CS240 Operating Systems, Communications and Concurrency

Unix Process Management and Interprocess Communication

Overview

- How to create and terminate processes and control behaviour from within C programs
- The Linux Boot sequence – Creating the first process
- Design considerations for communication, different implementation approaches

CS240 Operating Systems, Communications and Concurrency

Unix Process Creation

Every process has a unique ID

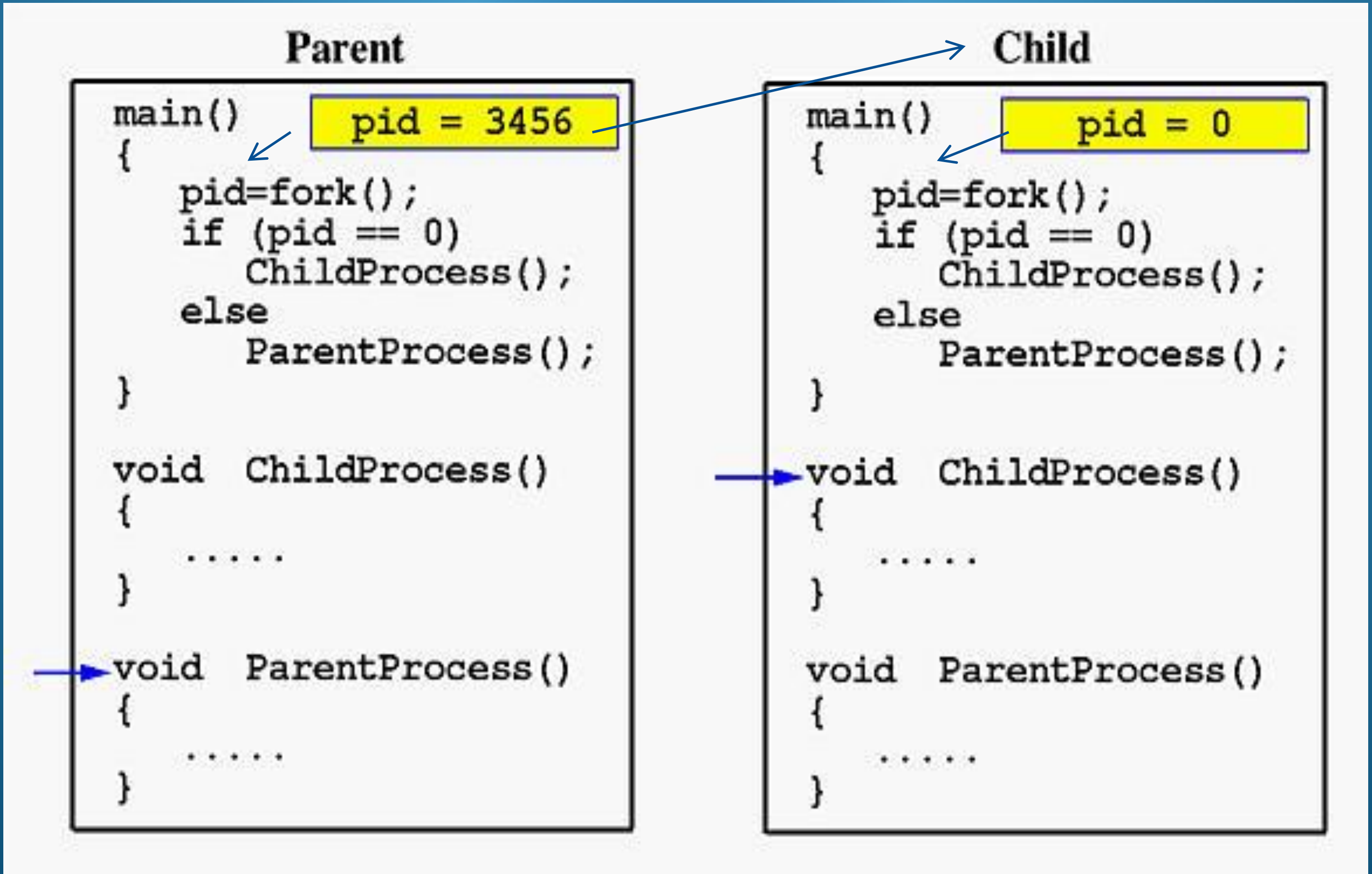
A process is created using the `fork()` system call within C program

Initial process is the Parent and new process is the Child

Child process is an exact copy of the parent including a copy of the parent's I/O descriptor table, so it inherits access to all the parent's open I/O devices.

CS240 Operating Systems, Communications and Concurrency

Unix Process Creation



```
main()
{
    pid=fork();
    if (pid == 0)
        ChildProcess();
    else
        ParentProcess();
}

void  ChildProcess()
{
    .....
}

void  ParentProcess()
{
    .....
}
```

CS240 Operating Systems, Communications and Concurrency

Unix Process Creation

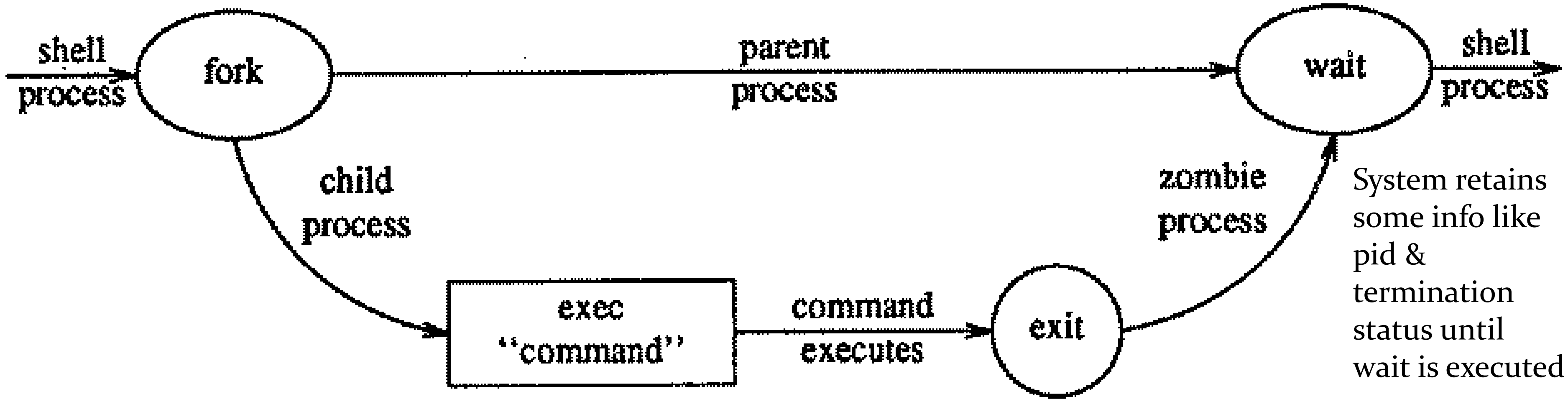
Typically, the **`exec()`** system call is used by child after a `fork()`

The `exec()` system call loads the specified binary file into the process's memory (replacing the existing program image) and starts execution of it.

CS240 Operating Systems, Communications and Concurrency

Unix Process Creation

Eg: Implementation of command initiation at the shell



The **wait()** system call suspends execution of the calling process until one of its children terminates.

wait() : on success, returns the process ID of the terminated child; on error, -1 is returned. The system reclaims the resources of the zombie child after wait has been performed.

