Operating Systems
CS2006

Lecture 6

Inter-Process Communication (IPC)

15th February 2023

Dr. Rana Asif Rehman

Introduction

- A process has access to the memory which constitutes its own address space.
- When a child process is created, the only way to communicate between a parent and a child process is:
  - The parent receives the exit status of the child
- So far, we've discussed communication mechanisms only during process creation/termination
- Processes may need to communicate during their life time.

Cooperating Processes

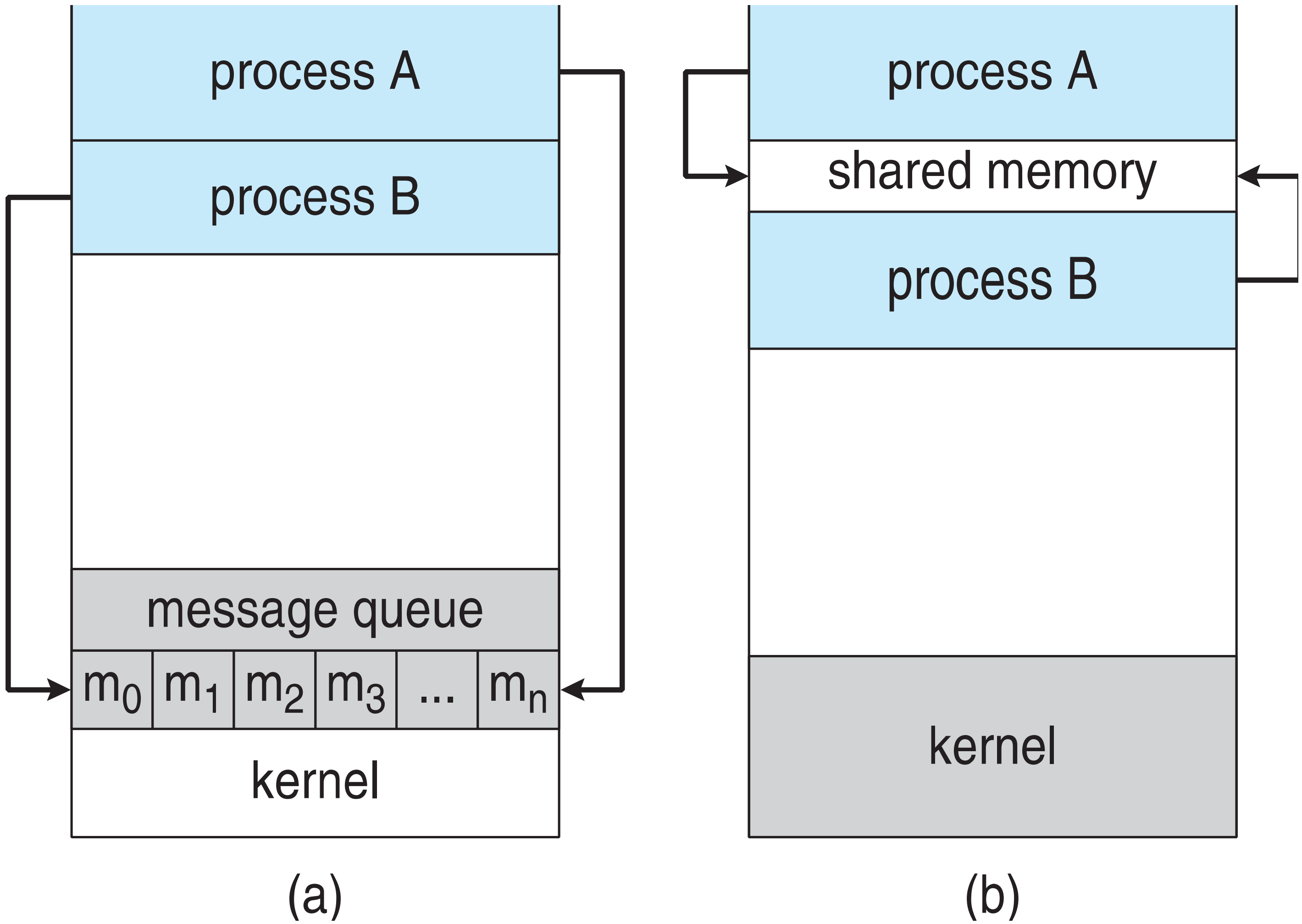
- **Independent** process cannot affect or be affected by the execution of another process.
- **Cooperating** process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience
- Dangers of process cooperation
  - Data corruption, deadlocks, increased complexity
  - Requires processes to synchronize their processing

Inter-Process Communication (IPC)

- Mechanism for processes to communicate and to synchronize their actions.
- Two main types
  - Shared Memory
  - Message Passing

Communications Models

(a) Message passing. (b) shared memory.



Interprocess Communication – Shared Memory

- An area of memory shared among the processes that wish to communicate
- The communication is under the control of the users processes not the operating system.
- Major issues is to provide mechanism that will allow the user processes to synchronize their actions when they access shared memory.
- Synchronization is discussed in great details in Chapter 6.

Producer-Consumer Problem

- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process
- To allow producer and consumer processes to run concurrently, we must have available a buffer of items that can be filled by the producer and emptied by the consumer.
- This buffer will reside in a region of memory that is shared by the producer and consumer processes.
- A producer can produce one item while the consumer is consuming another item.
- The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.

Producer-Consumer Problem

Two types of buffers can be used.

- **unbounded-buffer** places no practical limit on the size of the buffer
- **bounded-buffer** assumes that there is a fixed buffer size

- Let's look more closely at how the bounded buffer illustrates inter- process communication using shared memory….

