

CS343 - Operating Systems

Module-3A

Inter Process Communication



Dr. John Jose

Assistant Professor

Department of Computer Science & Engineering

Indian Institute of Technology Guwahati, Assam.

<http://www.iitg.ac.in/johnjose/>

Session Outline

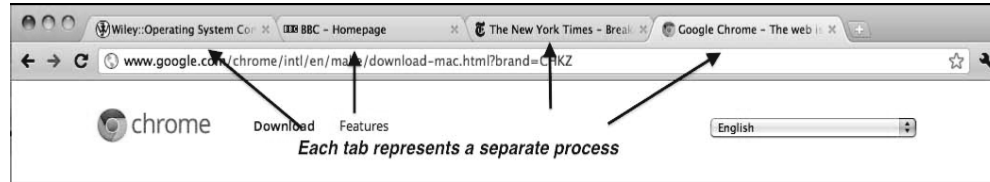
- ❖ **Multitasking/Multiprocessing Applications**
- ❖ **Review of process management functions**
- ❖ **Process creation and termination**
- ❖ **Inter Process Communication (IPC)**
- ❖ **Producer-Consumer problem**
- ❖ **IPC- shared memory**
- ❖ **IPC-message passing**
- ❖ **Direct vs indirect communication**

Multitasking in Mobile Systems

- ❖ Some mobile systems allow only one process to run, others suspended
- ❖ Due to screen space limits, user interface limits iOS provides for a
 - ❖ **Single foreground process-** controlled via user interface
 - ❖ **Multiple background processes**— in memory, running, but not on the display, and with limits
 - ❖ Limits include single, short task, receiving notification of events, specific long-running tasks like audio playback
- ❖ **Android runs foreground and background**, with fewer limits
 - ❖ Background process uses a service to perform tasks
 - ❖ Service can keep running even if background process is suspended
 - ❖ Service has no user interface, small memory use

Multi-process Application

- ❖ Many web browsers ran as single process (some still do)
 - ❖ If one web site causes trouble, entire browser can hang or crash



- ❖ Google Chrome Browser is multiprocess with 3 types of processes:
 - ❖ **Browser** process manages user interface, disk and network I/O
 - ❖ **Renderer** process renders web pages, deals with HTML, Javascript.
 - ❖ **Plug-in** process for each type of plug-in

Process Management

- ❖ Creating and deleting both user and system processes
- ❖ Suspending and resuming processes (context switching, scheduling)
- ❖ Providing mechanisms for process communication
- ❖ Providing mechanisms for process synchronization
- ❖ Providing mechanisms for deadlock handling

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

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
 - ❖ 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

Process Termination

- ❖ Some OS do not allow child to exist if its parent has terminated.
- ❖ **Cascading termination:** If a process terminates, then all its children, grand children, etc. must also be terminated.
- ❖ 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**

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
- ❖ Time dependent on hardware support

Process Management

- ❖ Creating and deleting both user and system processes
- ❖ Suspending and resuming processes (context switching, scheduling)
- ❖ Providing mechanisms for process communication
- ❖ Providing mechanisms for process synchronization
- ❖ Providing mechanisms for deadlock handling

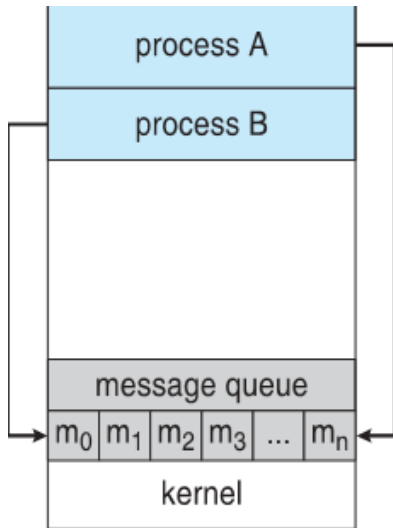
Inter-process Communication

- ❖ Processes within a system may be **independent** or **cooperating**
- ❖ **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
 - ❖ Computation speedup
 - ❖ Modularity
 - ❖ Convenience

