Operating Systems

Practice 3.
Process and IPC

AI Software, Gachon University

Objective

- Create children processes
- Send/Receive messages between processes

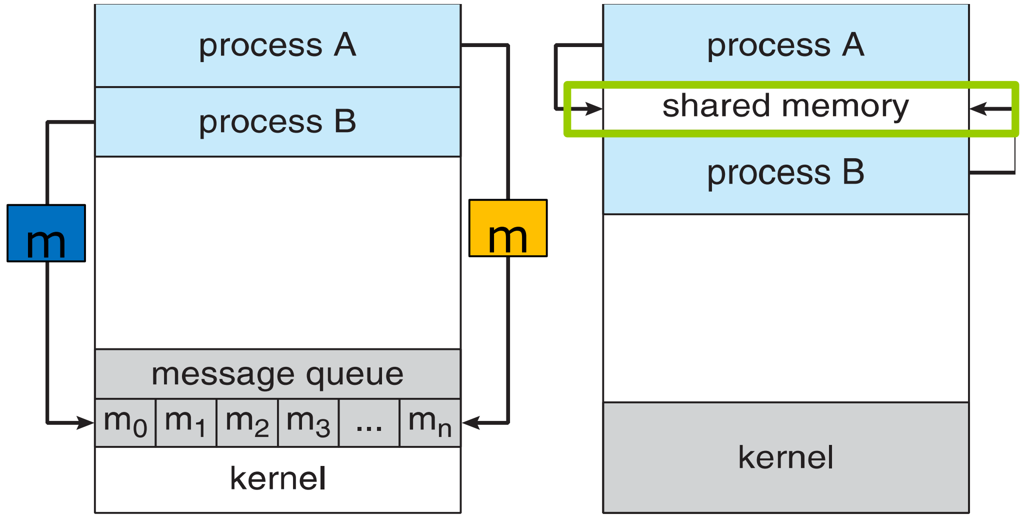
Communications Models

Mechanism

- for processes to **communicate** & to **synchronize** their actions

Two fundamental models of IPC

- **Shared memory**
- **Message passing**



(a) Message passing (b): Shared memory

Interprocess Communication – Message Passing

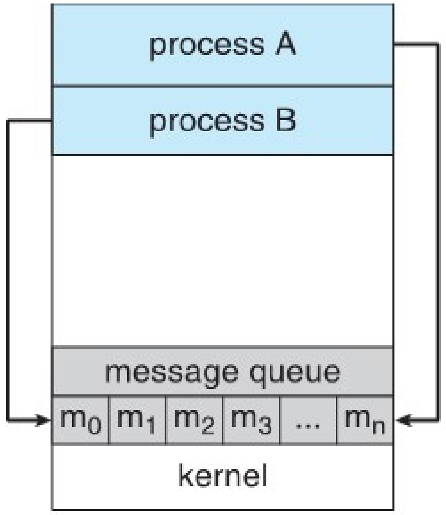
Message system – processes communicate with each other by **messages** without resorting to shared variables

IPC facility provides two operations:

- **`send`**(*message*) **`receive`**(*message*)

