OS 2025 Lab1

Shared Memory & Message Passing

Due Date: 2025/10/17 17:00 (before lab1 course finishes)

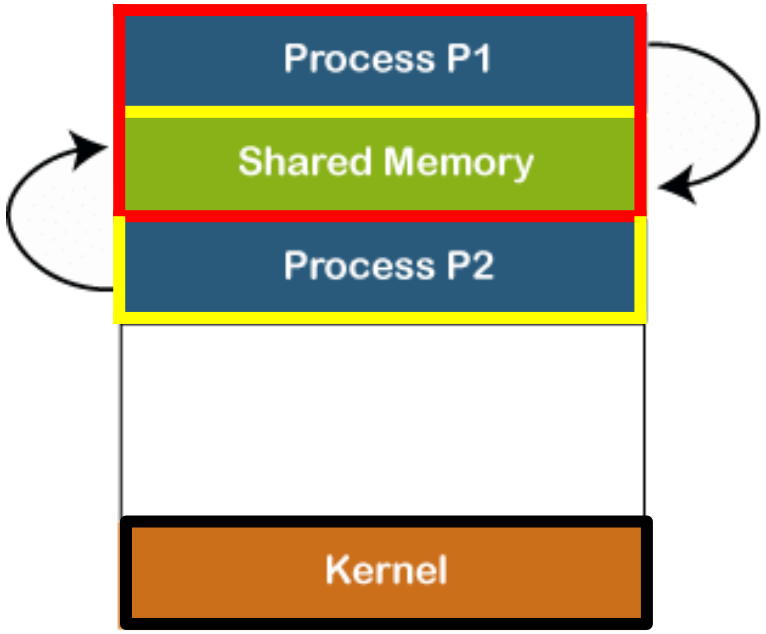
TA：鄭宇辰、張庭瑋、黃柏盛
Email：p76134715@gs.ncku.edu.tw、p76144906@gs.ncku.edu.tw、vx6142035@gs.ncku.edu.tw

Outline

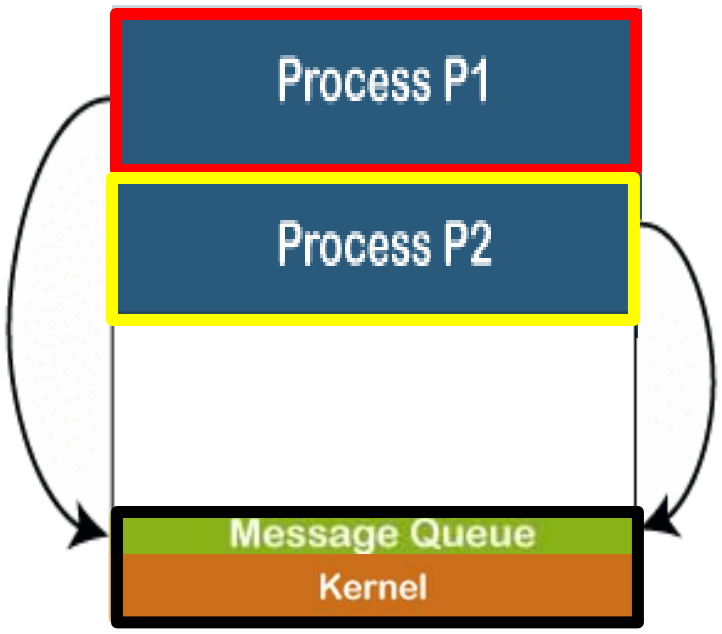
1. Overview
2. Requirement & Flow
3. Related Works

Overview - Inter Process Communication (IPC)

