# 【2017 Network System Programming Homework 4】

## ★ In this assignment, you should：

1. Know the signal inheritance of between parent process and child process
2. Understand how much time did the process spend in user/kernel space

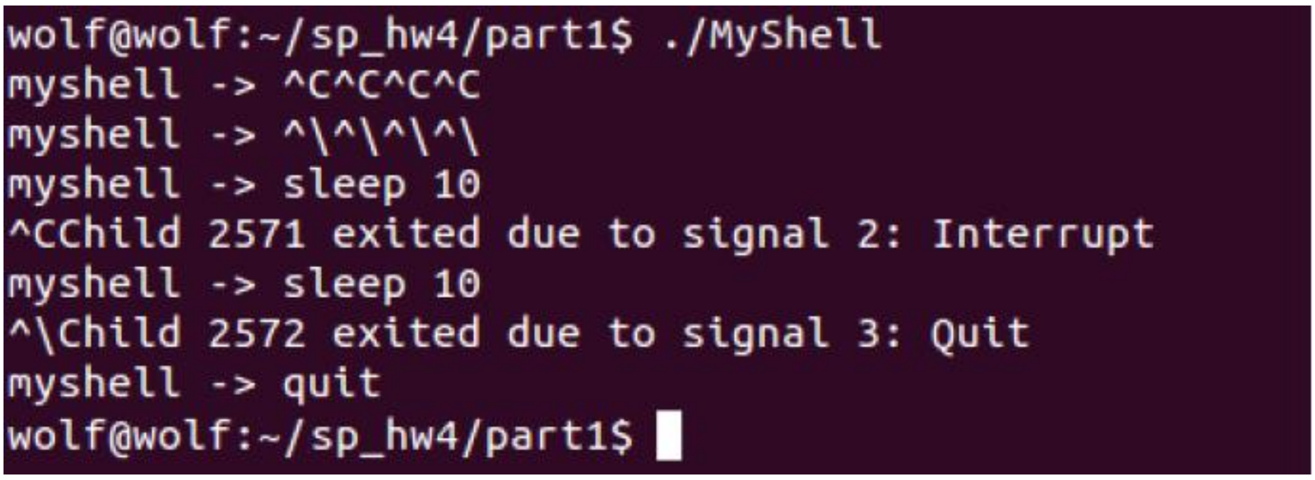
## ★ **Part 1**：

1. Following the part2 of Homework 2 to do this part. When fork() create a child process, it clones the signal handling from the parent. If the parent is set up to ignore signals before forking, the child will inherit this behavior as well.

   Signals ignored by a parent are propagated to the child automatically. The child can then catch or ignore the signal as appropriate.

   Edit the shell.c file so that the parent ignores the interrupt (Ctrl+c) and quit (Ctrl+\) signals. Edit the run_command.c file so that the child reverts to default signal handling.

2. The result is shown as follows：



```
wolf@wolf:~/sp_hw4/part1$ ./MyShell
myshell -> ^C^C^C^C
myshell -> ^\^\^\^\
myshell -> sleep 10
^CChild 2571 exited due to signal 2: Interrupt
myshell -> sleep 10
^\Child 2572 exited due to signal 3: Quit
myshell -> quit
wolf@wolf:~/sp_hw4/part1$ 
```

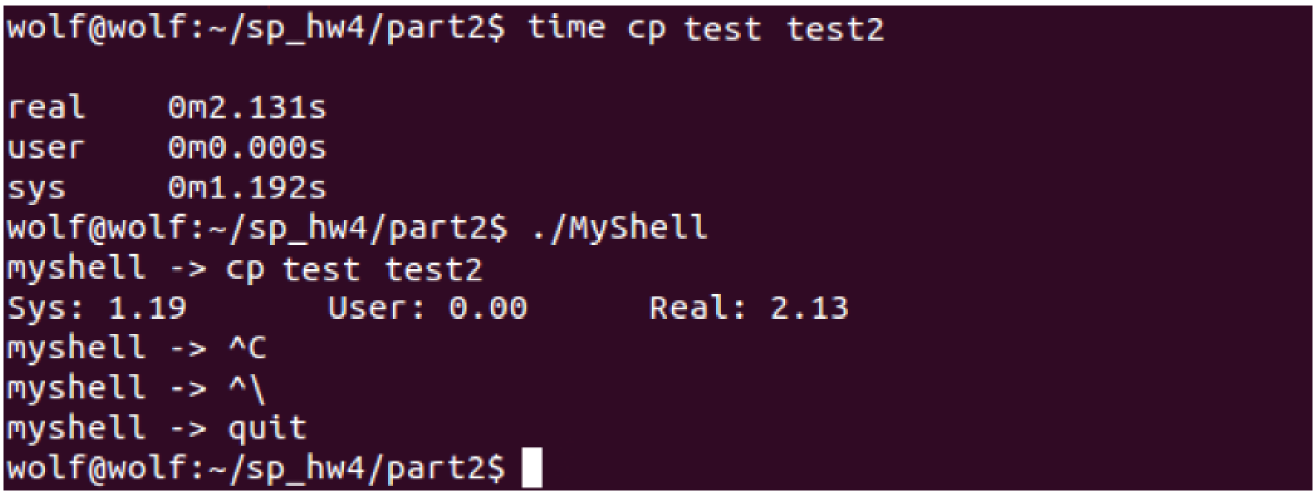
## ★ Part 2：<u>**In this part, you have to include the features of part 1**</u>.