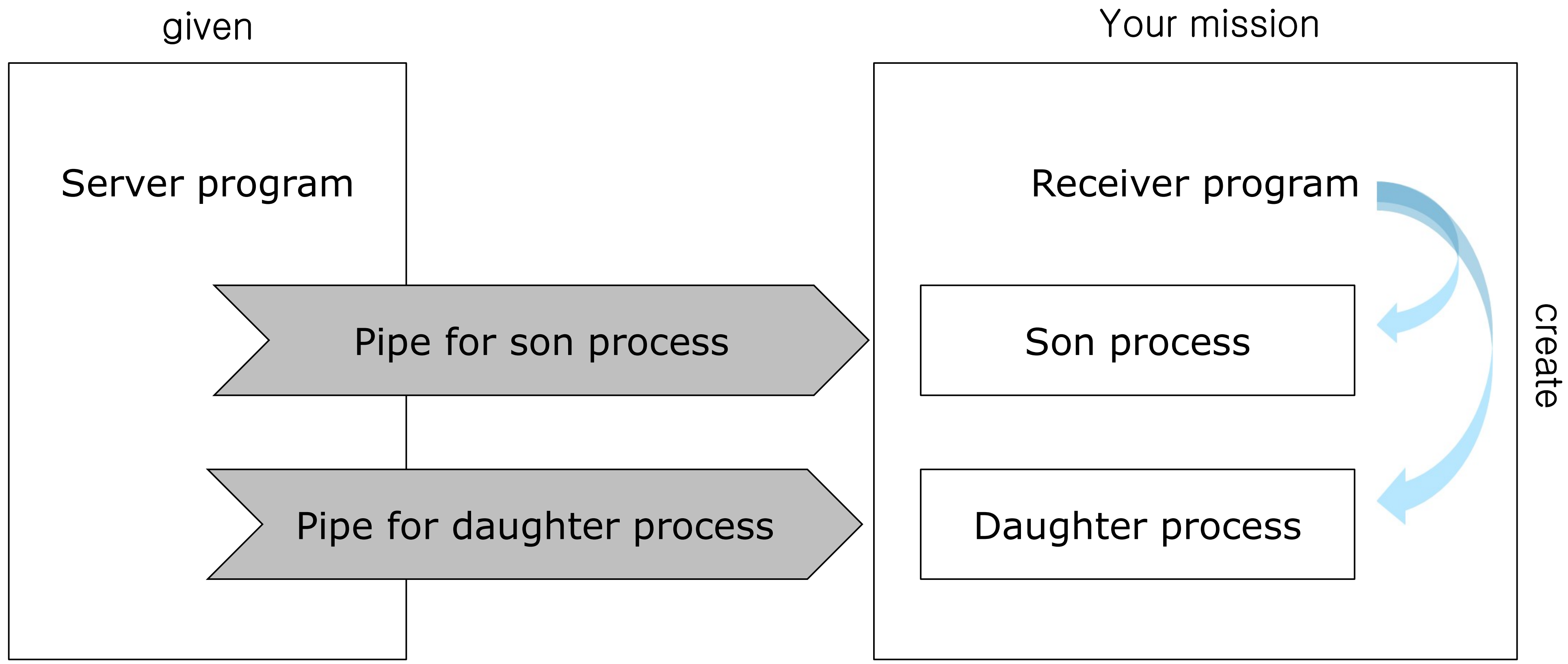
If *P* and *Q* wish to communicate, they need to:

- establish a ***communication link*** between them
- exchange messages via send/receive
  - need ***system call***
  - More time-consuming compared to shared memory
- e.g., **Microkernel structure**
- e.g., **Sockets** (networking), RPC (distributed systems)



Today’s mission

- Overview



Create child process

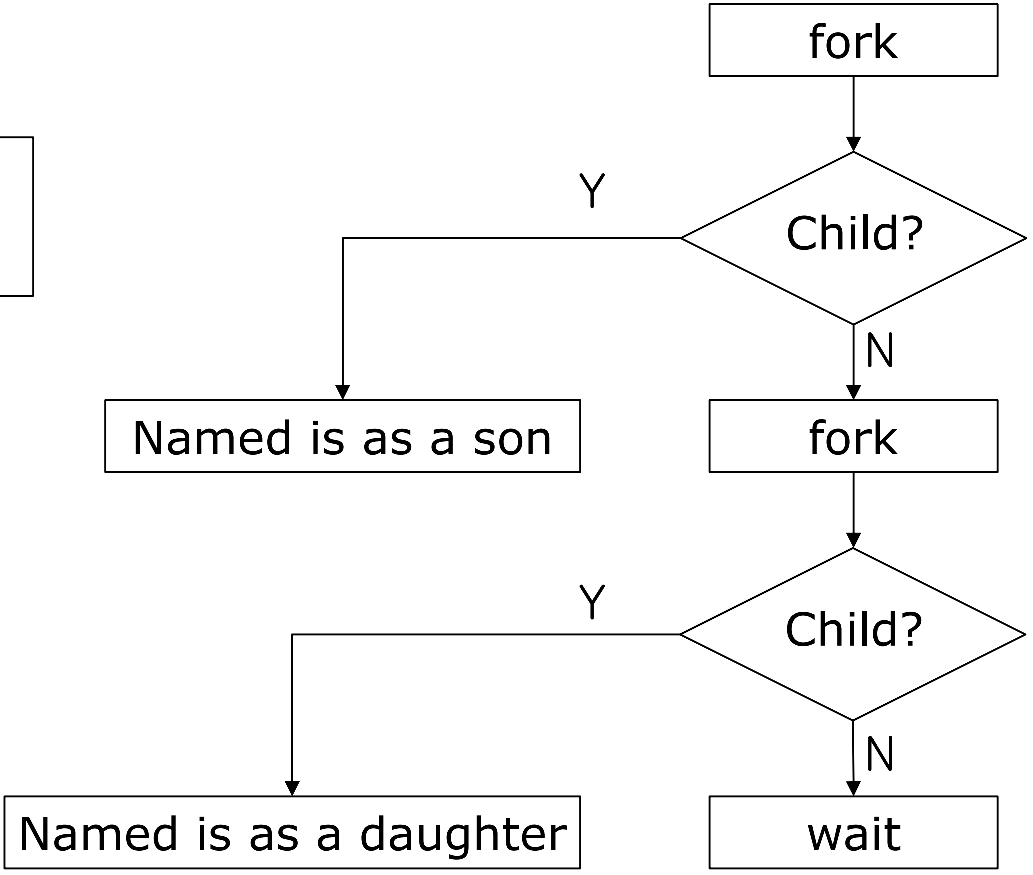
- A process can create child process using system call
  - Fork()
  - pid_t pid = fork()
    - pid < -1 : error
    - pid == 0 : child process
    - pid > 0 : parent process