Approaches to Inter Process Communication

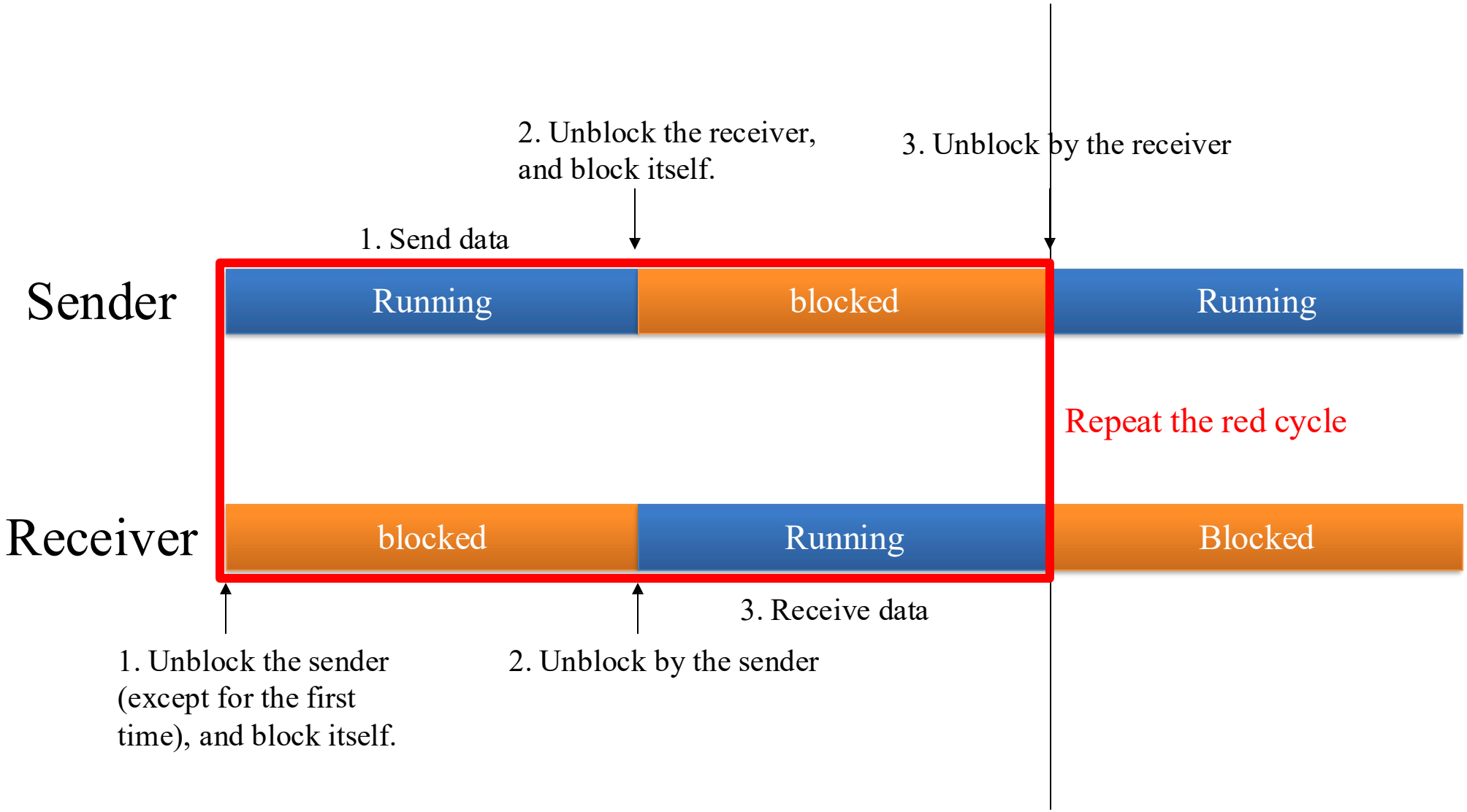


Shared Memory



Message Queue

Requirement & Flow



Requirement & Flow

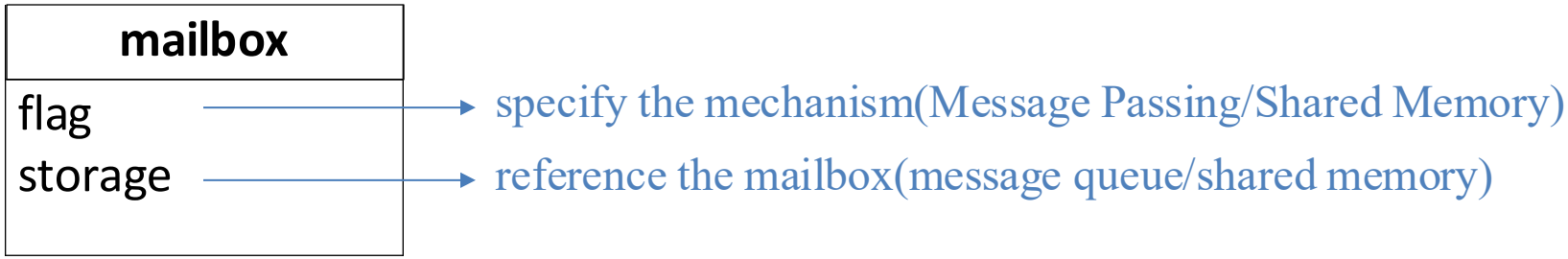
- Implement sender-receiver communication
  1. Implement two wrappers in sender.c and receiver.c
     1) send (message, &mailbox) in sender.c
     2) receive (&message, &mailbox) in receiver.c
     - Implement these wrappers with two mechanisms
       1) Message Passing(Using Message Passing system calls)
       2) Shared Memory(Using Shared Memory)

