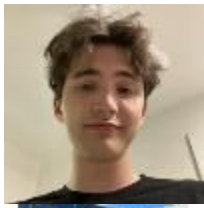
CSc 360
Operating Systems
Inter-process communications

Jianping Pan

Fall 2023

Feedback on A0

- We have a diverse student body
  - cs, se, with va, math, geo, hinf, music, psyc
- All have used some operating systems
  - and want to know more on “what’s **in** OS”
- Thanks to our course reps: AAA
  - T01: Hayden
  - T02: Jack (Ziming)
  - T03: Cristina



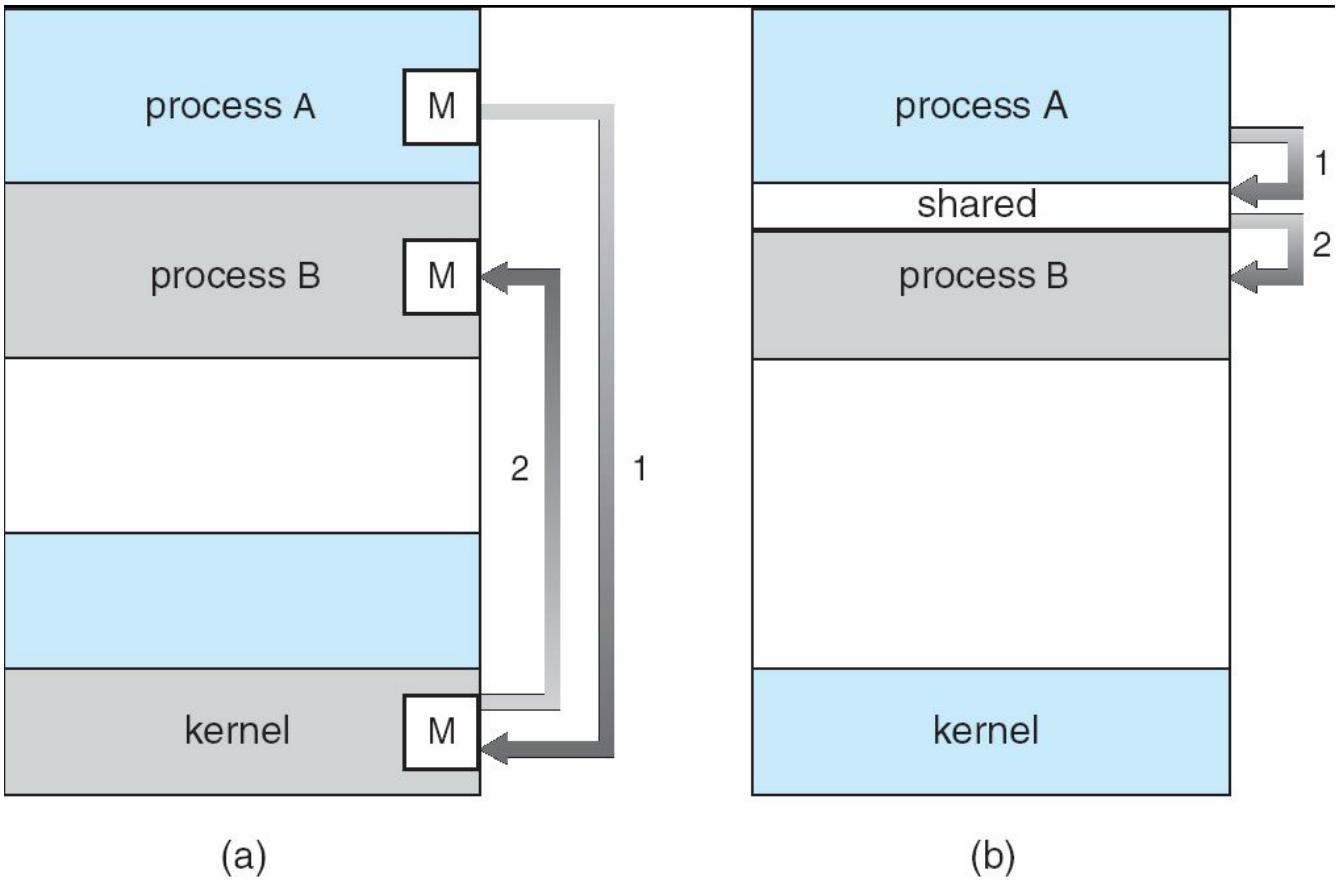

* we welcome students feedback directly as well

Review: the need to communicate

- Independent process
  - standalone process
- Cooperating process
  - affecting or affected by other processes
    - sharing, parallel, modularity, convenience
- Process communication
  - shared memory
  - message passing

* why fork() and then exec() for a new program? functionality? perf?

Message passing vs shared memory



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

Shared memory solution

- 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

* what's the limitation of such a ring buffer?

Shared memory: 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 (busy loop, will be improved later) */
buffer[in] = nextProduced;
in = (in + 1) % BUFFER_SIZE;   }
```


* how to fully utilize the ring buffer?