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       return 1;
    }
    else if (pid == 0) { /* child process */
       execlp("/bin/ls","ls",NULL);
    }
    else { /* parent process */
       /* parent will wait for the child to complete */
       wait(NULL);
       printf("Child Complete");
    }

    return 0;
}
```

Create child process

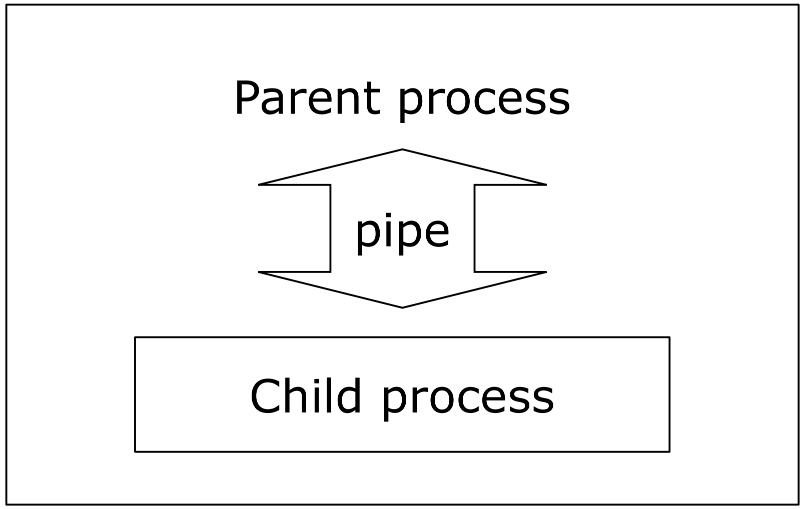
- How to create multiple children?
  - Is this right way?

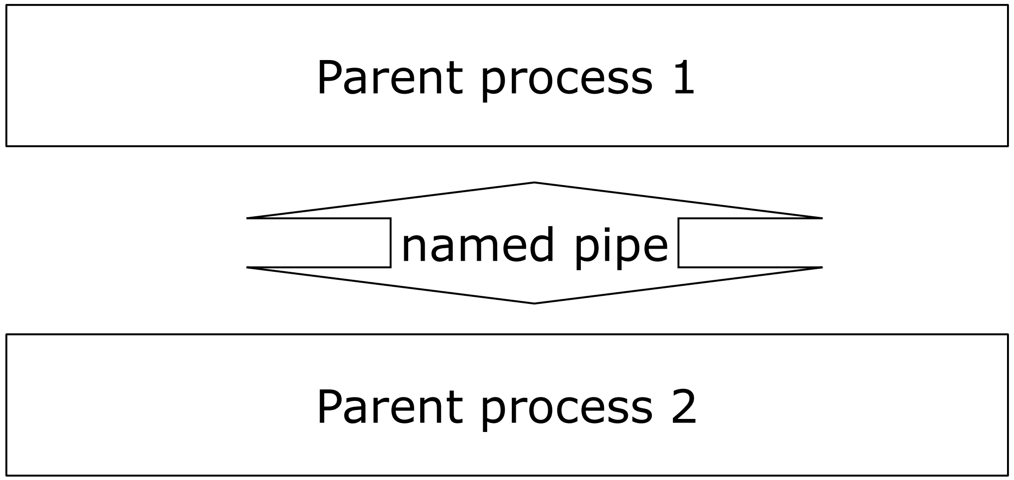
```
pid_t pid1 = fork();
pid_t pid2 = fork();
```



Named pipe

- What is differences between pipe and named pipe
  - Pipe can be used when two processes knows file descriptors' name (parent ←→ child process)
  - Named pipe can be used by any processes which know pipe name





Named pipe

- Create
  - 1. create FIFO(first In First Out)