Requirement & Flow

2. Implement main() in sender.c and receiver.c respectively
   - In main() of sender.c ：
     1) Call send(message, &mailbox) according to the flow in slide 4
     2) Measure the total sending time
     3) Get the mechanism and the input file from command line arguments
        - e.g. ./sender 1 input.txt
          (1 for Message Passing, 2 for Shared Memory)
     4) Get the messages to be sent from the input file
     5) Print information on the console according to the output format
     6) If the message form the input file is EOF, send an exit message to the receiver.c
     7) Print the total sending time and terminate the sender.c
   - In main() of receiver.c ：
     1) Call receive(&message, &mailbox) according to the flow in slide 4
     2) Measure the total receiving time
     3) Get the mechanism from command line arguments
        - e.g. ./receiver 1
     4) Print information on the console according to the output format
     5) If the exit message is received, print the total receiving time and terminate the receiver.c

Mailbox Structure

- TA will provide the mailbox structure for you to implement these two mechanisms



Format of Input File

- Lines of messages
  - Message size: 1-1024 bytes
  - No blank lines

```
1     first message
2     second message
3     third message
4     fourth message
5     fifth message
6     sixth message
7     seventh message
8     eighth message
9     ninth message
10    tenth message
```

Output Format

Sender

```
mephen@2024oslab-VirtualBox:/media/sf_shared_folder/lab1_sender_recevier_modified$ ./sender 1 input.txt
Message Passing
Sending message:  first message
Sending message:  second message
Sending message:  third message
Sending message:  fourth message
Sending message:  fifth message
Sending message:  sixth message
Sending message:  seventh message
Sending message:  eighth message
Sending message:  ninth message
Sending message:  tenth message
End of input file! exit!
Total time taken in sending msg: 0.000053 s
```

Receiver

```
mephen@2024oslab-VirtualBox:/media/sf_shared_folder/lab1_sender_recevier_modified$ ./receiver 1
Message Passing
Receiving message: first message
Receiving message: second message
Receiving message: third message
Receiving message: fourth message
Receiving message: fifth message
Receiving message: sixth message
Receiving message: seventh message
Receiving message: eighth message
Receiving message: ninth message
Receiving message: tenth message
Sender exit!
Total time taken in receiving msg: 0.000041 s
```


Time Measurement

- How to measure time spent on sending / receiving messages
  - Only measure the time spent on action related to communication:
    - Sending / Receiving messages via Message Passing system call
    - Accessing the shared memory
  - Don’t measure the time spent on action unrelated to communication, like:
    - Waiting to be unblocked
    - Printing messages

Example:

```c
#include <time.h>
struct timespec start, end;
double time_taken;

clock_gettime(CLOCK_MONOTONIC, &start);
send(message, &mailbox);
clock_gettime(CLOCK_MONOTONIC, &end);

time_taken = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) * 1e-9;
```

Related Works - Semaphore

- Semaphore can be used for Synchronization
- Semaphore S – integer variable
- Two standard operations modify S: wait() and signal()

```
– wait (S) {
        while S <= 0
            ; // no-op
        S--;
    }
– signal (S) {
        S++;
    }
```

Related Works - Deadlock

- Deadlock – two or more processes are waiting infinitely for an event that can be caused by only one of the waiting processes
- Let S and Q be two semaphores initialized to 1