Bounded-Buffer – Shared-Memory Solution

- The following variables reside in a region of memory shared by the producer and consumer processes:
- Shared data

```
#define BUFFER_SIZE 10
typedef struct {
  . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

Lets code this….

Bounded-Buffer – Producer

```
item next_produced;
while (true) {
        /* produce an item in next produced */
        while (((in + 1) % BUFFER_SIZE) == out)
                ; /* buffer is full - do nothing */
        buffer[in] = next_produced;
        in = (in + 1) % BUFFER_SIZE;
}
```

Bounded Buffer – Consumer

```
item next_consumed;
while (true) {
        while (in == out)
                ; /* buffer empty - do nothing */
        next_consumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;

        /* consume the item in next consumed */
}
```

- This scheme allows at most BUFFER SIZE − 1 items in the buffer at the same time.

Message Passing (IPC)

- Message system – processes communicate with each other without resorting to shared variables.
- IPC facility provides two operations:
  - **send**(message) – message size fixed or variable
  - **receive**(message)
- If P and Q wish to communicate, they need to:
  - establish a communication link between them
  - exchange messages via send/receive
- Implementation of communication link
  - physical (i.e., shared memory, hardware bus)
  - logical (direct/indirect, blocking/non-blocking, automatic/explicit buffering)

Implementation Questions

- How are links established?
- Can a link be associated with more than two processes?
- How many links can there be between every pair of communicating processes?
- What is the capacity of a link?
- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bi-directional?

Message Passing Systems

- Exchange messages over a communication link
- Methods for implementing the communication link and primitives (send/receive):
  1. Direct or Indirect communications (Naming)
  2. Symmetric or Asymmetric communications (blocking versus non-blocking)
  3. Buffering

Direct Communication

- Processes must name each other explicitly:
  - **send** (P, message) – send a message to process P
  - **receive**(Q, message) – receive a message from process Q
- Properties of communication link
  - Links are established automatically.
  - A link is associated with exactly one pair of communicating processes.
  - Between each pair there exists exactly one link.
  - Processes need to know each other's identity
  - The link may be unidirectional, but is usually bi-directional.

Disadvantage: a process must know the name or ID of the process(es) it wishes to communicate with

Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports).
  - Each mailbox has a unique id.
  - Processes can communicate only if they share a mailbox.
- Properties of communication link
