# HW4 REPORT

*To send an entire folder to BBG to root folder, where home is name of folder.

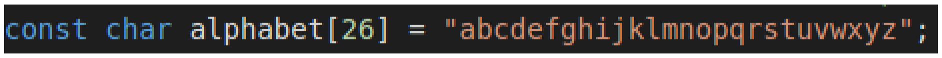
Command: sudo scp -r home root@10.0.0.16:/root

## Q.1. pThread Practice on BBG [link]

This problem requires us to implement pthreads on BBG. The main thread or the main, spawns 2 child threads child1 and child2. Th child1 sorts the gdb.txt to print out the alphabets that has occurrence less than 100 in the entire text. The child 2 run in a continuous loop and prints the cpu utilization every 100 milliseconds. Both the threads can be exited by giving SIGUSR1 or SIGUSR2 or SIGINT(CTRL-C). There is no concept of priority of threads as of now.

### 1) A brief description of the search/map processing algorithm and data structures employed in Child Thread 1 to track occurrences.

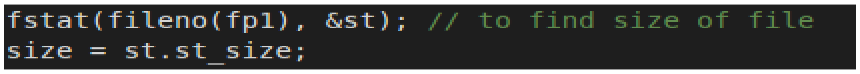
- ➔ The thread 1 in my case is called "CHILD1". The data structures used to implement sorting is **arrays** which has been used to emulate **map.** Initially I declare a const array of all alphabetical characters.



```
const char alphabet[26] = "abcdefghijklmnopqrstuvwxyz";
```

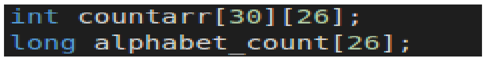
These will be my '**key**' against which the number of occurrences will be my '**value**' as in hash maps.

- ➔ Then I proceed to find the size of the entire gdb.txt



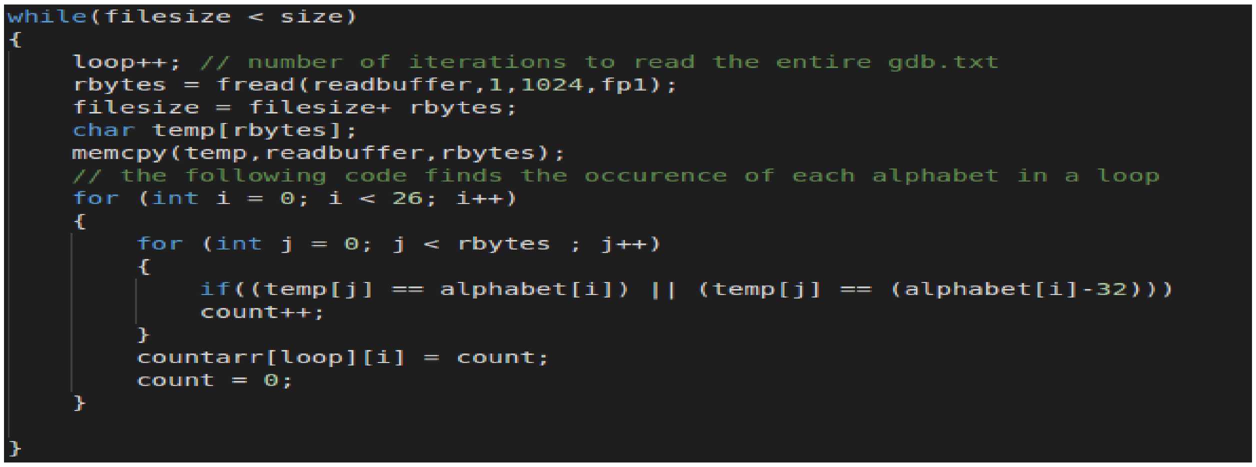
```
fstat(fileno(fp1), &st); // to find size of file
size = st.st_size;
```

- ➔ I process the file 1 KB at a time. And I keep count of the number of loops to cover the entire file. Each time it processes 1 KB , I find the occurrences of all the alphabets in the 1 KB. And store in in 2D array as below:



```
int countarr[30][26];
long alphabet_count[26];
```

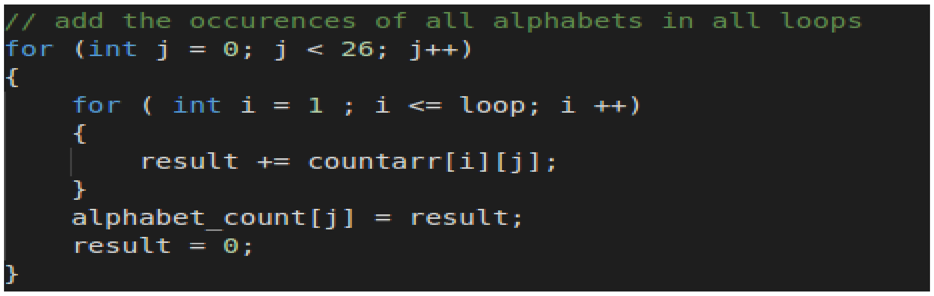
In total to cover the entire file in steps of 1 KB each it requires me 30 loops. The 2nd dimension of the 2D array countarr is the total number of alphabets. In each loop I compare the characters in the file with the characters in my const char alphabet array. If match found , I increment the count for that index of alphabet in that loop. The main sorting process is as shown:



```
while(filesize < size)
{
    loop++; // number of iterations to read the entire gdb.txt
    rbytes = fread(readbuffer,1,1024,fp1);
    filesize = filesize+ rbytes;
    char temp[rbytes];
    memcpy(temp,readbuffer,rbytes);
    // the following code finds the occurence of each alphabet in a loop
    for (int i = 0; i < 26; i++)
    {
        for (int j = 0; j < rbytes ; j++)
        {
            if((temp[j] == alphabet[i]) || (temp[j] == (alphabet[i]-32)))
            count++;
        }
        countarr[loop][i] = count;
        count = 0;
    }

}
```

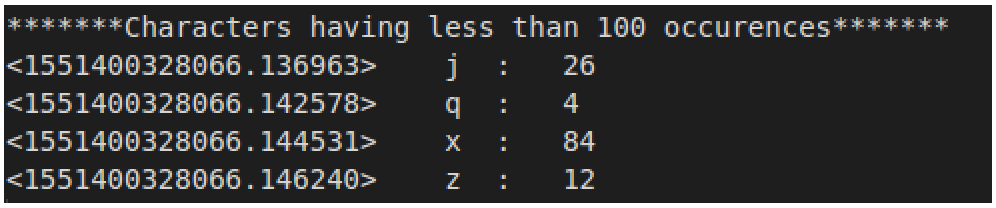
The sorting covers both upper and lower cases.

➔ Finally we come to the last step of sorting, we add all the occurrences of the alphabets based on their index in the countarr 2D array and store them in the array called alphabet count.



```
// add the occurences of all alphabets in all loops
for (int j = 0; j < 26; j++)
{
    for ( int i = 1 ; i <= loop; i ++)
    {
        result += countarr[i][j];
    }
    alphabet_count[j] = result;
    result = 0;
}
```

➔ In the file we display only the characters that have occurrences less than 100.

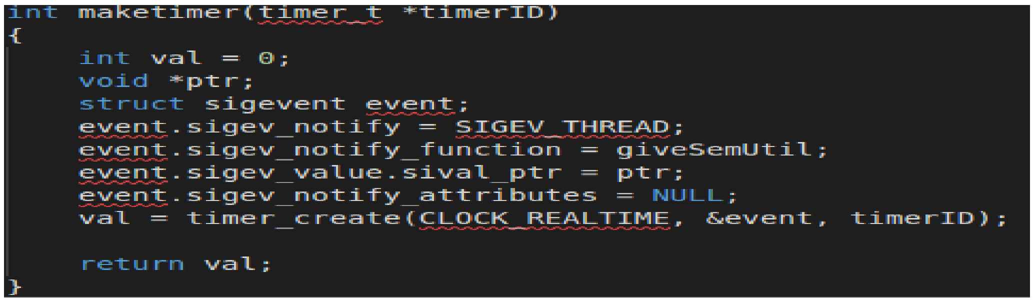


```
*******Characters having less than 100 occurences*******
<1551400328066.136963>    j  :   26
<1551400328066.142578>    q  :   4
<1551400328066.144531>    x  :   84
<1551400328066.146240>    z  :   12
```

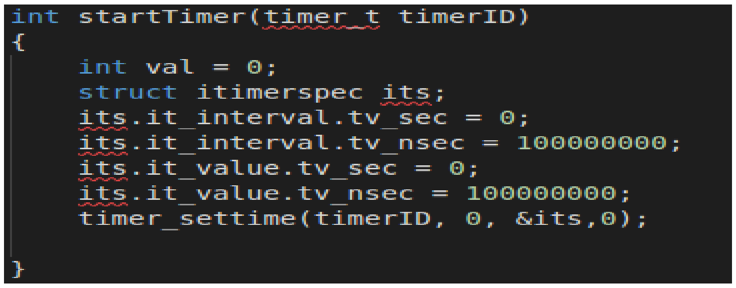
➔ Pitfalls of this approach is that, since I knew ahead of time, the file to process, the number of characters in alphabets, it was relatively easier for me to implement this sorting using arrays. Since the sorting of the array is limited to only child1 and is not shared among other threads, this approach worked for me. Linked List are more efficient in multi threaded implementations as changes tend to be local - affecting only a pointer or two for insert and remove at a localized part of the data structure. So, I can have many threads working on the same linked list. In a situation where a process is continuously writing in gdb.txt and another process is continuously sorting, linked list will be used in this case. With an array, any change that modifies the size of the array is likely to require locking a large portion of the array, and that is rarely done.