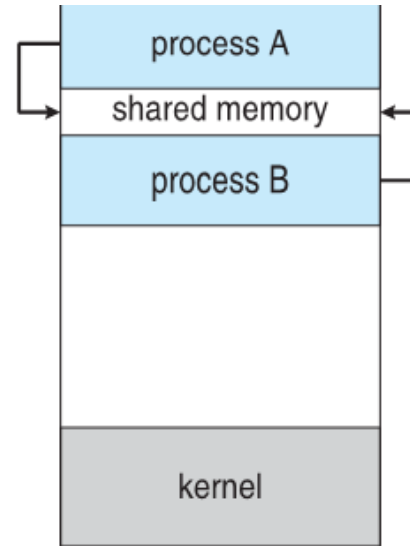
Communications Models

- ❖ Cooperating processes need **interprocess communication (IPC)**
- ❖ Two models of IPC:

Message passing



Shared memory



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

Bounded-Buffer – Producer & Consumer

```
item buffer[BUFFER_SIZE]; int in = 0; int out = 0;
```

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 */

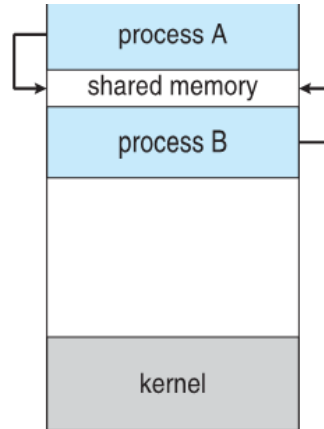
    next_consumed = buffer[out];

    out = (out + 1) % BUFFER_SIZE;
    /* consume the item in next
    consumed */

}
```

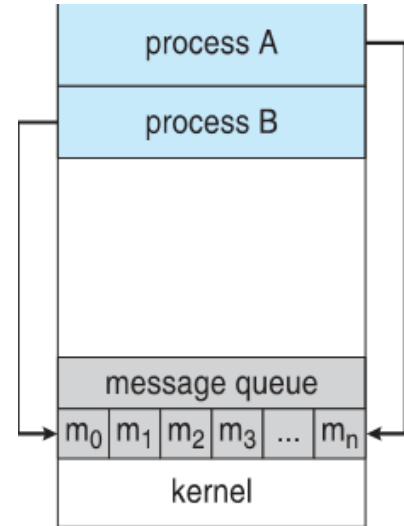
IPC – Shared Memory

- ❖ 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.



IPC – 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



IPC – Message Passing

- ❖ If processes P and Q wish to communicate, they need to:
 - ❖ Establish a communication link between them
 - ❖ Exchange messages via send/receive
- ❖ Implementation issues:
 - ❖ How are links established?
 - ❖ Can a link be associated with more than two processes?
 - ❖ How many links between a pair of communicating processes?
 - ❖ What is the capacity of a link?
 - ❖ Unidirectional or bi-directional link?
 - ❖ Is the size of a message in the link fixed or variable?

IPC – Message Passing

- ❖ Implementation of communication link

- ❖ Physical:

- ❖ Shared memory

- ❖ Hardware bus

- ❖ Network

- ❖ Logical:

- ❖ Direct or indirect

- ❖ Synchronous or asynchronous

- ❖ Automatic or explicit buffering

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
- ❖ Properties of communication link
 - ❖ Links are established automatically
 - ❖ 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

Indirect Communication

- ❖ Messages are directed and received from mailboxes
 - ❖ 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
 - ❖ Link may be unidirectional or bi-directional

Indirect Communication

- ❖ Operations
 - ❖ create a new mailbox (port)
 - ❖ 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

Indirect Communication

- ❖ Mailbox sharing
 - ❖ P_1 , P_2 , and P_3 share mailbox A
 - ❖ P_1 sends; P_2 and P_3 receive
- ❖ 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. Sender is notified who the receiver was.

Synchronization

- ❖ Message passing may be either blocking or non-blocking
- ❖ **Blocking** is considered **synchronous**
 - ❖ **Blocking send** -- the sender is blocked until the message is received
 - ❖ **Blocking receive** -- the receiver is blocked until a message is available

Synchronization

- ❖ Message passing may be either blocking or non-blocking
- ❖ **Non-blocking** is considered **asynchronous**
 - ❖ **Non-blocking send** -- the sender sends the message and continue
 - ❖ **Non-blocking receive** -- the receiver receives:
 - ❖ A valid message, or
 - ❖ Null message

Buffering

- ❖ Queue of messages attached to the link.
- ❖ Implemented in one of three ways
 1. Zero capacity – no messages are queued on a link.
Sender must wait for receiver
 2. Bounded capacity – finite length of n messages
Sender must wait if link full
 3. Unbounded capacity – infinite length
Sender never waits

Thank you

johnjose@iitg.ac.in

<http://www.iitg.ac.in/johnjose/>