CS240 Operating Systems, Communications and Concurrency

Unix Process Creation - C program template

```
#include <stdio.h>  /*C standard I/O functions */
#include <unistd.h> /*provides access to the POSIX API */
#include <stdlib.h> /*Standard library functions for C */

int main(int argc, char *argv[])
{

Program statements go here


}
```

At a command prompt, when we type a command with parameters separated by spaces, the number of strings (representing the command and its arguments) is passed to the main function as an integer (`argc`) and each string as a character array accessible as `argv[0], argv[1], argv[2]` etc.

CS240 Operating Systems, Communications and Concurrency

Unix Process Creation – C Program Statements

```
{
int pid;
pid = fork(); /*create a child process */

/* Two processes now exist executing
copies of this code (unless fork failed)
but with different pid values */

   if (pid < 0) {  /*Testing for errors */
      printf("Fork failed");
      exit(-1);  /* program ends */
   }
```

CS240 Operating Systems, Communications and Concurrency

Unix Process Creation

```
 else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
    }

else {  /* Parent process */
    /* Wait suspends execution of current
      process until a child has ended. */
       wait(NULL);
       printf("Child complete");
       exit(0); /* parent ends */
    }
}
```

CS240 Operating Systems, Communications and Concurrency

How many processes are created by this program?

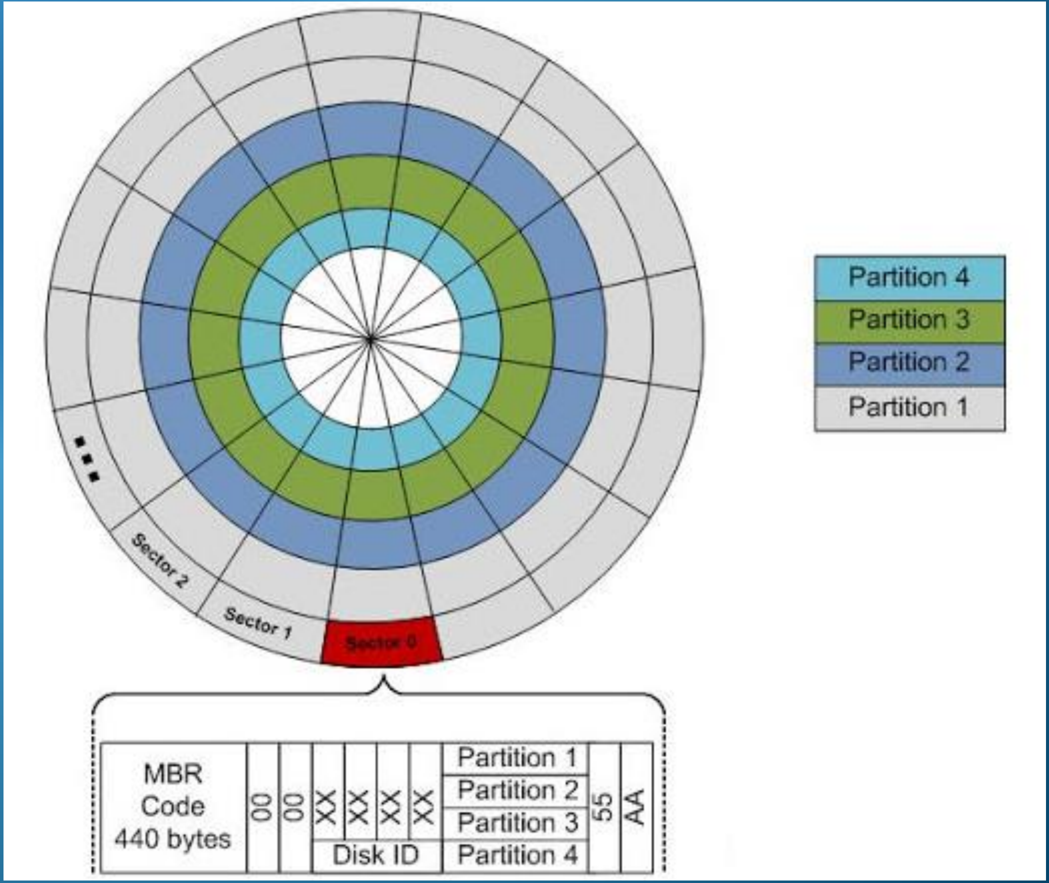
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
int pid; int i;

