Operating Systems EE5012 - Laboratory

Graham Claffey

18296661

14/Feburary/2019

SDip in Embedded Systems Engineering

Laboratory Assignment #4

Assignment objectives

1. Learn how to use named pipes for IPC (inter-process communication)
2. Learn how to wrtie shell script program to manage concurrent processes
3. Learn how to write a simple Bash shell function to trap signals

Description of solution

The solution for proc_A.sh was to wrtie some sudo code as to what I wanted to achieve. This made it easier to understand what I was to expect at each point in the code. The first thing was to check if there was a pipe named pipe1. If this was not found, we created it in proc_A.sh.

Then I sent a message from proc_A.sh into the pipe1 pipe 5 times, when 5 messages were sent the script sent a final message to indicated to proc_B.sh that we are finished sending data.

The next step was to read data from pipe2. First thing I done was to check if there was a pipe called pipe2, if not we created here just like pipe1. (This part was uneccessary, becaue pipe2 would be created by proc_B.sh anyway.)

After the pipe was created the script would just sit there and wait to read something on pipe2, when the data finally comes it pints it to the screen(stdout). The data was the PID for proc_B.sh, so when we recieve this data we used it to send a SIGHUP and the PID of proc_B.

Finally remove pipe1 and exit.

The solution for proc_B.sh was similar. The first thing was create a loop that ran until it recieved a specific string from pipe1 ("Last Message!"). Within this loop we are checking if pipe1 is created, and if it is not created we would create it here. (I done this because if I started proc_B.sh before proc_A.sh I would get message printing "pipe1 does not exist" and it bothered me, so have this little if statement here to keep me sane).

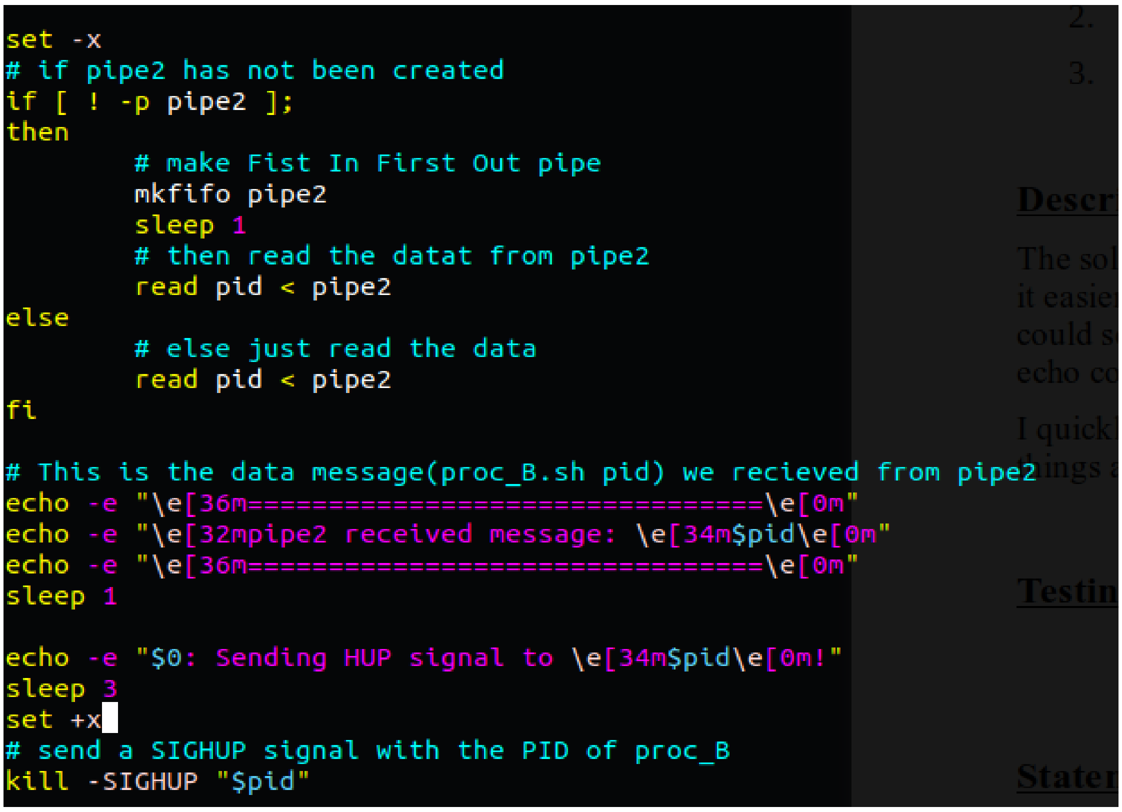
So we just sit in the loop reading from pipe1 until the string matches the condition and we then move on. Next was to create a pipe called pipe2 so we could use it to send out PID.