➔ The above approach has a time complexity of **O(n^2)** for all cases. The space complexity will vary with the size of the file to process.

- For “CHILD2” which logs cpu utilization every 100 milliseconds, posix timer as been used. The two functions timer_create and timer_settime. These functions have been used in two of the functions in my program called maketimer and starttimer.

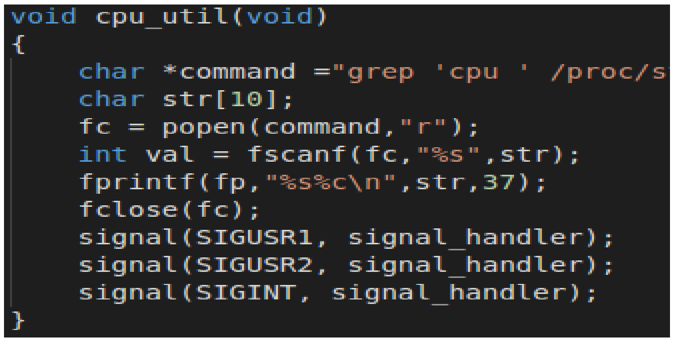


```
int maketimer(timer_t *timerID)
{
    int val = 0;
    void *ptr;
    struct sigevent event;
    event.sigev_notify = SIGEV_THREAD;
    event.sigev_notify_function = giveSemUtil;
    event.sigev_value.sival_ptr = ptr;
    event.sigev_notify_attributes = NULL;
    val = timer_create(CLOCK_REALTIME, &event, timerID);

    return val;
}
```



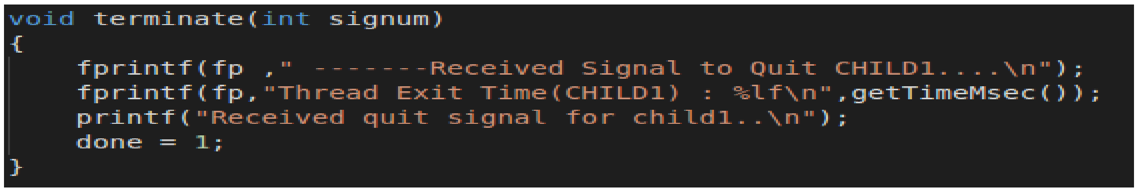
```
int startTimer(timer_t timerID)
{
    int val = 0;
    struct itimerspec its;
    its.it_interval.tv_sec = 0;
    its.it_interval.tv_nsec = 100000000;
    its.it_value.tv_sec = 0;
    its.it_value.tv_nsec = 100000000;
    timer_settime(timerID, 0, &its,0);

}
```

- The call back function of the timer interrupt is giveSemUtil, which just posts the semaphore for the child2 to start printing the cpu utilization on the given log file. The cpu utilization function is as below:



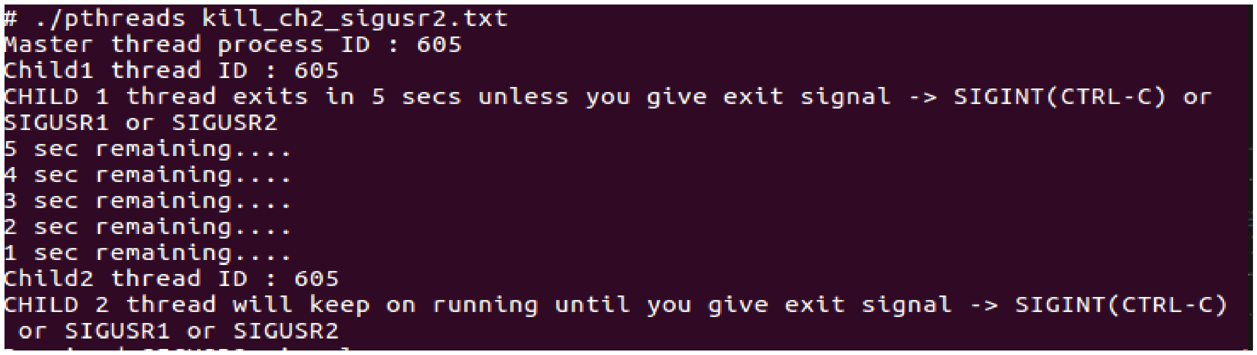
```
void cpu_util(void)
{
    char *command ="grep 'cpu ' /proc/s
    char str[10];
    fc = popen(command,"r");
    int val = fscanf(fc,"%s",str);
    fprintf(fp,"%s%c\n",str,37);
    fclose(fc);
    signal(SIGUSR1, signal_handler);
    signal(SIGUSR2, signal_handler);
    signal(SIGINT, signal_handler);
}
```

- In this program I have used signal function to terminate child2 and sigaction to terminate child1.



```
void terminate(int signum)
{
    fprintf(fp ," -------Received Signal to Quit CHILD1....\n");
    fprintf(fp,"Thread Exit Time(CHILD1) : %lf\n",getTimeMsec());
    printf("Received quit signal for child1..\n");
    done = 1;
}
```

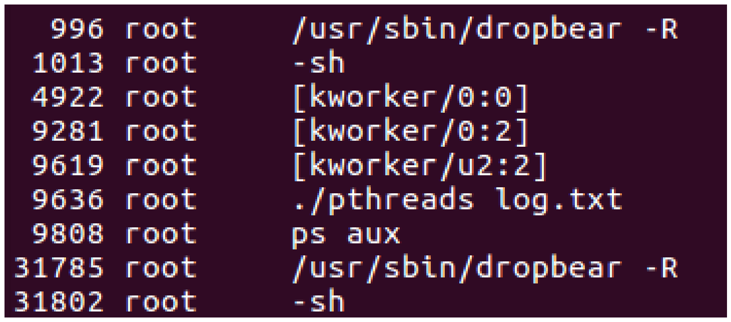
- After the child1 has completed its sorting, it waits for a period of 5 secs before terminating. On termination it gives sem_post to child2 thread.



```
# ./pthreads kill_ch2_sigusr2.txt
Master thread process ID : 605
Child1 thread ID : 605
CHILD 1 thread exits in 5 secs unless you give exit signal -> SIGINT(CTRL-C) or
SIGUSR1 or SIGUSR2
5 sec remaining....
4 sec remaining....
3 sec remaining....
2 sec remaining....
1 sec remaining....
Child2 thread ID : 605
CHILD 2 thread will keep on running until you give exit signal -> SIGINT(CTRL-C)
 or SIGUSR1 or SIGUSR2
```

