Inter-Process Communication

CPEN333 – Software Design for Engineers II
2023 W1
University of British Columbia

Electrical and Computer Engineering

Introduction

- We have already discussed cooperating processes.

- Cooperating processes need **interprocess communication** (**IPC**) that would allow them to exchange data and information.

- In this set of slides, we examine different mode of communications and methods.

Objectives

- To describe the communication within processes
  - message passing
  - shared-memory
- To describe communication in client-server systems

Cooperating processes

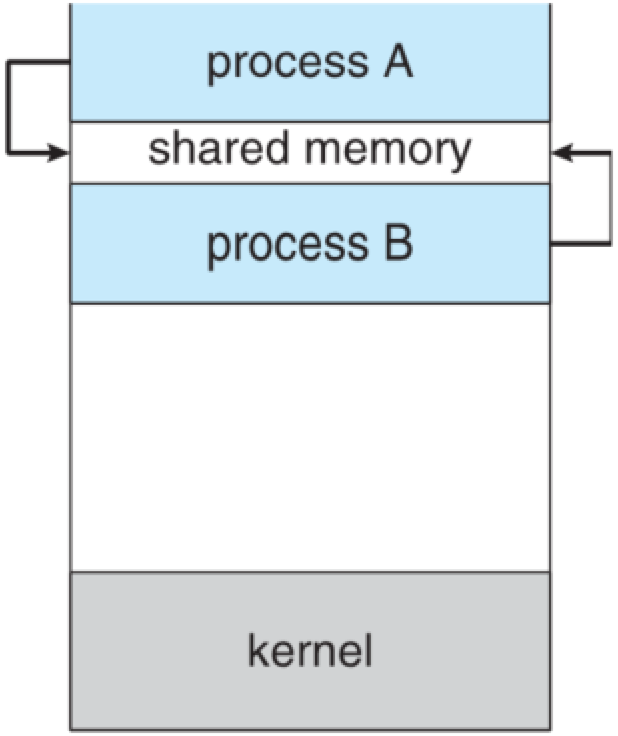
- Reasons for cooperating processes could be:
  - Information sharing: e.g. several users may be interested in the same piece of information
  - Computation speedup: e.g. breaking a particular task into subtasks being executed in parallel
  - Modularity: e.g. dividing a system functions into separate processes in a modular fashion
  - Convenience: even an individual user may work on many tasks at the same time

Communications Models

- Two main models of IPC
  - Shared memory: a region of memory is used that is shared by the cooperating processes
  - Message passing: Communication takes place by means of messages exchanged between cooperating processes
- Both of these two models are common in the OSs, and many systems implement both.

Shared Memory Systems

- Recall that, normally, the OS tries to prevent one process from accessing another process's memory.
  - Shared-memory requires that two or more processes agree to remove this restriction.

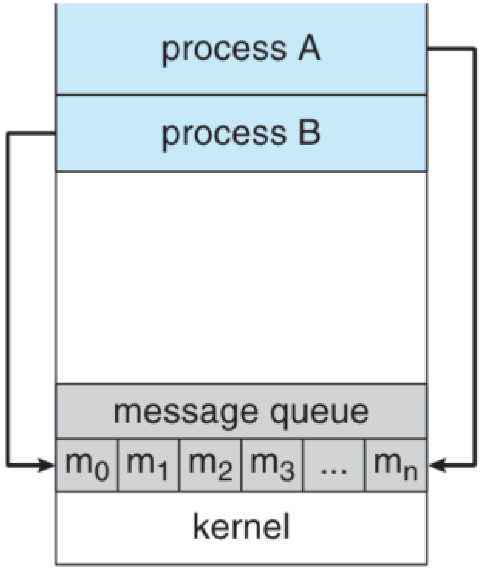


Shared Memory (cont.)

- Usually the shared-memory resides in the address space of the process creating the shared-memory segment
- Then those processes can exchange information by reading and writing data in the shared areas (not under the OS's control)
- Synchronization may be needed, of course: e.g., the processes are responsible for ensuring that they are not writing to the same location simultaneously.
  - We have already seen this as an example in the producer-consumer problem.

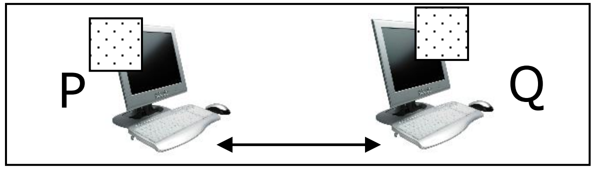
Message Passing Systems

- Message passing is the other method that provides a mechanism for processes to communicate and to synchronize their actions
  - Processes communicate with each other without resorting to shared variables
  - The operating system is to provide the means for cooperating processes to communicate with each other via a message-passing facility.

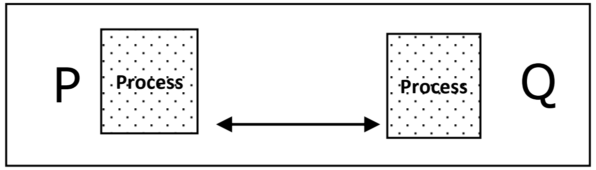


Message Passing (cont.)

