# Operating Systems

Dr. Shu Yin

# Part II: Process Management

- Processes
- Threads
- Process Synchronization
- CPU Scheduling
- Deadlocks

# Goals

- Interprocess Communication
- Process Synchronization

# Interprocess Communication

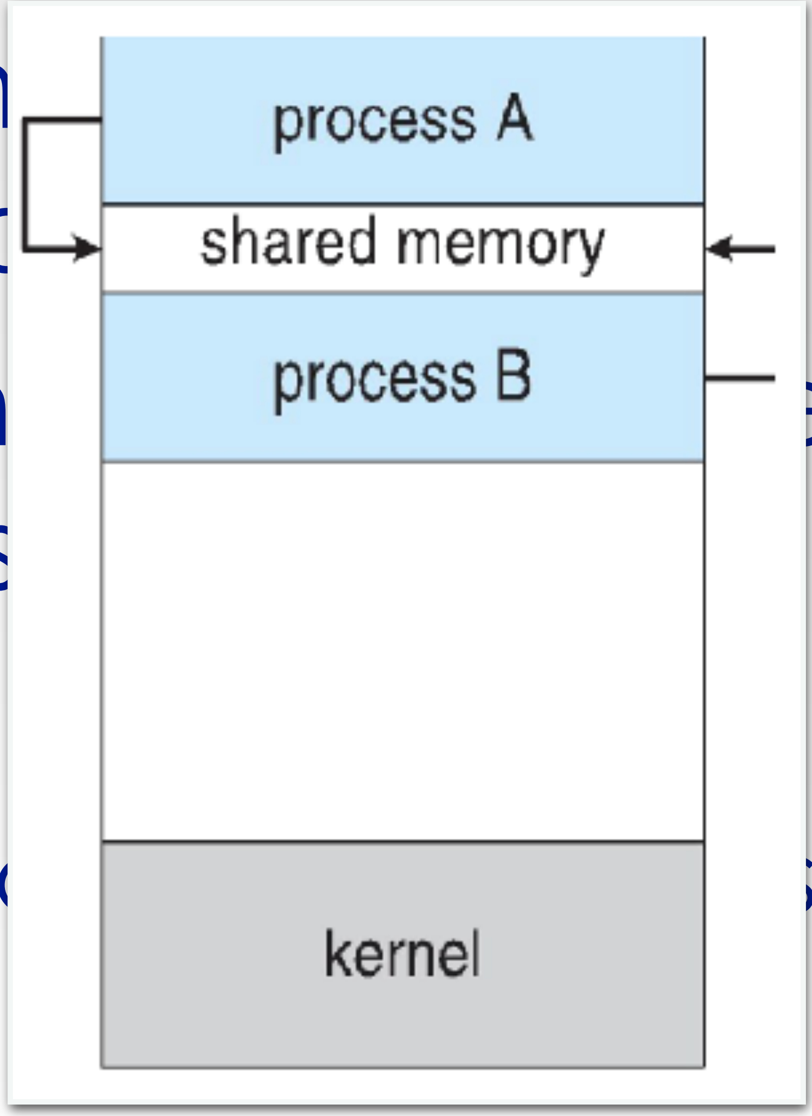
- Process within a system may be *independent* or *cooperating*
- Cooperating process can affect or be affected by other processes, including sharing data

# Interprocess Communication (cont.)

- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience (e.g. editing, printing, compiling)
- Cooperating processes need IPC
- Two models of IPC
  - Shared memory
  - Message passing

# IPC: Shared Memory

- An area of m among the processes tha municate
- The commun er the control of the proces
- Major issue:
  - Synchronizati sed later)