  - Link established only if processes have a shared mailbox
  - A link may be associated with many processes.
  - Each pair of processes may share several communication links.
  - Link may be unidirectional or bi-directional.

Indirect Communication (Cont.)

- Operations provided by the OS
  - create a new mailbox
  - send and receive messages through mailbox
  - destroy a mailbox
- Primitives are defined as:
  - **send**(A, message) – send a message to mailbox A
  - **receive**(A, message) – receive a message from mailbox A

Indirect Communication (Cont.)

- Mailbox sharing
  - P1, P2, and P3 share mailbox A.
  - P1, sends; P2 and P3 receive.
  - Who gets the message?
- Solutions
  - Allow a link to be associated with at most two processes.
  - Allow only one process at a time to execute a receive operation.
  - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

Synchronization

- Message passing can be blocking or non-blocking
- Four types of synchronization
  - Blocking send:
    - sender blocked until message received by mailbox or process
  - Nonblocking send:
    - sender resumes operation immediately after sending
  - Blocking receive:
    - receiver blocks until a message is available
  - Nonblocking receive:
    - receiver returns immediately with either a valid or null message.

Buffering

- Messages exchanged by processes reside in temporary queue
- Three ways to implement queues
  1. Zero capacity

     No messages may be queued within the link, requires sender to block until receiver retrieves message.
  2. Bounded capacity

     Link has finite number of message buffers. If no buffers are available then sender must block until one is freed up.
  3. Unbounded capacity

     Link has unlimited buffer space, consequently send never needs to block.

Communication in Client/Server Systems

- Sockets
- Remote procedure calls
- Pipes
- What types of Inter-Process Communication might each of these use?

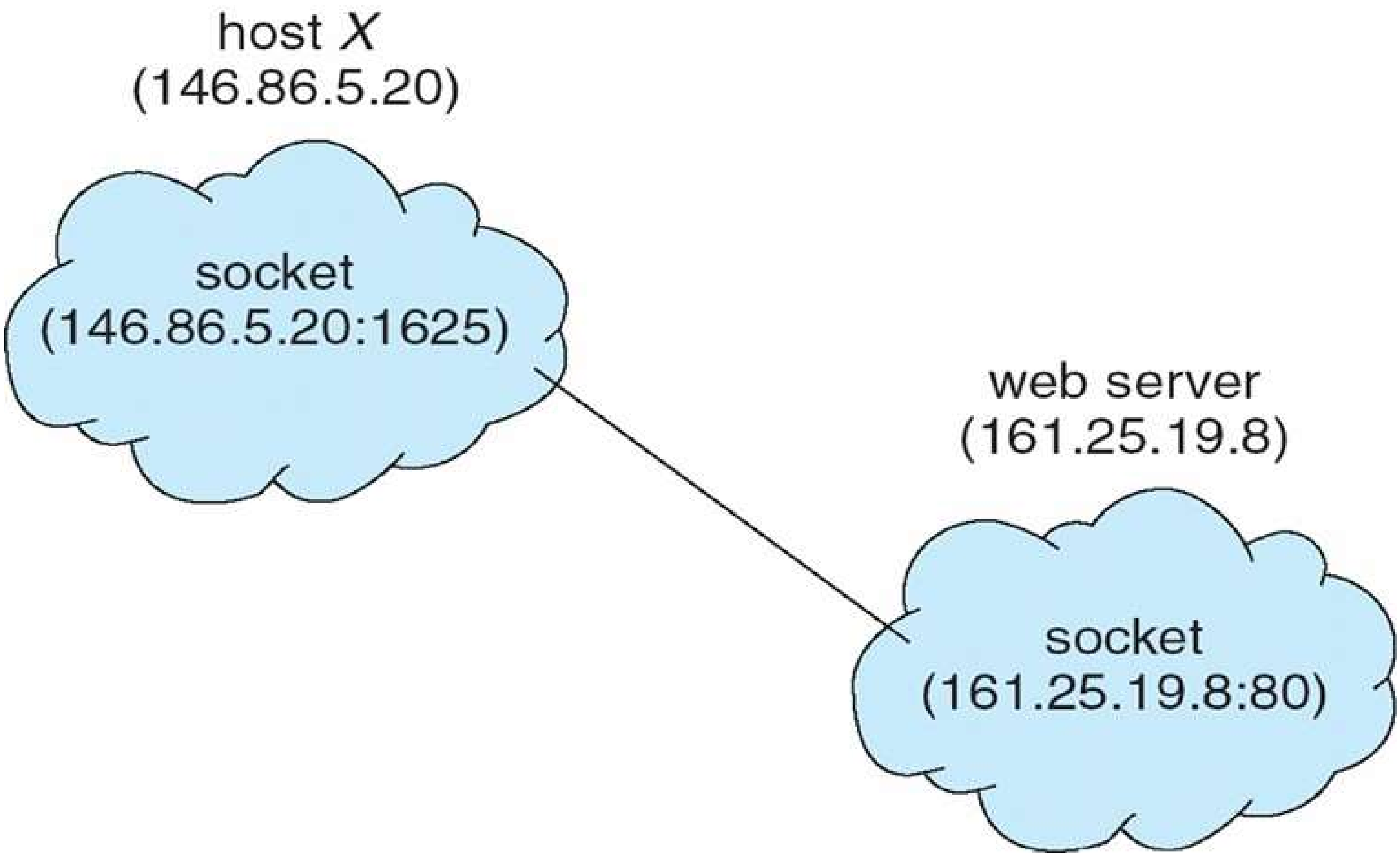
1. Sockets

- Defined as a point of communication
- A pair of process communicating over a network employee a pair of sockets
- Socket = IP Address : Port Number
- All ports below 1024 are considered well known
  - FTP: 21
  - Telnet: 23
  - Web server: 80

Sockets

- Client initiates a requests for connection, it is assigned a port by host computer
- Connection on one host must be unique
- Sockets can be
  - Connection-oriented (TCP) socket
  - Connectionless (UDP) socket
- Considered Low level mode of communication
  - Shares unstructured stream of bytes

Sockets


host *X*
(146.86.5.20)
socket
(146.86.5.20:1625)
web server
(161.25.19.8)
socket
(161.25.19.8:80)

Sockets in Java

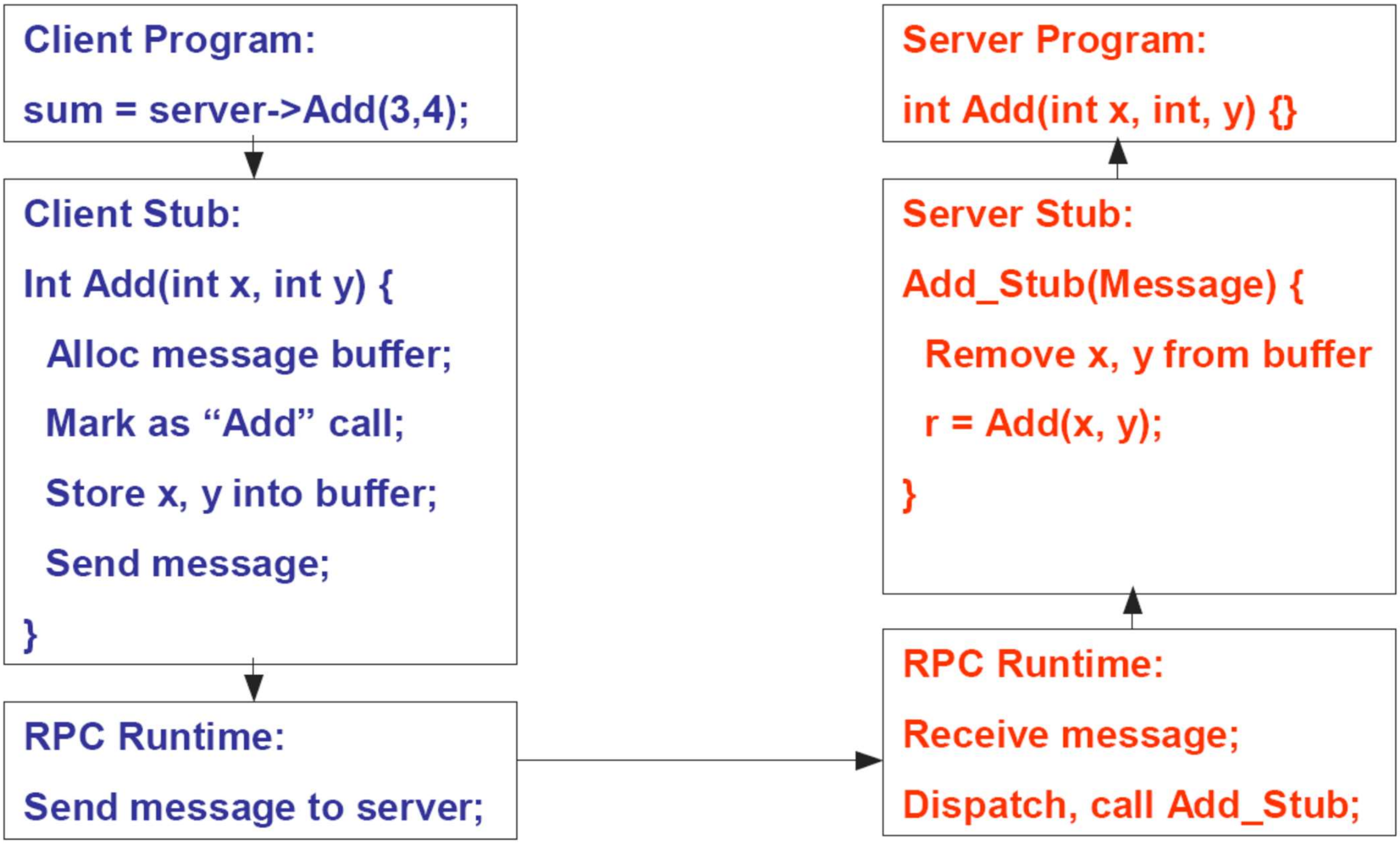
- Three types of sockets
  - **Connection-oriented** (**TCP**)
  - **Connectionless** (**UDP**)
  - **`MulticastSocket`** class– data can be sent to multiple recipients