- A general message passing facility provides at least two operations:
  - **`send(message)`** and **`receive(`*`message`*`)`**
- If two processes *P* and *Q* wish to communicate, they need to:
  - establish a *communication link* between them
  - exchange messages via send/receive
- A particularly useful and practical method in distributed environment (e.g. chat programs)



P and Q on two different machines



P and Q on the same machine

Implementation Questions

- So far, we have discussed general message passing concepts.
- For the implementation, we might decide on several options.
- How are links established?
  - A communication link must exist between the cooperating processes.
  - The link can be
    - physical, e.g. a hardware bus, a communication network (LAN or Internet)
    - logical, e.g. a message passing queue
- Can a link be associated with more than two processes?
- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bi-directional?

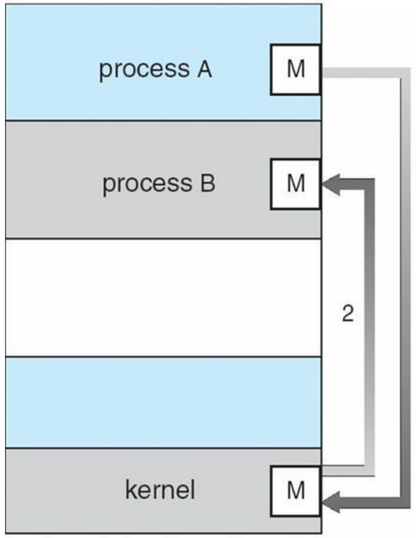
Blocking vs non-blocking

- Communication between processes takes place through calls to **`send()`** and **`receive()`** primitives
- There are different design options for implementing each primitive:
  - Message passing may be either blocking or non-blocking
- Blocking is considered *synchronous*
  - **Blocking send** has the sender block until the message is received
  - **Blocking receive** has the receiver block until a message is available
- Non-blocking is considered *asynchronous*
  - **Non-blocking send** has the sender send the message and continue
  - **Non-blocking receive** has the receiver receive a valid message or a null
- Different combination of **`send()`** and **`receive()`** are possible. When Both send and receive are blocking, we have a rendezvous between them.

Buffering

- Whether communication is direct or indirect, messages exchanged by communicating processes reside in a temporary queue.

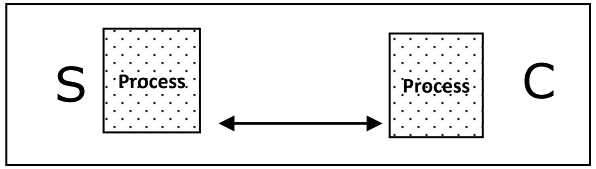
- Such queues can be implemented in one of three ways
  - Zero capacity (queue max length is zero, i.e., 0 messages)
    - Sender must wait for the receiver
  - Bounded capacity (finite length of n messages)
    - Sender must block if the link is full, otherwise it can continue without waiting
  - Unbounded capacity (infinite length)
    - Sender never waits

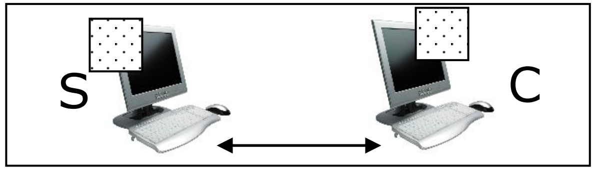


Communications in Client-Server Systems

- So far, it was described how processes can communicate using:
  - shared memory and
  - message passing

- Strategies for communication in client-server systems:
  - Sockets
  - Pipes
  - Remote Procedure Calls (RPC)