    - mkfifo([fifo_name], 0666);
  - 2. Create file descriptor
    - int fd = open([fifo_name], O_WRONLY);

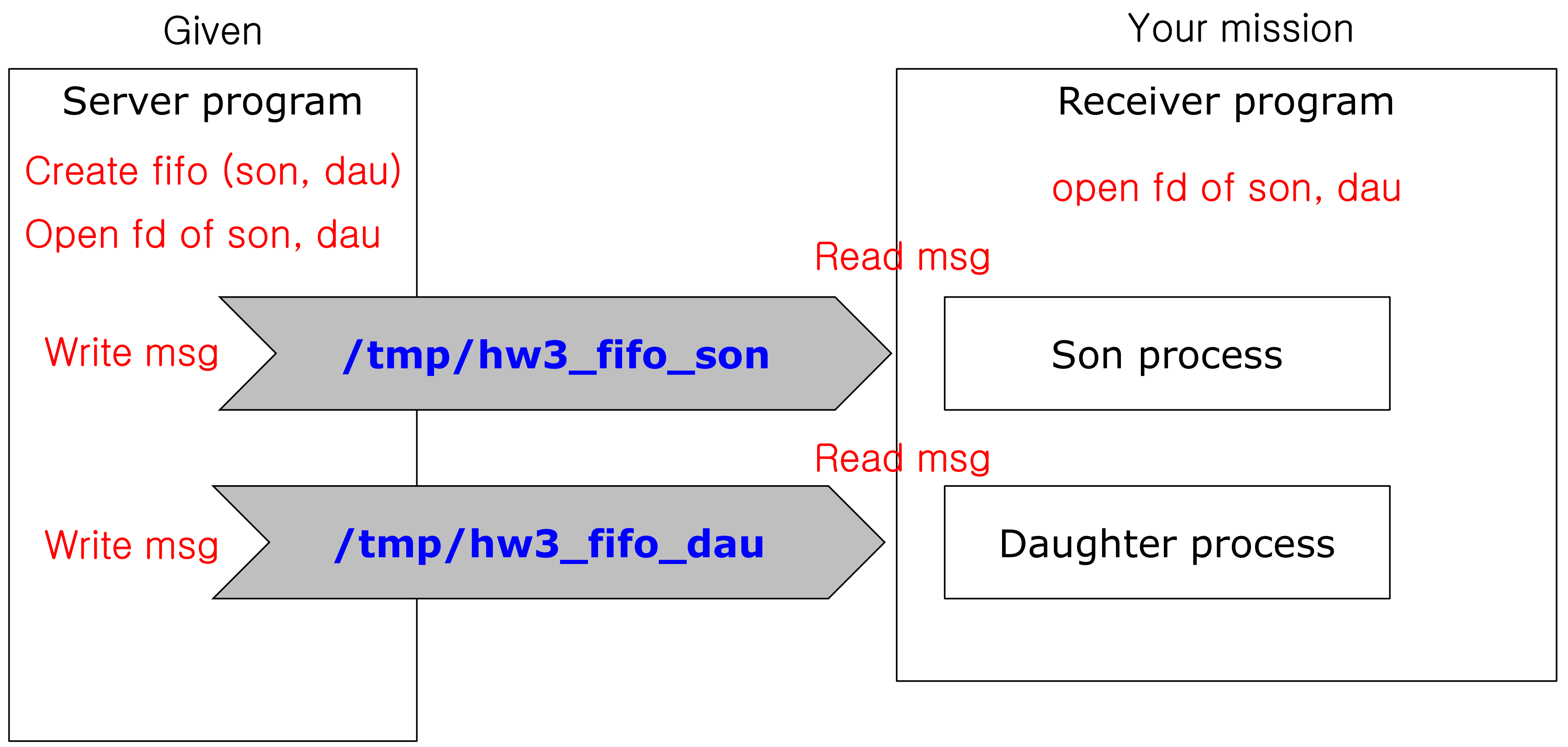
- Terminate
  - Unlink
    - Unlink([fifo_name]);

Named pipe

- Send message
  - Using system call : write
    - write(fd, message(string), sizeof(message));
- Receive Message
  - Using system call: read
    - read(fd, buffer(string of char*), sizeof(buffer));

Today's mission again

- Overview



What to do?

- A server program is given
  - Text-based menu will be shown.
  - First, select a destination of a message.
    - Who do you want to send a message? (1:son/2:daughter/0:end)
  - Then, write a message
    - What is a message?
    - Internally, message would be sent trough each named-pipe.

What to do?

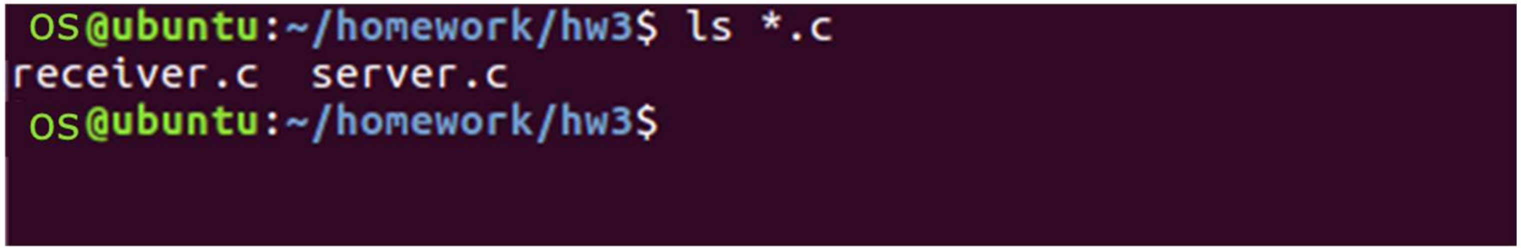