1. In the beginning, you should understand what the values mean in the struct tms. Add two functions, set_timer() and stop_timer(), to the timer.c file. Insert calls to these functions in the run_command.c file.
   Hint: set timer before forking; parent stops the timer after wait.

2. Files provided：

   timer.c

3. The result is shown as follows：(It takes the time which cp spends as an example. In addition, it is allowed to have little error amount of time. )



```
wolf@wolf:~/sp_hw4/part2$ time cp test test2

real    0m2.131s
user    0m0.000s
sys     0m1.192s
wolf@wolf:~/sp_hw4/part2$ ./MyShell
myshell -> cp test test2
Sys: 1.19       User: 0.00      Real: 2.13
myshell -> ^C
myshell -> ^\
myshell -> quit
wolf@wolf:~/sp_hw4/part2$ 
```

## ★ Part 3：

1. Develop a client-server model that communicate through named pipes and look up words in the dictionary. The following steps provide some information about the client and server you are going to implement.
   I. The server is told the name of a FIFO (call it the request FIFO) on which it will listen for requests. It opens the request FIFO and waits for a request. When a request comes in, it does the dictionary look up and then opens the reply FIFO (the name of which is part of the request) and sends the response back through it. The request is of type Client, defined in dict.h.
   II. The client creates a FIFO to receive a response (call it the reply FIFO), then prepares and sends a packet to the server with a word to look up and the name of reply FIFO it just created. It then waits for the reply.
   III. The server is called fifo_server and is built from lookup2.c (a module which performs the dictionary look ups) and fifo_sever.c (which interfaces with the client through the FIFOs).
   IV. The client is called fifo_client and is built from main.c (the user interface module) and lookup3.c (which interfaces with the server through the FIFOs).

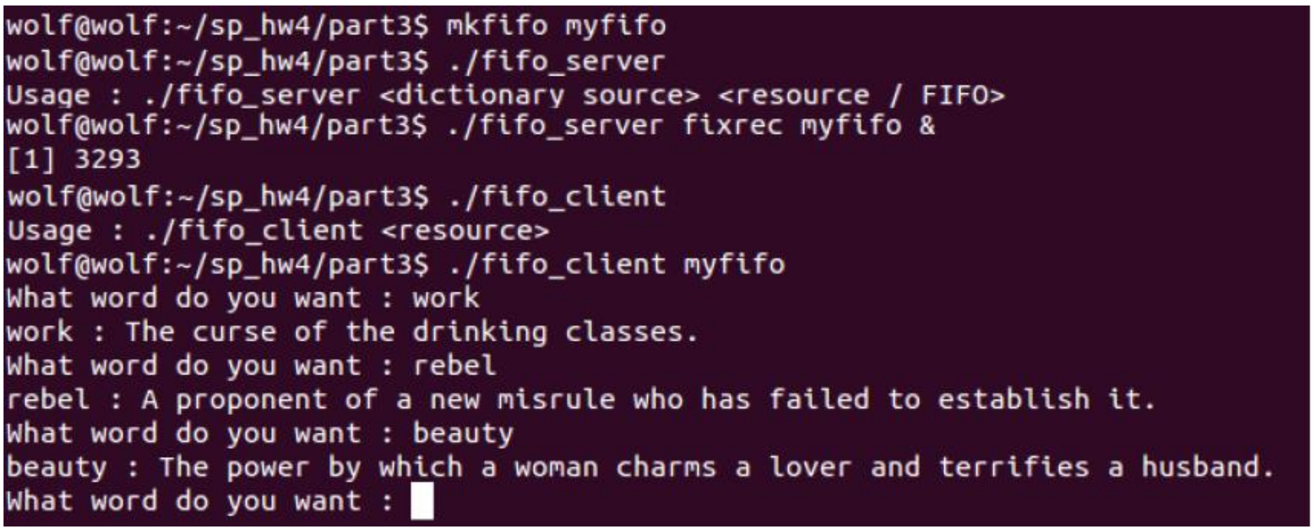
2. The following provides you with some help on how to use the files provided to develop the client and server for this task.