for (i = 1; i<=3; i++) {
   pid = fork();
   if (pid != 0) {
      printf("Process %d\n", pid);
      execlp("/bin/ls", "ls", NULL);
      }
   }
}
```

Booting a Linux System - Master Boot Record

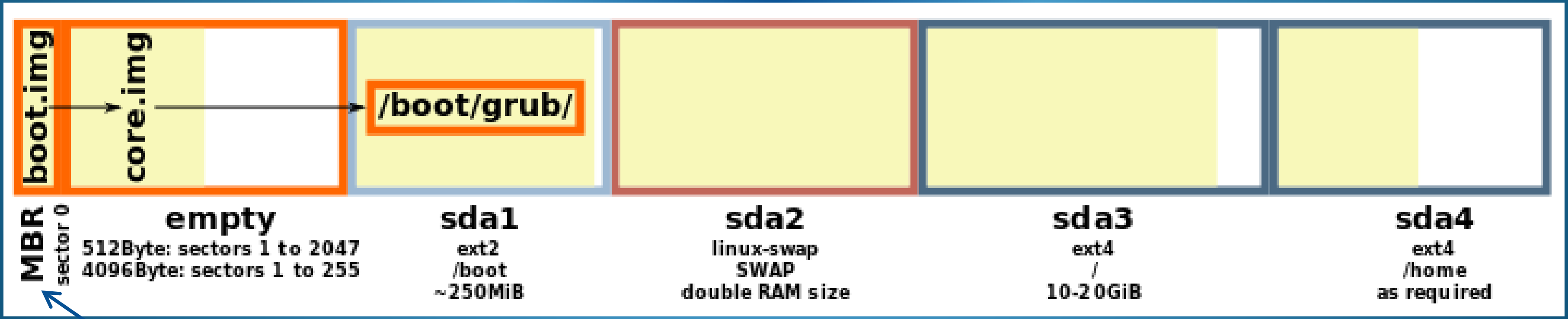
After power on self test and hardware identification by firmware routines, the first boot device is selected and the Master Boot Record, the first sector on the drive, is read from that device. The MBR contains initial bootstrapping code and information about the active partition.



CS240 Operating Systems, Communications and Concurrency

Booting Linux

Hard Disk Partitioning

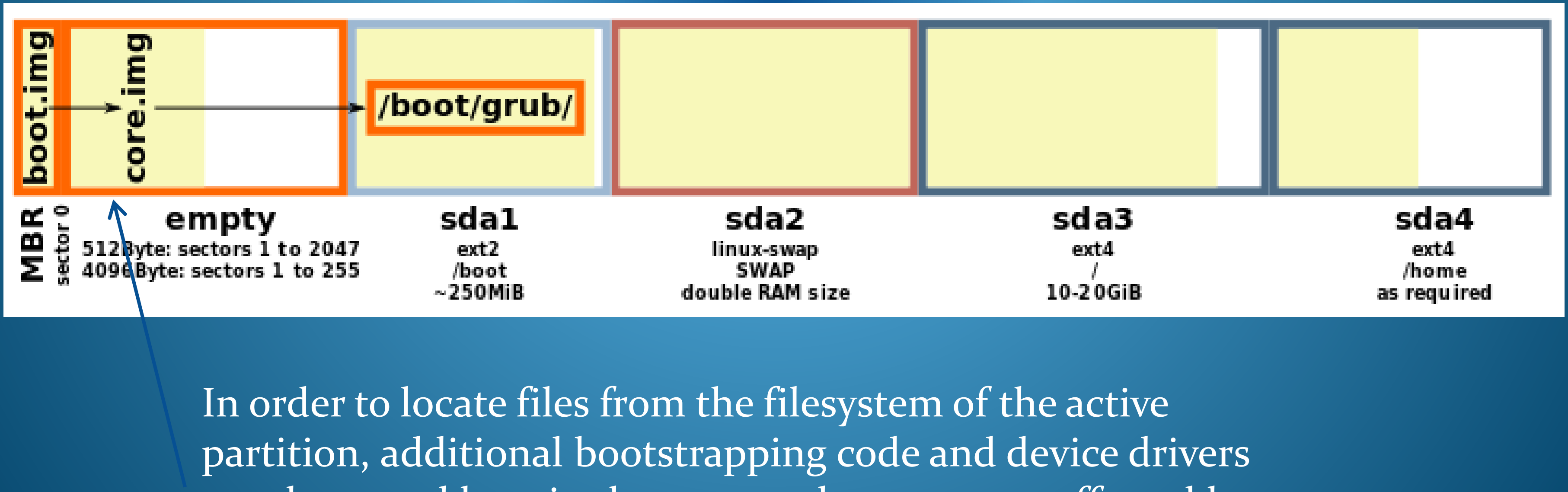


Master Boot Record (MBR) contains initial bootstrapping code called by the BIOS and partition information about the primary boot device and the active partition.
It is always located in the first sector on the disk. Raw binary 440 bytes of code.

CS240 Operating Systems, Communications and Concurrency

Booting Linux

Hard Disk Partitioning

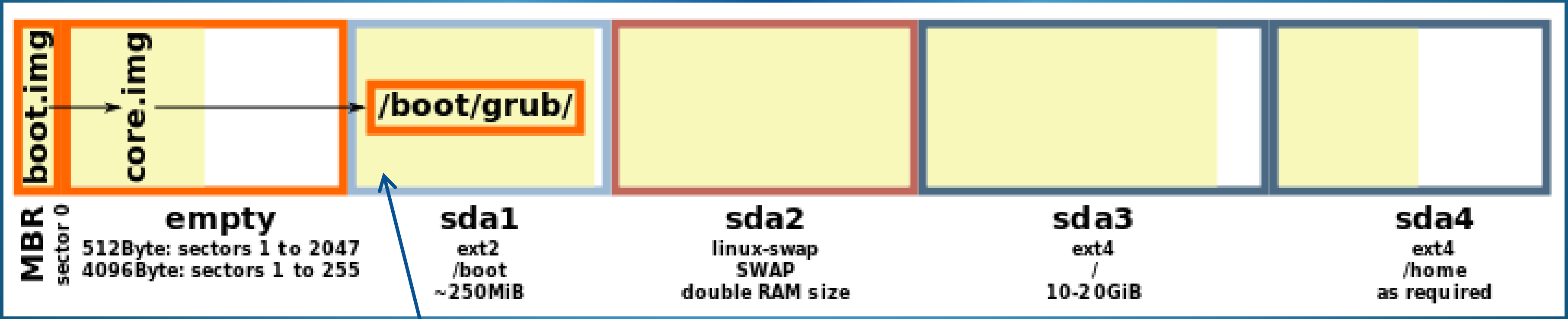


In order to locate files from the filesystem of the active partition, additional bootstrapping code and device drivers may be stored here in these unused sectors, not affected by filesystem formatting.

CS240 Operating Systems, Communications and Concurrency

Booting Linux

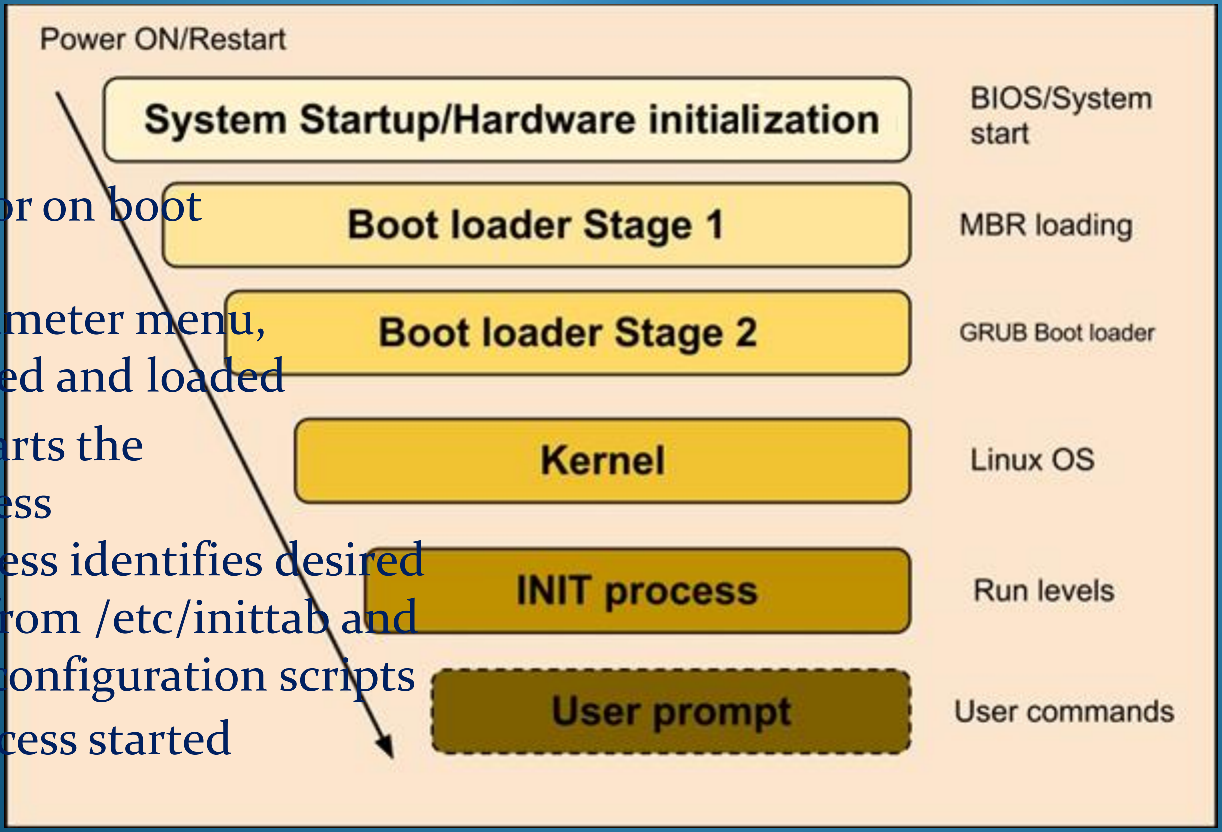
Hard Disk Partitioning



This gives enough code then to locate and load the main boot loader program which presents an interface asking the user to select the operating system to boot.

