Week 11

XMUT-NWEN 241 - 2024 T2

Systems Programming

Felix Yan

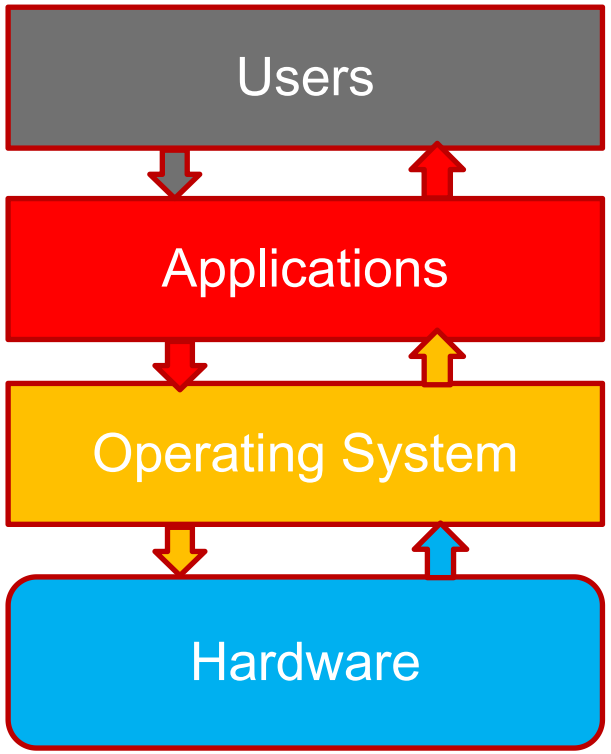
School of Engineering and Computer Science

Victoria University of Wellington

Content

- System calls
- Interprocess communication

System calls - What and Why?



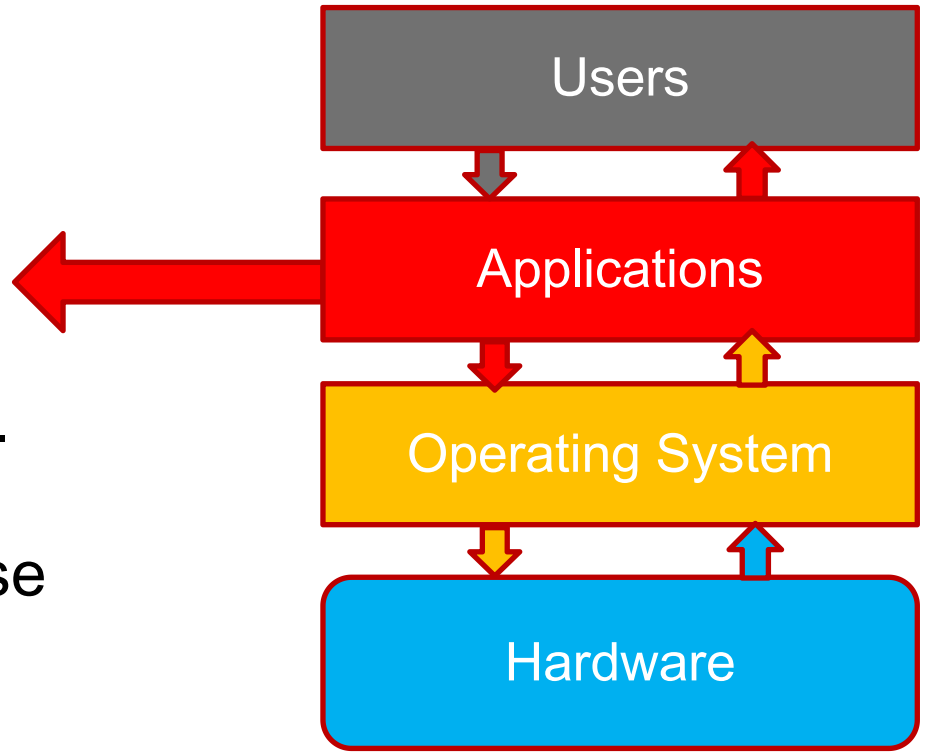
Conceptual View of a Computer System

System calls - What and Why?

- Typically needs access to **system resources**.

- System resources can be:
  a) **physical** – e.g. input devices, screen displays.
  OR
  b) **Virtual** – e.g. files, network connections, threads.