   I. The request FIFO used to send the server a request is created before running the client or server, lookup3. c must create a new reply FIFO for each request because the FIFO's name is part of the request. Edit the lookup3.c file to createthe reply FIFO.
   II. Edit lookup3. c to send a request down the request FIFO and wait for a response.
   III. Edit fifo_server.c to read the request FIFO, invoke the routine in-lookup2.c to do the search, and send the reply back to the client.
   IV. After the files have been edited, type make, or make fifo_server and make fifo_client.
   V. When you get the prompt, make the request FIFO, and run the fifo_server and fifo_client as shown in the sample output

3. Files provided：

   dict.h, main.c, lookup3.c, fifo_server.c, fixrec

4. The result is shown as follows：



```
wolf@wolf:~/sp_hw4/part3$ mkfifo myfifo
wolf@wolf:~/sp_hw4/part3$ ./fifo_server
Usage : ./fifo_server <dictionary source> <resource / FIFO>
wolf@wolf:~/sp_hw4/part3$ ./fifo_server fixrec myfifo &
[1] 3293
wolf@wolf:~/sp_hw4/part3$ ./fifo_client
Usage : ./fifo_client <resource>
wolf@wolf:~/sp_hw4/part3$ ./fifo_client myfifo
What word do you want : work
work : The curse of the drinking classes.
What word do you want : rebel
rebel : A proponent of a new misrule who has failed to establish it.
What word do you want : beauty
beauty : The power by which a woman charms a lover and terrifies a husband.
What word do you want : 
```

## ★ ★Part 4：

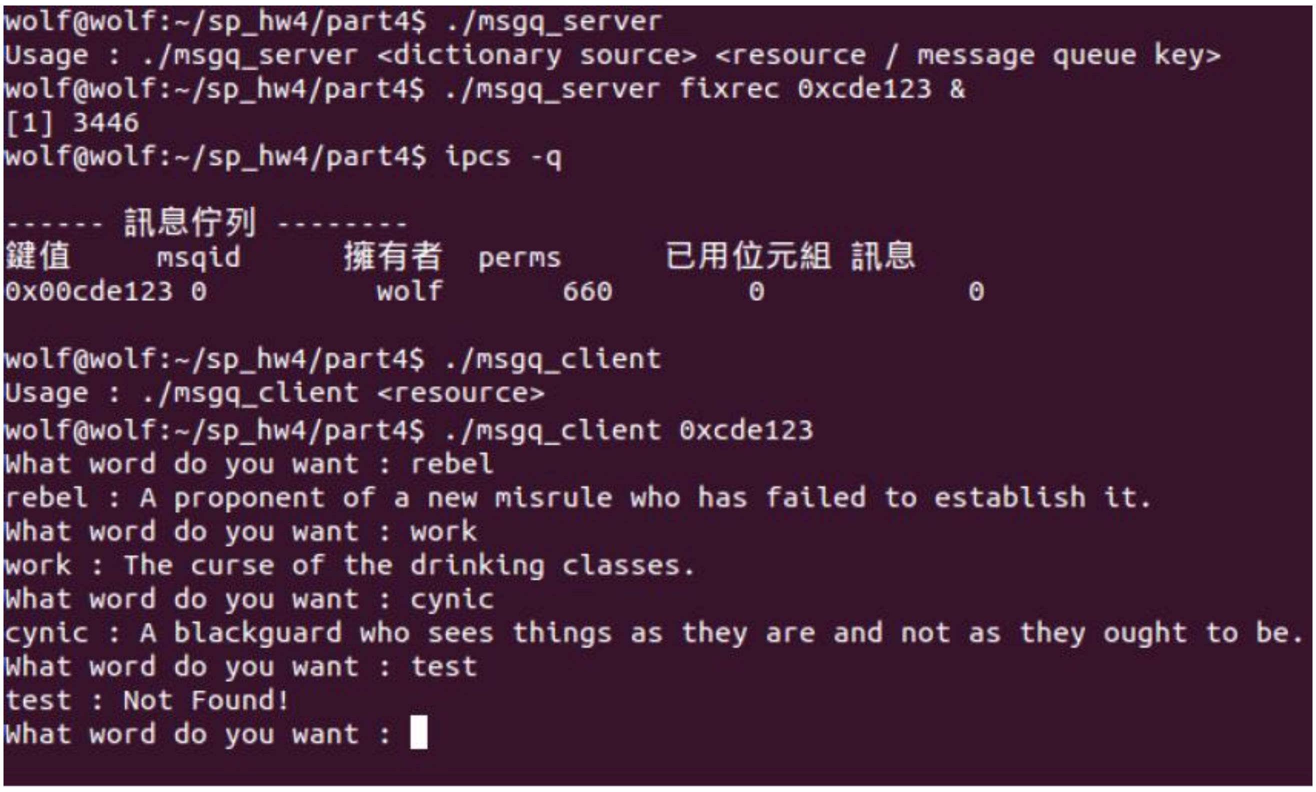
1. Develop a client and server that communicate through a message queue and look up words in the dictionary. The following steps provide some information about the client and Server you are going to develop.
   I. The server opens up a message queue on which it will listen for requests. Type 1 messages are designated to be requests; the server listens only for these. When a request comes in, the server does the dictionary look up and sends the reply. Reply messages are keyed with a type equal in value to the PID of the sender.
   II. The client opens up the message queue, then prepares and sends a message containing the word to look up and the PID of the sender, to the server. It then waits for the reply.
   III. Messages from the client are of type ClientMessage. Messages from the server are of type ServerMessaqe.
   IV. The server is called msgq_server and is built from lookup2.c (the module which performs the dictionary took ups) and msgq_server.c (which interfaces with the client through the message queue).
   V. The client is called msgq_client and is built from main.c (the user interface module) and lookup4.c (which interfaces with the server through the message queue).