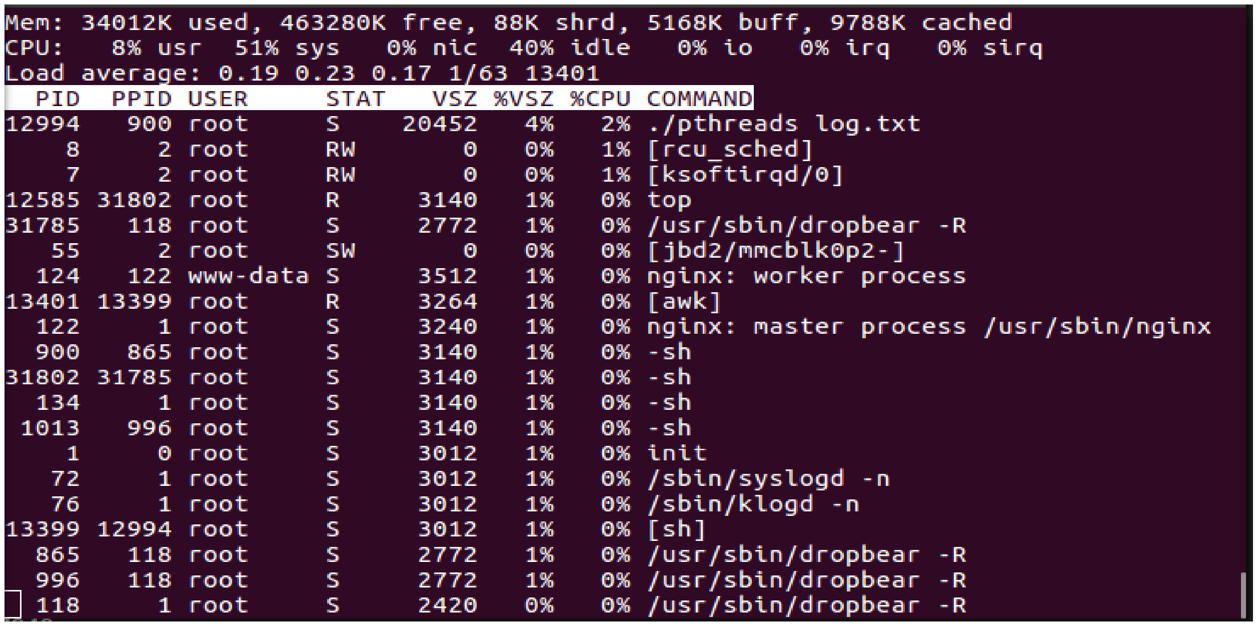
## 2) A screenshot capture of the ps command showing your threads running.

Ps on BBG showing the pthread process running.



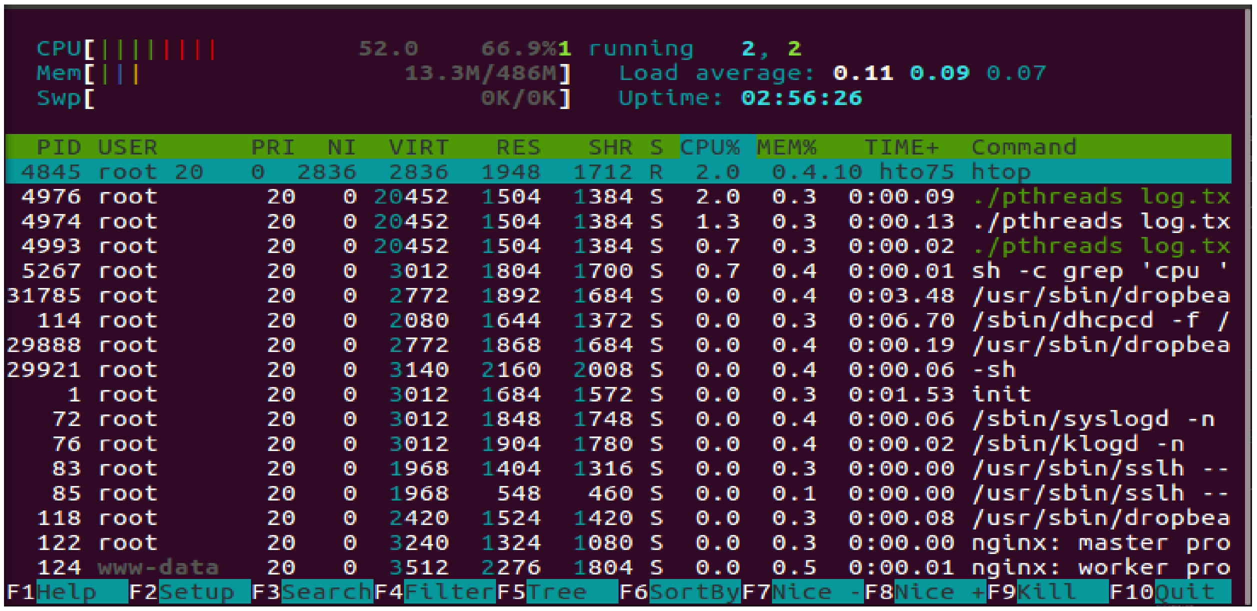
```
  996 root     /usr/sbin/dropbear -R
 1013 root     -sh
 4922 root     [kworker/0:0]
 9281 root     [kworker/0:2]
 9619 root     [kworker/u2:2]
 9636 root     ./pthreads log.txt
 9808 root     ps aux
31785 root     /usr/sbin/dropbear -R
31802 root     -sh
```

Top on BBG showing pthread process running



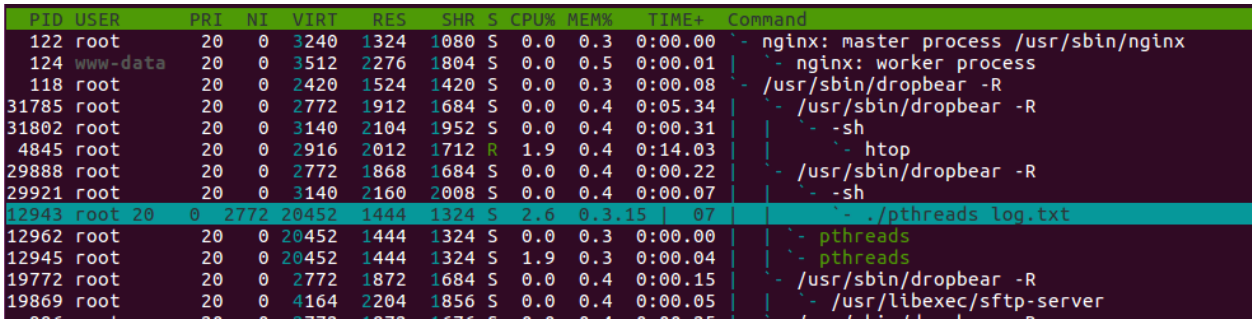
```
Mem: 34012K used, 463280K free, 88K shrd, 5168K buff, 9788K cached
CPU:   8% usr  51% sys   0% nic  40% idle   0% io   0% irq   0% sirq
Load average: 0.19 0.23 0.17 1/63 13401
  PID  PPID USER     STAT   VSZ %VSZ %CPU COMMAND
12994   900 root     S    20452   4%   2% ./pthreads log.txt
    8     2 root     RW       0   0%   1% [rcu_sched]
    7     2 root     RW       0   0%   1% [ksoftirqd/0]
12585 31802 root     R     3140   1%   0% top
31785   118 root     S     2772   1%   0% /usr/sbin/dropbear -R
   55     2 root     SW       0   0%   0% [jbd2/mmcblk0p2-]
  124   122 www-data S     3512   1%   0% nginx: worker process
13401 13399 root     R     3264   1%   0% [awk]
  122     1 root     S     3240   1%   0% nginx: master process /usr/sbin/nginx
  900   865 root     S     3140   1%   0% -sh
31802 31785 root     S     3140   1%   0% -sh
  134     1 root     S     3140   1%   0% -sh
 1013   996 root     S     3140   1%   0% -sh
    1     0 root     S     3012   1%   0% init
   72     1 root     S     3012   1%   0% /sbin/syslogd -n
   76     1 root     S     3012   1%   0% /sbin/klogd -n
13399 12994 root     S     3012   1%   0% [sh]
  865   118 root     S     2772   1%   0% /usr/sbin/dropbear -R
  996   118 root     S     2772   1%   0% /usr/sbin/dropbear -R
  118     1 root     S     2420   0%   0% /usr/sbin/dropbear -R
```