- A receiver program should be built by your own
  - Create two children processes
    - Son and daughter (or daughter and son)
      - When a process is created, print out their process name.
    - Each process should open assigned named-pipe
      - Son: /tmp/hw3_fifo_son
      - Daughter: /tmp/hw3_fifo_dau
    - Wait until new message comes.
      - Then, display new messages from server

Order of run (example)

- Make two source files
  - server.c (will be given) / receiver.c
- Build each source file
  - gcc –o server server.c
  - gcc –o receiver receiver.c
  - Then, server and receiver file would be created.
- Run server first
- Then run receiver (using other terminal)
- Send message using server menu

Order of run

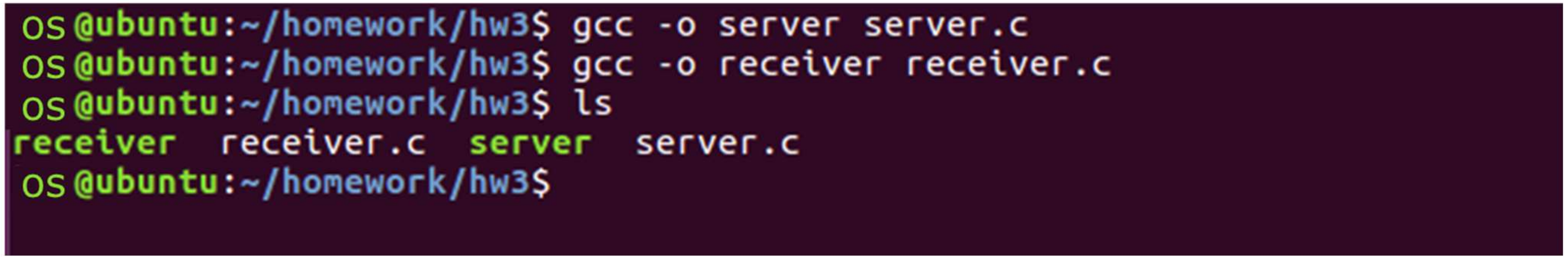
- Make two source files
  - server.c (will be given) / receiver.c



