Agenda

- Assignment 3 due Friday November 10
- Today’s lecture
  - IPC
    - Message Passing
    - Pipes
    - Signals
  - Memory management
  - Textbook Reading: 8.1-8.3
  - Next Lecture: Partitioning, Paging, Segmentation

Where we are…

- The OS provides three key abstractions
  - Process → CPU
  - Memory (Address) Space → RAM
  - Files → Secondary storage, Network, and Peripheral devices

Message Passing

- When processes interact with one another two fundamental requirements must be satisfied:
  - Synchronization
  - Communication
- Message passing is one approach to providing both of these functions
  - Works with distributed systems *and* shared memory multiprocessor and uniprocessor systems
- Features:
  - Post-office or telephone model
  - Processes cannot read each others’ memory
  - To communicate, processes ask the system to send a message on their behalf
  - Real life example: the Internet

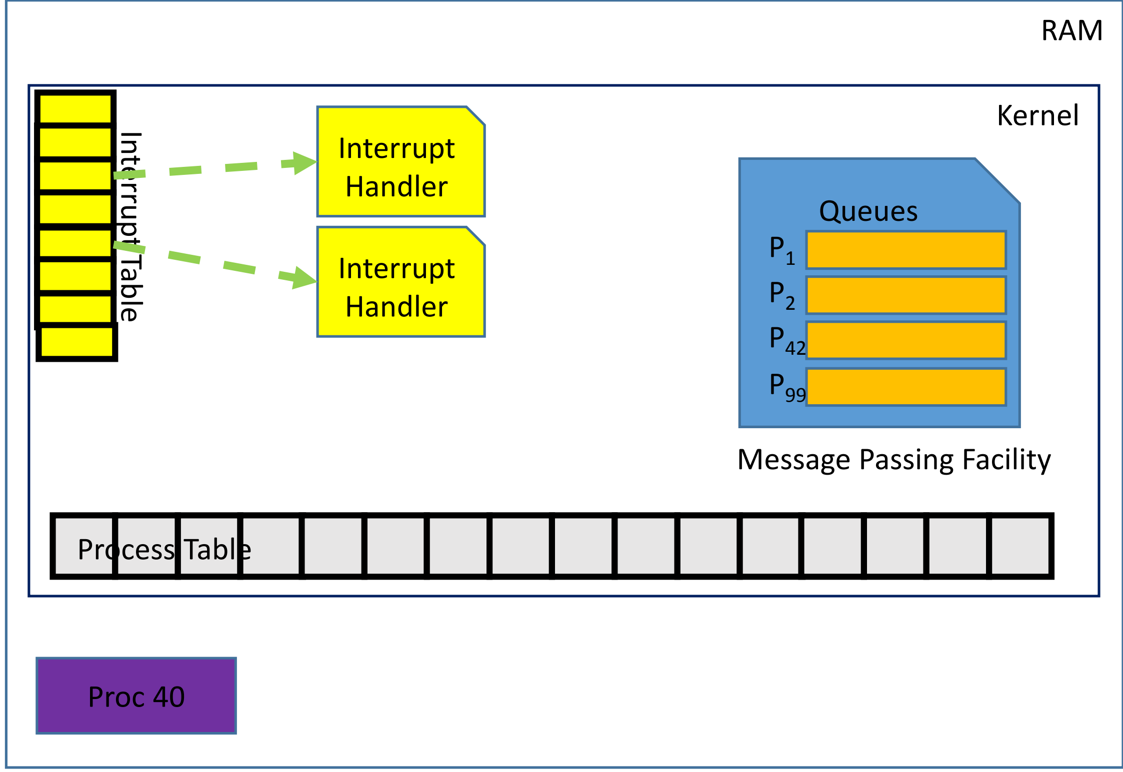
Message Passing Operation

- The OS provides a “Post-office” facility
- The actual function is normally provided in the form of a pair of primitives:

```
send (destination, message)
receive (source, message)
```