After we input our PID into pipe2, this script looped forever.

Whilst in the loop, proc_A.sh will send the SIGHUP at some point and the trap_functoion will kick in, which renices' our proc_B.sh script, removes pipe2 and exits.

Testing and results

When working with pipes I could see how tricky they are to debug when things do go according to plan. At first I was using echo commdand to test exactly where my code was haulting.
I quickly moved up to the set -x and set +x functions to test blocks of the bash script, which made things a bit eaiser. To activate/turn on this feature, you must put the set -x command before the block of code you wish to test and then use the set +x to signal the end of the block. This method really helped me out when I was working on the proc_B.sh script.

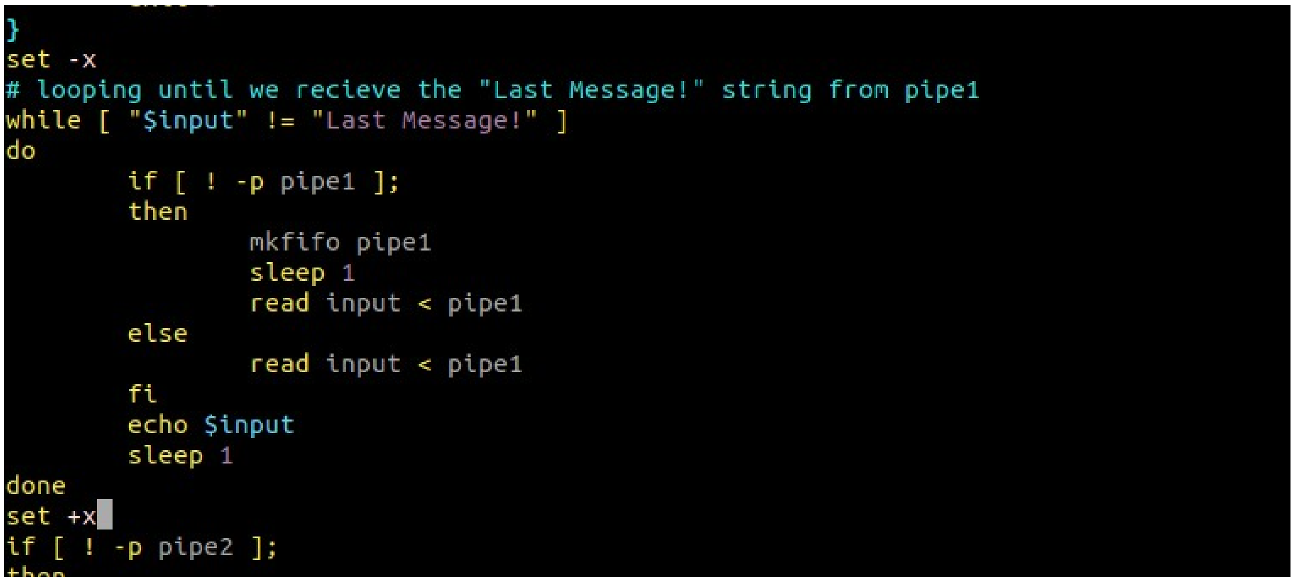


```
set -x
# if pipe2 has not been created
if [ ! -p pipe2 ];
then
        # make Fist In First Out pipe
        mkfifo pipe2
        sleep 1
        # then read the datat from pipe2
        read pid < pipe2
else
        # else just read the data
        read pid < pipe2
fi

# This is the data message(proc_B.sh pid) we recieved from pipe2
echo -e "\e[36m=================================\e[0m"
echo -e "\e[32mpipe2 received message: \e[34m$pid\e[0m"
echo -e "\e[36m=================================\e[0m"
sleep 1

echo -e "$0: Sending HUP signal to \e[34m$pid\e[0m!"
sleep 3
set +x
# send a SIGHUP signal with the PID of proc_B
kill -SIGHUP "$pid"
```