CS240 Operating Systems, Communications and Concurrency

Unix Boot Sequence – Creating the first process



Firmware

First sector on boot device

Boot parameter menu, OS selected and loaded

Kernel starts the first process

First process identifies desired runlevel from /etc/inittab and executes configuration scripts

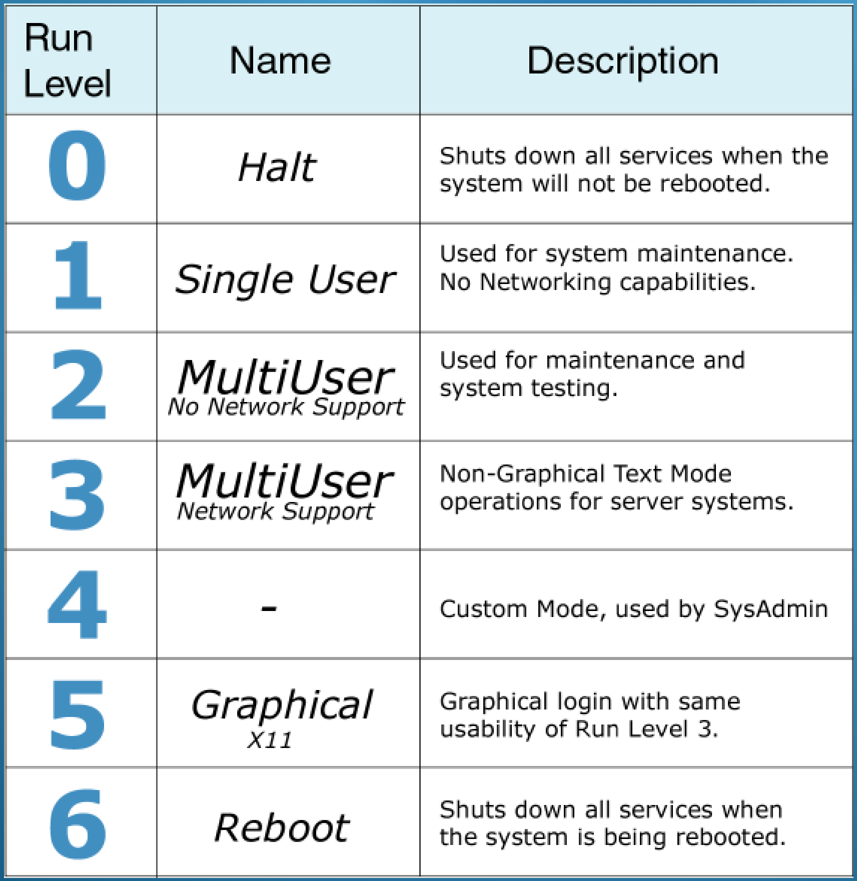
Getty process started

Understanding the Linux Boot Process

Boot Process in Linux

CS240 Operating Systems, Communications and Concurrency

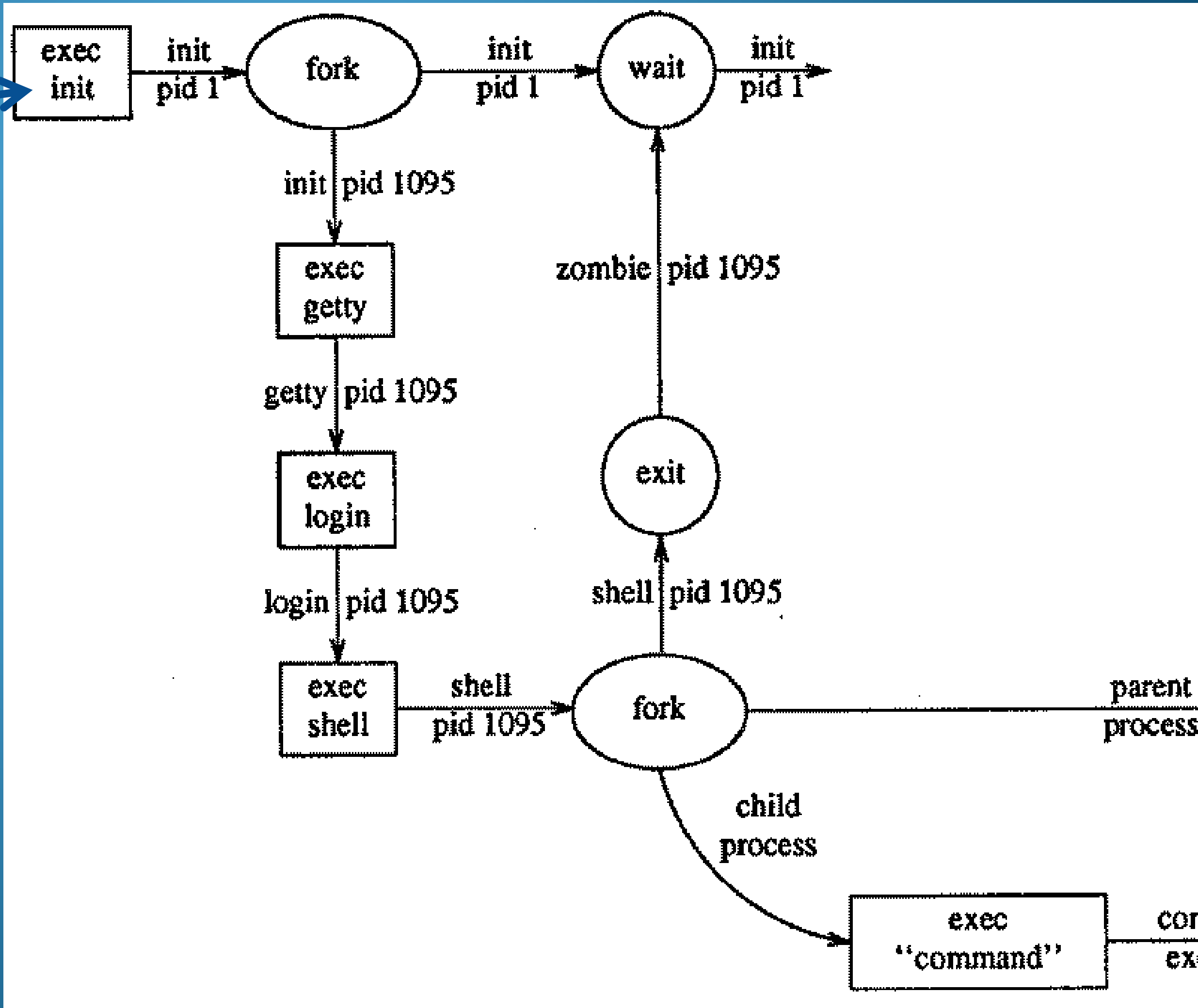
Linux Run Levels



Run Level	Name	Description
0	*Halt*	Shuts down all services when the system will not be rebooted.
1	*Single User*	Used for system maintenance. No Networking capabilities.
2	*MultiUser* *No Network Support*	Used for maintenance and system testing.
3	*MultiUser* *Network Support*	Non-Graphical Text Mode operations for server systems.
4	-	Custom Mode, used by SysAdmin
5	*Graphical* *X11*	Graphical login with same usability of Run Level 3.
6	*Reboot*	Shuts down all services when the system is being rebooted.

CS240 Operating Systems, Communications and Concurrency

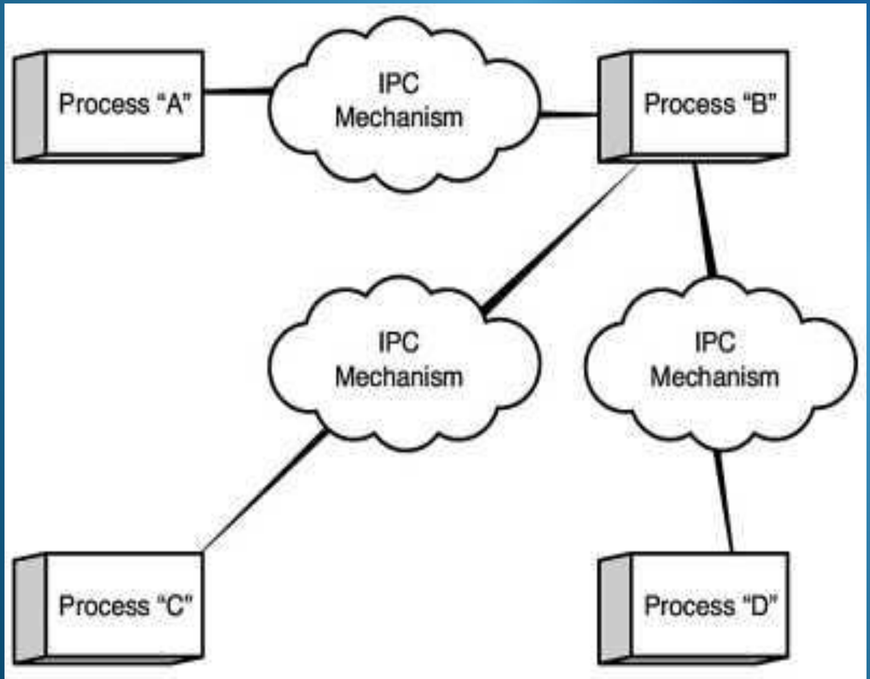
Init is the initial start up process. Its primary role is to create processes from configuration information stored in the file /etc/inittab during boot up depending on the run level selected.



CS240 Operating Systems, Communications and Concurrency