- A process sends information in the form of a *message* to another process designated by a *destination*
- A process receives information by executing the `receive` primitive, indicating the *source* and the *message*

The Message Passing Facility



- Queue P_i stores messages (or senders) to the process P_i
- The head of queue P_i is the next message to be received by P_i
- Sending processes:
  - must know the ID or name of the process they are sending to
  - ask the kernel to enqueue the message on the corresponding queue
- To receive a message, it is not (always) necessary for the process to explicitly specify the receiver
  - The process asks the kernel to remove a message from their queue and return it
- Analogy: A queue is the same as a mailbox

Message Passing – Indirect Addressing

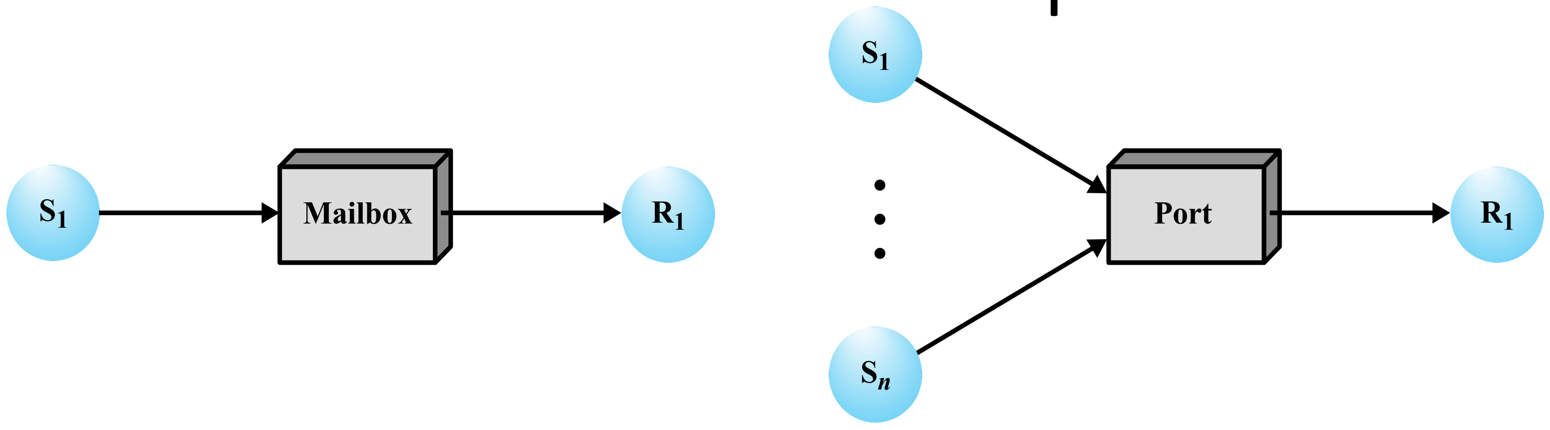
Messages are sent to a shared data structure consisting of queues that can temporarily hold messages

Queues are referred to as *mailboxes*

One process sends a message to the mailbox and the other process picks up the message from the mailbox