Htop showing instances of threads



```
  CPU[||||||||          52.0     66.9%1 running    2, 2
  Mem[|||                  13.3M/486M]    Load average: 0.11 0.09 0.07
  Swp[                         0K/0K]    Uptime: 02:56:26

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 4845 root 20  0  2836  2836  1948  1712 R  2.0  0.4.10 hto75 htop
 4976 root       20   0 20452  1504  1384 S  2.0  0.3  0:00.09 ./pthreads log.tx
 4974 root       20   0 20452  1504  1384 S  1.3  0.3  0:00.13 ./pthreads log.tx
 4993 root       20   0 20452  1504  1384 S  0.7  0.3  0:00.02 ./pthreads log.tx
 5267 root       20   0  3012  1804  1700 S  0.7  0.4  0:00.01 sh -c grep 'cpu '
31785 root       20   0  2772  1892  1684 S  0.0  0.4  0:03.48 /usr/sbin/dropbea
  114 root       20   0  2080  1644  1372 S  0.0  0.3  0:06.70 /sbin/dhcpcd -f /
29888 root       20   0  2772  1868  1684 S  0.0  0.4  0:00.19 /usr/sbin/dropbea
29921 root       20   0  3140  2160  2008 S  0.0  0.4  0:00.06 -sh
    1 root       20   0  3012  1684  1572 S  0.0  0.3  0:01.53 init
   72 root       20   0  3012  1848  1748 S  0.0  0.4  0:00.06 /sbin/syslogd -n
   76 root       20   0  3012  1904  1780 S  0.0  0.4  0:00.02 /sbin/klogd -n
   83 root       20   0  1968  1404  1316 S  0.0  0.3  0:00.00 /usr/sbin/sslh --
   85 root       20   0  1968   548   460 S  0.0  0.1  0:00.00 /usr/sbin/sslh --
  118 root       20   0  2420  1524  1420 S  0.0  0.3  0:00.08 /usr/sbin/dropbea
  122 root       20   0  3240  1324  1080 S  0.0  0.3  0:00.00 nginx: master pro
  124 www-data   20   0  3512  2276  1804 S  0.0  0.5  0:00.01 nginx: worker pro
F1Help  F2Setup  F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

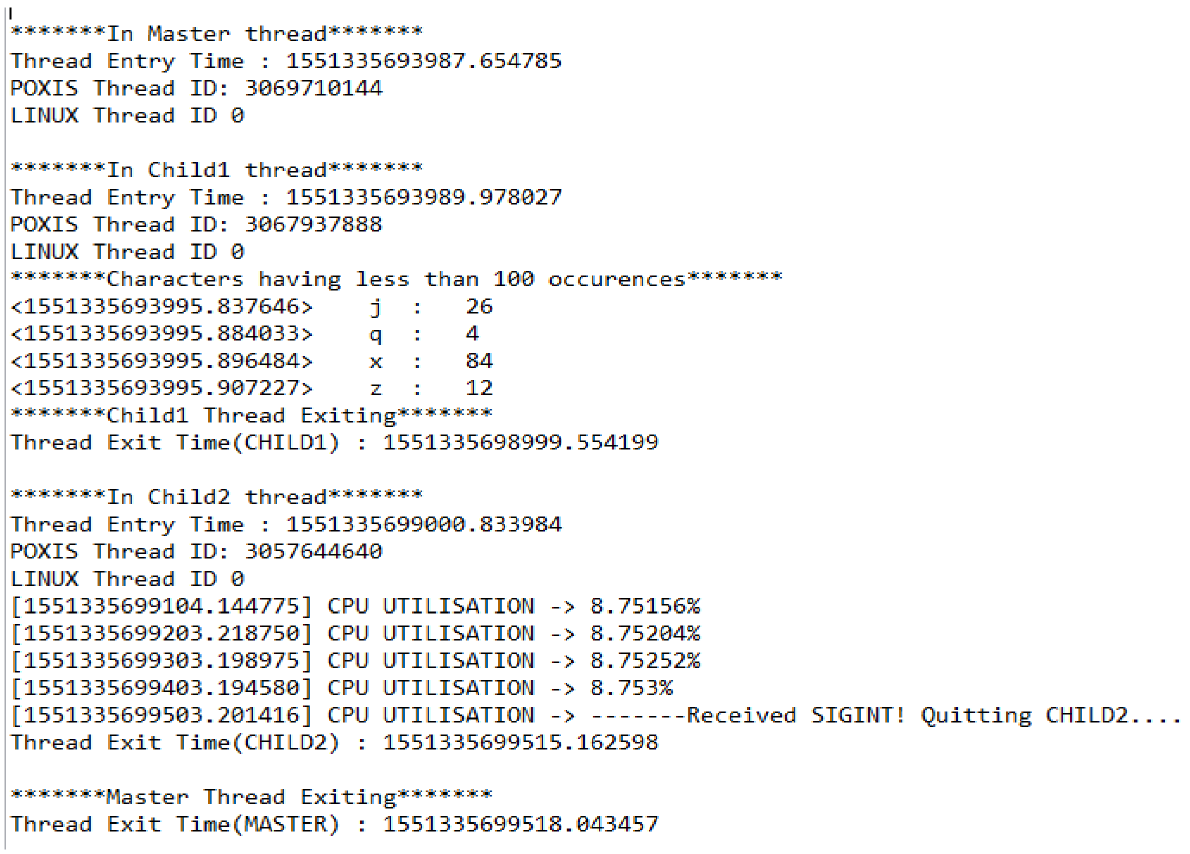
To enable thread views in htop, launch htop, and press <F2> to enter htop setup menu. Choose "Display option" under "Setup" column, and toggle on "Tree view" and "Show custom thread names" options. Presss <F10> to exit the setup.
http://ask.xmodulo.com/view-threads-process-linux.html



```
  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
  122 root       20   0  3240  1324  1080 S  0.0  0.3  0:00.00 `- nginx: master process /usr/sbin/nginx
  124 www-data   20   0  3512  2276  1804 S  0.0  0.5  0:00.01 |  `- nginx: worker process
  118 root       20   0  2420  1524  1420 S  0.0  0.3  0:00.08 `- /usr/sbin/dropbear -R
31785 root       20   0  2772  1912  1684 S  0.0  0.4  0:05.34 |  `- /usr/sbin/dropbear -R
31802 root       20   0  3140  2104  1952 S  0.0  0.4  0:00.31 |  |  `- -sh
 4845 root       20   0  2916  2012  1712 R  1.9  0.4  0:14.03 |  |     `- htop
29888 root       20   0  2772  1868  1684 S  0.0  0.4  0:00.22 |  `- /usr/sbin/dropbear -R
29921 root       20   0  3140  2160  2008 S  0.0  0.4  0:00.07 |  |  `- -sh
12943 root 20  0  2772 20452  1444  1324 S  2.6  0.3.15 |   07 |  |     `- ./pthreads log.txt
12962 root       20   0 20452  1444  1324 S  0.0  0.3  0:00.00 |  |  `- pthreads
12945 root       20   0 20452  1444  1324 S  1.9  0.3  0:00.04 |  |  `- pthreads
19772 root       20   0  2772  1872  1684 S  0.0  0.4  0:00.15 |  `- /usr/sbin/dropbear -R
19869 root       20   0  4164  2204  1856 S  0.0  0.4  0:00.05 |  |  `- /usr/libexec/sftp-server
```

## 3) Log file text(screenshot)

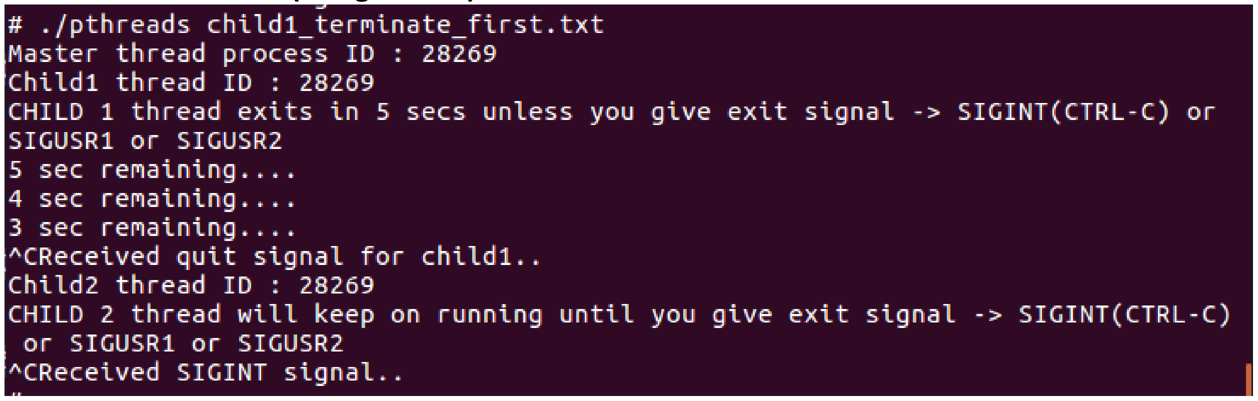
The original log file is attached to my repository in HW4->pthreads folder.
The log of pthreads terminated via SIGINT(CTRL-C)



```
*******In Master thread*******
Thread Entry Time : 1551335693987.654785
POXIS Thread ID: 3069710144
LINUX Thread ID 0

*******In Child1 thread*******
Thread Entry Time : 1551335693989.978027
POXIS Thread ID: 3067937888
LINUX Thread ID 0
*******Characters having less than 100 occurences*******
<1551335693995.837646>    j  :   26
<1551335693995.884033>    q  :   4
<1551335693995.896484>    x  :   84
<1551335693995.907227>    z  :   12
*******Child1 Thread Exiting*******
Thread Exit Time(CHILD1) : 1551335698999.554199

