CISC 3320 MW3

Example IPC Systems and Libraries

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Acknowledgement

- These slides are a revision of the slides by the authors of the textbook

Outline

- Examples of IPC Systems
  - Shared memory (POSIX, Windows)
  - Mailboxes/Ports (POSIX message queue, Windows mailslot)
  - Pipes (POSIX, Windows; named and ordinary/anonymous)
  - Windows advanced local procedure call (Your reading)
  - Mach message passing (Your reading)
- Communication in Client-Server Systems

POSIX Shared Memory

- Create or open an existing shared memory segment
  - System call **`shm_open()`**
  - Process first creates shared memory segment, e.g., **`shm_fd = shm_open(name, O CREAT | O RDWR, 0666);`**
  - Also used to open an existing segment
- Set the size of the segment, e.g.,
  - **`ftruncate(shm_fd, 4096);`**
- Map the shared memory segment to cooperating processes' address space
  - Use **`mmap()`** to memory-map a file pointer to the shared memory object
- Reading and writing to shared memory is done by using the pointer returned by **`mmap()`.**

Windows Shared Memory

- Create a shared memory segment
  - API **`CreateFileMapping(INVALID_HANDLE_VALUE, ...)`**
- Open an existing shared memory segment
  - API **`OpenFileMapping()`**
- Map the shared memory segment to cooperating processes' address space
  - Use **`MapViewOfFile()`** to memory-map a file pointer to the shared memory object
- Reading and writing to shared memory is done by using the pointer returned by **`MapViewOfFile()`.**

Shared Memory

- Example programs
  - POSIX (using Linux)
  - Windows

Questions?

- POSIX shared memory
- Windows shared memory
- Essential system calls and APIs
- Example programs
- Don’t forget to do clean-up!

Mailbox/Port: POSIX Message Queue

- Essential system calls
  - mq_open
  - mq_send
  - mq_receive
  - mq_close
  - mq_unlink

Mailbox/Port: Windows Mailslots

- Create mailslot
  - CreateFile
    - Mailslot name must be a "mailslot", e.g.,
    - \\.\mailslot\mymailslot
- Write to mailslot
  - WriteFile
- Read from mailslot
  - ReadFile

Mailboxes

- Example programs
  - POSIX (using Linux)
  - Windows

Questions

- POSIX message queue
- Windows mailslot
- Essential system calls and APIs
- Example programs
- Don’t forget to do clean-up!

Pipes

- Acts as a conduit allowing two processes to communicate
  - The communication pattern follows message passing
  - But, pipes may be implemented using shared memory

Pipes: Design Issues

- Issues:
  - Is communication unidirectional or bidirectional?
  - In the case of two-way communication, is it half or full-duplex?
  - Must there exist a relationship (i.e., ***parent-child***) between the communicating processes?
  - Can the pipes be used over a network?
- **Ordinary pipes** – cannot be accessed from outside the process that created it. Typically, a parent process creates a pipe and uses it to communicate with a child process that it created.
- **Named pipes** – can be accessed without a parent-child relationship