```
os@ubuntu:~/homework/hw3$ ls *.c
receiver.c  server.c
os@ubuntu:~/homework/hw3$
```

Order of run (example)

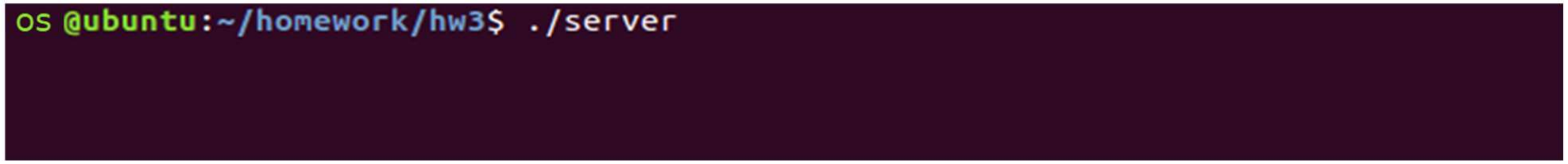
- Build each source file
  - gcc –o server server.c
  - gcc –o receiver receiver.c
  - Then, server and receiver file would be created.



```
os@ubuntu:~/homework/hw3$ gcc -o server server.c
os@ubuntu:~/homework/hw3$ gcc -o receiver receiver.c
os@ubuntu:~/homework/hw3$ ls
receiver  receiver.c  server  server.c
os@ubuntu:~/homework/hw3$
```

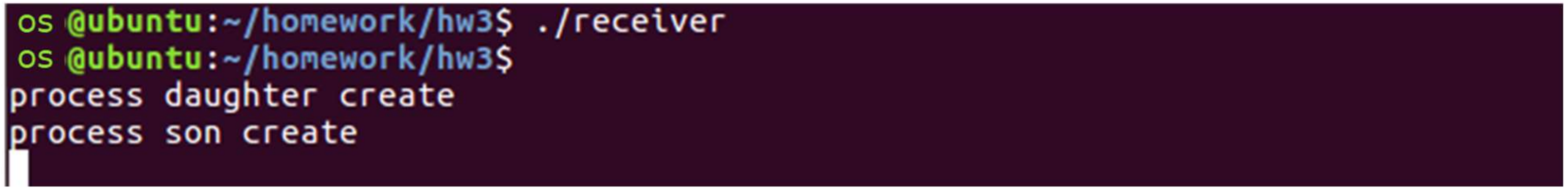
Order of run (example)

- Run server first
  - Nothing is shown (actually, this is waiting for connecting of FIFO)



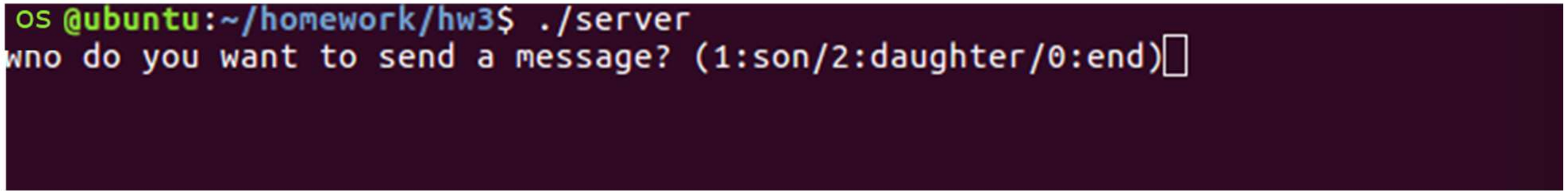
```
os @ubuntu:~/homework/hw3$ ./server
```

- Then run receiver (using other terminal)