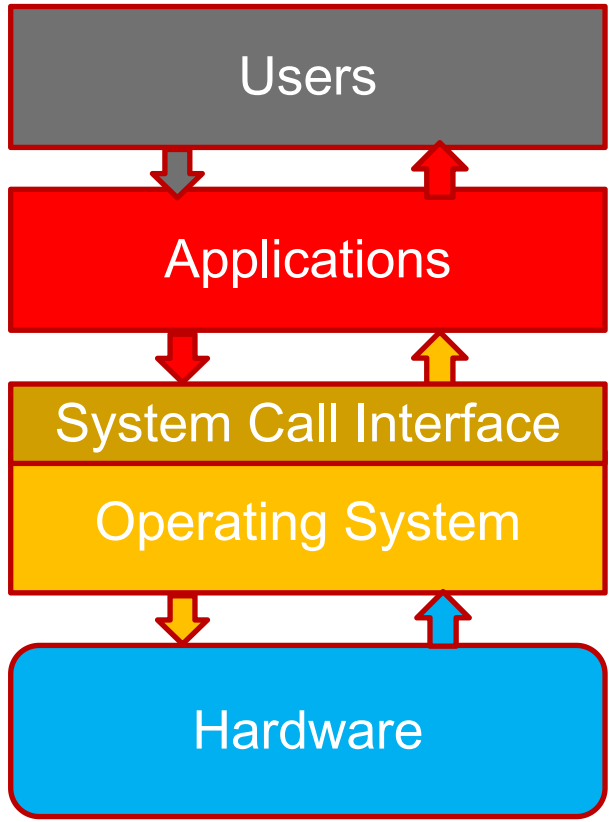
- Applications need O.S. to enable them access these resources.



Conceptual View of a Computer System

System calls - What and Why?

- Operating Systems **do not** **allow application software to access system resources** **directly** due to security and reliability issues.
- A program can **request** the services of system resources from O.S through **system calls**.
- **System calls** are **function invocations made from application into the OS** in order to request some service or resource from the operating system.
- Application developers often do not have direct access to system calls but can access them through a **system call API**, which in turn invokes the system call.



An example of a system call usage

- Consider the following example:

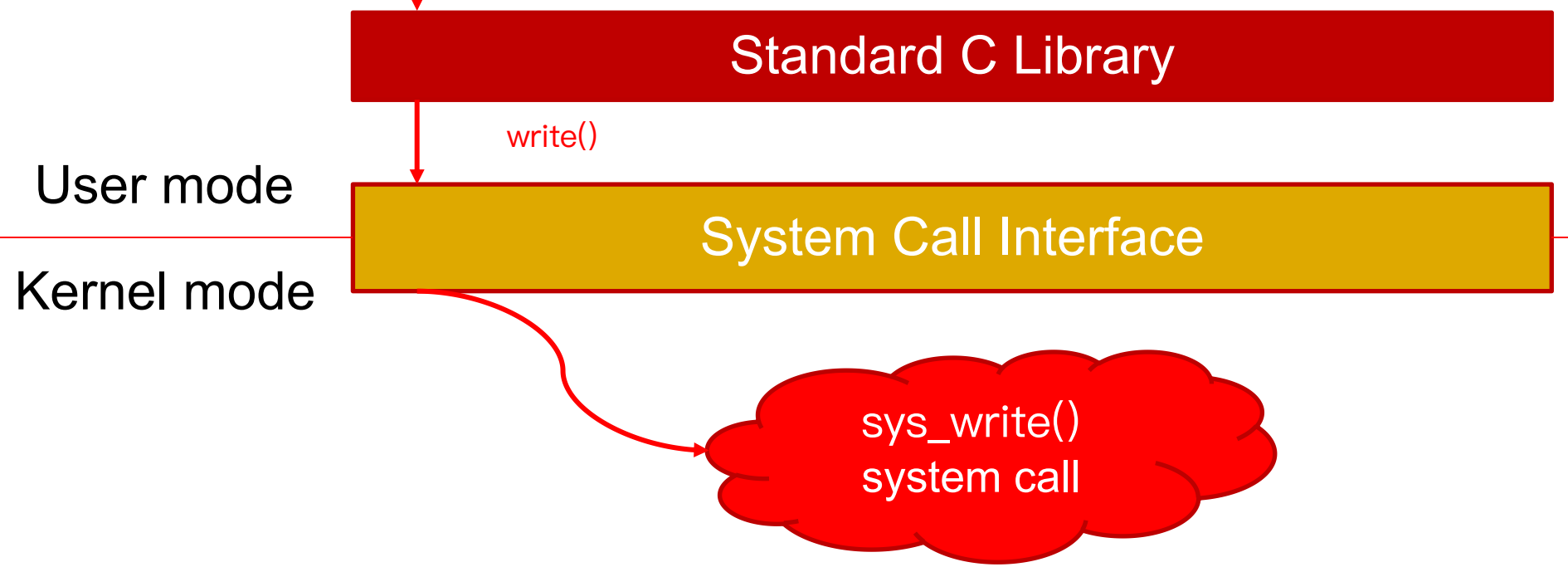
```
#include<stdio.h>

int main()
{
  printf("Hello World");
  return 0;
}
```