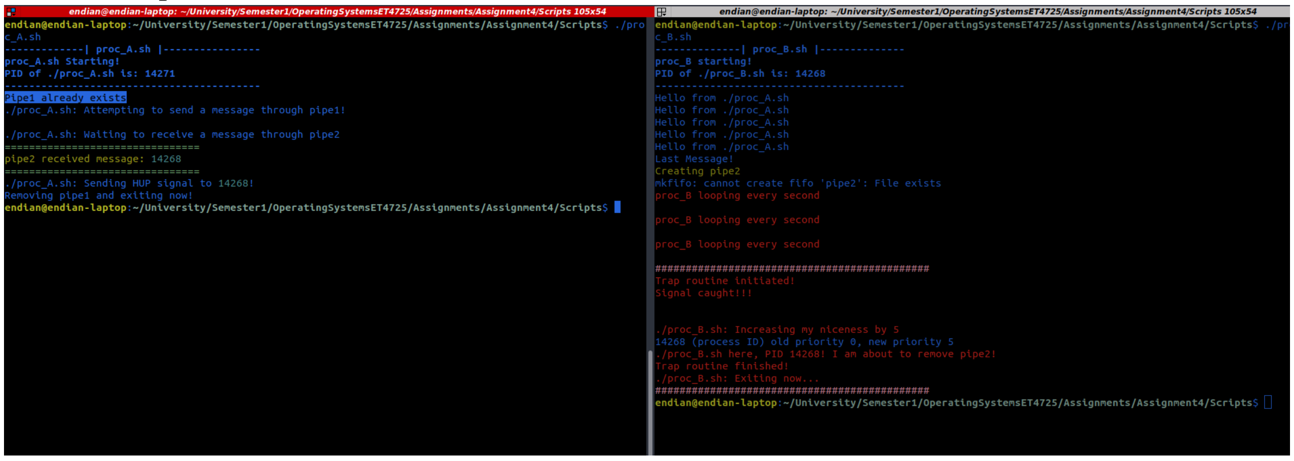
Moving over to proc_B.sh, I had some issues with sending the PID of proc_B.sh (this is what I thought for quite some time). I couldn't understand why I could not read the data from proc_A.sh.



```
}
set -x
# looping until we recieve the "Last Message!" string from pipe1
while [ "$input" != "Last Message!" ]
do
        if [ ! -p pipe1 ];
        then
                mkfifo pipe1
                sleep 1
                read input < pipe1
        else
                read input < pipe1
        fi
        echo $input
        sleep 1
done
set +x
if [ ! -p pipe2 ];
```

Turns out in the end I had a typo for the final message "Last Message!" that was sent from proc.sh, proc_A.sh was sending this string >> "Last Message!" and the while loop in proc_B.sh was checking for this string >> "Last message!".