Ordinary Pipes

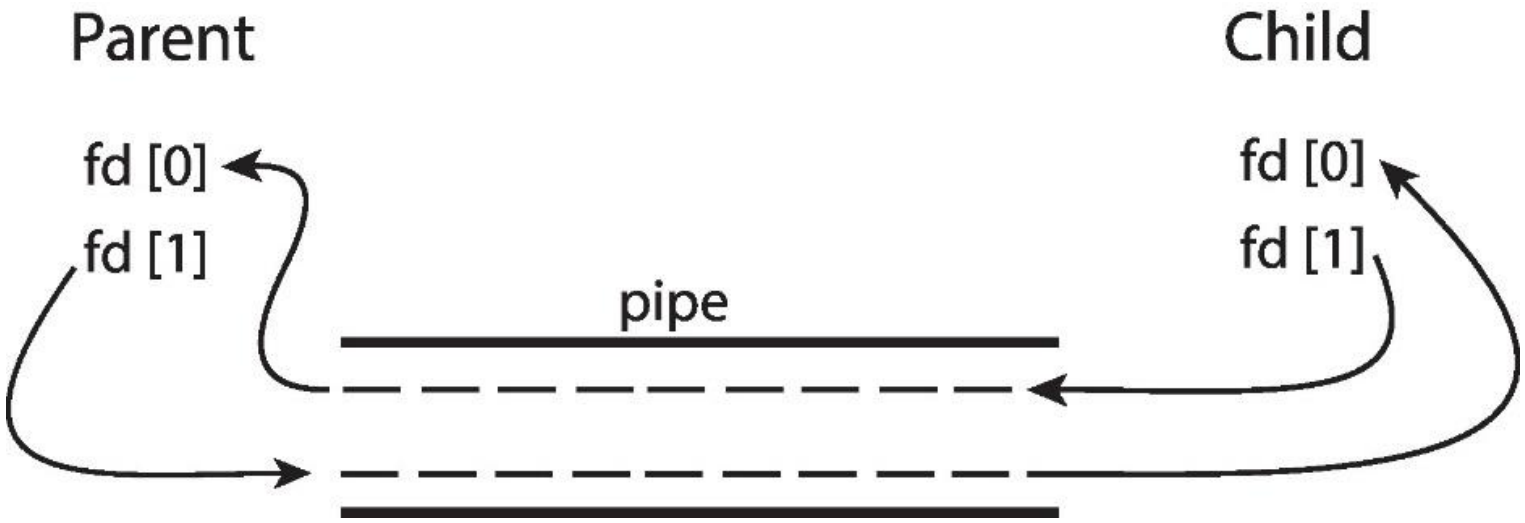
- Ordinary Pipes allow communication in standard producer-consumer style: unidirectional
- Producer writes to one end (the **write-end** of the pipe)
- Consumer reads from the other end (the **read-end** of the pipe)
- Ordinary pipes are therefore unidirectional
- Require parent-child relationship between communicating processes
- Windows calls these **anonymous pipes**

POSIX Ordinary Pipes

- Relies on processes' parent-child relationship via the fork system call
  - The pipe system call creates a pipe, e.g., fd[0], fd[1]
    - Read end: fd[0]
    - Write end: fd[1]

Ordinary Pipes: Parent-Child relationship

- To use the pipe in two processes, fork a child process, resulting in a two unidirectional communication links



Ordinary Pipe: Example Applications

Windows Ordinary Pipes

- Called anonymous pipes
- Similar to UNIX, an anonymous pipe on Windows is unidirectional, and emply parent-child relationship
- Example application

Questions?

- Concept of pipes
- Ordinary pipes
- POSIX ordinary pipes
- Windows anonymous pipes

Named Pipes

- Named Pipes are more powerful than ordinary pipes
  - Communication is bidirectional
  - No parent-child relationship is necessary between the communicating processes
  - Several processes can use the named pipe for communication
- Provided on both UNIX and Windows systems

UNIX Named Pipes: Example Program

- Called “fifo”, See example programs

Questions?