C Library function **printf** **"asks" the operating system to print for the calling program** by using the system call API routines

System call invocation –*Example*

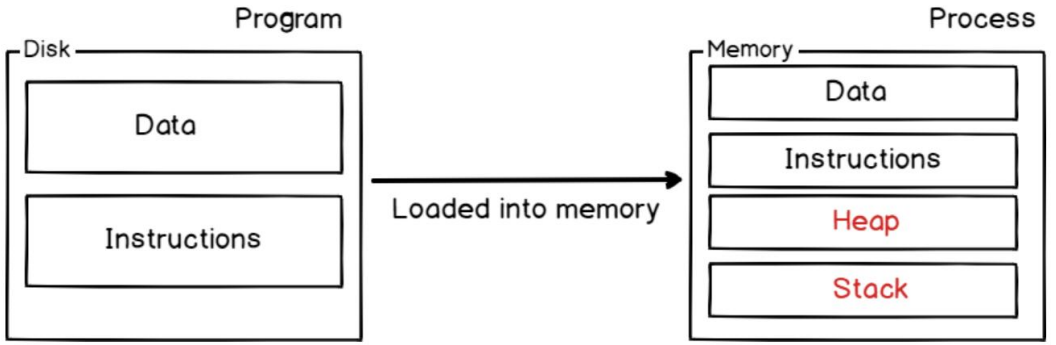
```
#include <stdio.h>
void main(void)
{
    printf("Hello, world\n");
    exit(0);
}
```



Interprocess Communication

What is a process ?

- Program and process are related terms.



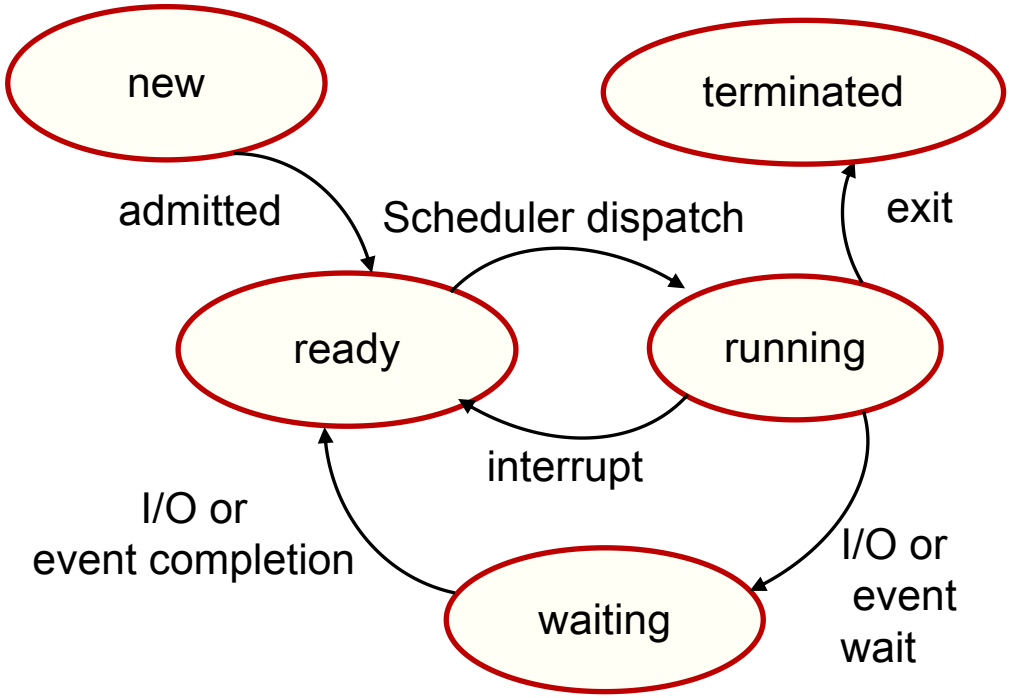
Program is a set of instructions to carry out a specified task	**Process** is a program in execution