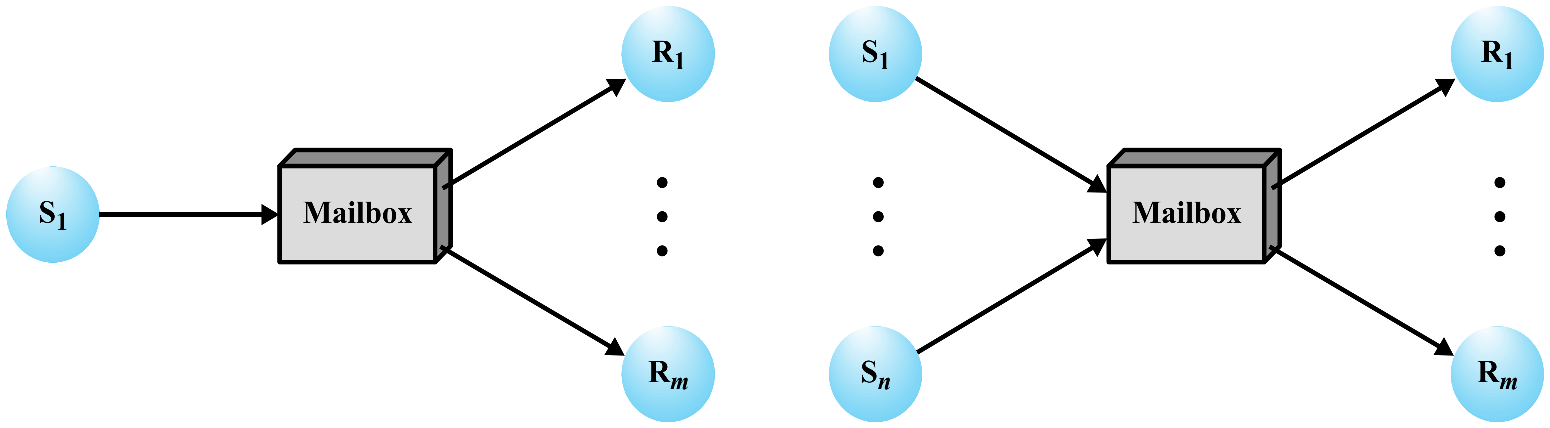
Allows for greater flexibility in the use of messages