Interprocess Communication

Interprocess communication refers to the exchange or sharing of information between separate independent schedulable tasks.
There are several reasons why you might want to do this.



Data Parallelization for Computational Speedup

Modular, Pipelined or Layered Division of functions or microservices within an application

Client use of a service on a network.

CS240 Operating Systems, Communications and Concurrency

Interprocess Communication

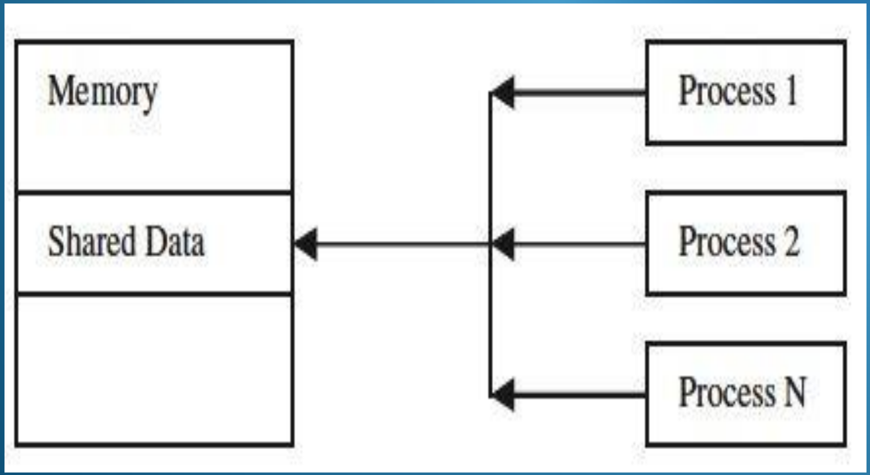
There are two basic ways of implementing interprocess communication:

By utilising a shared region of memory accessible to all the communicating parties

By using explicit message passing primitives provided by a communication subsystem of the operating system.

Shared Memory Communication

Usually, the address space accessible to independent processes is separate.

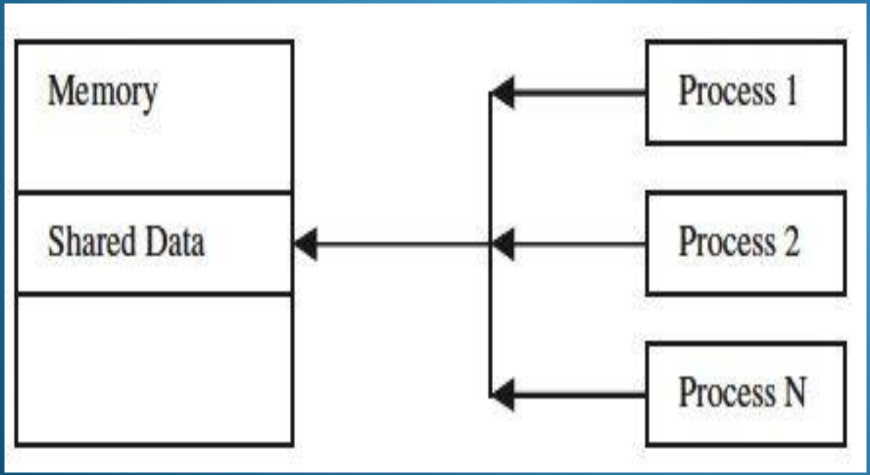


Shared memory communication would require the processes to use the operating system to establish a shared region of memory between them.

CS240 Operating Systems, Communications and Concurrency

Shared Memory Communication

Usually communication requires processes to be on the same host, they are usually cooperative parts of the same application.



Model is application oriented, suits cooperating processes willing to share memory.

Implicit communication through read/write operations.

Highly efficient, no communication protocols.