*******In Child2 thread*******
Thread Entry Time : 1551335699000.833984
POXIS Thread ID: 3057644640
LINUX Thread ID 0
[1551335699104.144775] CPU UTILISATION -> 8.75156%
[1551335699203.218750] CPU UTILISATION -> 8.75204%
[1551335699303.198975] CPU UTILISATION -> 8.75252%
[1551335699403.194580] CPU UTILISATION -> 8.753%
[1551335699503.201416] CPU UTILISATION -> -------Received SIGINT! Quitting CHILD2....
Thread Exit Time(CHILD2) : 1551335699515.162598

*******Master Thread Exiting*******
Thread Exit Time(MASTER) : 1551335699518.043457
```

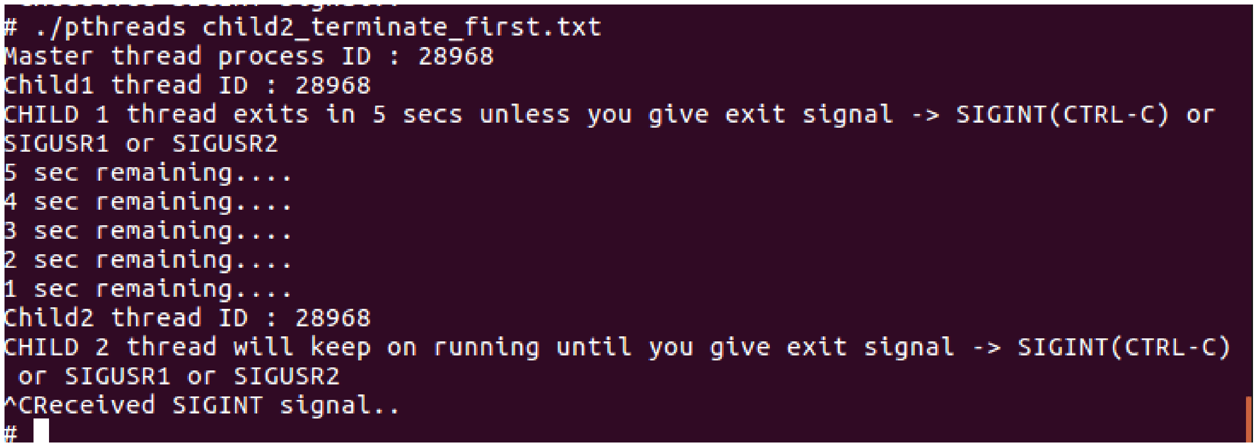
## 4) As a separate run/capture sequence, show the use of USR1 and/or USR2 signals to terminate the Child Threads.

- Both the threads can be terminated by any one of the 3 signals, SIGINT,SIGUSR1,SIGUSR2.
- To kill using SIGINT a simple ctrl-c from the keyboard is sufficient. Otherwise, to kill from another terminal, use the command “kill -<signal> <pid>
- **Termination of child1(using SIGINT)**



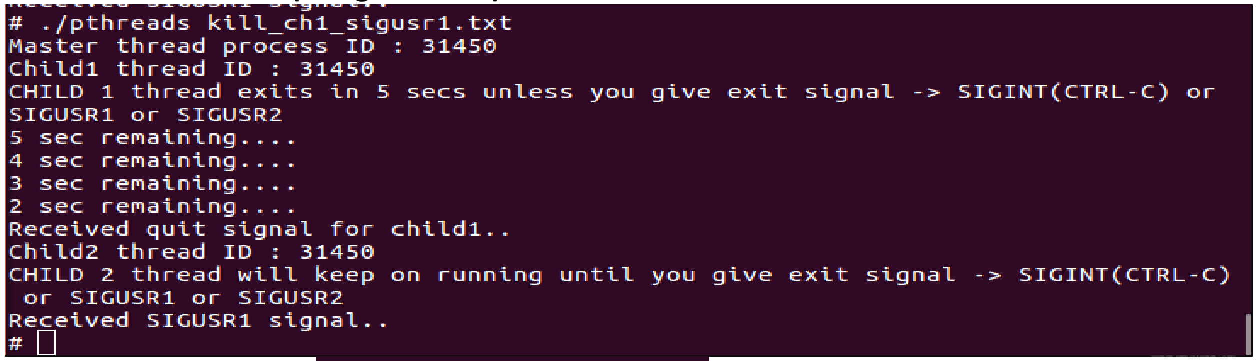
```
# ./pthreads child1_terminate_first.txt
Master thread process ID : 28269
Child1 thread ID : 28269
CHILD 1 thread exits in 5 secs unless you give exit signal -> SIGINT(CTRL-C) or
SIGUSR1 or SIGUSR2
5 sec remaining....
4 sec remaining....
3 sec remaining....
^CReceived quit signal for child1..
Child2 thread ID : 28269
CHILD 2 thread will keep on running until you give exit signal -> SIGINT(CTRL-C)
 or SIGUSR1 or SIGUSR2
^CReceived SIGINT signal..
```

➔ **Child 1 runs to completion, Child2 terminated using SIGINT**

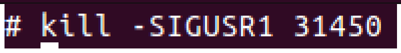


```
# ./pthreads child2_terminate_first.txt
Master thread process ID : 28968
Child1 thread ID : 28968
CHILD 1 thread exits in 5 secs unless you give exit signal -> SIGINT(CTRL-C) or
SIGUSR1 or SIGUSR2
5 sec remaining....
4 sec remaining....
3 sec remaining....
2 sec remaining....
1 sec remaining....
Child2 thread ID : 28968
CHILD 2 thread will keep on running until you give exit signal -> SIGINT(CTRL-C)
 or SIGUSR1 or SIGUSR2
^CReceived SIGINT signal..
#
```

➔ **Termination of child1(using SIGUSR1)**

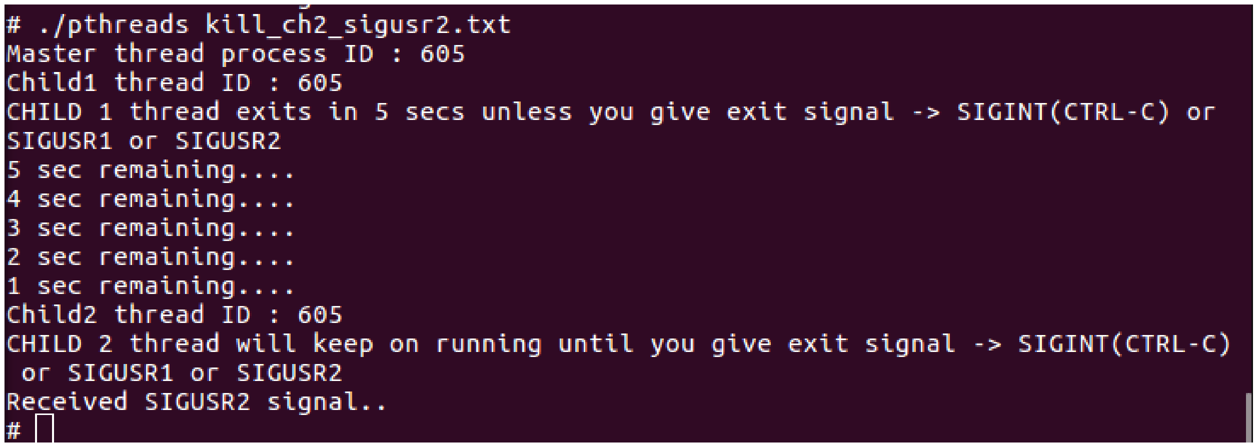


```
# ./pthreads kill_ch1_sigusr1.txt
Master thread process ID : 31450
Child1 thread ID : 31450
CHILD 1 thread exits in 5 secs unless you give exit signal -> SIGINT(CTRL-C) or
SIGUSR1 or SIGUSR2
5 sec remaining....
4 sec remaining....
3 sec remaining....
2 sec remaining....
Received quit signal for child1..
Child2 thread ID : 31450
CHILD 2 thread will keep on running until you give exit signal -> SIGINT(CTRL-C)
 or SIGUSR1 or SIGUSR2
Received SIGUSR1 signal..
#
```

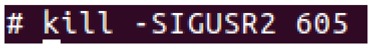
**Command used to kill :** 