```java
import java.net.*;
import java.io.*;

public class DateServer
{
  public static void main(String[] args) {
    try {
      ServerSocket sock = new ServerSocket(6013);

      /* now listen for connections */
      while (true) {
        Socket client = sock.accept();

        PrintWriter pout = new
         PrintWriter(client.getOutputStream(), true);

        /* write the Date to the socket */
        pout.println(new java.util.Date().toString());

        /* close the socket and resume */
        /* listening for connections */
        client.close();
      }
    }
    catch (IOException ioe) {
      System.err.println(ioe);
    }
  }
}
```

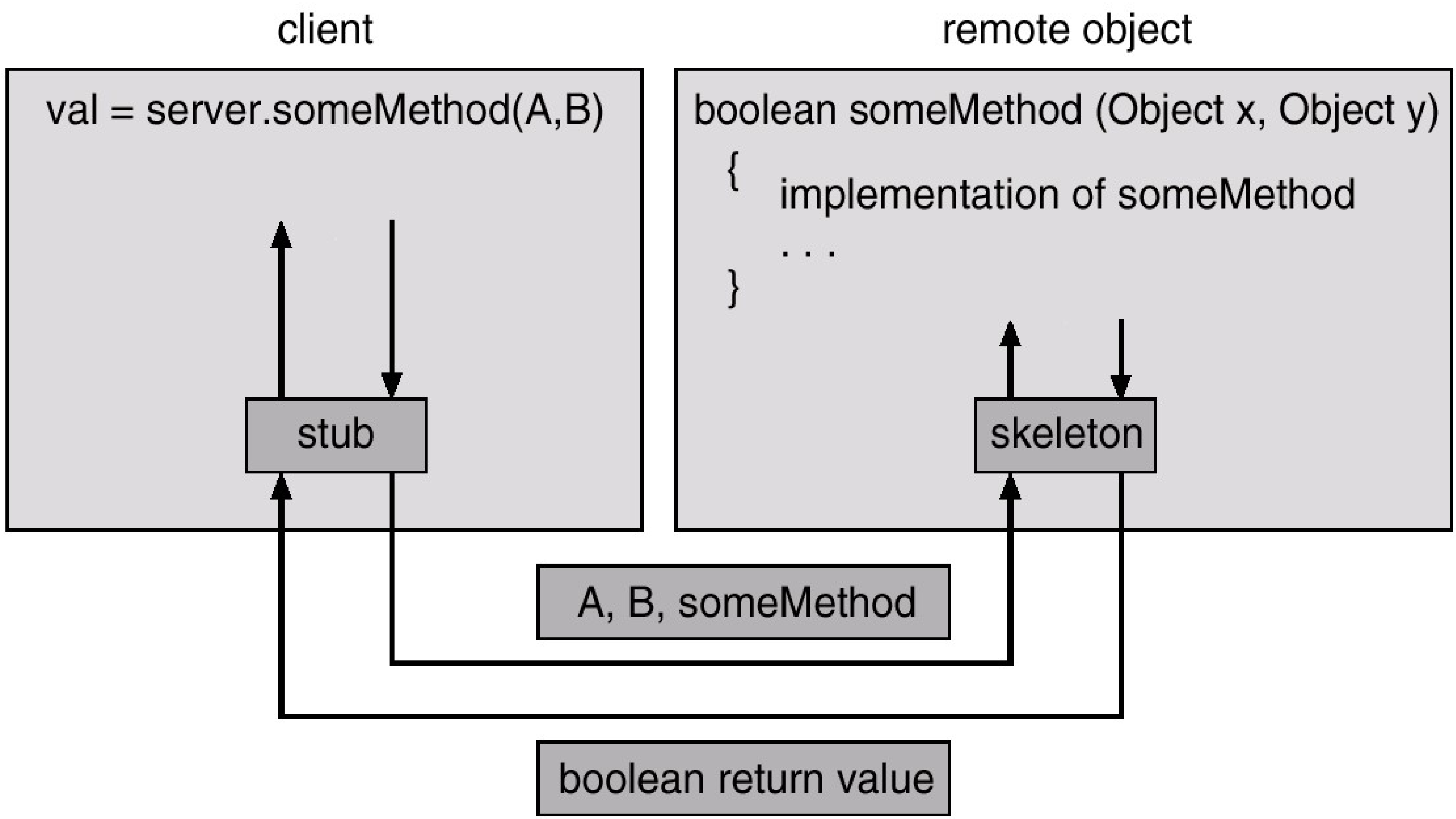
2. Remote Procedure Calls (RPC)

- Remote procedure call (RPC) abstracts procedure calls between processes on networked systems.