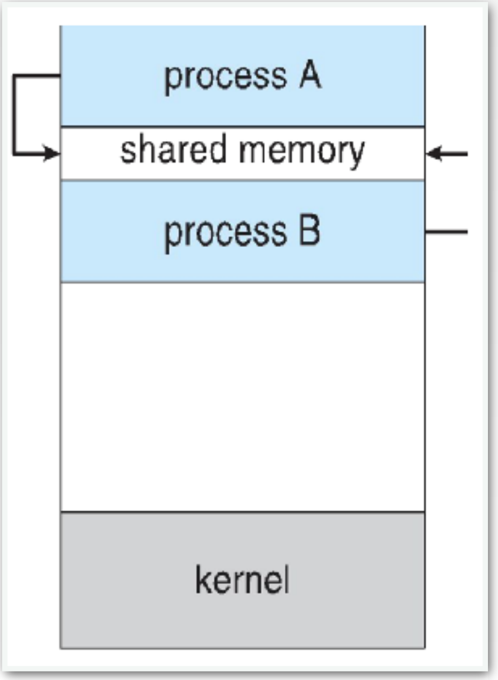





上海科技大学
ShanghaiTech University

# IPC: Shared Memory (cont.)

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



# IPC: Shared Memory (cont.)

- Bounded-buffer
- Shared-Memory Solution

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

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

shared data

```
item next_consumed;
while (true){
  while (in == out)
   ;/*do nothing*/
  next_consumed = buffer[out];
  out = (out + 1) % BUFFER_SIZE;
  /*consume the item in the next
  consumed*/
}
```

consumer

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

producer



上海科技大学 ShanghaiTech University

# IPC: Shared Memory (cont.)

- Bounder-Buffer
- How many elements in the buffer can be used at most a a given time?

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

producer

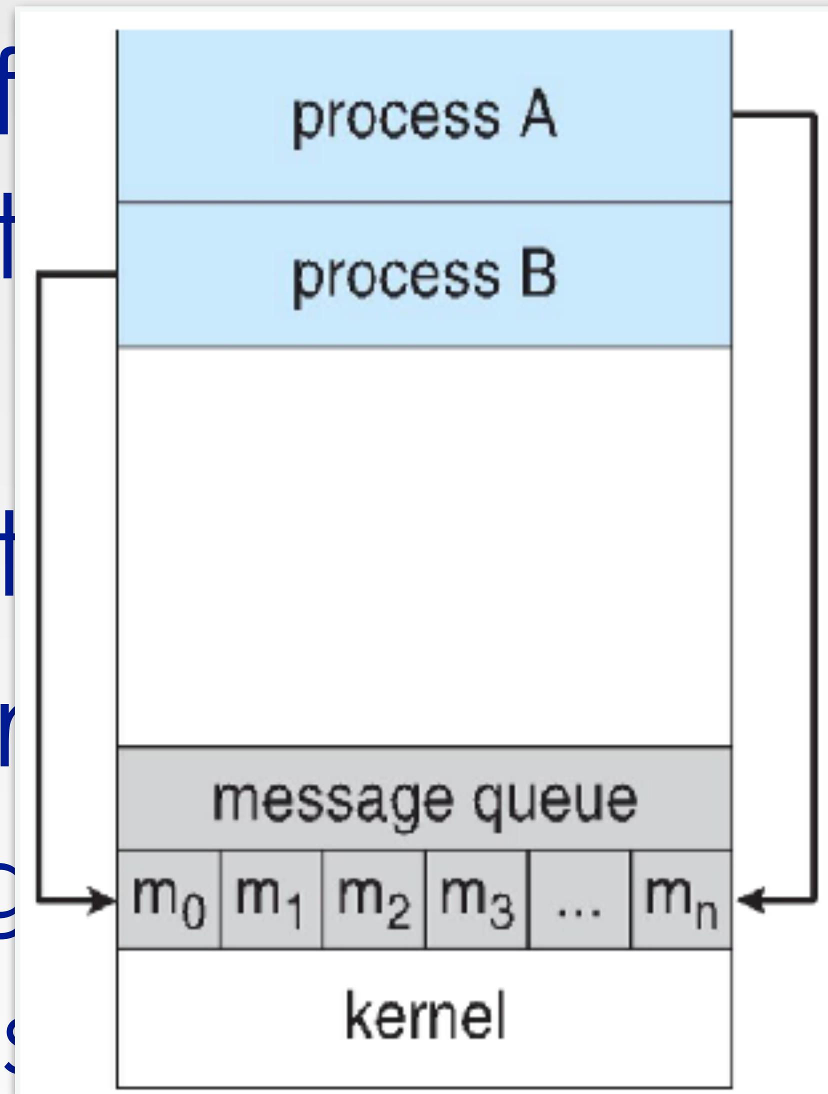
```
item next_consumed;
while (true){
  while (in == out)
   ;/*do nothing*/
  next_consumed = buffer[out];
  out = (out + 1) % BUFFER_SIZE;
  /*consume the item in the next
  consumed*/
}
```