```
os @ubuntu:~/homework/hw3$ ./receiver
os @ubuntu:~/homework/hw3$
process daughter create
process son create
```

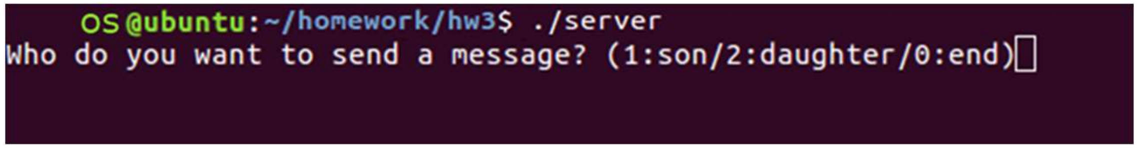
- Then the menu of server will be shown



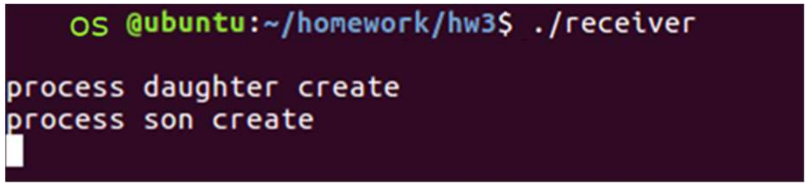
```
os @ubuntu:~/homework/hw3$ ./server
who do you want to send a message? (1:son/2:daughter/0:end)
```

Order of run (example)

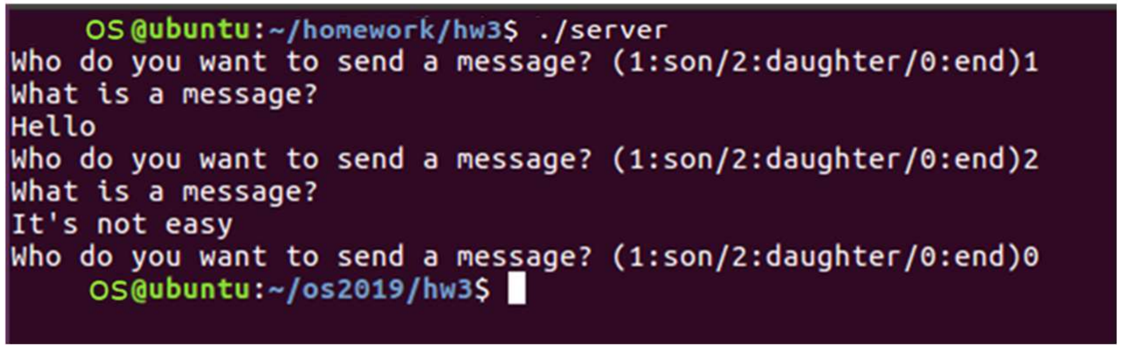
- Send message using server menu



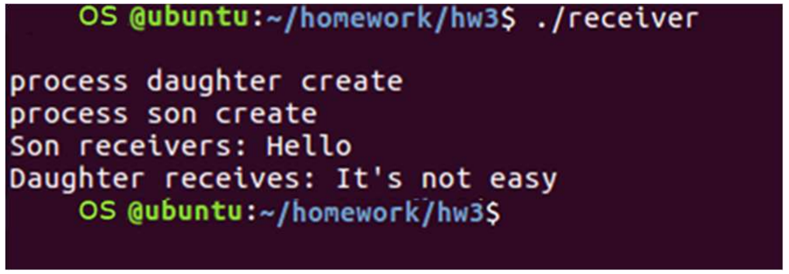
```
    OS @ubuntu:~/homework/hw3$ ./server
Who do you want to send a message? (1:son/2:daughter/0:end)
```



```
    OS @ubuntu:~/homework/hw3$ ./receiver

process daughter create
process son create
```



```
    OS @ubuntu:~/homework/hw3$ ./server
Who do you want to send a message? (1:son/2:daughter/0:end)1
What is a message?
Hello
Who do you want to send a message? (1:son/2:daughter/0:end)2
What is a message?
It's not easy
Who do you want to send a message? (1:son/2:daughter/0:end)0
    OS@ubuntu:~/os2019/hw3$
```



```
    OS @ubuntu:~/homework/hw3$ ./receiver

process daughter create
process son create
Son receivers: Hello
Daughter receives: It's not easy
    OS @ubuntu:~/homework/hw3$
```

code analysis

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define MAX_BUF1024

int main(int argc, char *argv[])
{
    // variable
    char ch, tch;
    int terminate = 0;

    // init named pipe
    char message[MAX_BUF];
    int file_desc_son;
    int file_desc_dau;