- **Stubs** – client-side proxy for the actual procedure on the server.
- The client-side stub locates the server and marshals the parameters.
- The server-side stub receives this message, unpacks the marshalled parameters, and performs the procedure on the server.

RPC: An Example



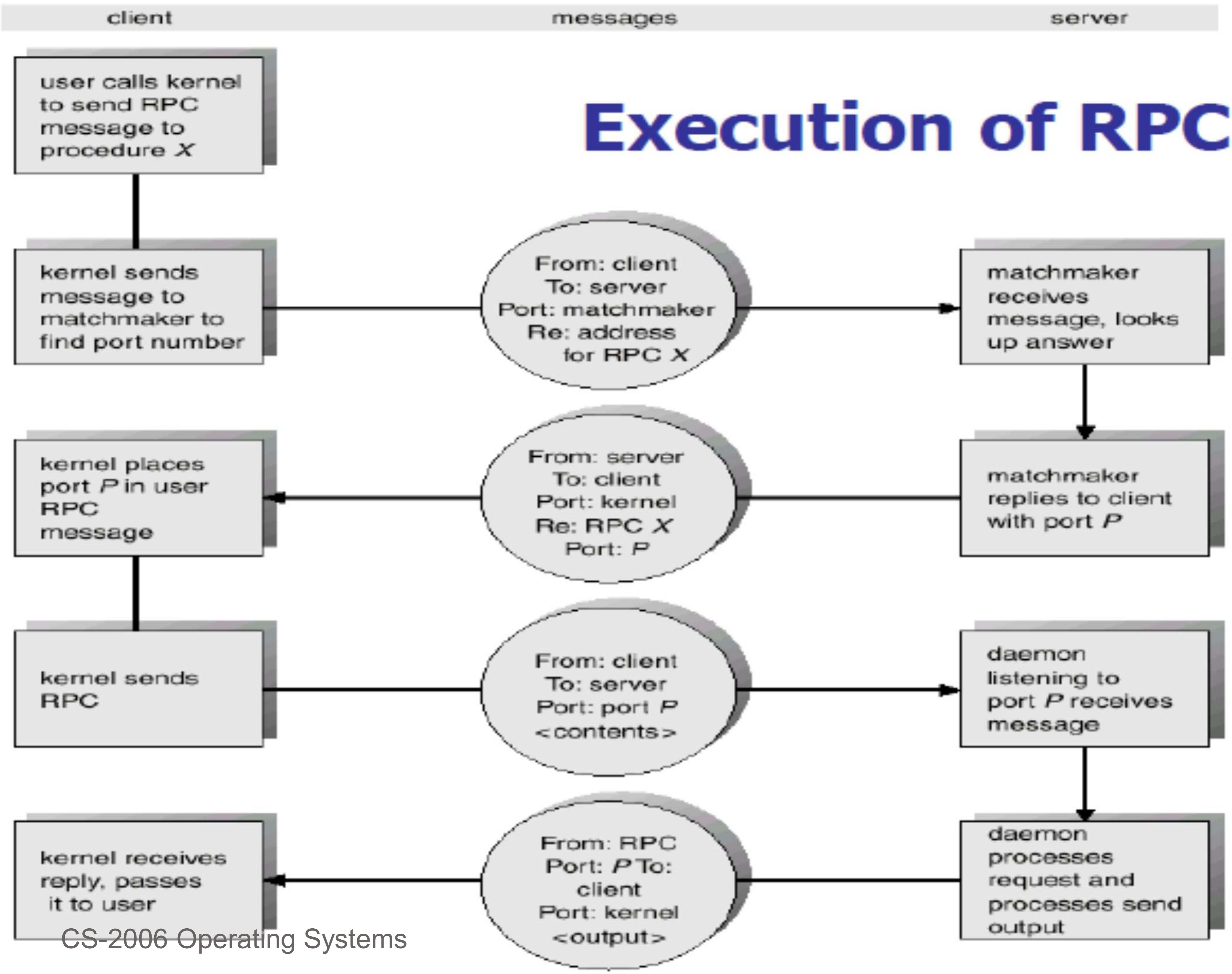
Marshalling Parameters



Remote Procedure Calls

- Problem exists in data representation
  - Big endian
  - Little endian
- Many RPC systems define a machine independent representation
  - External Data Representation (XDR)
- Semantics of a call
  - Messages are acted on *exactly once* rather than *at-most once*
- Port binding of RPC server
  - Fixed at compile time
  - Done dynamically by **rendezvous** mechanism (*match maker*)