consumer

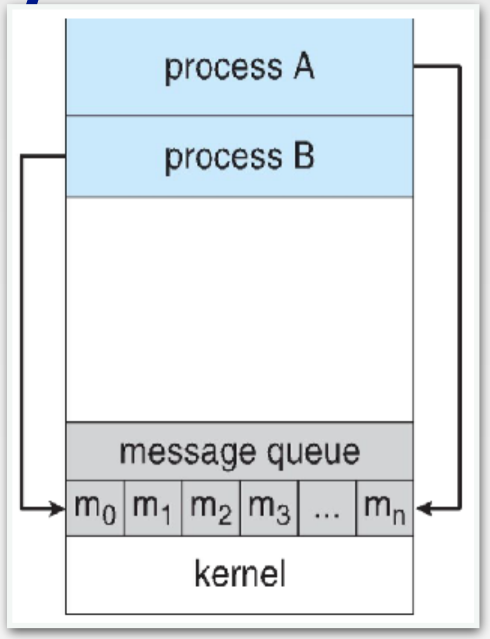
上海科技大学 ShanghaiTech University

# IPC: Message Passing

- Mechanism f … o communicat … hronize their action
- Message syst …
- IPC facility pr … erations:
  - **send**(messag …
  - **receive**(mes …
- Message size: fixed or variable



# IPC: Message Passing (cont.)




- If processes A and B with to communicate
  - Establish a communicate link
  - Exchange messages via send/receive
- Implementation issues:
  - How are links established
  - Can a link be associated with one or more processes
  - How many linksWhat's the capacity of a link
  - Is the size of message fixed/variable
  - Is a link unidirectional or bi-directional

# IPC: Message Passing (cont.)

- Implementation of communication link
  - Physical
    - Shared memory
    - HW bus
    - Network

Before we further discuss the MP, the communication should be discussed first

# Direct Communication

- Processes must name each other explicitly
  - **`send`**(A, message)
    - Send a message to process A
  - **`receive`**(B, message)
    - Receive a message from process B
- Properties of communication link
  - Established automatically
  - Exists exactly one link
  - May be uni-directional, but usually bi-directional
  - A link is associated with exactly one pair

# 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
  - Share a common mailbox
  - May be associated with many processes
  - Each pair may share several links
  - May be uni-directional or bi-directional

# Indirect Communication (cont.)

- Operations
  - Create a new mailbox (port)
  - 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
  - $P_1$, $P_2$, and $P_3$ share mailbox A
  - $P_1$ sends; $P_2$ and $P_3$ 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 may be either blocking or non-blocking
- Blocking - synchronous
  - Blocking send
  - Blocking receive
- Non-blocking - asynchronous
  - Non-blocking send
  - Non-blocking receive: Valid/ Null
- Different combinations possible
  - Both S/R are blocking, then Rendezvous

# Message Passing (cont.)

- Producer-consumer becomes trivial

```
item next_produced;
while (true){
  /*produce an item in next
  produced*/
  send (next_produced)
}
```

producer

```
item next_consumed;
while (true){
  while (in == out)
  receive(next_consumed);
  /*consume the item in the
  next consumed*/
}
```

consumer

# Buffering

- Queues of messages attached to the link
- Implemented in one of three ways
  - Zero capacity
  - Bounded capacity
  - Unbounded capacity

# Example of IPC: POSIX

- POSIX shared memory
  - Process first creates shared memory segment

```
shm_fd = shm_open(name, O CREAT | O RDWR, 0666);
```