Sender Receiver Relationship



(a) One to one

(b) Many to one

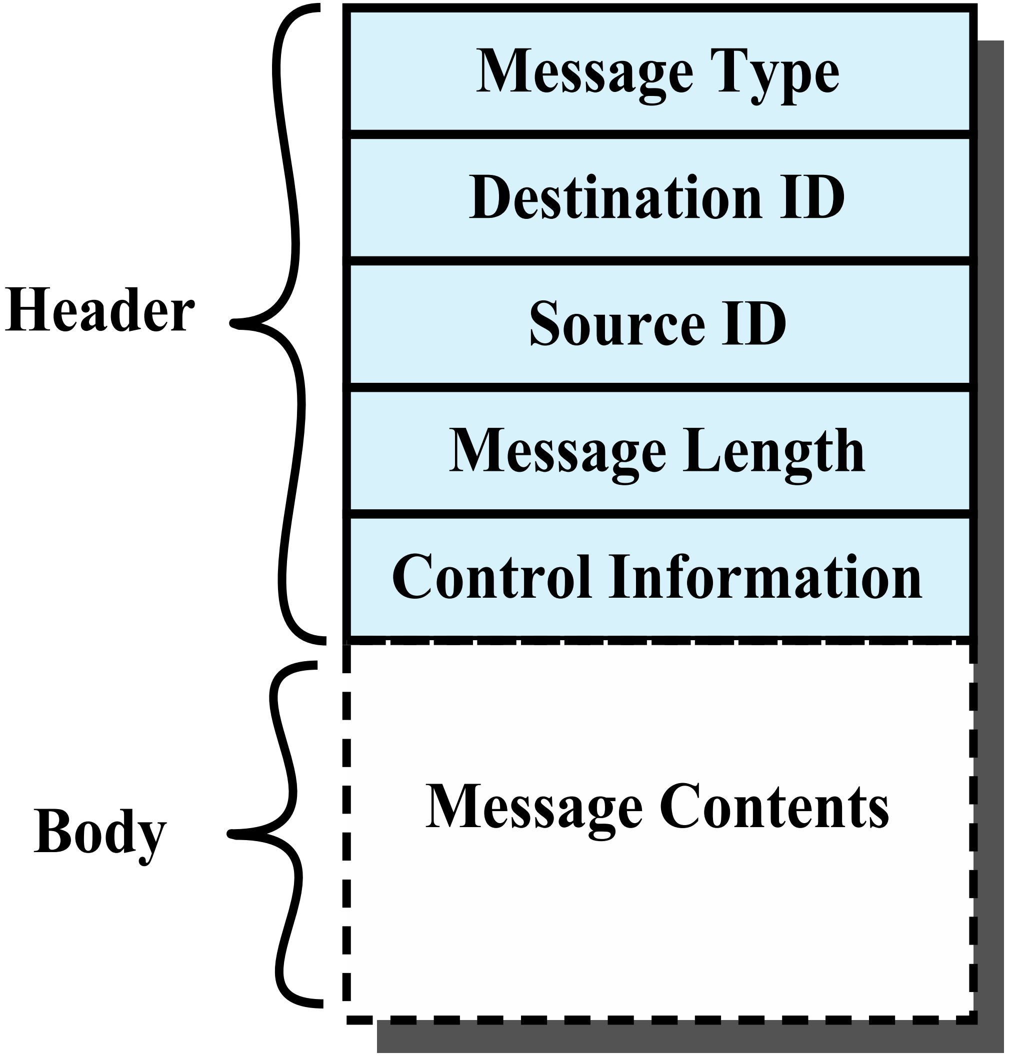


(c) One to many

(d) Many to many

Message Format

- The message is divided into two parts:
  - **header**, which contains information about the message
    - an identification of the source and intended destination of the message,
    - a length field, and a type field to discriminate among various types of messages
    - control information
  - **body**, which contains the actual contents of the message



Process Naming

- How processes are named and how process names are discovered is a whole topic beyond scope of this course

  Example: On the Internet
  - Processes are named by IP/port #
  - Use DNS to look up IP addresses and port #s of specific processes: e.g., 25 for e-mail (SMTP)
- Other Examples:
  - On a single system each process has unique process ID (pid) assigned by the OS
  - Each thread has a unique thread ID (tid), assigned by the OS or run-time system such as Java
- Another idea is to use a registry (e.g., DNS)

Synchronization

Communication of a message between two processes implies synchronization between the two

The receiver cannot receive a message until it has been sent by another process

When a receive primitive is executed in a process there are two possibilities:

If there is no waiting message the process is blocked until a message arrives or the process continues to execute, abandoning the attempt to receive

If a message has previously been sent the message is received and execution continues

Should Processes Block?

- Questions:
  - When a process sends a message, should it block until the message is received?
  - When a process receives a message, should it block until the message arrives?
- Answers depend on the system:
  - Typically, receivers will block until the message arrives
  - Whether senders should block depends on what type of behaviour is desirable:
    - Synchronous sends
    - Asynchronous sends

Blocking Send, Blocking Receive

- Both sender and receiver are blocked until the message is delivered
- Sometimes referred to as a *rendezvous*
- Allows for tight synchronization between processes
- At end of operation both sender and receiver have synchronized
  - i.e., Both know where they are
- Analogy: Telephone

Nonblocking Send

Nonblocking send, blocking receive

- Sender continues on but receiver is blocked until the requested message arrives
- Most useful combination
- Sends one or more messages to a variety of destinations as quickly as possible
- Example -- a service process that exists to provide a service or resource to other processes

Nonblocking send, nonblocking receive

- Neither party is required to wait
- Sender does not know when receiver gets the message
- Analogy: sending a letter

Producer Consumer Solution with Message Passing

```
void Producer() {
  while( 1 ) {
    p = produce();
    Send( consumer, p );
  }
}

void Consumer( ) {
  while( 1 ) {
    Recv( producer, p )
    consume( p );
  }
}
```

Pipes

- Special form of message passing
- Processes establish a direct connection (pipe) between themselves
- Can pass messages or data streams to each other.
- Pipes can either be
  - Local: both processes on the same machine
  - Interhost: (established via a network connection)
- Processes need to be able to identify each other