2. The following provides you with some help on how to use the files provided to develop the client and server for this task.

   I. Edit msgq_server.c to get type 1 messages, and send messages of type N, where N is the PID of the client. Use the look up routine in lookup2.c to do the searching. The client opens up the message queue, then prepares and sends a message containing the word to look up and the PID of the sender, to the server. It then waits for the reply.
   II. Edit lookup4.c to send messages down the queue. Link with main.c.
   III. After the files have been edited, type make, or make msgq_server and make msgq_client.
   IV. When you get the prompt, run the msgq server and msgq_client as shown in the sample output.

3. File provided:
   dict.h, main.c, lookup4.c, msgq_server.c, fixrec

4. The result is shown as follows：



```
wolf@wolf:~/sp_hw4/part4$ ./msgq_server
Usage : ./msgq_server <dictionary source> <resource / message queue key>
wolf@wolf:~/sp_hw4/part4$ ./msgq_server fixrec 0xcde123 &
[1] 3446
wolf@wolf:~/sp_hw4/part4$ ipcs -q

------ 訊息佇列 --------
鍵值     msqid      擁有者  perms      已用位元組 訊息
0x00cde123 0          wolf       660        0            0

wolf@wolf:~/sp_hw4/part4$ ./msgq_client
Usage : ./msgq_client <resource>
wolf@wolf:~/sp_hw4/part4$ ./msgq_client 0xcde123
What word do you want : rebel
rebel : A proponent of a new misrule who has failed to establish it.
What word do you want : work
work : The curse of the drinking classes.
What word do you want : cynic
cynic : A blackguard who sees things as they are and not as they ought to be.
What word do you want : test
test : Not Found!
What word do you want : 
```

## ★ Upload:

1. Please compress your homework into **zip** or **tar** archive.
2. Naming rules: “**StudentID_SP_HW4.zip**”.
   For example: M043040001_SP_HW4.zip
3. Upload your homework to National Sun Yat-sen Cyber University.
4. Attention ! Part1, 2 & Part3, 4 have different deadlines.
   Deadline of Part1 & Part2 : 2016/11/13(Mon.) 23:59
   Deadline of Part3 & Part4 : 2016/11/20(Mon.) 23:59

## ★ Rules:

1. Please use C language in this homework and run your program on Ubuntu 16.04.

2. Please provide Makefile to compile your homework; otherwise, you will get ZERO.

3. **Do not copy homework of others.** If it happened, you will get **ZERO** whether you are either the owner of the homework or the copycat.

4. You have to deeply understand what your program do because TA will ask you something about your program during the demo.

5. If you have any question, please send email to sp_ta@net.nsysu.edu.tw or come to EC5018, but TA does not help to debug.

6. If you do not submit your assignment on time, you will not hand in the delayed homework and get ZERO as well. If you have trouble, please advise in advance by email. Moreover, time and place for demo will be announced later.