  - Also used to open an existing segment to share it
  - Set the size of the object

```
ftruncate(shm fd, 4096);
```

  - Now the process could write to the shared memory

```
sprintf(shared memory, “Writing to shared
memory”);
```

上海科技大学 ShanghaiTech University

# Example of IPC: POSIX

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main()
{
/* the size (in bytes) of shared memory object */
const int SIZE = 4096;
/* name of the shared memory object */
const char *name = "OS";
/* strings written to shared memory */
const char *message_0 = "Hello";
const char *message_1 = "World!";

/* shared memory file descriptor */
int shm_fd;
/* pointer to shared memory obect */
void *ptr;

   /* create the shared memory object */
   shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);

   /* configure the size of the shared memory object */
   ftruncate(shm_fd, SIZE);

   /* memory map the shared memory object */
   ptr = mmap(0, SIZE, PROT_WRITE, MAP_SHARED, shm_fd, 0);

   /* write to the shared memory object */
   sprintf(ptr,"%s",message_0);
   ptr += strlen(message_0);
   sprintf(ptr,"%s",message_1);
   ptr += strlen(message_1);

   return 0;
}
```

Producer

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main()
{
/* the size (in bytes) of shared memory object */
const int SIZE = 4096;
/* name of the shared memory object */
const char *name = "OS";
/* shared memory file descriptor */
int shm_fd;
/* pointer to shared memory obect */
void *ptr;

   /* open the shared memory object */
   shm_fd = shm_open(name, O_RDONLY, 0666);

   /* memory map the shared memory object */
   ptr = mmap(0, SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);

   /* read from the shared memory object */
   printf("%s",(char *)ptr);

   /* remove the shared memory object */
   shm_unlink(name);

   return 0;
}
```

Consumer

# Communications in C-S Systems

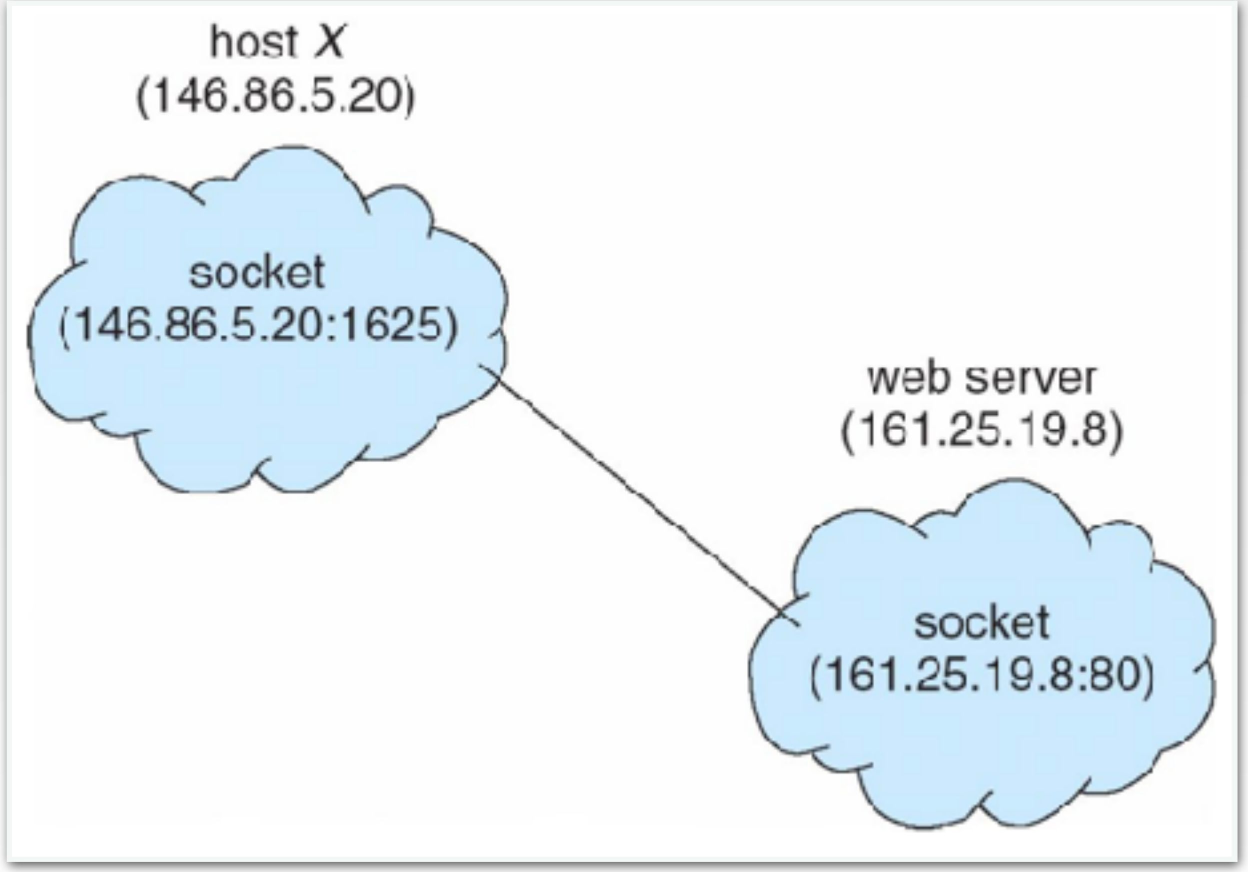
- Sockets
- Remote Procedure Calls
- Pipes