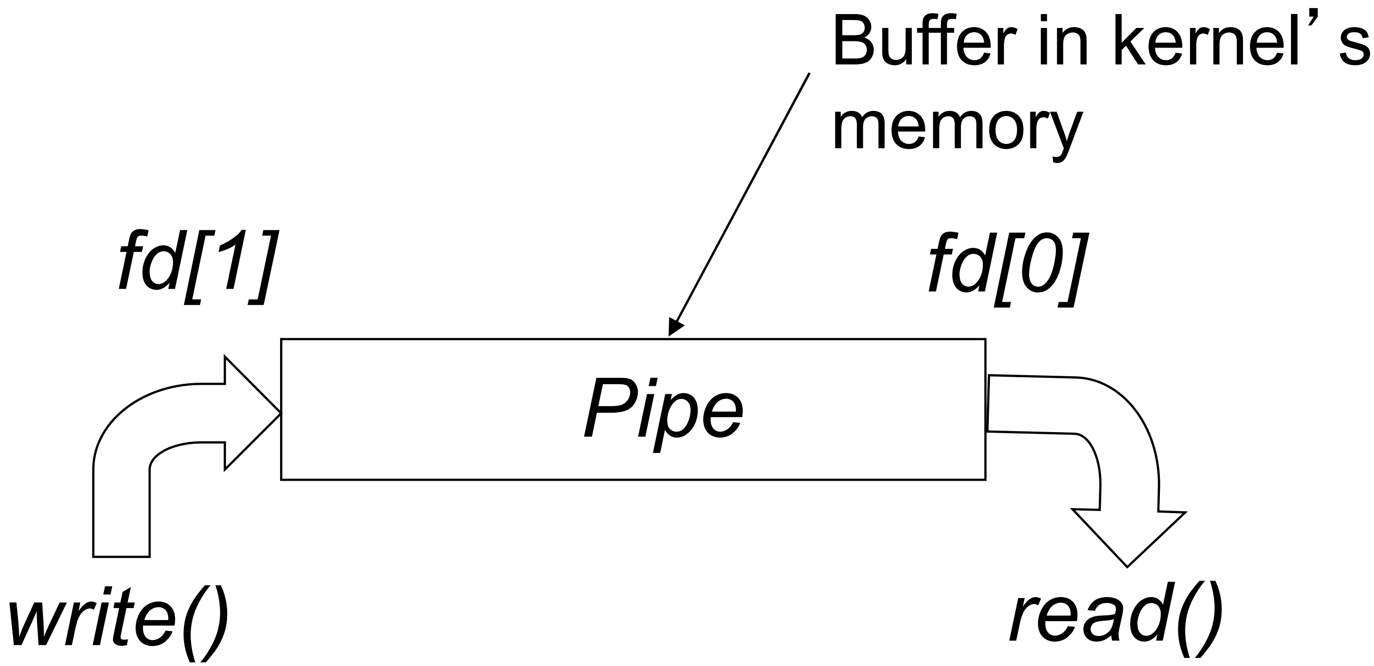
Execution of RPC



3. Pipes

- IPC mechanism in early UNIX systems
- Four issues must be considered
  - Unidirectional or bi-directional communication
  - If bidirectional then is it half duplex or full duplex
  - Must a relationship (such as *parent-child*) exist between the communicating processes?
  - Can pipe communicate over a network?

Pipes: Shared info in kernel's memory



Pipe Creation (Unix)

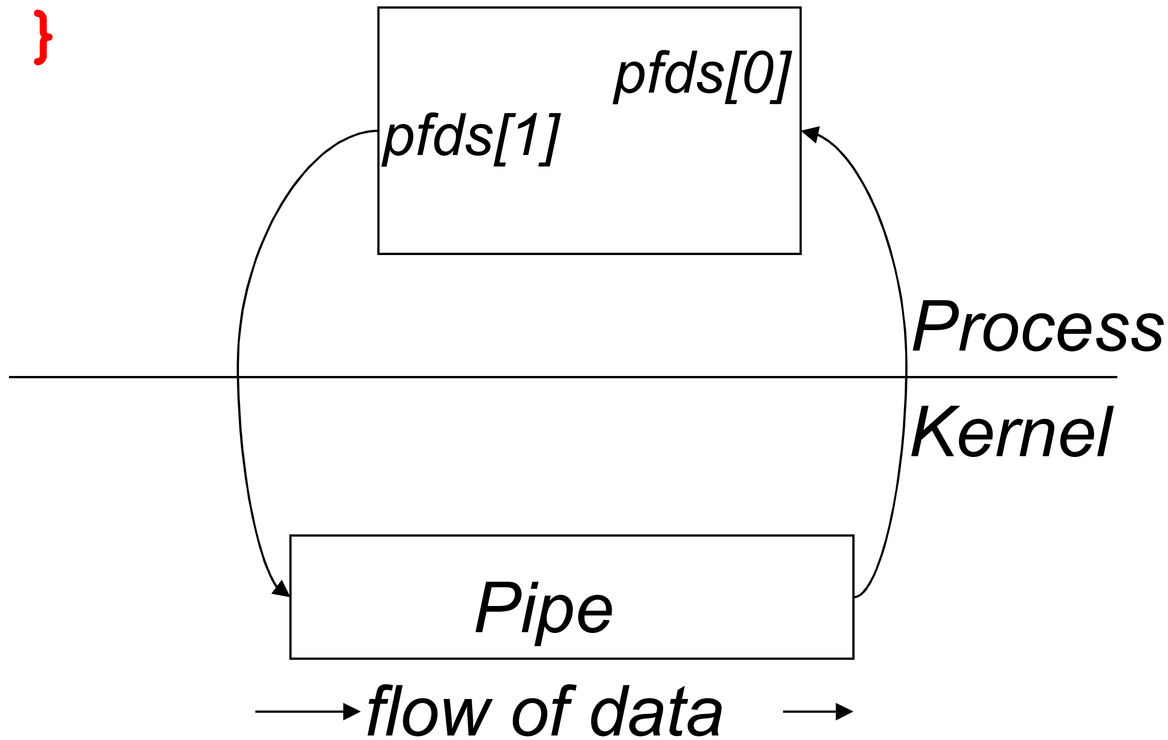
```c
#include <unistd.h>
int pipe(int filedes[2]);
```

- Creates a pair of file descriptors pointing to a pipe inode
- Places them in the array pointed to by filedes
  - filedes[0] is for reading
  - filedes[1] is for writing.

- On success, zero is returned.
- On error, -1 is returned

Pipe Creation