```

Include files

Maximum buffer size

File descriptor for son/dau fifo

code analysis

- // open named fifo
- char *fifo_son = "/tmp/hw3_fifo_son";
- char *fifo_dau = "/tmp/hw3_fifo_dau";
- mkfifo(fifo_son, 0666);
- mkfifo(fifo_dau, 0666);
- file_desc_son = open(fifo_son, O_WRONLY);
- file_desc_dau = open(fifo_dau, O_WRONLY);

Assign fifo names

Create fifos

File open to use fifo (write only)
You also use this code when receiver use file descriptor. However, in that case you should open file descriptors using read only option.

code analysis

- // open named fifo
- char *fifo_son = "/tmp/hw3_fifo_son";
- char *fifo_dau = "/tmp/hw3_fifo_dau";
- mkfifo(fifo_son, 0666);
- mkfifo(fifo_dau, 0666);
- file_desc_son = open(fifo_son, O_WRONLY);
- file_desc_dau = open(fifo_dau, O_WRONLY);

Assign fifo names

Create fifos

File open to use fifo (write only)
You also use this code when receiver use file descriptor. However, in that case you should open file descriptors using read only option.

code analysis

```
while(!terminate)
{
    printf("Who do you want to send a message? (1:son/2:daughter/0:end)");
    ch = getchar();
    if (ch == '0') {
        terminate = 1;
        write(file_desc_son, "0", sizeof(MAX_BUF));
        write(file_desc_dau, "0", sizeof(MAX_BUF));
        continue;
    }
    else if (ch == '1' || ch == '2') {
        scanf("%c", &tch);
        printf("What is a message?\n");
        scanf("%[^\n]", message);
        // send message
        if (ch == '1') write(file_desc_son, message, sizeof(message));
        else write(file_desc_dau, message, sizeof(message));
        // flush remaining new line command
        scanf("%c", &tch);
    }
    else if (ch == '\n') continue;
}
```

Menu handler

1. Select destination of message
2. Send message
3. If ‘0’ is selected, this code send ‘0’ to son/daughter process and terminate. So, receiver code also terminate if received message is ‘0’

code analysis

- // file close
- close(file_desc_son);
- close(file_desc_dau);

- // unlink fifo
- unlink(fifo_son);
- unlink(fifo_dau);

- return 1;
- }

Unlink fifo

Hint

- How to design receiver.c
  1. Fork twice (son, daughter)
  2. For each child process (use if pid == 0)
     1. Set a buffer (Ex. Char buf[10000])
     2. Open message_queue: Same with server but use O_RDONLY instead of O_WRONLY
     3. Use the same name with server (Ex. /tmp/hw3_fifo_son or /tmp/hw3_fifo_dau)
     4. For Loop
        1. Read buffer (refer page 10)
        2. Display the received message
     5. Until get ‘0’ (means buf[0] == ‘0’)
     6. Close message_queue (Ex. Close(queue_name))
     7. Parent should wait its children (Ex. Wait(NULL))

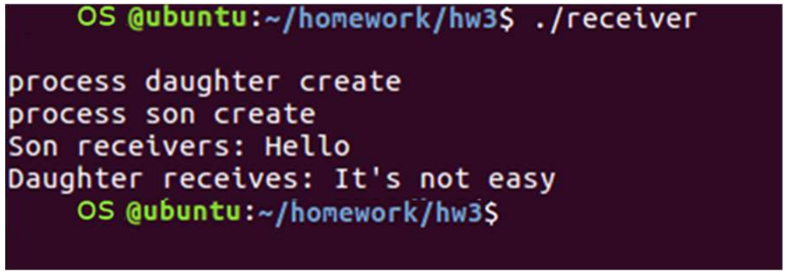
Due

- Due to April. 13. 23:59

- Upload receiver.c file only
  - **Change name to "receiver_ID_NAME.c", e.g., receiver_202212345_JungchanCho.c**
  - **Insert your ID and Name on the top of the code as comments (//)**

- Check point
  - 1. Build success (20)
  - 2. When I run 'receiver', two messages which mean son and daughter process are created well are shown. (30)
  - 3. When I send message to son and daughter processes, they should display the message.(50)

- Good Luck



```
OS @ubuntu:~/homework/hw3$ ./receiver

process daughter create
process son create
Son receivers: Hello
Daughter receives: It's not easy
OS @ubuntu:~/homework/hw3$
```