This took me quite some time to find because I thought the script was getting further than it actually had. After a few more minor problems (for some reason I had an intermittent problem with sending the PID to proc_A.sh, at this current point in time I can't remember how I resolved this, I think it was piece of testing code messing with my results) but I managed to figure these problems out and now the scripts work as intended.



```
endian@endian-laptop: ~/University/Semester1/OperatingSystemsET4725/Assignments/Assignment4/Scripts 105x54
endian@endian-laptop:~/University/Semester1/OperatingSystemsET4725/Assignments/Assignment4/Scripts$ ./pro
c_A.sh
-------------| proc_A.sh |----------------
proc_A.sh Starting!
PID of ./proc_A.sh is: 14271
------------------------------------------
Pipe1 already exists
./proc_A.sh: Attempting to send a message through pipe1!

./proc_A.sh: Waiting to receive a message through pipe2
================================
pipe2 received message: 14268
================================
./proc_A.sh: Sending HUP signal to 14268!
Removing pipe1 and exiting now!
endian@endian-laptop:~/University/Semester1/OperatingSystemsET4725/Assignments/Assignment4/Scripts$
```

```
endian@endian-laptop: ~/University/Semester1/OperatingSystemsET4725/Assignments/Assignment4/Scripts 105x54
endian@endian-laptop:~/University/Semester1/OperatingSystemsET4725/Assignments/Assignment4/Scripts$ ./pr
c_B.sh
-------------| proc_B.sh |-------------
proc_B starting!
PID of ./proc_B.sh is: 14268
----------------------------------------
Hello from ./proc_A.sh
Hello from ./proc_A.sh
Hello from ./proc_A.sh
Hello from ./proc_A.sh
Hello from ./proc_A.sh
Last Message!
Creating pipe2
mkfifo: cannot create fifo 'pipe2': File exists
proc_B looping every second

proc_B looping every second

proc_B looping every second

#############################################
Trap routine initiated!
Signal caught!!!


./proc_B.sh: Increasing my niceness by 5
14268 (process ID) old priority 0, new priority 5
./proc_B.sh here, PID 14268! I am about to remove pipe2!
Trap routine finished!
./proc_B.sh: Exiting now...
#############################################
endian@endian-laptop:~/University/Semester1/OperatingSystemsET4725/Assignments/Assignment4/Scripts$
```

<u>Statement of completion</u>

With this assignment, I learned a lot about process inter-operability and inter-communcation. Working with the named pipes it's easy to see how complicated communication between processes can get. Another important thing i've learned was when have multiple concurrent processes running it's important to setup the oder of how you want them to run. They could start to missbehave depending on scheduling or priorities if you don't accomadate for this in your code.

Between reading, writing the code and writing this report, I think this assignment took about 10-13 hours.