```
int main()
{  int pfds[2];
  if (pipe(pfds) == -1)
    {  perror("pipe");
       exit(1); }
  }
```



Reading/Writing from/to a Pipe

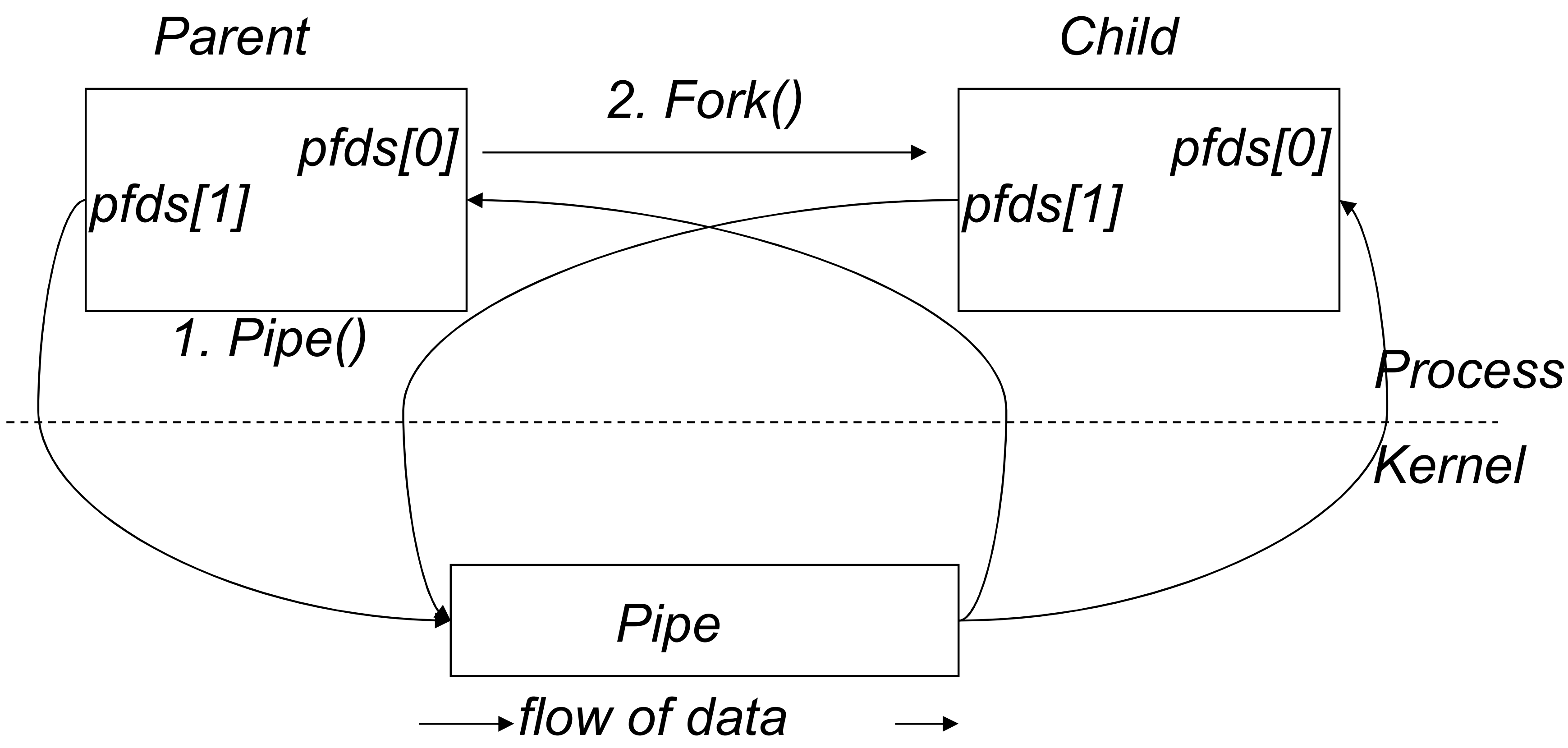
- `int read(int filedescriptor, char *buffer, int bytetoread);`

- `int write(int filedescriptor,char *buffer,int bytetowrite);`

Example

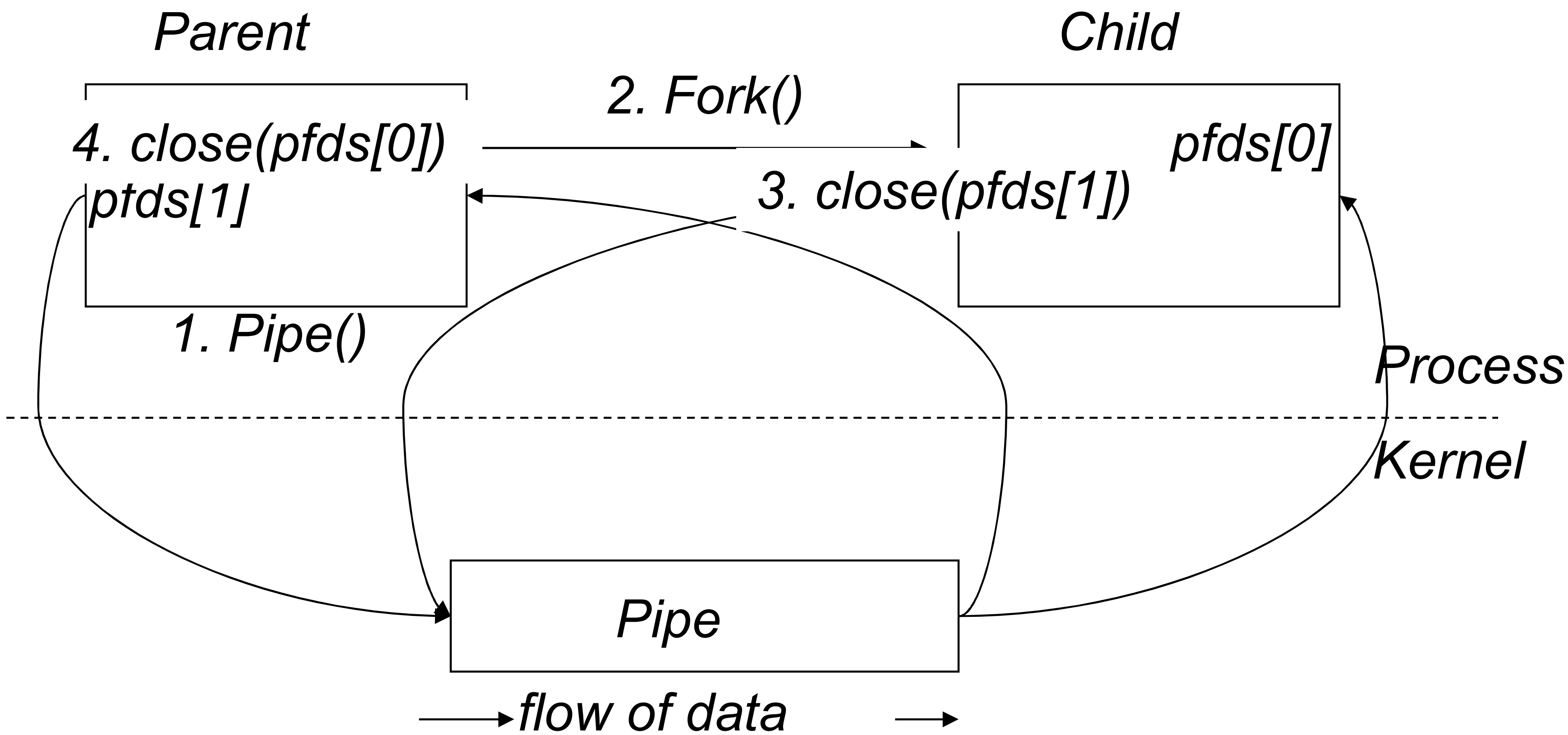
```
int main()
{ int pfds[2];
  char buf[30];
  if (pipe(pfds) == -1) {
          perror("pipe");
          exit(1); }
  printf("writing to file descriptor #%d\n", pfds[1]);
   write(1, "test", 5);?