Passive entity	Active entity
Program is a stored in disk and does not require any other resource.	**Process** requires system resources such as CPU, memory, I/O etc.
Life span - Longer	Life span – limited
Each time a program is run a new process is created.	

Process lifecycle

As a process executes, it changes **state**

- **new**: The process is being created
- **ready**: The process is waiting to be assigned to a processor
- **running**: Instructions are being executed
- **waiting**: The process is waiting for some event to occur
- **terminated**: The process has finished execution



Process management system calls

The following system calls are used for basic process management.

- fork()
- exec()

Defined in unistd.h

- wait()

Defined in sys/wait.h

- exit()

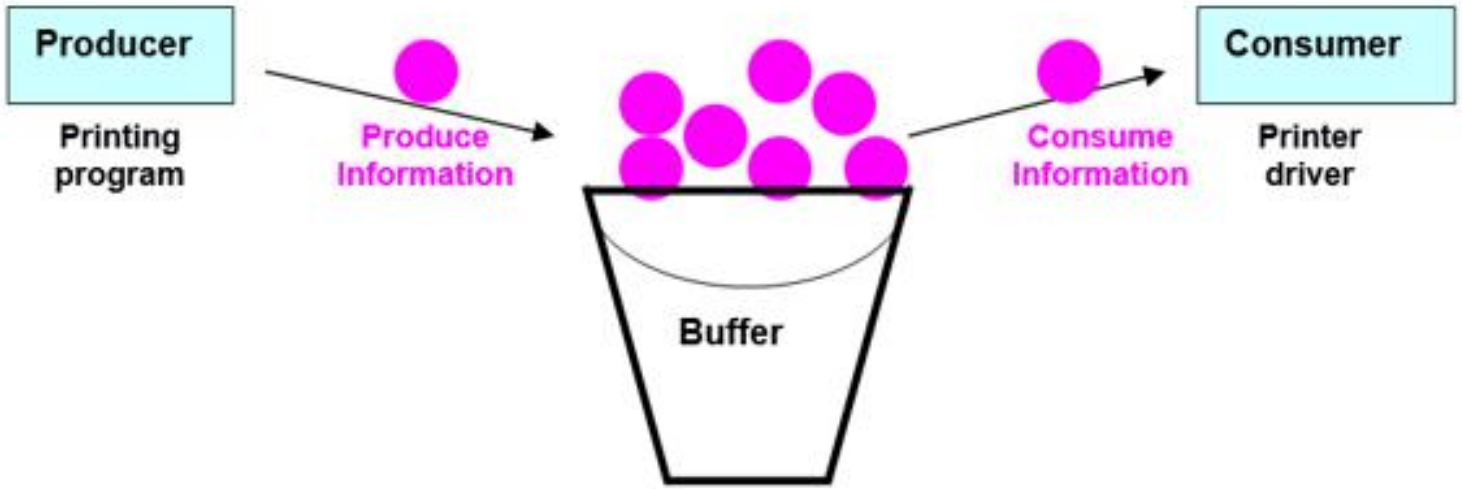
Defined in stdlib.h

Process - Independent Vs Cooperating

- **Independent** processes: processes that don't interact with other processes
- **Cooperating** processes: process can affect or be affected by other processes.
- In order to co-operate processes, need to **communicate**
  - **Inter Process Communication**