# Sockets

- An endpoint for communication
- Concatenation of IP and Port
- The socket 172.16.254.1:22
  - port 22 on host 172.16.254.1
- Communication consists between a pair of sockets
- All ports below 1024 are well known
  - Used for standard services
- Special IP address 127.0.0.1
  - Loopback
  - Refers to system on which process is running

# Socket Communication

- Three types
  - TCP(Connection-oriented)
  - UDP(Connectionless)
  - MulticastSocket




上海科技大学
ShanghaiTech University

# Connection-oriented Communication

- Session is established before transferring
- Delivered in the same order as it was sent
- Acknowledge after successful delivery
- TCP

# Connectionless Communication

- Message sent from one end to another w/o prior arrangement
- Send w/o ensuring if available and ready
- IP, UDP
- Lower overhead
- Allows for multicast and broadcast
- Error correction to reduce error effects

# Sockets in C

server

```c
/* A simple server in the internet domain using TCP
   The port number is passed as an argument */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

void error(char *msg)
{
    perror(msg);
    exit(1);
}

int main(int argc, char *argv[])
{
     int sockfd, newsockfd, portno, clilen;
     char buffer[256];
     struct sockaddr_in serv_addr, cli_addr;
     int n;
     if (argc < 2) {
         fprintf(stderr,"ERROR, no port provided\n");
         exit(1);
     }
     sockfd = socket(AF_INET, SOCK_STREAM, 0);
     if (sockfd < 0)
        error("ERROR opening socket");
     bzero((char *) &serv_addr, sizeof(serv_addr));
     portno = atoi(argv[1]);
     serv_addr.sin_family = AF_INET;
     serv_addr.sin_addr.s_addr = INADDR_ANY;
     serv_addr.sin_port = htons(portno);
     if (bind(sockfd, (struct sockaddr *) &serv_addr,
              sizeof(serv_addr)) < 0)
              error("ERROR on binding");
     listen(sockfd,5);
     clilen = sizeof(cli_addr);
     newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
     if (newsockfd < 0)
          error("ERROR on accept");
     bzero(buffer,256);
     n = read(newsockfd,buffer,255);
     if (n < 0) error("ERROR reading from socket");
     printf("Here is the message: %s\n",buffer);
     n = write(newsockfd,"I got your message",18);
     if (n < 0) error("ERROR writing to socket");
     return 0;
}
```

client

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

void error(char *msg)
{
    perror(msg);
    exit(0);
}

int main(int argc, char *argv[])
{
    int sockfd, portno, n;