```
# kill -SIGUSR1 31450
```

➔ **Termination of  child2(using SIGUSR2)**



```
# ./pthreads kill_ch2_sigusr2.txt
Master thread process ID : 605
Child1 thread ID : 605
CHILD 1 thread exits in 5 secs unless you give exit signal -> SIGINT(CTRL-C) or
SIGUSR1 or SIGUSR2
5 sec remaining....
4 sec remaining....
3 sec remaining....
2 sec remaining....
1 sec remaining....
Child2 thread ID : 605
CHILD 2 thread will keep on running until you give exit signal -> SIGINT(CTRL-C)
 or SIGUSR1 or SIGUSR2
Received SIGUSR2 signal..
#
```

**Command used to kill :** 

```
# kill -SIGUSR2 605
```

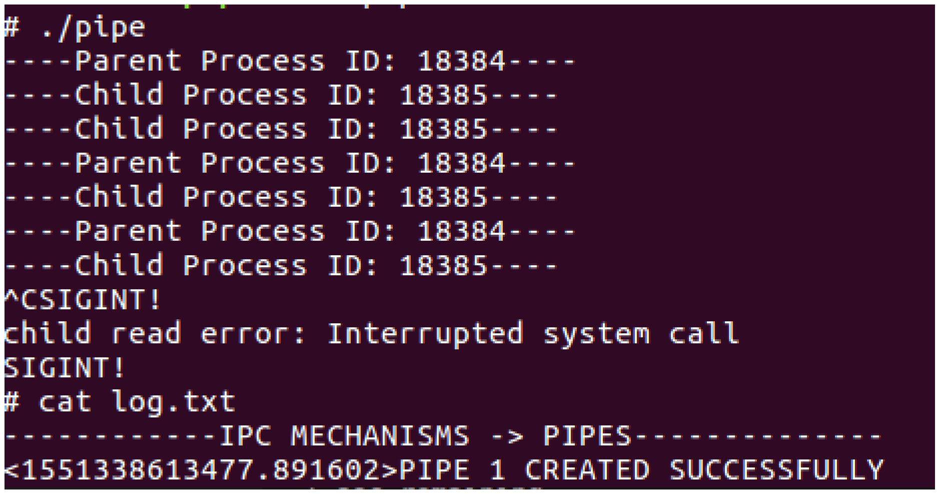
All the above log text files are given in my repo in HW4->pthreads folder.

## Q2. (IPC) Multiple Ways to communicate between two processes on BBG[link]

4 types of IPCs are implemented. They all send back and forth a structure of 10 messages including both commands string, timestamp and their respective pids.
The payload is the same for all the IPCs.

1. Pipes: (link)

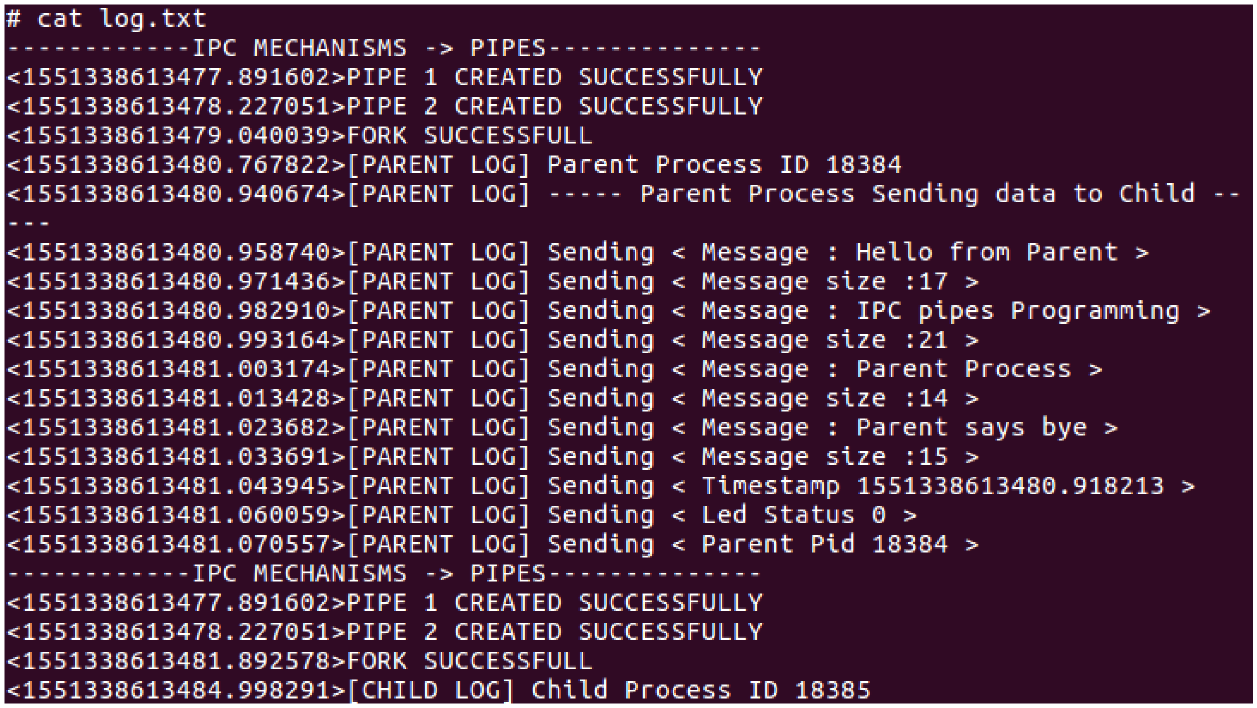
Proof of execution of pipes



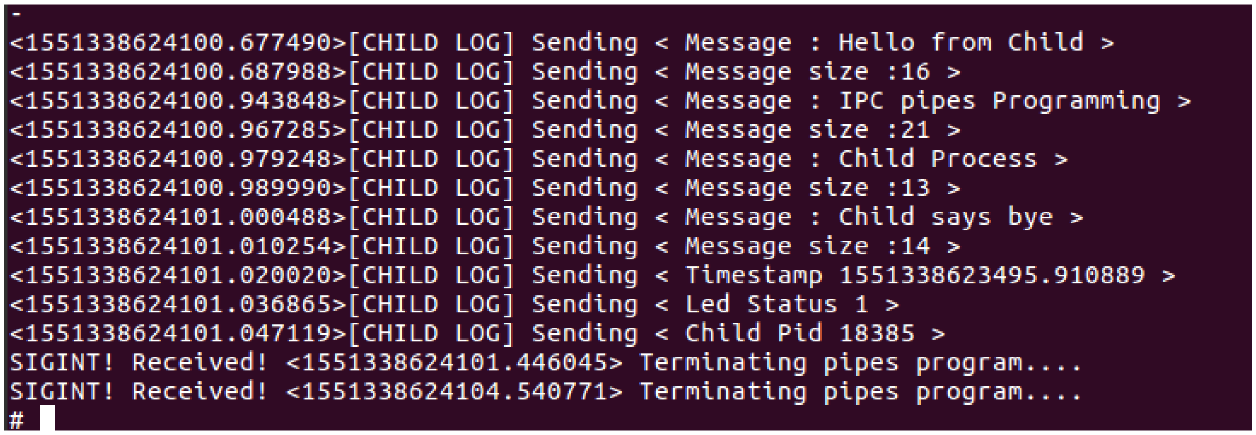
```
# ./pipe
----Parent Process ID: 18384----
----Child Process ID: 18385----
----Child Process ID: 18385----
----Parent Process ID: 18384----
----Child Process ID: 18385----
----Parent Process ID: 18384----
----Child Process ID: 18385----
^CSIGINT!
child read error: Interrupted system call
SIGINT!
# cat log.txt
------------IPC MECHANISMS -> PIPES-------------
<1551338613477.891602>PIPE 1 CREATED SUCCESSFULLY
```

The entire file log.txt is included in the pipes folder of IPC in HW4 in my repository

cat log.txt



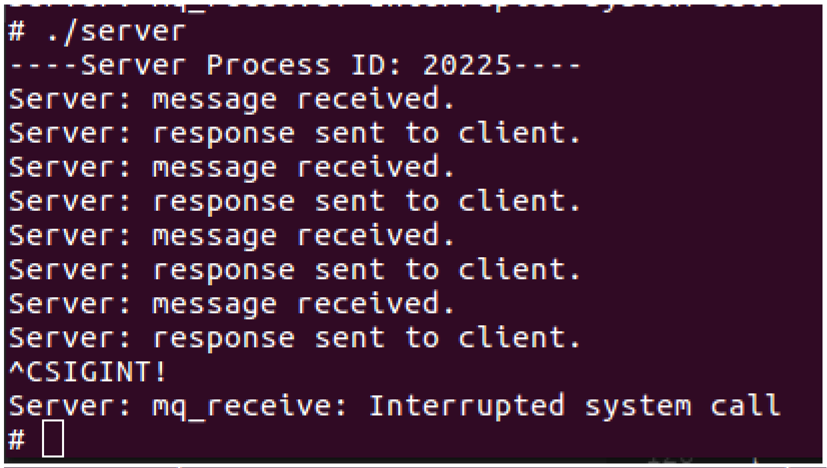
```
# cat log.txt
------------IPC MECHANISMS -> PIPES-------------
<1551338613477.891602>PIPE 1 CREATED SUCCESSFULLY
<1551338613478.227051>PIPE 2 CREATED SUCCESSFULLY
<1551338613479.040039>FORK SUCCESSFULL
<1551338613480.767822>[PARENT LOG] Parent Process ID 18384
<1551338613480.940674>[PARENT LOG] ----- Parent Process Sending data to Child --
---
<1551338613480.958740>[PARENT LOG] Sending < Message : Hello from Parent >
<1551338613480.971436>[PARENT LOG] Sending < Message size :17 >
<1551338613480.982910>[PARENT LOG] Sending < Message : IPC pipes Programming >
<1551338613480.993164>[PARENT LOG] Sending < Message size :21 >
<1551338613481.003174>[PARENT LOG] Sending < Message : Parent Process >
<1551338613481.013428>[PARENT LOG] Sending < Message size :14 >
<1551338613481.023682>[PARENT LOG] Sending < Message : Parent says bye >