Shared memory: consumer

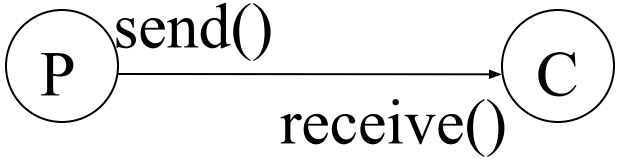
- Consumer
  - wait for an available item
  - update out

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


* how procedure and consumer synchronize in this example?

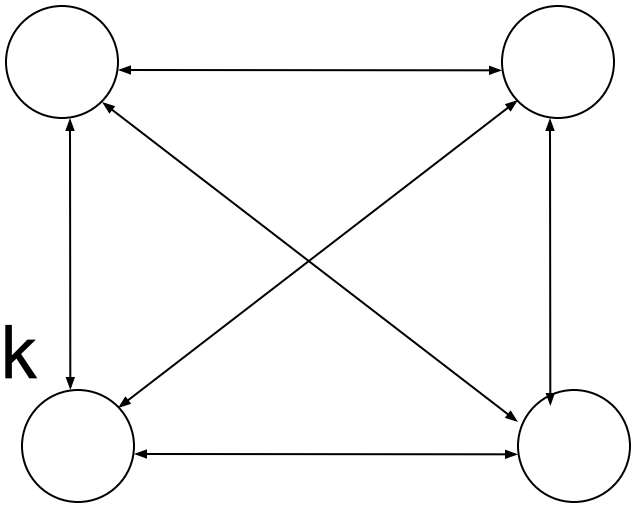
Message passing

- Message passing: an interface
- Send message
  - send()
- Receive message
  - receive()
- Communication *link*
  - physical (e.g., memory, bus, network)
  - logical (e.g., logical properties)



Direct communication

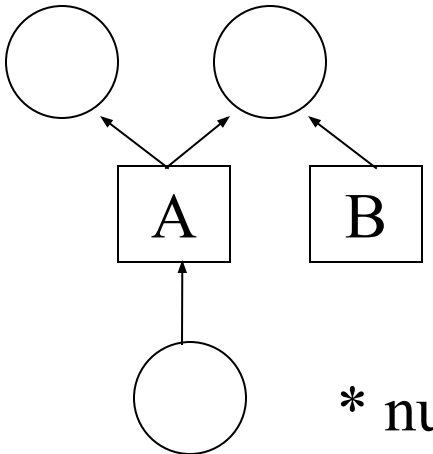
- Send a message to process C
  - **send**(*C, message*)
- Receive a message from process P
  - **receive**(*P, message*)
- Communication links
  - one link for one pair
  - one pair needs one link
    - usually bi-directional



* number of links?

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



* number of links?

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

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

This lecture

- IPC
  - shared memory
  - message passing
    - direct vs indirect
    - blocking vs non-blocking
    - buffered vs unbuffered
- Explore further
  - self-test questions in textbook

Next lecture

- Threads
  - read OSC7 Chapter 4 (or OSC6 Chapter 5)