    struct sockaddr_in serv_addr;
    struct hostent *server;

    char buffer[256];
    if (argc < 3) {
       fprintf(stderr,"usage %s hostname port\n", argv[0]);
       exit(0);
    }
    portno = atoi(argv[2]);
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        error("ERROR opening socket");
    server = gethostbyname(argv[1]);
    if (server == NULL) {
        fprintf(stderr,"ERROR, no such host\n");
        exit(0);
    }
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,
         (char *)&serv_addr.sin_addr.s_addr,
         server->h_length);
    serv_addr.sin_port = htons(portno);
    if (connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr)) < 0)
        error("ERROR connecting");
    printf("Please enter the message: ");
    bzero(buffer,256);
    fgets(buffer,255,stdin);
    n = write(sockfd,buffer,strlen(buffer));
    if (n < 0)
         error("ERROR writing to socket");
    bzero(buffer,256);
    n = read(sockfd,buffer,255);
    if (n < 0)
         error("ERROR reading from socket");
    printf("%s\n",buffer);
    return 0;
}
```

上海科技大学
ShanghaiTech University

# Remote Procedure Calls

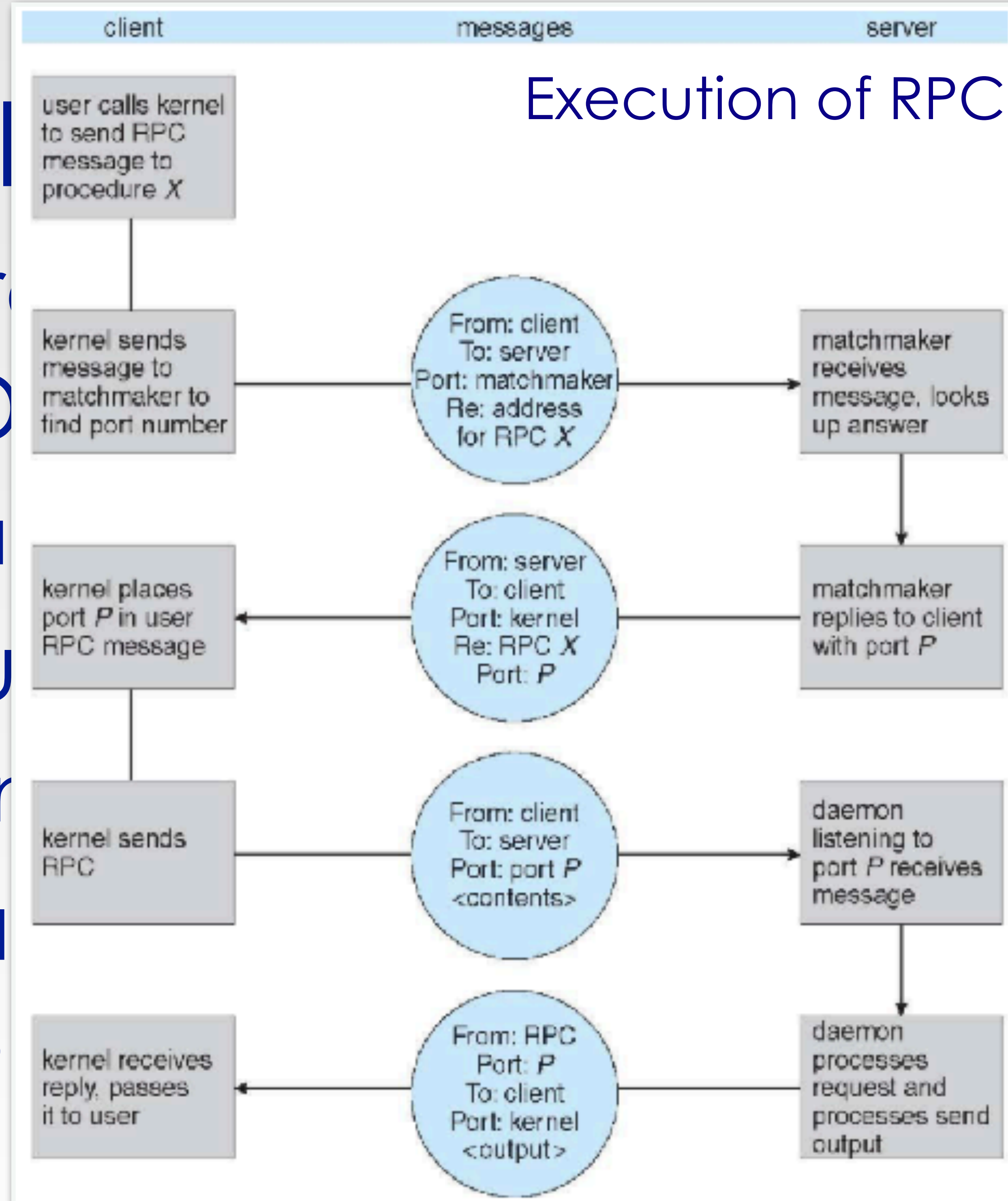
- RPC abstracts procedure calls between processes on networked systems
  - Uses port for service differentiation
- Stubs
  - Client-side proxy for the actual procedure on the server

# Remote Procedure Calls (cont.)

- Client-side stub locates the server and marshalls the parameters
- Server-side stub receives this message, unpacks the marshalled parameters, and performs the procedure on the server