Cooperating Processes

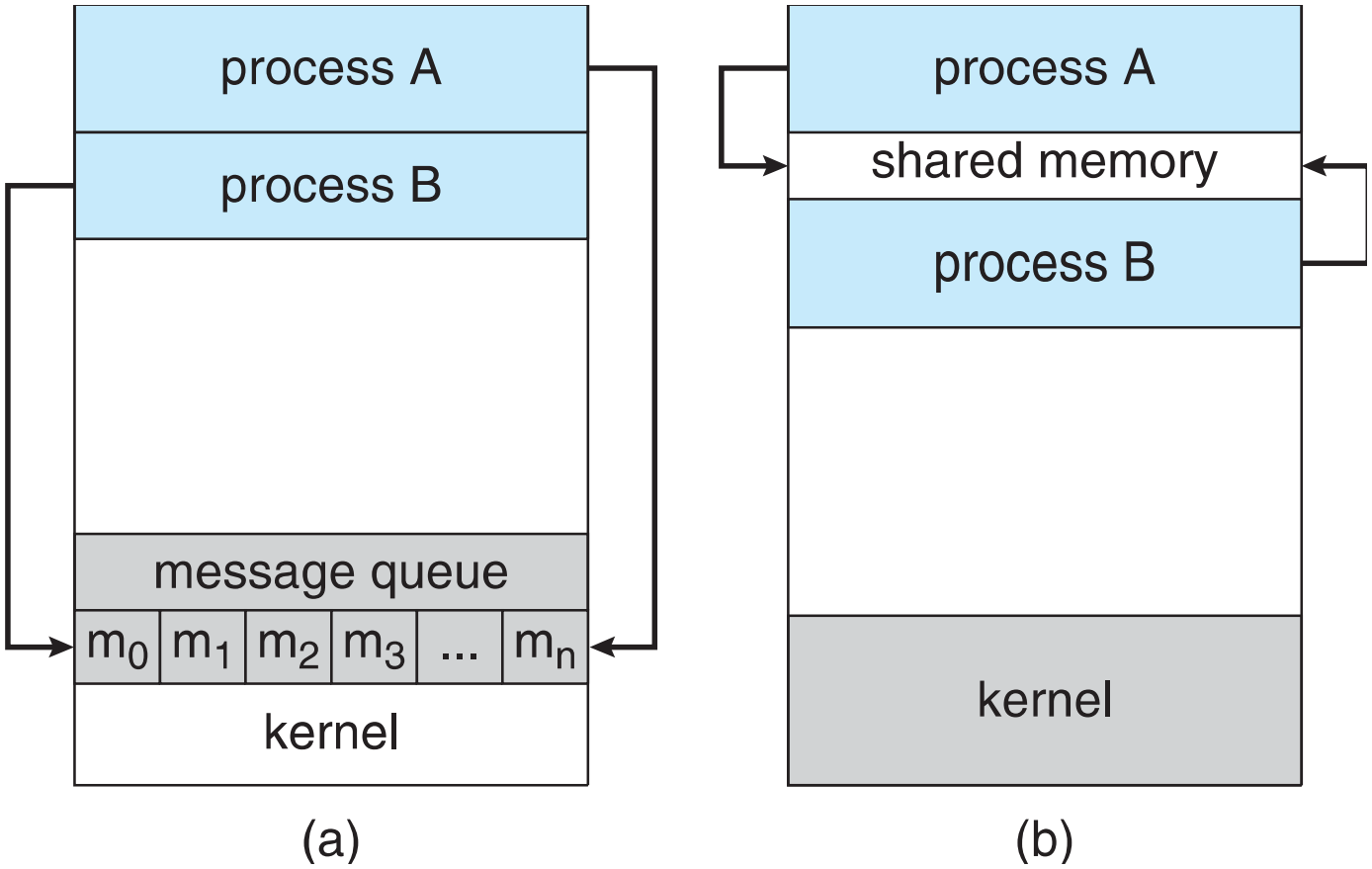
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience