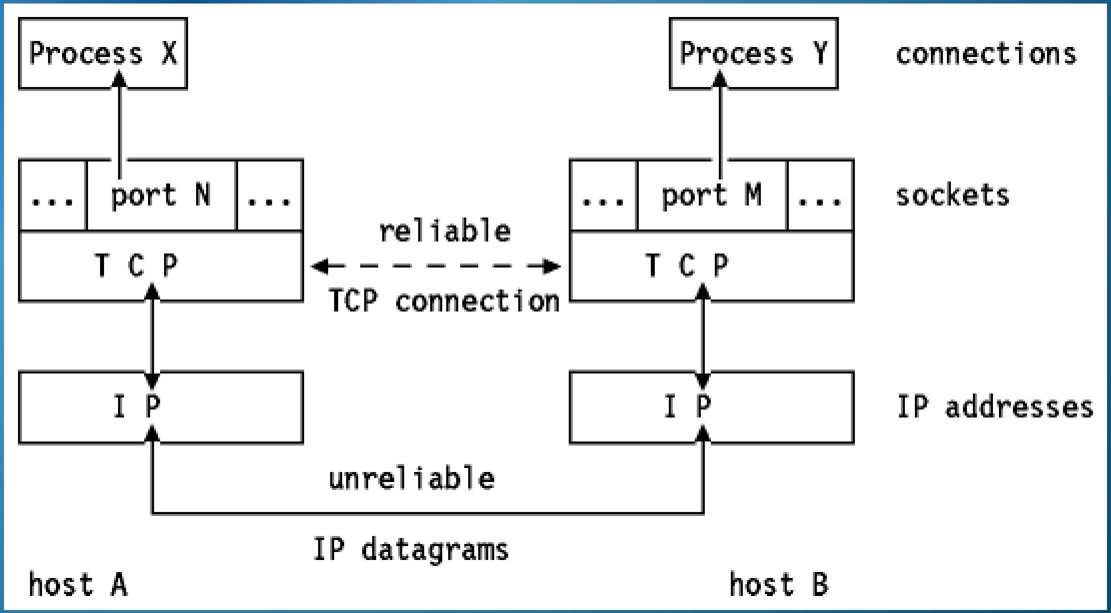
Need synchronisation mechanisms.

CS240 Operating Systems, Communications and Concurrency

Interprocess Communication

Independent processes need a different means of communication as they cannot access each other's disjoint address space. Processes may not trust each other and may reside on different hosts.

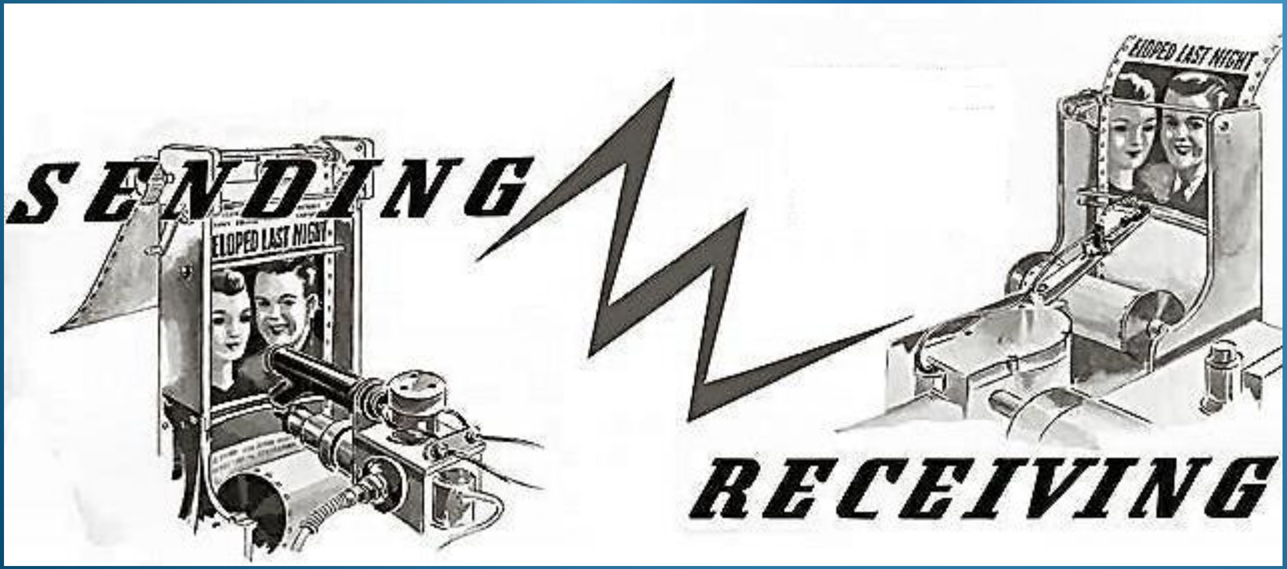
A message passing facility is any intermediary operating system mechanism which can take data from the address space of one process and place it in an area accessible to the other.



CS240 Operating Systems, Communications and Concurrency

Interprocess Communication

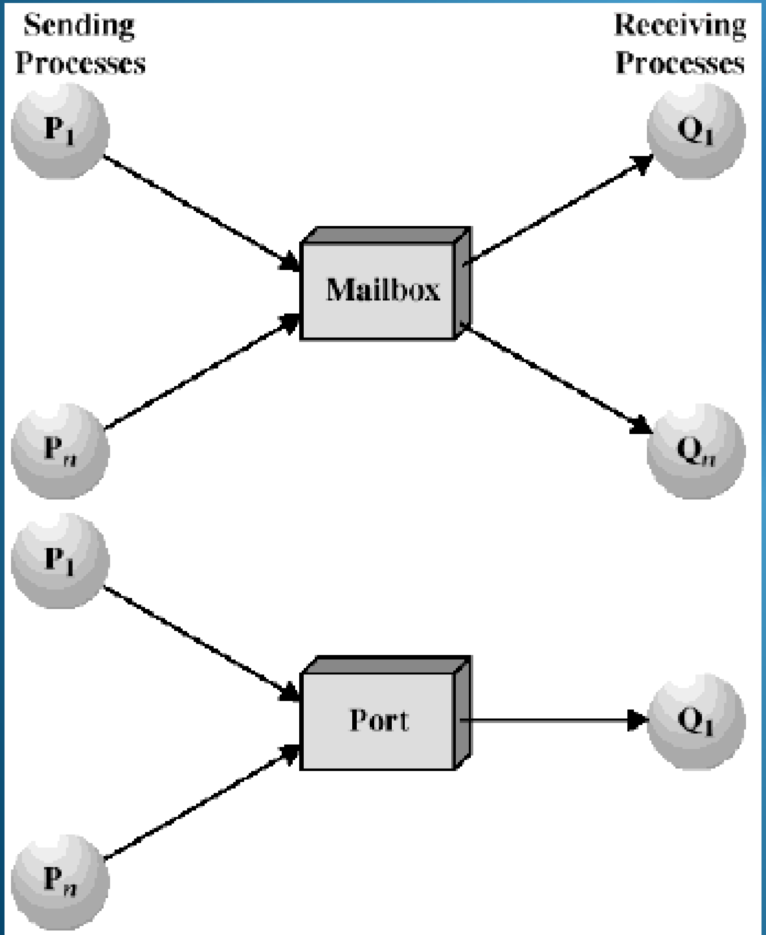
In its simplest form, a message passing mechanism must implement two basic primitives, **send** and **receive**, or their equivalent, which processes explicitly invoke to exchange messages.



Higher level abstractions, such as remote procedure call, are extensions of these basic primitives.

CS240 Operating Systems, Communications and Concurrency

Design Issues for Interprocess Communication



Naming

How is the link between two processes established?

The sender may identify (as a parameter to the Send primitive) a process ID or a mailbox or port ID to which a message is to be sent.

Indirect vs direct communication.

Single channel or multichannel.

Design Issues for Interprocess Communication

Synchronisation

The message passing operations Send and Receive may be either blocking or non-blocking.