# Remote I (cont.)

- Data repr a External D DL)
  - Big-endia
- More failu vs. Local)
  - Exactly or
- OS typica ous service to erver



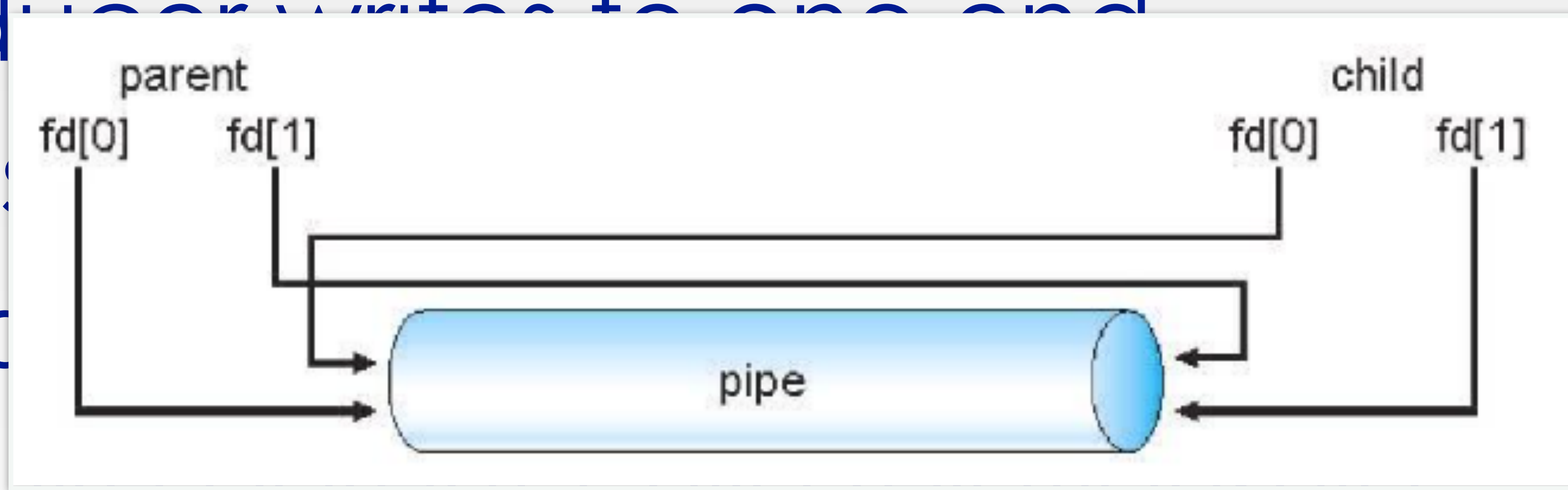

上海科技大学 ShanghaiTech University

# Pipes

- Acts as a conduit allowing two processes to communicate
- Ordinary pipes
  - Can NOT be accessed from the outside
- Named pipes
  - Can be accessed w/t a parent-child relationship

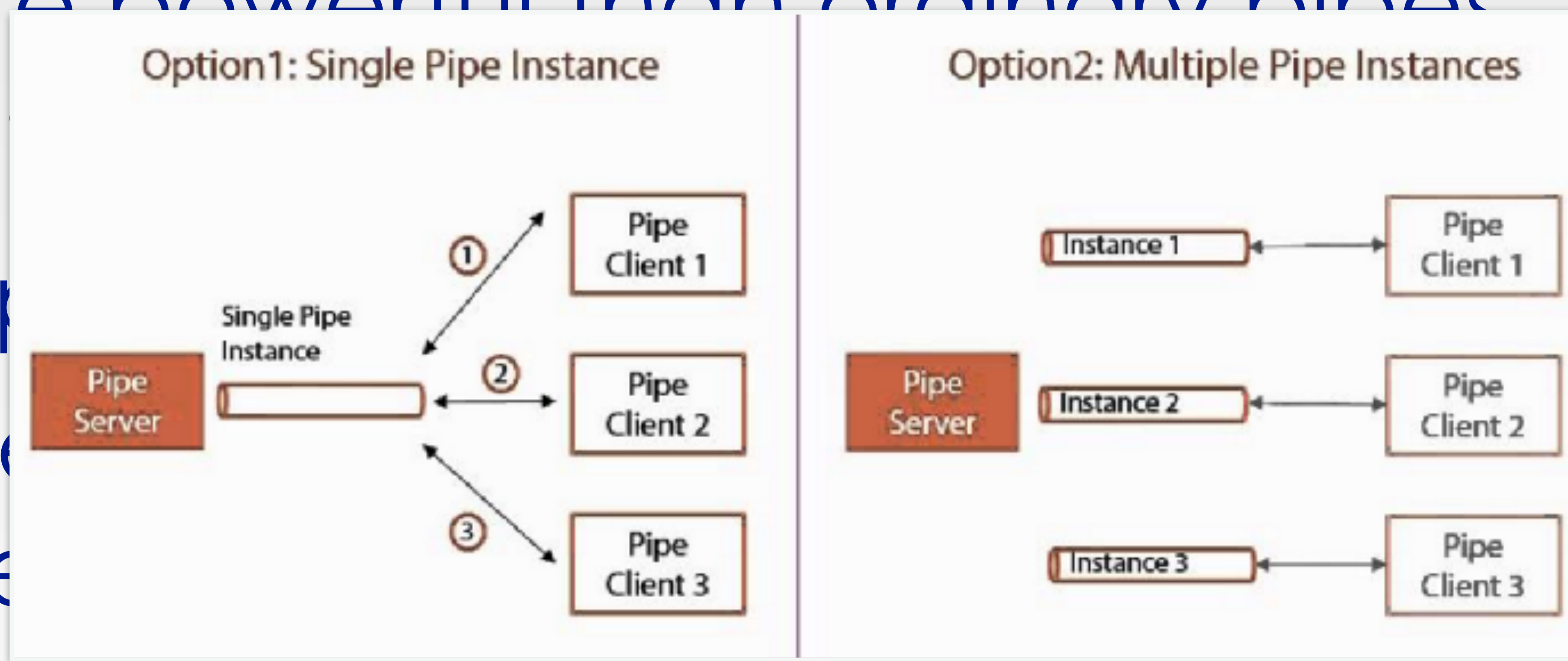
# Ordinary Pipes

- Allow communication in standard Producer-Consumer style
- Producer writes to one end
- Con
- Uni-c
- Require parent-child relationship



# Named Pipes

- More powerful than ordinary pipes
- Bi-d
- No p
- Seve ed
  pipe




- Provided on both UNIX and Windows

# Summary

- IPC
  - Shared Memory
  - Message Passing
  - Sockets



上海科技大学
ShanghaiTech University