- Concept of pipes
- Ordinary pipes
- Named pipes
- Example programs

Mach Message Passing

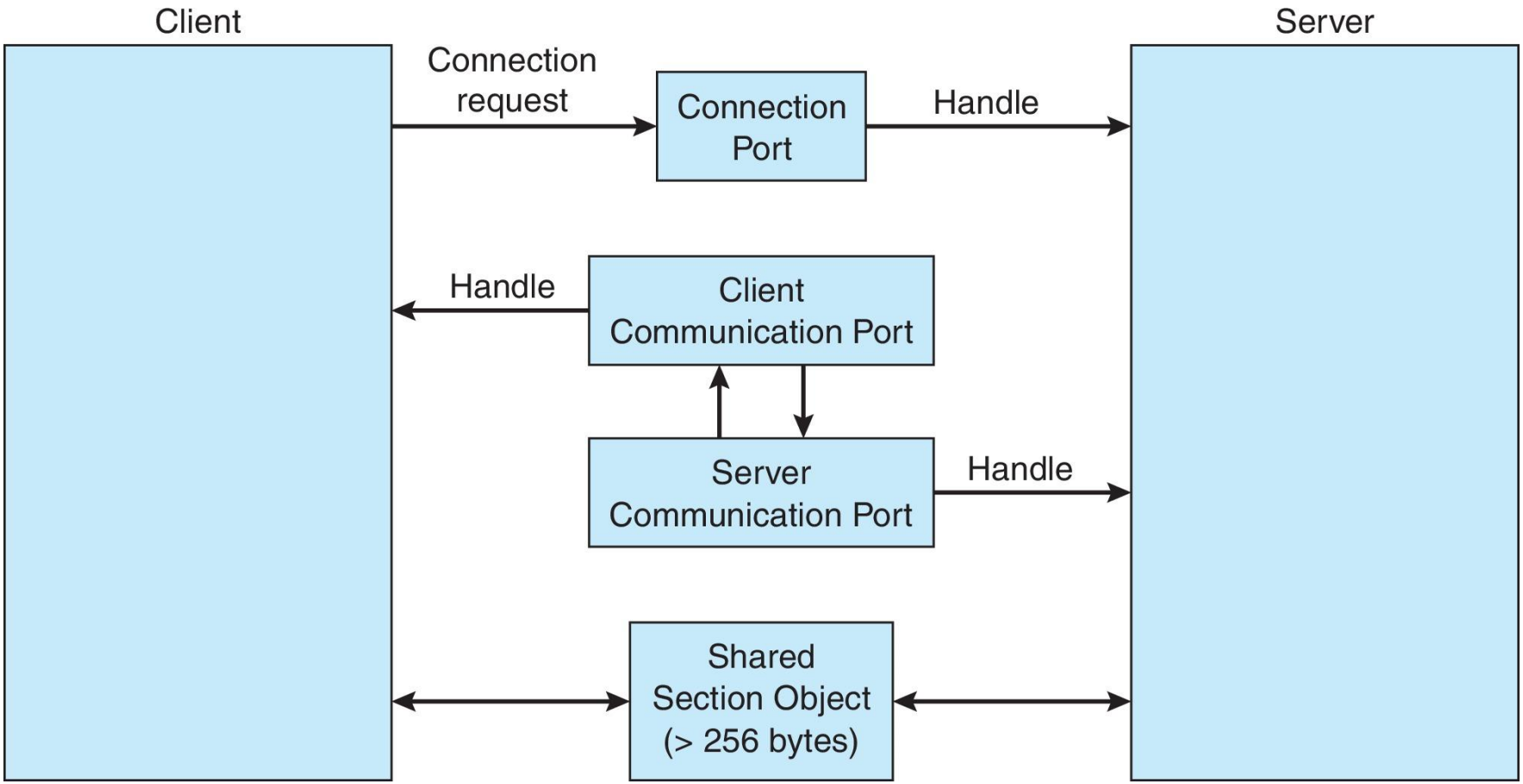
- Mach communication is message based
  - Even system calls are messages
  - Each task gets two ports at creation - Kernel and Notify
  - Messages are sent and received using the **`mach_msg()`** function
  - Ports needed for communication, created via

    `mach_port_allocate()`
  - Send and receive are flexible, for example four options if mailbox full:
    - Wait indefinitely
    - Wait at most n milliseconds
    - Return immediately
    - Temporarily cache a message

Windows IPC

- Message-passing centric via **advanced local procedure call (LPC)** facility
  - Only works between processes on the same system
  - Uses ports (like mailboxes) to establish and maintain communication channels
  - Communication works as follows:
    - The client opens a handle to the subsystem's **connection port** object.
    - The client sends a connection request.
    - The server creates two private **communication ports** and returns the handle to one of them to the client.
    - The client and server use the corresponding port handle to send messages or callbacks and to listen for replies.

Local Procedure Call (LPC)



Questions?

- Mach message passing
- Windows ALPC