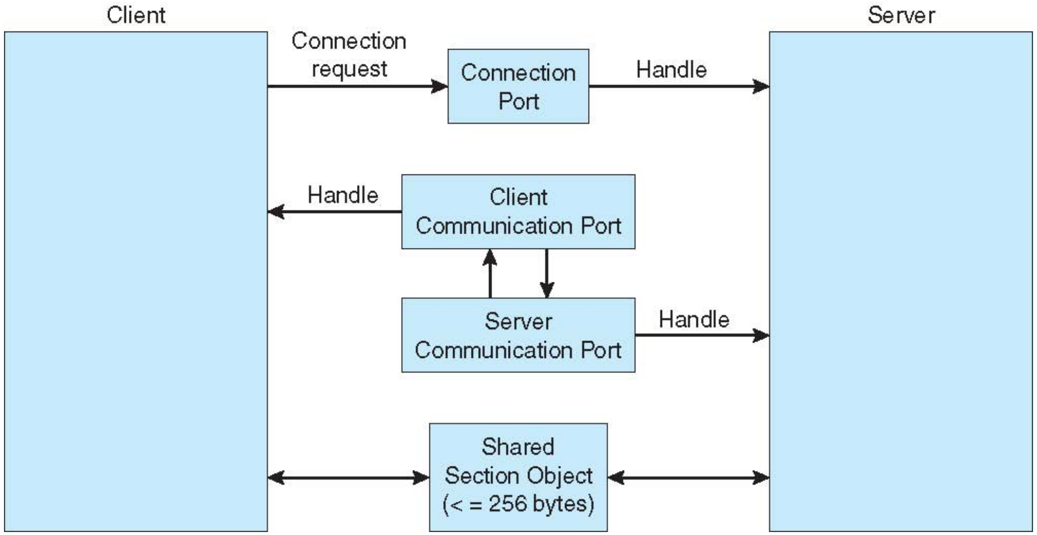
```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main()
{
/* the size (in bytes) of shared memory object */
const int SIZE = 4096;
/* name of the shared memory object */
const char *name = "OS";
/* shared memory file descriptor */
int shm_fd;
/* pointer to shared memory obect */
void *ptr;

   /* open the shared memory object */
   shm_fd = shm_open(name, O_RDONLY, 0666);

   /* memory map the shared memory object */
   ptr = mmap(0, SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);

   /* read from the shared memory object */
   printf("%s",(char *)ptr);

   /* remove the shared memory object */
   shm_unlink(name);

   return 0;
}
```

# Examples of IPC Systems - Mach

- Mach communication is message based
  - Even system calls are messages
  - Mailboxes needed for communication, created via **`port_allocate()`**
  - Send and receive are flexible, for example four options if mailbox full:
    - Wait indefinitely
    - Wait at most n milliseconds
    - Return immediately
    - Temporarily cache a message

# Examples of IPC Systems - Windows

- Message-passing centric via advanced local procedure call (LPC) facility
- One of the three message-passing technique is chosen:
  1. For small messages, port's message queue is use.
  2. Larger messages passed through section object.
  3. Amount of data is too large, then read and write directly.



# Credential for slides

Silberschatz, Galvin and Gagne