Blocking Send:	Sending process is blocked until the message passing mechanism has delivered the message to the receiving process or to a mailbox.
Non-Blocking Send:	The sending process sends a message and can continue other tasks immediately while the message passing mechanism delivers the message.
Blocking Receive:	Receiver waits until a message is received by it.
Non-Blocking Receive:	Receiver either gets an available message or a null value returned by the Receive primitive.

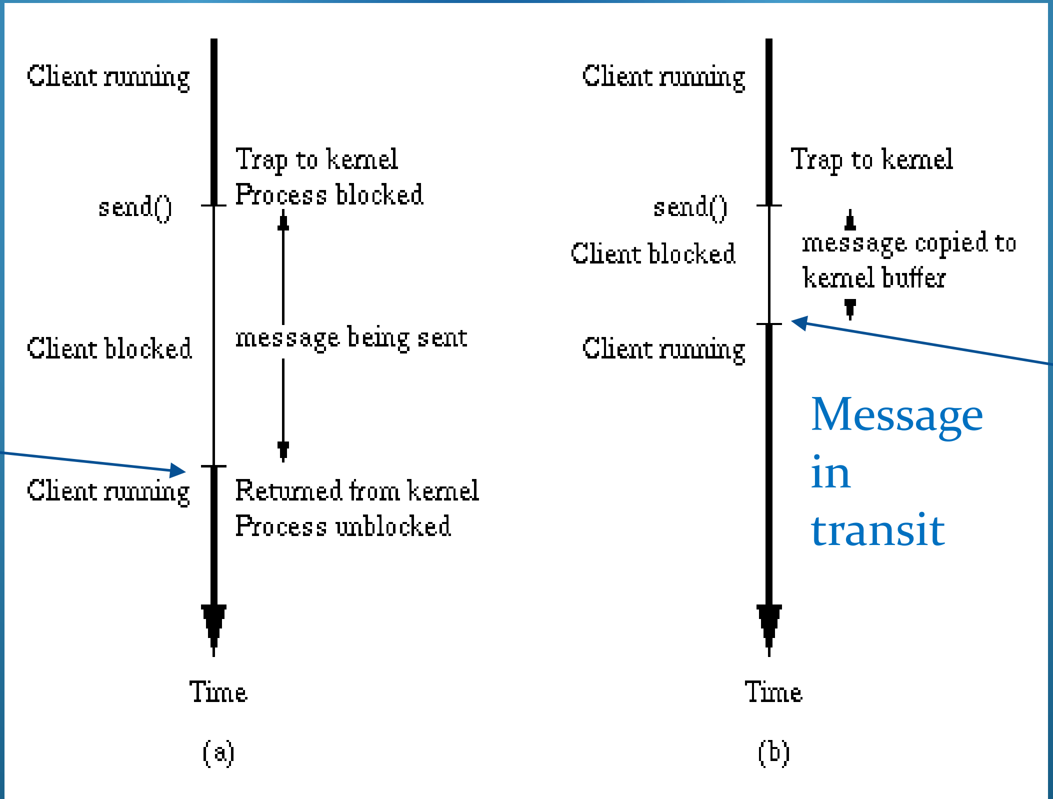
CS240 Operating Systems, Communications and Concurrency

Design Issues for Interprocess Communication

The implementation of some of these semantics requires the management of a message buffer/queue.

Blocking send() - one message in transit at a time

Client waits until each message is delivered



Client waits only for message to be received by local host

Non blocking send() – multiple messages can be in transit

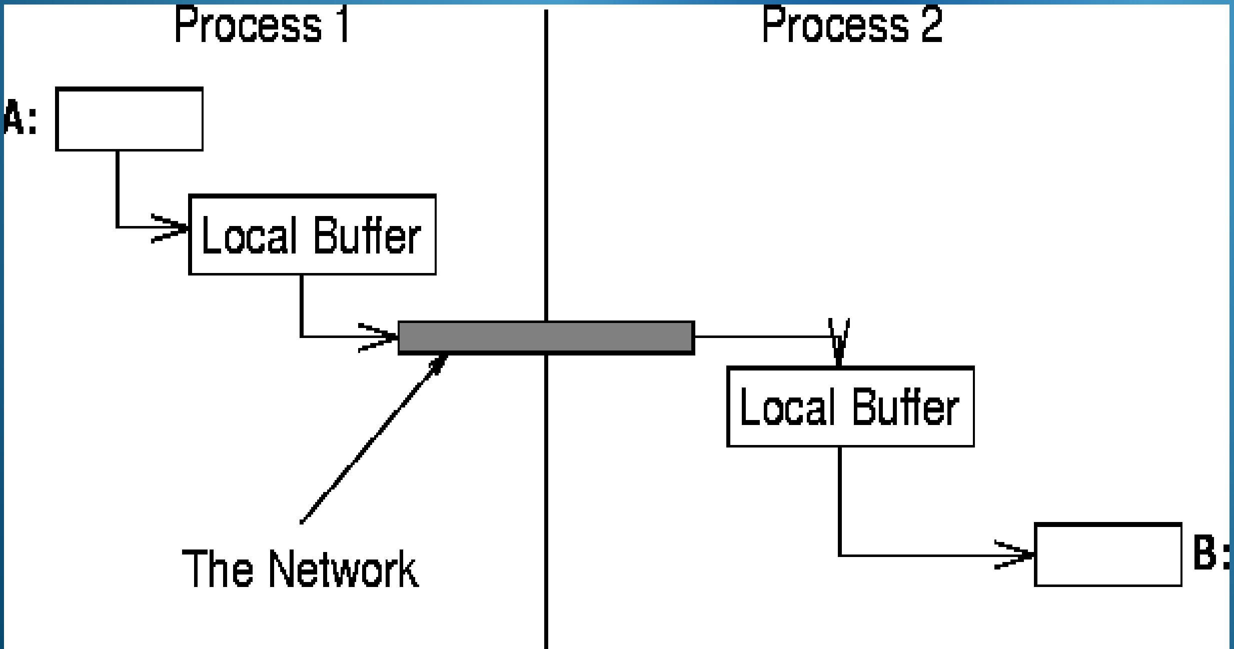
CS240 Operating Systems, Communications and Concurrency

Design Issues for Interprocess Communication

Buffering

What is the capacity of a Link?

Buffering is a requirement for implementing non blocking semantics. A link can have zero, limited or unlimited* capacity.



Buffering adds complexity and the need for message sequencing.

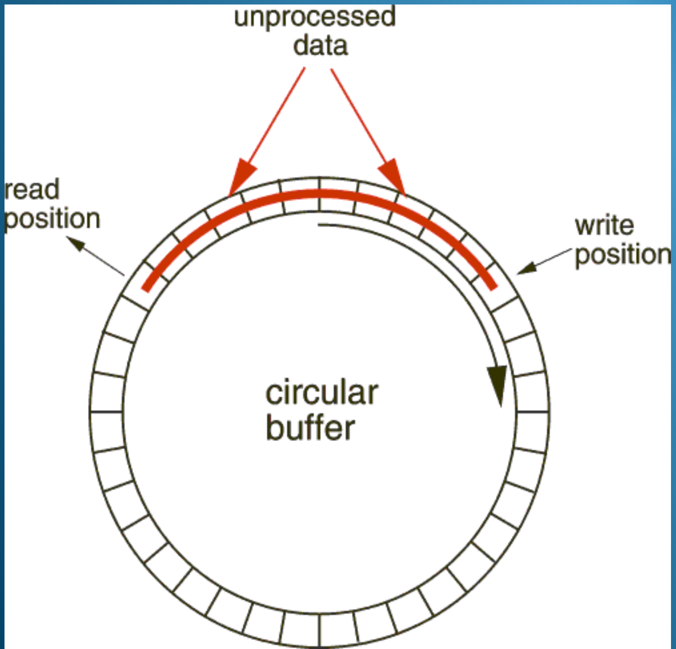
CS240 Operating Systems, Communications and Concurrency