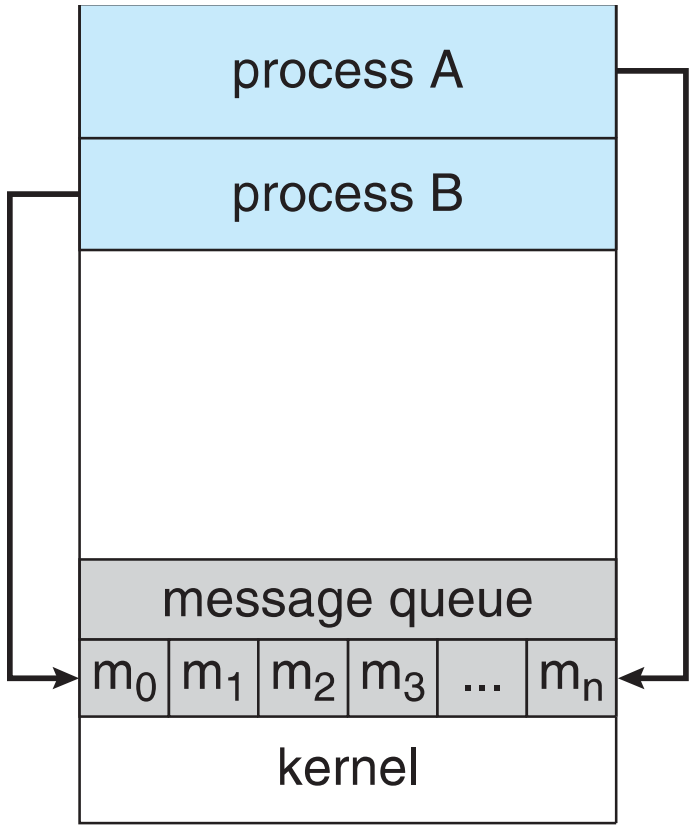
- Cooperating processes can reside on **same machine** or in **different machines (on a network)**.

Inter-process communication

- Cooperating processes need **Inter-process communication** (**IPC**)
- Two primary models of IPC
  - **Message passing**
  - **Shared memory**