  printf("reading from file descriptor
  #%d\n", pfds[0]);
   read(0, buf, 5);?
  printf("read %s\n", buf);
}
```

A Channel between two processes



A Channel between two processes

- To allow one way communication each process should close one end of the pipe.



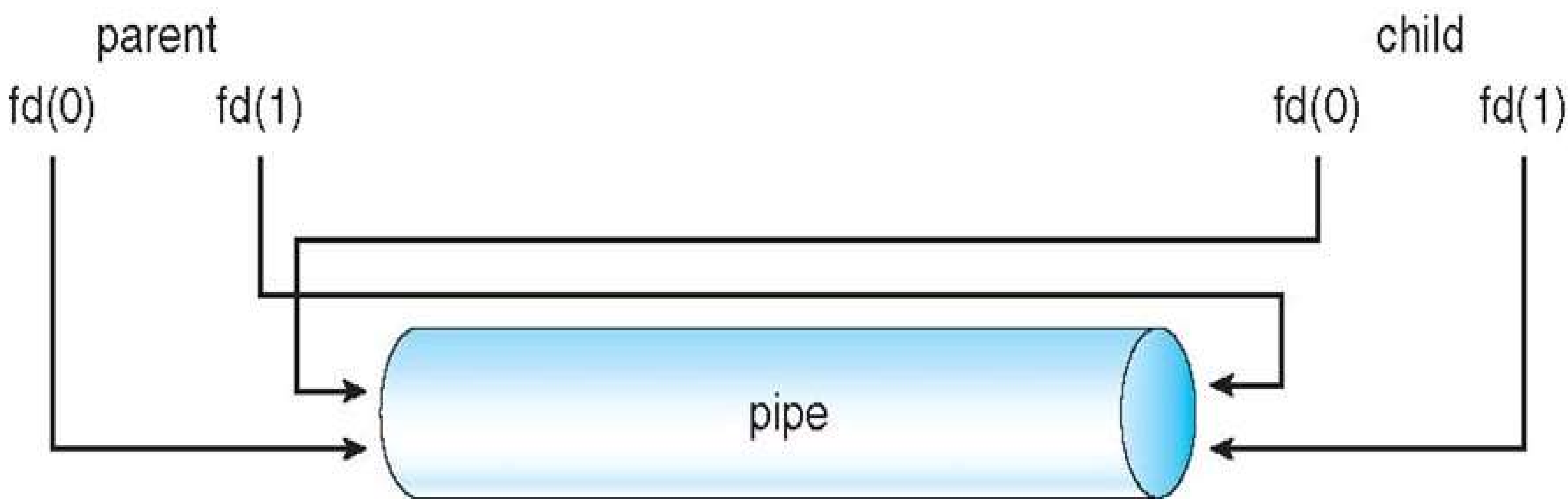
Pipes

- Two types of pipes
  - Ordinary pipes
  - Named pipes

Ordinary pipes

- **Ordinary pipes** allow communication in standard producer-consumer form
- **Producer** writes to one end of the pipe (the *write-end*)
- **Consumer** reads from other end of the pipe (the *read-end* of the pipe)
- Ordinary pipes are therefore unidirectional
- Required parent-child relationship between communicating processes
- In windows known as *Anonymous pipes*

Ordinary pipe



Named pipes

- Named pipes are more powerful than ordinary pipes
- Communication is bidirectional
- UNIX provides half duplex while Windows provides full duplex communication
- 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
- Called FIFO in UNIX
- In UNIX, communicating process reside on the same machine (socket is required for intermachine comm.)
- In Windows, communicating process may reside on either the same machine or different machines

References

- Operating System Concepts (Silberschatz, 9th edition) Chapter 3