IPC

Chapter 3.4, 3.5

Agenda

1. The need to communicate
2. Shared memory
3. Message passing

1. The need to communicate (1)

Independent process

- standalone process

Cooperating process

- affecting or affected by other processes
  - sharing, parallel, modularity, convenience

Process communication

- shared memory
- message passing

1. The need to communicate (2):The producer-consumer problem

Producer

- produce info to be consumed by consumer

Consumer

- consume information produced by producer

Buffer

- unbounded: unlimited buffer size
- bounded: limited buffer size
  - more practical

2. Shared memory solution (1)

Shared memory: memory mapping

- allocated in the calling process's address space
- attached to other processes' address space

Data structure: bounded, circular

```
#define BUFFER_SIZE 10
Typedef struct {. . .} item;
item buffer[BUFFER_SIZE];
int in = 0; int out = 0;
```

- empty, full, # of items

2. Shared memory (2): producer

Producer

- wait for an available space
- update in

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

2. Shared memory (3): consumer

Consumer

- wait for an available item
- update out

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


3. Message passing (1)

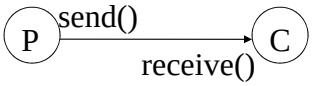
Message passing: an interface

Send message

- send()

Receive message

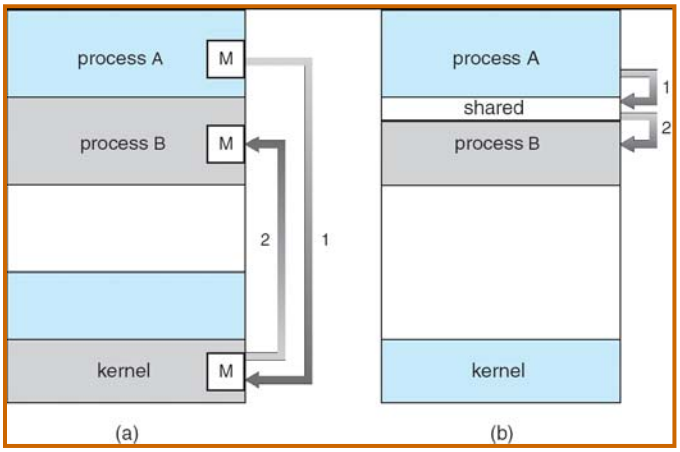
- receive()



Communication *link*

- physical (e.g., memory, bus, network)
- logical (e.g., logical properties)

3. Message passing (2): message passing vs shared memory



3. Message passing (3): Direct communication

Send a message to process C

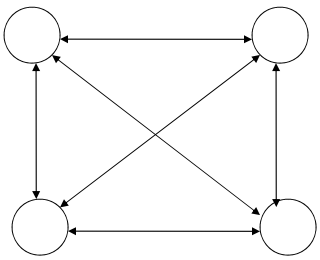
- **send** (*C, message*)

Receive a message from process P

- **receive**(*P, message*)

Communication links

- one link for one pair
- one pair needs one link
  - usually bi-directional



3. Message passing (4): Indirect communication

Send a message to mailbox A

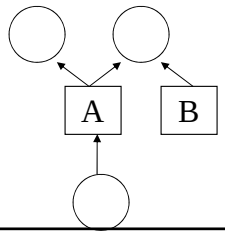
- **send**(*A, message*)

Receive a message from mailbox A

- **receive**(*A, message*)

Communication links and mailboxes

- one link by many pairs
- many links for one pair
  - mailbox owner



3. Message passing (5): Synchronization

Blocking vs non-blocking

- blocking send
  - caller blocked until send is completed
- blocking receive
  - caller blocked until receive is finished
- non-blocking send
- non-blocking receive

Blocking: a means of synchronization

3. Message passing (6): Buffering

Buffer: to hold message temporary

- zero capacity
  - sender blocks until receiver is ready
  - otherwise, message is lost
- bounded capacity
  - when buffer is full, sender blocks
  - when buffer is not full, no need to block sender
- unbounded capacity
  - no need to block sender