Source code

```
proc_A.sh
#! /bin/bash

# Title:	proc_A.sh
# Description:  This script will demonstrate sending data between proc_A.sh and proc_B.sh
#		and in proc_A.sh we use the PID that we recieve from proc_B to send a SIGHUP
#		signal.
# Author:	Graham Claffey
# ID:		18296661


# Two proccesses called proc_A and proc_B communicate via a single named pipe1
# proc_A starts first: sends a message on pipe1 5 times. Then it sends the "Last message"
# to indicated the communication from proc_A is completed.
# proc_B starts in a separate terminal: when proc_B receives the message "Last message" on pipe1
# It then creates a pipe named pipe2 and sends its PID over pipe2.
# Then it prints "proc_B looping" every second.
# The proc_A will then read pipe2 and write the message that was received to the screen stdout
# proc_A will send a "HUP" signal to proc_B using the PID that was received over pipe2.
# It will then remove pipe1 and exit.
# The proc_B process will use a trap function to intercept the HUP signal and it will also lower it's priority by 5.
# The proc_B will remove pipe2 and then exit.



echo -e "\e[1m-------------| proc_A.sh |----------------\e[0m"
echo -e "\e[1mproc_A.sh Starting!\e[0m"
echo -e "\e[1mPID of $0 is: $$\e[0m"
echo -e "\e[1m------------------------------------------\e[0m"

# if pipe1 does not exist
if [ ! -p pipe1 ];
then
	# create pipe1
```

```
        echo -e "\e[32mCreating pipe1\e[0m"
        mkfifo pipe1
else
        # echo this message if pipe1 exists
        echo -e "\e[7mPipe1 already exists\e[0m"
fi



echo "$0: Attempting to send a message through pipe1!"
# send message 5 times
for((x=0; x<5; x++));
do
        # echo this message into pipe1
        echo "Hello from $0" > pipe1
        sleep 1
done

# echo the final message into pipe1
echo "Last Message!" > pipe1
sleep 1

echo -e "\n$0: Waiting to receive a message through pipe2"

# if pipe2 has not been created
if [ ! -p pipe2 ];
then
        # make Fist In First Out pipe
        mkfifo pipe2
        sleep 1
        # then read the datat from pipe2
        read pid < pipe2
else
        # else just read the data
        read pid < pipe2
fi
```

```
# This is the data message(proc_B.sh pid) we recieved from pipe2
echo -e "\e[36m==================================\e[0m"
echo -e "\e[32mpipe2 received message: \e[34m$pid\e[0m"
echo -e "\e[36m==================================\e[0m"
sleep 1

echo -e "$0: Sending HUP signal to \e[34m$pid\e[0m!"
sleep 3
# send a SIGHUP signal with the PID of proc_B
kill -SIGHUP "$pid"

# remove pipe1
rm pipe1
echo "Removing pipe1 and exiting now!"
# wait for the child process to finish
wait
exit 0
```

proc_B.sh

```
#! /bin/bash

# Title:       proc_B.sh
# Description: This script will demonstrate sending data between proc_B.sh and proc_B.sh
#              and in proc_B.sh we recieve some data from proc_A.sh via pipe1
#              then we send the PID of this process to proc_A.sh and loop until we recieve
#              a SIGHUP signal. Using the trap function we can hangup out program
# Author:      Graham Claffey
# ID:          18296661

echo -e "\e[1m--------------| proc_B.sh |--------------\e[0m"
echo -e "\e[1mproc_B starting!\e[0m"
echo -e "\e[1mPID of $0 is: $$\e[0m"
echo -e "\e[1m----------------------------------------\e[0m"

# setting up the trap_function to trigger when we recieve the hangup signal
```

```
trap 'trap_function' SIGHUP

trap_function()
{
	# Print some information and increase the niceness of this process
	# so we can hangup then exit the process
	echo -e "\e[95m#################################################\e[0m"
	echo -e "\e[31mTrap routine initiated!\e[0m"
	echo -e "\e[31mSignal caught!!!\e[0m\n\n"

	echo -e "\e[31m$0: Increasing my niceness by 5\e[0m"
	renice 5 $$
	echo -e "\e[31m$0 here, PID $$! I am about to remove pipe2!\e[0m"
	rm pipe2
	echo -e "\e[31mTrap routine finished!\e[0m"
	echo -e "\e[31m$0: Exiting now...\e[0m"
	echo -e "\e[95m#################################################\e[0m"
	exit 0
}

# looping until we recieve the "Last Message!" string from pipe1
while [ "$input" != "Last Message!" ]
do
	if [ ! -p pipe1 ];
	then
		mkfifo pipe1
		sleep 1
		read input < pipe1
	else
		read input < pipe1
	fi
	echo $input
	sleep 1
done
```

```
if [ ! -p pipe2 ];
then
	echo -e "\e[32mCreating pipe2\e[0m"
	mkfifo pipe2
else
	echo -e "\e[7mPipe2 already exists\e[0m"
fi

# put this processes PID into the PID variable
PID=$$
sleep 1

# send the PID variable to pipe2
echo "$PID" > pipe2
# if you don't want to make a variable to send the PID
# echo "$$" > pipe2
# this works the same way
sleep 1

# loop forever printing every second
while true
do
	echo -e "\e[31mproc_B looping every second\n\e[0m"
	sleep 1
done

wait
```