<1551338613481.033691>[PARENT LOG] Sending < Message size :15 >
<1551338613481.043945>[PARENT LOG] Sending < Timestamp 1551338613480.918213 >
<1551338613481.060059>[PARENT LOG] Sending < Led Status 0 >
<1551338613481.070557>[PARENT LOG] Sending < Parent Pid 18384 >
------------IPC MECHANISMS -> PIPES-------------
<1551338613477.891602>PIPE 1 CREATED SUCCESSFULLY
<1551338613478.227051>PIPE 2 CREATED SUCCESSFULLY
<1551338613481.892578>FORK SUCCESSFULL
<1551338613484.998291>[CHILD LOG] Child Process ID 18385
```



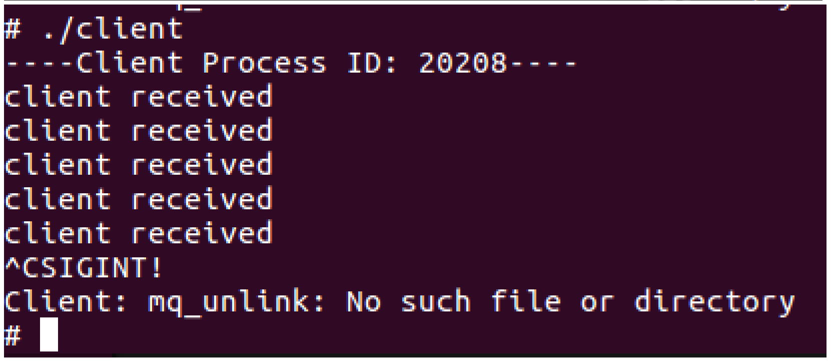
```
-
<1551338624100.677490>[CHILD LOG] Sending < Message : Hello from Child >
<1551338624100.687988>[CHILD LOG] Sending < Message size :16 >
<1551338624100.943848>[CHILD LOG] Sending < Message : IPC pipes Programming >
<1551338624100.967285>[CHILD LOG] Sending < Message size :21 >
<1551338624100.979248>[CHILD LOG] Sending < Message : Child Process >
<1551338624100.989990>[CHILD LOG] Sending < Message size :13 >
<1551338624101.000488>[CHILD LOG] Sending < Message : Child says bye >
<1551338624101.010254>[CHILD LOG] Sending < Message size :14 >
<1551338624101.020020>[CHILD LOG] Sending < Timestamp 1551338623495.910889 >
<1551338624101.036865>[CHILD LOG] Sending < Led Status 1 >
<1551338624101.047119>[CHILD LOG] Sending < Child Pid 18385 >
SIGINT! Received! <1551338624101.446045> Terminating pipes program....
SIGINT! Received! <1551338624104.540771> Terminating pipes program....
#
```

2. POSIX message queue (link)

Proof of execution of posix_q



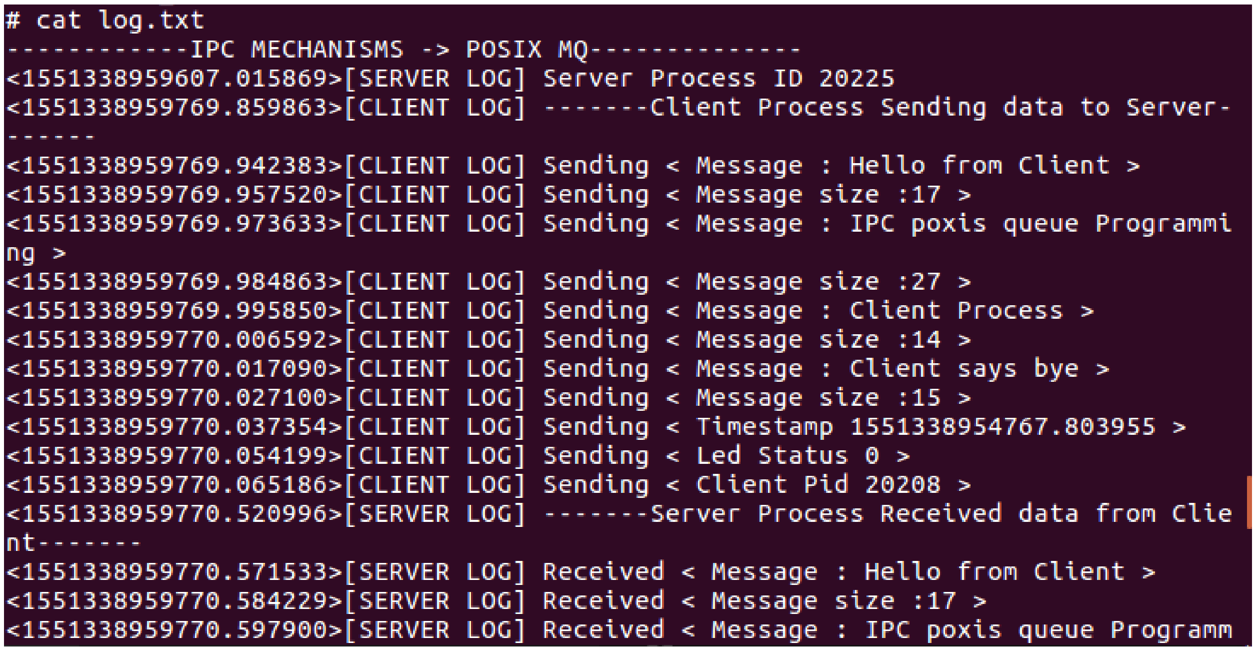
```
# ./server
----Server Process ID: 20225----
Server: message received.
Server: response sent to client.
Server: message received.
Server: response sent to client.
Server: message received.
Server: response sent to client.
Server: message received.
Server: response sent to client.
^CSIGINT!
Server: mq_receive: Interrupted system call
#
```



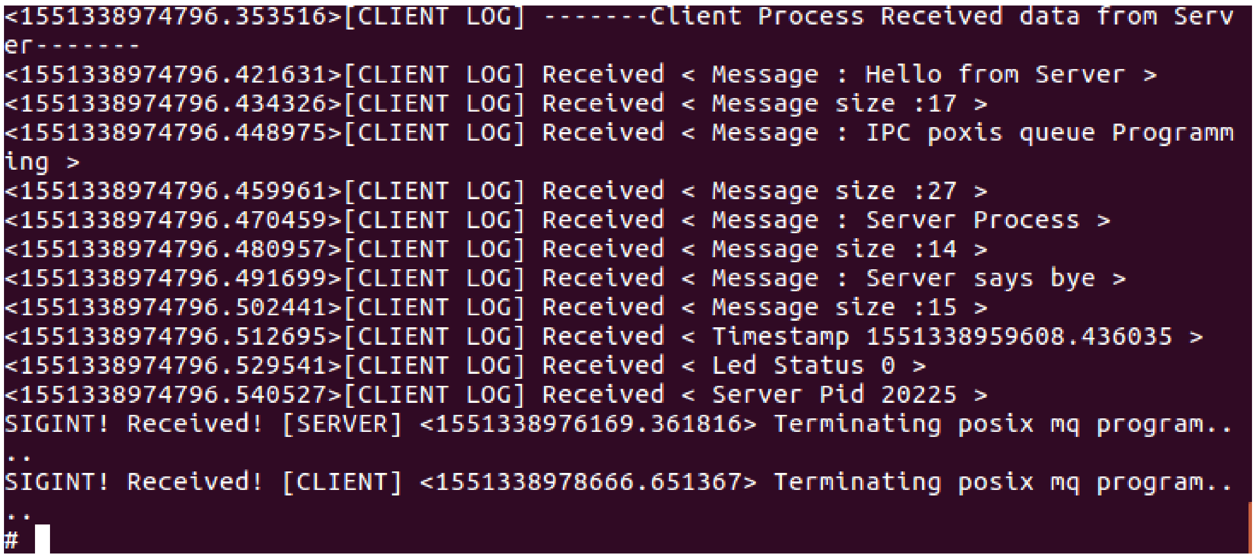
```
# ./client
----Client Process ID: 20208----
client received
client received
client received
client received
client received
^CSIGINT!
Client: mq_unlink: No such file or directory
#
```

The entire file log.txt is included in the posix_q folder of IPC in HW4 in my repository.

To execute start server or client in any order .



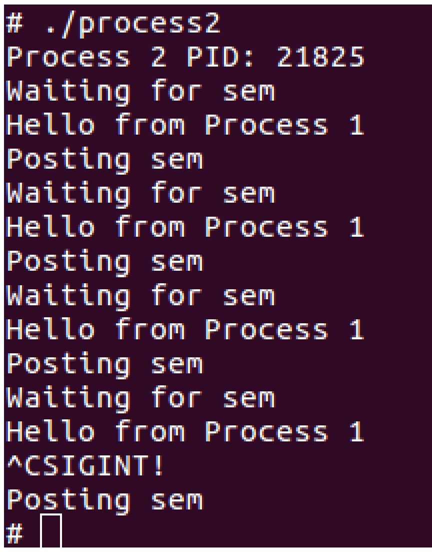
```
# cat log.txt
------------IPC MECHANISMS -> POSIX MQ-------------
<1551338959607.015869>[SERVER LOG] Server Process ID 20225
<1551338959769.859863>[CLIENT LOG] -------Client Process Sending data to Server-
------