Signals

- Processes can asynchronously notify other processes
- All processes running on the same machine
- All processes controlled by the same OS
- One process asks the OS to notify another process on its behalf

Memory Management

Objectives

- Discuss requirements for memory management
- Various memory-management techniques
  - Partitioning
  - Paging
  - Segmentation

Motivation

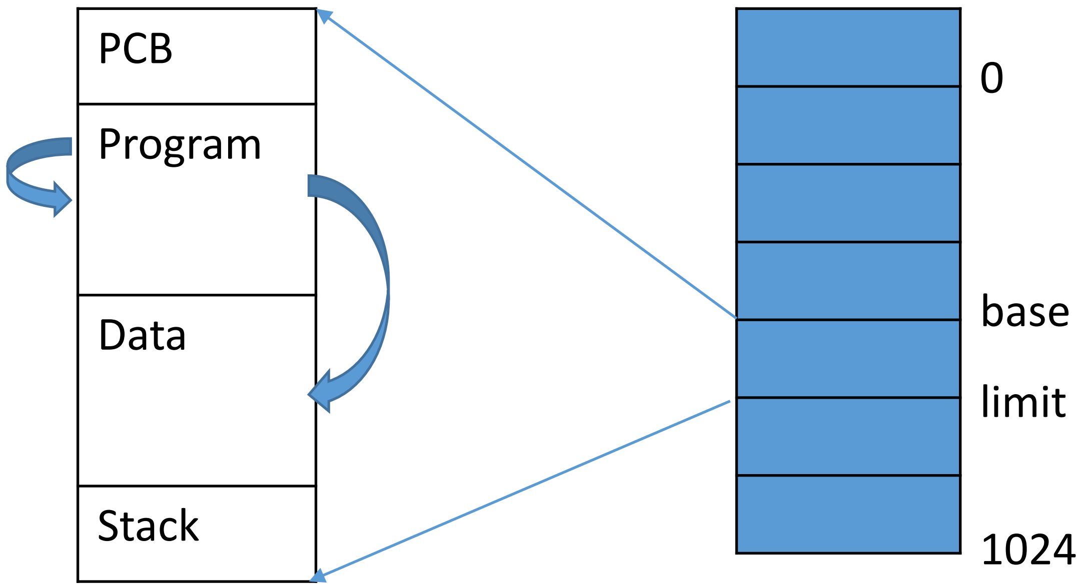
- Main memory and registers are only storage CPU can access directly
  - Register access in one CPU clock (or less)
  - Main memory can take many cycles
  - **Cache** sits between main memory and CPU registers
- Memory is allocated when
  - A process is created
  - A process needs more heap/stack
  - The OS needs to use cache
  - The OS needs to allocate data structures

Motivation

- Memory is a finite resource and needs to be managed
  - Fair allocation among processes (sharing)
  - Efficient allocation --> OS has enough memory
  - Protection of memory required to ensure correct operation
- Issues:
  - Program may not be at address 0x00000000h
    - Linking and loading issues
  - Virtual memory may be used
    - Mapping and paging issues
  - Caching issues
    - Components may be duplicated
- How does the OS handle these situations?

What is Memory?

- Computer memory consists of a linear array of addressable storage cells that are similar to registers
- Program must be brought (from disk) into memory and placed within a process for it to be run
  - Each process has a base/limit address to specify its accessible addresses
  - ”segmentation fault” error generated when trying to access memory outside of the allowable address space