Design Issues for Interprocess Communication

Message Parameters

What size should a message packet be?

Packets can be of fixed or variable size, typed or untyped messages.



Fixed sized messages easier to buffer and manage bandwidth.

Typed message useful in strongly typed language environments.

Design Issues for Interprocess Communication

Other issues - Reliability & Security

The mechanism must deal with the problems of networked communication such as lost and scrambled messages, and also with security issues.

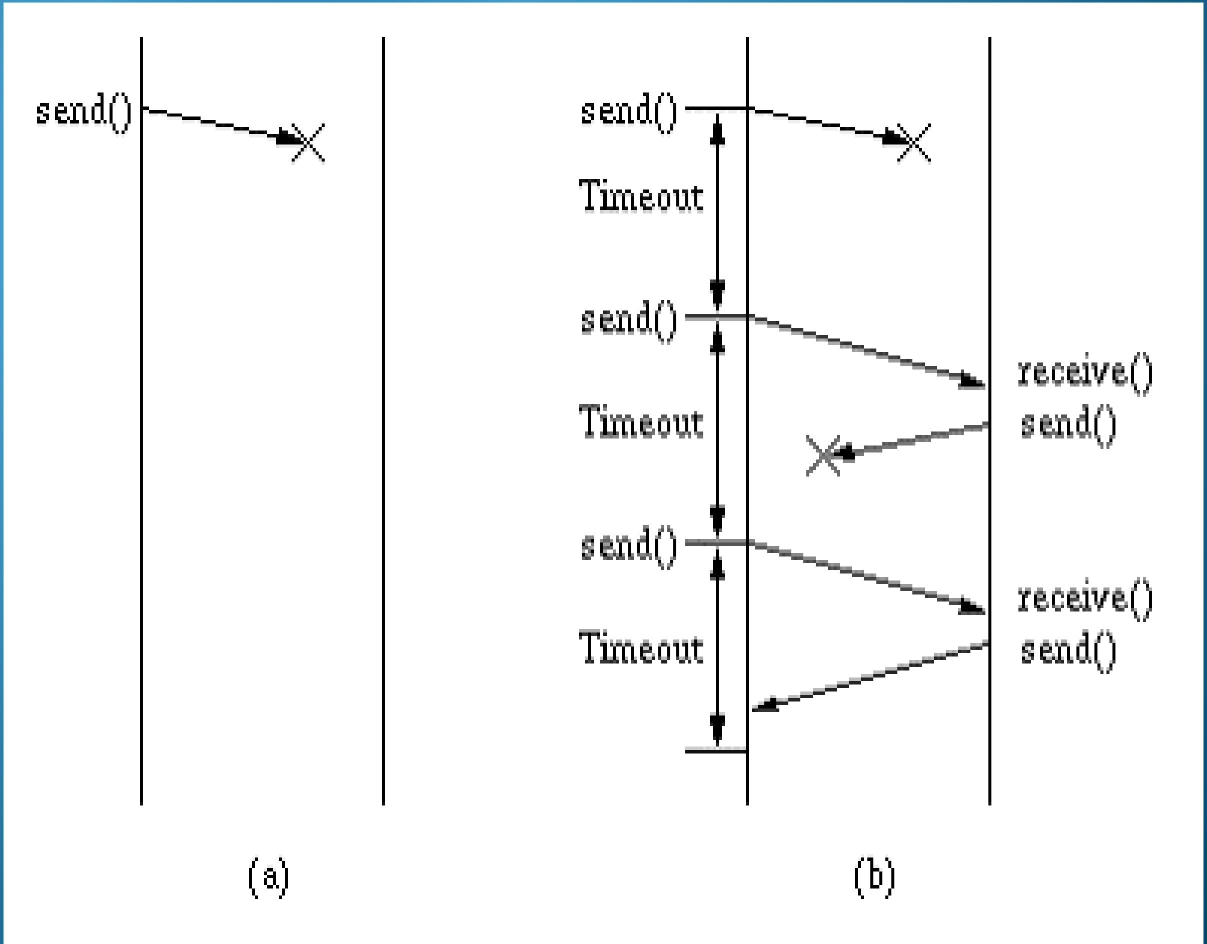
The system should be reliable, fast and easy to use for programmers.

CS240 Operating Systems, Communications and Concurrency

Design Issues for Interprocess Communication

Unreliable vs Reliable protocol.

A reliable transmission protocol will use sequence numbers and timeouts/acks/retransmission mechanisms to ensure message sequences arrive correctly.

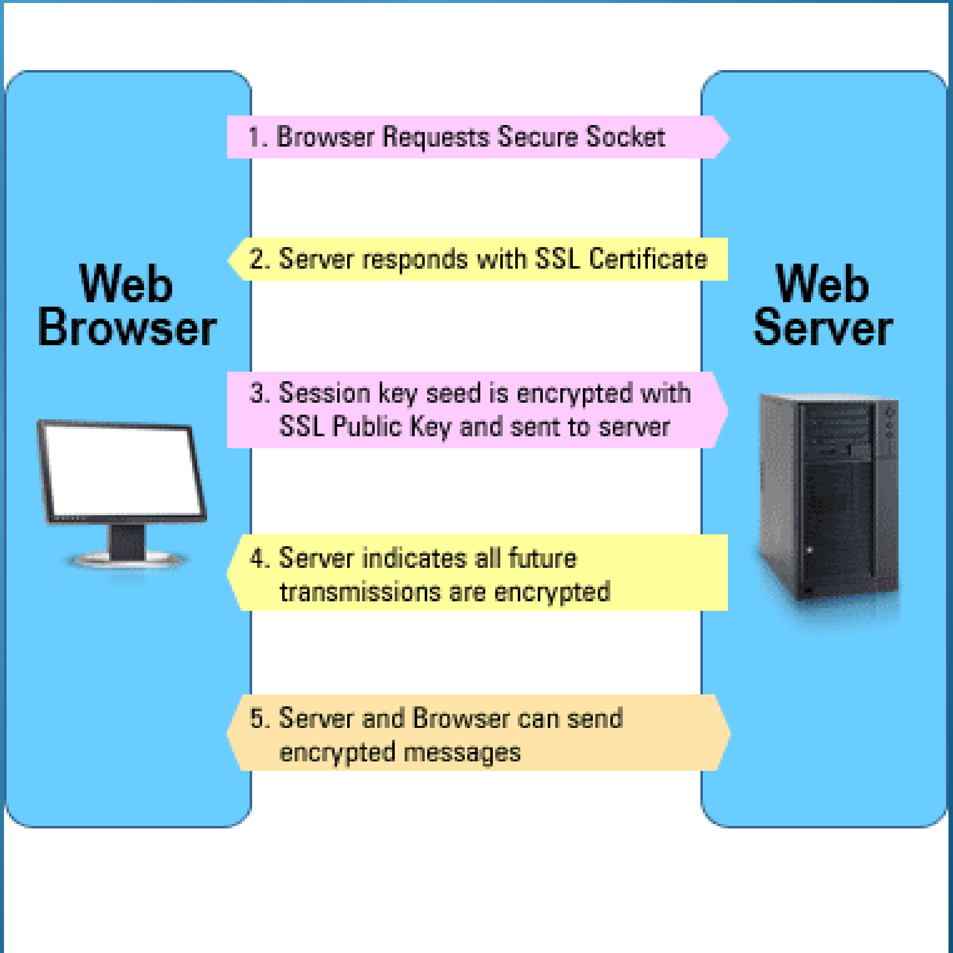


CS240 Operating Systems, Communications and Concurrency

Design Issues for Interprocess Communication

Eg. Encryption with SSL or IPSec

On a public network, it is often desirable for messages to be encoded so that only communicating parties can read them.


Web Browser
Web Server
1. Browser Requests Secure Socket
2. Server responds with SSL Certificate
3. Session key seed is encrypted with SSL Public Key and sent to server
4. Server indicates all future transmissions are encrypted
5. Server and Browser can send encrypted messages