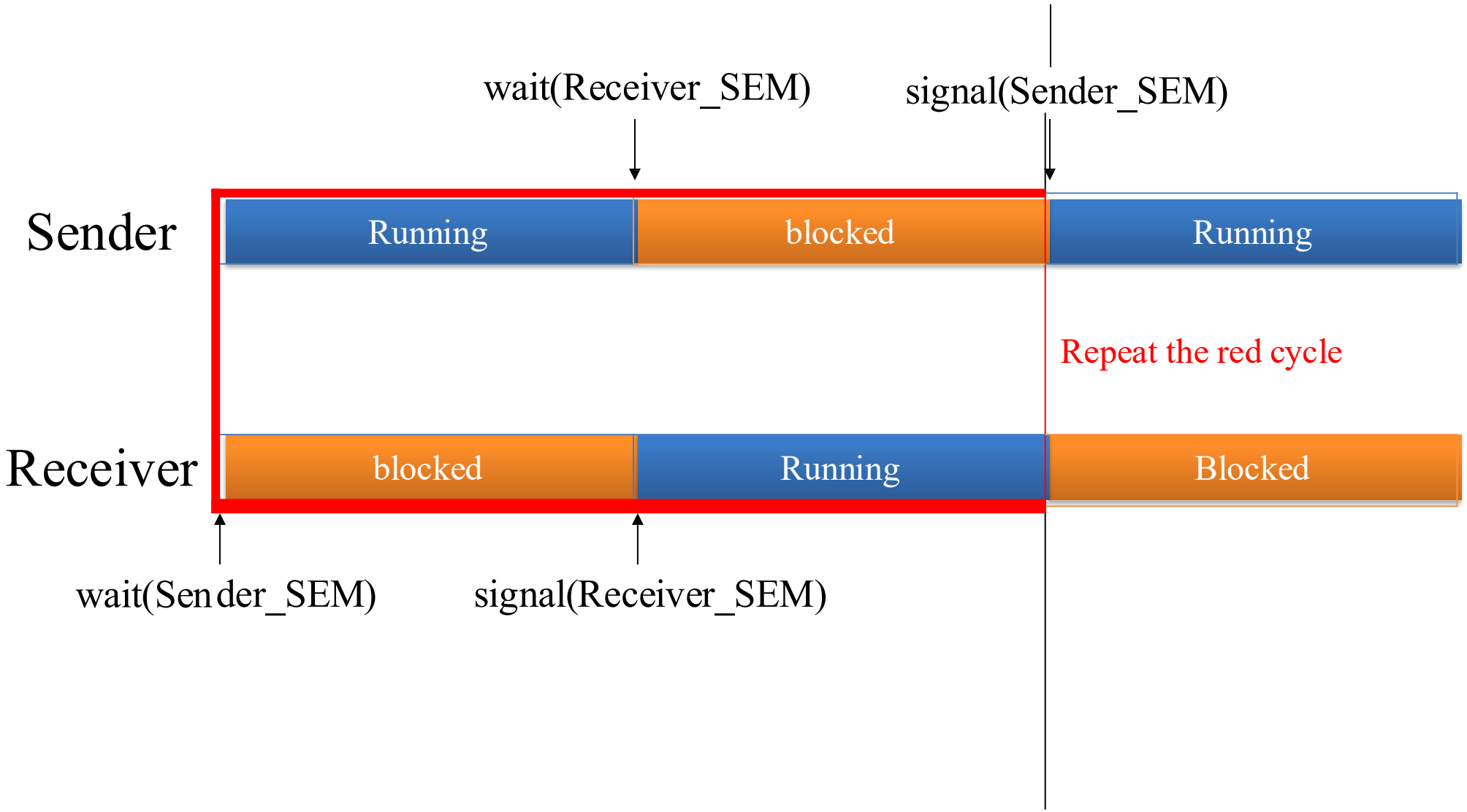
P_0	P_1
wait (S);	wait (Q);
wait (Q);	wait (S);
.	.
.	.
.	.
signal (S);	signal (Q);
signal (Q);	signal (S);

- Consider if P_0 executes wait(S) and P_1 wait(Q). When P_0 executes wait(Q), it must wait for P_1. However, P_1 also waits for P_0 when it executes wait(S.)

Related Works – Initialization Deadlock

- If the semaphore was (accidently) initialized as 0, it could also cause the deadlock because no process can get the semaphore.

Related Works - Semaphore



Related Works

- Semaphore APIs
  - System V API： sem_open(), sem_wait(), sem_post(), sem_close(), sem_unlink()
  - POSIX API： semget(), semop(), semctl()
- Shared Memory APIs
  - System V API：shmget(), shmat(), shmdt(), shmctl()
  - POSIX API：shm_open(), mmap(), munmap(), shm_unlink()
- Message Passing APIs
  - System V API：msgget(), msgsnd(), msgrcv(), msgctl()
  - POSIX API：mq_open(), mq_send(), mq_receive(), mq_close(), mq_unlink()

Demo & Grading

1. (2.5 points) Show communication information based on Message Passing.(follow the output format)
2. (2.5 points) Show communication information based on Shared Memory.(follow the output format)
3. (2 points) Compare their performance according to these communication information.
   - Shared-Memory shall be faster than Message-Passing
4. (3 points) Answer 3 questions about your code.

Precautions

- Due Date: 2025/10/17 17:00 (before lab1 course finishes)
- You should implement lab1 with C language.
- You will get 6 files: sender.c/.h, receiver.c/.h, message.txt(input file), makefile from os_2025_lab1_template
  - You can modify makefile as you want, but make sure your makefile can compile your codes and create the executable successfully.