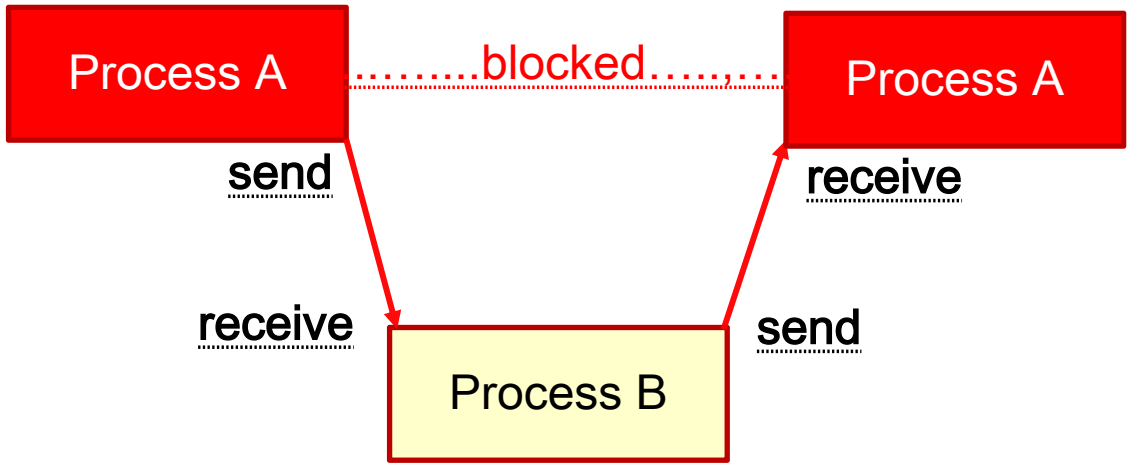
Message passing



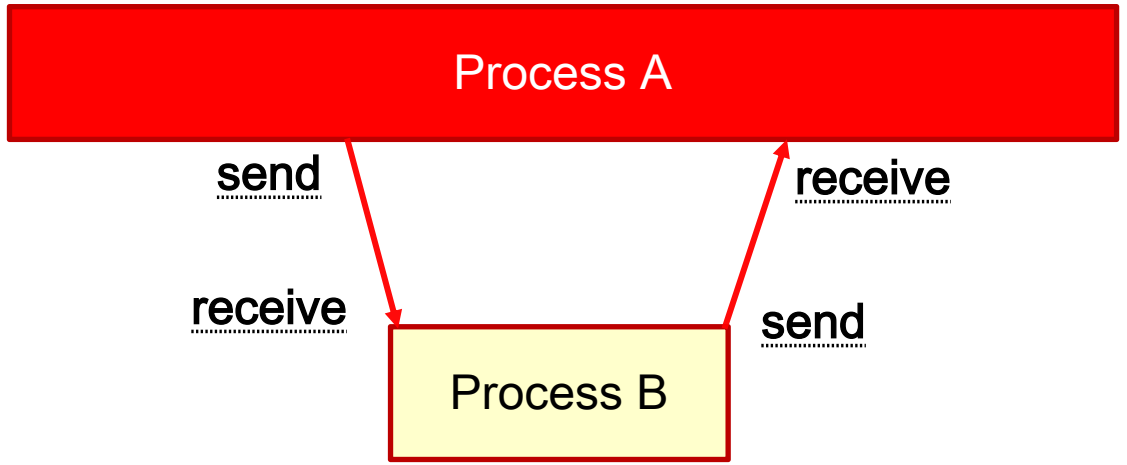
- Processes communicate with each other without resorting to shared variables
- IPC facility provides two primitive operations:
  - **send**(*message*)
  - **receive**(*message*)
- If *A* and *B* wish to communicate, they need to:
  - establish a ***communication link*** between them
  - exchange messages via send/receive

Design options - Synchronization

Blocking (Synchronous)



Non-blocking (Asynchronous)



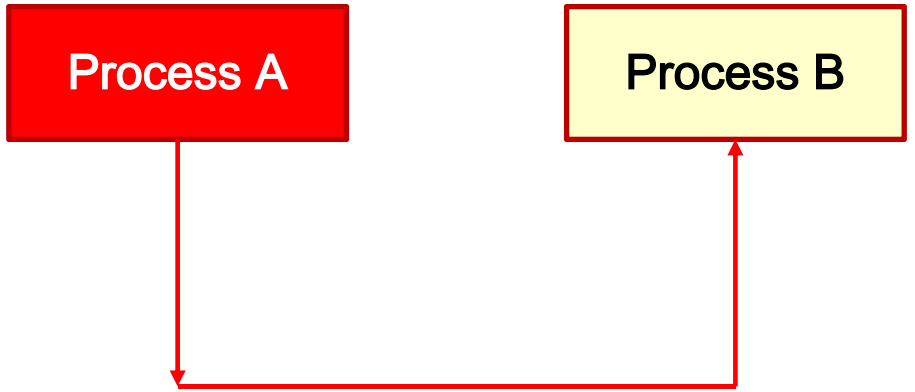
Design options - Synchronization

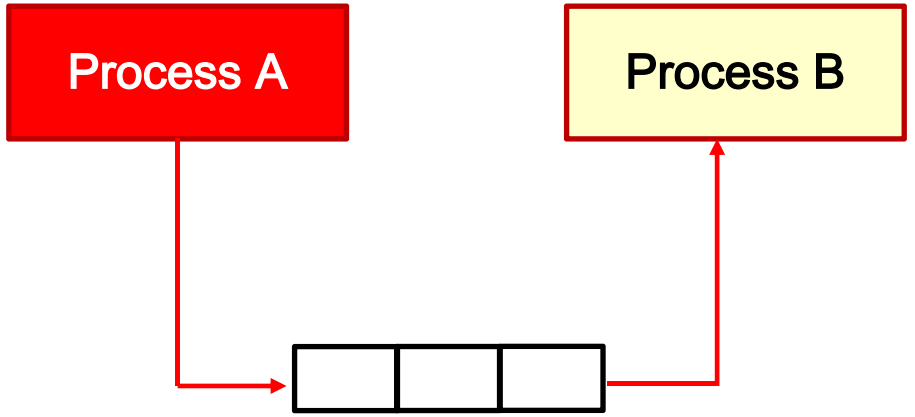
	Blocking	Non - Blocking
Send	Has the sender block until the message is received	Has the sender send the message and continue
Receive	Has the receiver block until a message is available	Has the receiver shown its willing to receive message and continue