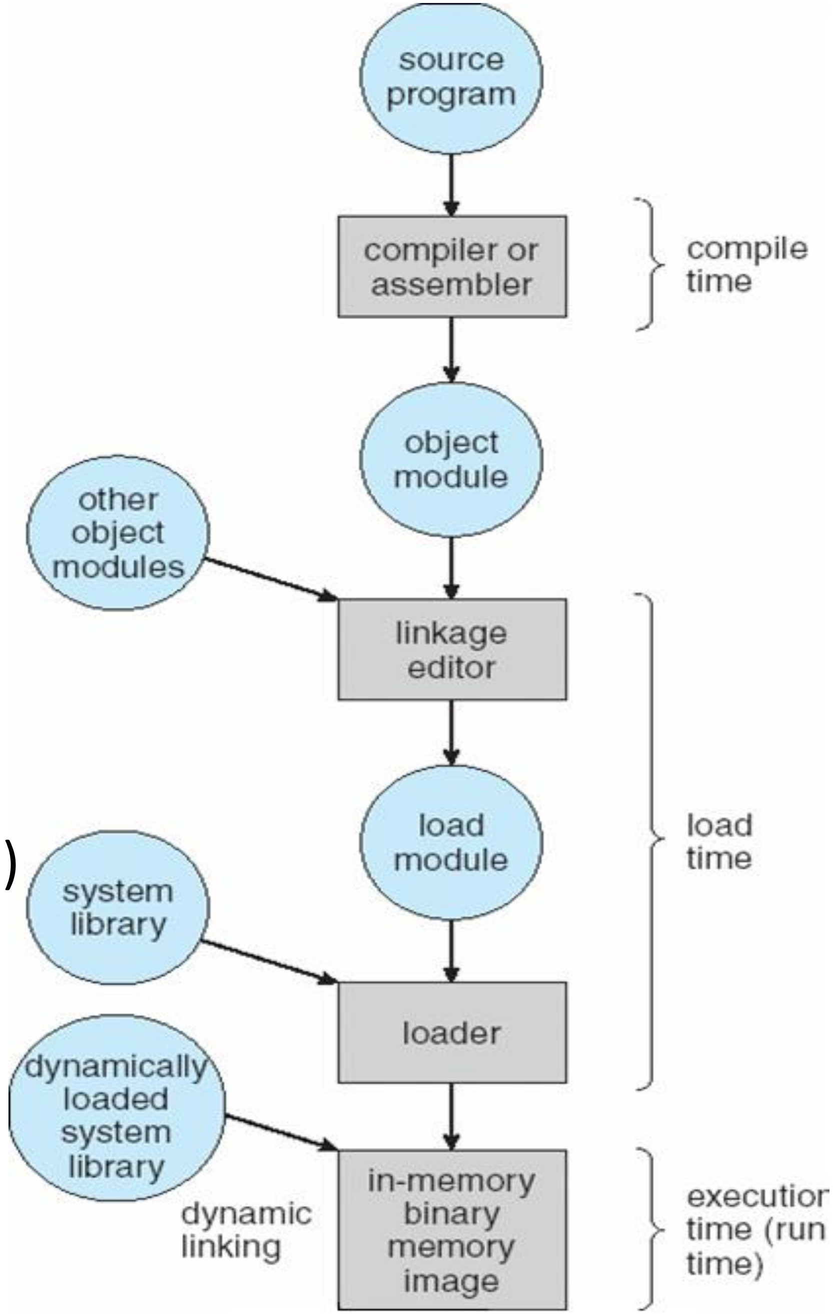


Addressing requirements for a process

Memory array

Address Binding

- Linking a symbolic address in a program to an actual memory location in main memory
- **Compile time**
  - The compiler puts physical addresses in the program (**absolute code**)
- **Loading time**
  - Compiler uses offsets from some base which is determined at load time
  - The addresses are inserted by the loader (**relocatable code**)
- **Execution time**
  - Need hardware support (base and limit registers) to map addresses
- Most modern Oss use virtual addressing instead



Memory Management Goals

- Make sure each process has sufficient memory
- Keep as many processes in memory as possible
- Allocate memory efficiently
    - Allocate as much as is needed for a process – not more
    - Leave some free space for new starting processes
    - Maximize memory utilization
- Memory is a finite resources
- These goals cannot be simultaneously met

OS Strategies

- Swapping
- Segmentation
- Paging
- Virtual memory

We will discuss those strategies in the next few lectures.