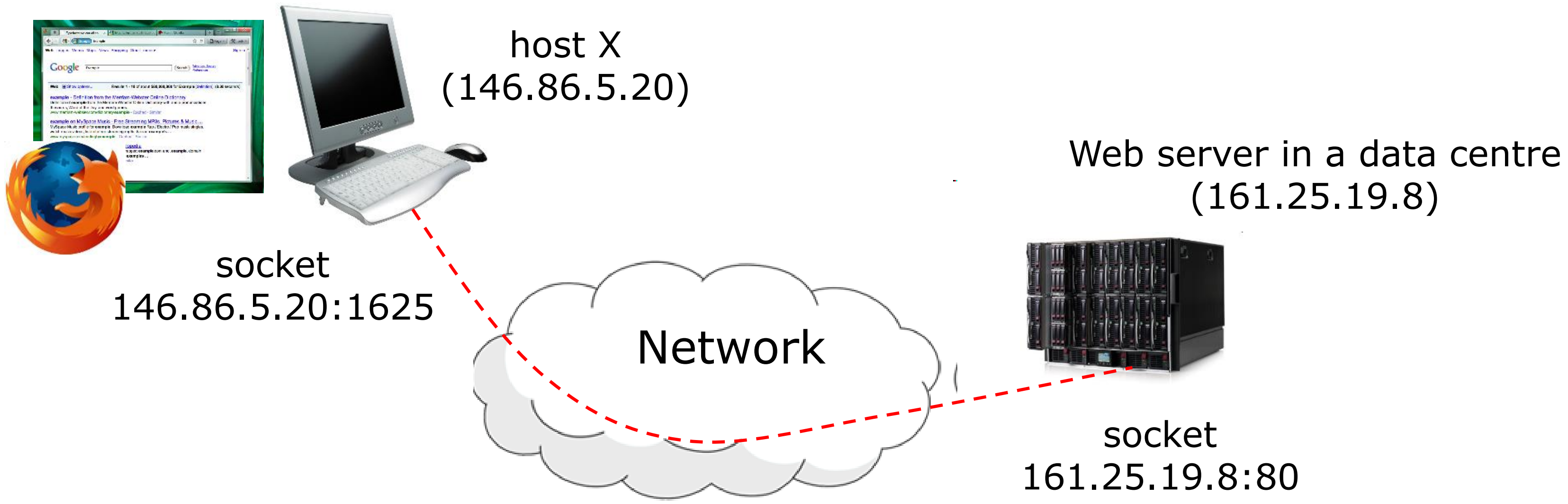
Client-server model

- The primary model used in the Internet: email, web, …

- The client-server describes the relationship of the cooperating processes.
  - A server is the service provider, and generally is constantly awaiting to receive incoming requests to provide service.
    - A servers is described by the service it provides, e.g. a file server or a web server.
  - Clients are service requesters and initiate communication sessions with the server to receive service.
    - Similarly we have email client, web client (e.g. a web browser), …

Sockets

- A socket is defined as an *endpoint for communication*
  - It is identified by an IP address concatenated with a port number, in its simplest form.
    - e.g., the socket **146.86.5.20:1625** refers to port **1625** on host **146.86.5.20**
- A pair of processes communicating over network employ a pair of sockets.



TCP vs UDP sockets

- Two very popular types of sockets are TCP and UDP.
  - TCP (transmission control protocol) and UDP (user datagram protocol) are the Internet's primary transport protocols.

- TCP sockets are *connection-oriented*, that is a connection is first established and then data can transfer.

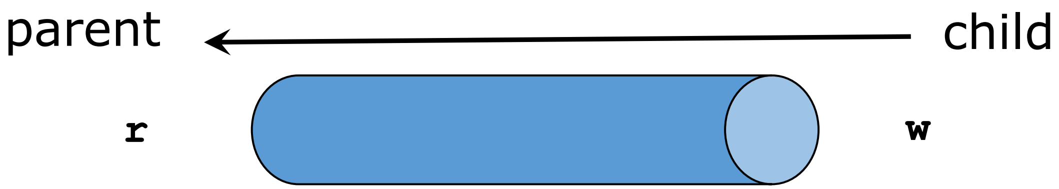
- UDP sockets are *connectionless*, that is, no connection is established first. Each data segment is individually addressed and routed to be sent.

- UDP and TCP sockets are different in some other respects, discussed later.

Pipes

- A pipe acts as a conduit providing one of the simpler ways for processes to communicate
  - Ordinary pipes allow two processes to communicate in standard producer-consumer fashion
    - The producer writes to one end of the pipe (write-end) and the consumer reads from the other end (read-end)

- A pipe creates an (r, w) file descriptor: r is for reading and w for writing.
  - For example, if the parent process wants to receive data from the child process, it keep r open, and the child keeps w open



Pipes (cont.)

- Pipes are used quite often in the command-line environment (originally from the UNIX OS) in which the output of one command serves as input to the second.

- A pipe can be constructed on the CLI using the | character
  - Example (macOS or Linux terminal): ls | more
  - Equivalent example (Windows cmd/powershell): dir | more

Remote Procedure Calls

- Remote procedure call (RPC) abstracts procedure calls between processes on networked systems
- The semantics of RPCs allow a client to invoke a procedure on a remote host as it would invoke a procedure locally
- We must use a message-based communication to provide remote service
  - The messages are well structured
  - Each message is addressed to an RPC daemon (e.g. background server) listening to a port on a remote system, and contains
    - the identifier of the function to execute and
    - the parameters to pass to that function

Remote Procedure Calls (cont.)

- An important issue that must be dealt with concerns differences in data representation of the client and server machines.
- Endianness describes the order of bytes of a word in a computer memory.
  - Little-endian: some systems store the least significant byte first.
    - e.g. Intel's x86 processors and their clones are little endian.
  - Big-endian: some systems store the most significant byte first.
    - e.g. Motorola 6800, PowerPC
  - Bi-endian: allows switchable endianness
    - e.g. ARM architecture since version 3 (little-endian generally)

References

- Some sections of chapter 3 of Operating Systems Concepts

Acknowledgement: This set of slides is partly based on the PPTs provided by the Wiley's companion website for the operating system concepts book (including textbook images, when not explicitly mentioned/referenced).