Different combinations possible

Design options - Buffering

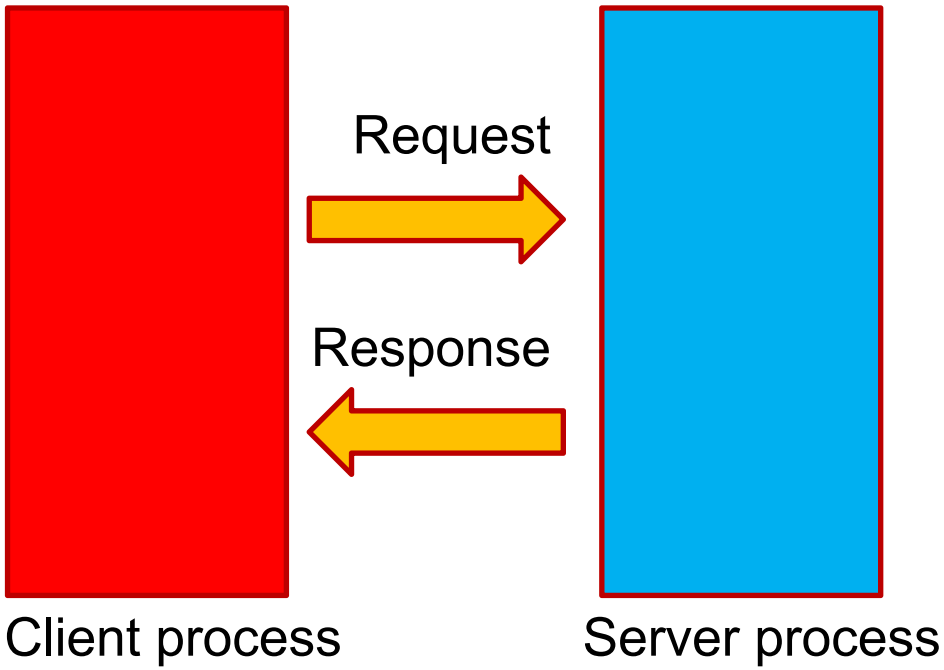
- **Queue** of messages attached to the link
- Implemented in one of three ways:
  - **Zero capacity** – 0 messages
    Sender must wait for receiver
  - **Bounded capacity** – finite length of *n* messages
    Sender must wait if link full
  - **Unbounded capacity** – infinite length
    Sender never waits





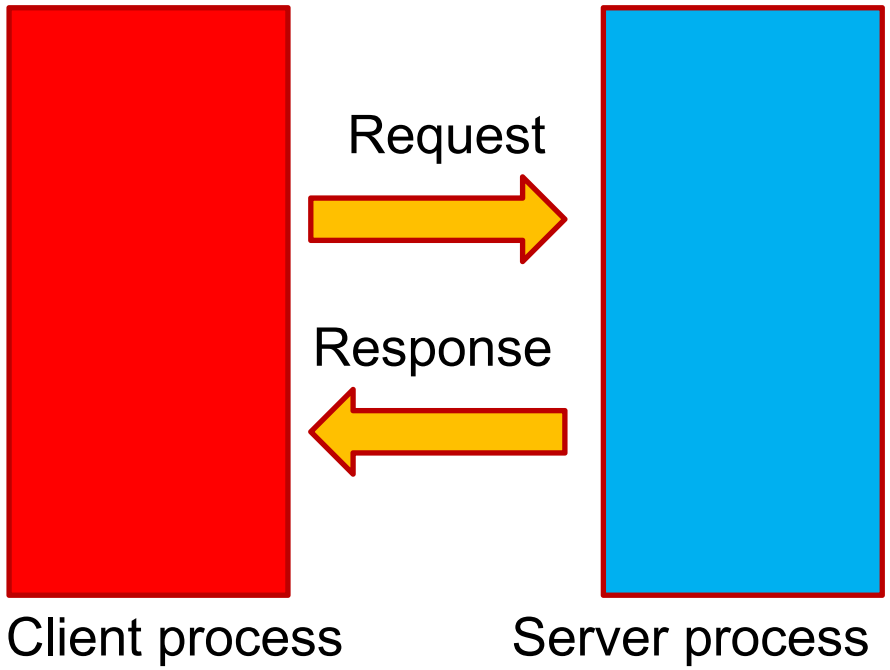
Client-server model

- Most common IPC paradigm
- Based on the producer-consumer model of process cooperation
- Client makes the request for some resource or service to the server process
- Server process handles the request and sends the response (result) back to the client



Client-server model

- **Client process** needs to know the existence and the address of the server
- However, the **Server** does not need to know the existence or address of the client prior to the connection
- **Once a connection** is established, both sides can send and receive information



Next Lecture

- TCP Socket Programming