<1551338959769.942383>[CLIENT LOG] Sending < Message : Hello from Client >
<1551338959769.957520>[CLIENT LOG] Sending < Message size :17 >
<1551338959769.973633>[CLIENT LOG] Sending < Message : IPC poxis queue Programmi
ng >
<1551338959769.984863>[CLIENT LOG] Sending < Message size :27 >
<1551338959769.995850>[CLIENT LOG] Sending < Message : Client Process >
<1551338959770.006592>[CLIENT LOG] Sending < Message size :14 >
<1551338959770.017090>[CLIENT LOG] Sending < Message : Client says bye >
<1551338959770.027100>[CLIENT LOG] Sending < Message size :15 >
<1551338959770.037354>[CLIENT LOG] Sending < Timestamp 1551338954767.803955 >
<1551338959770.054199>[CLIENT LOG] Sending < Led Status 0 >
<1551338959770.065186>[CLIENT LOG] Sending < Client Pid 20208 >
<1551338959770.520996>[SERVER LOG] -------Server Process Received data from Clie
nt-------
<1551338959770.571533>[SERVER LOG] Received < Message : Hello from Client >
<1551338959770.584229>[SERVER LOG] Received < Message size :17 >
<1551338959770.597900>[SERVER LOG] Received < Message : IPC poxis queue Programm
```



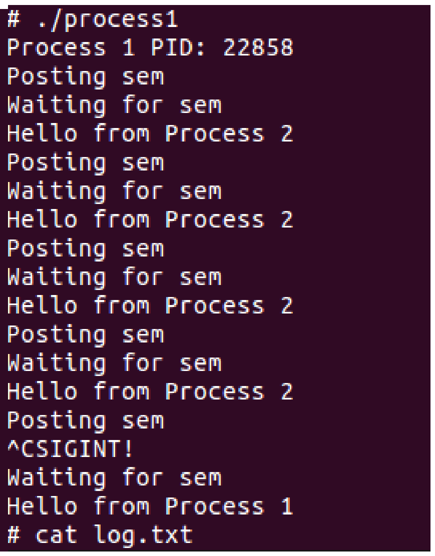
```
<1551338974796.353516>[CLIENT LOG] -------Client Process Received data from Serv
er-------
<1551338974796.421631>[CLIENT LOG] Received < Message : Hello from Server >
<1551338974796.434326>[CLIENT LOG] Received < Message size :17 >
<1551338974796.448975>[CLIENT LOG] Received < Message : IPC poxis queue Programm
ing >
<1551338974796.459961>[CLIENT LOG] Received < Message size :27 >
<1551338974796.470459>[CLIENT LOG] Received < Message : Server Process >
<1551338974796.480957>[CLIENT LOG] Received < Message size :14 >
<1551338974796.491699>[CLIENT LOG] Received < Message : Server says bye >
<1551338974796.502441>[CLIENT LOG] Received < Message size :15 >
<1551338974796.512695>[CLIENT LOG] Received < Timestamp 1551338959608.436035 >
<1551338974796.529541>[CLIENT LOG] Received < Led Status 0 >
<1551338974796.540527>[CLIENT LOG] Received < Server Pid 20225 >
SIGINT! Received! [SERVER] <1551338976169.361816> Terminating posix mq program..
..
SIGINT! Received! [CLIENT] <1551338978666.651367> Terminating posix mq program..
..
#
```

3. Shared Memory(link)

Proof of execution of shared memory

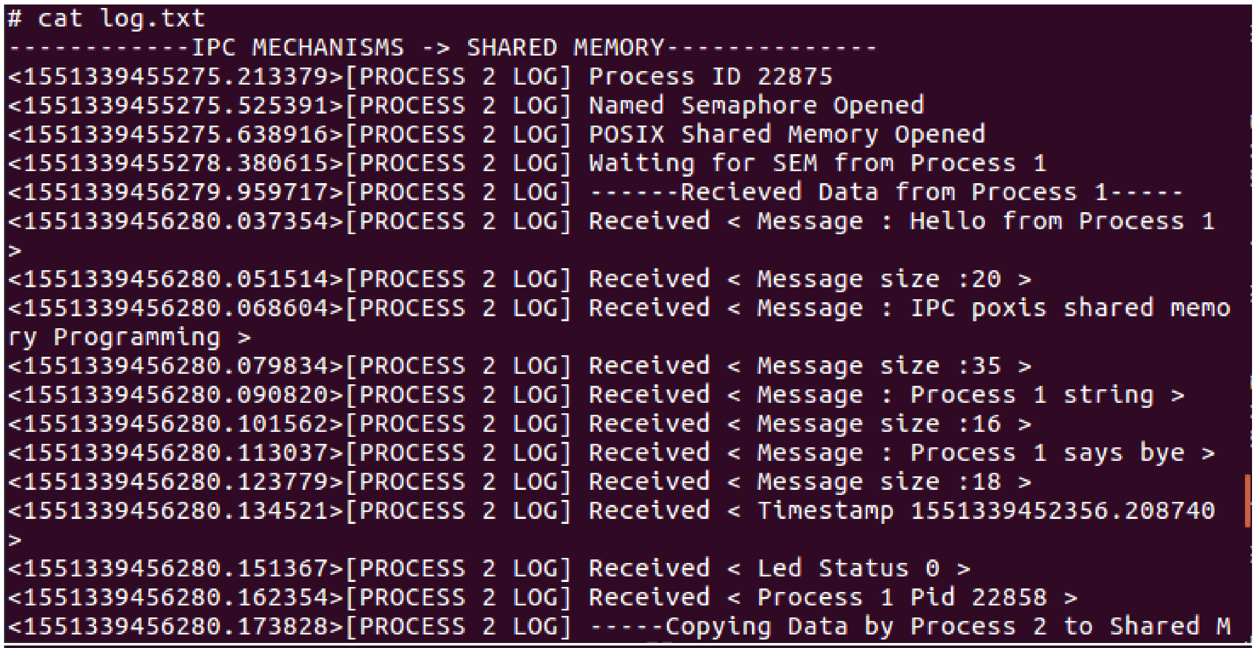


```
# ./process2
Process 2 PID: 21825
Waiting for sem
Hello from Process 1
Posting sem
Waiting for sem
Hello from Process 1
Posting sem
Waiting for sem
Hello from Process 1
Posting sem
Waiting for sem
Hello from Process 1
^CSIGINT!
Posting sem
# 
```

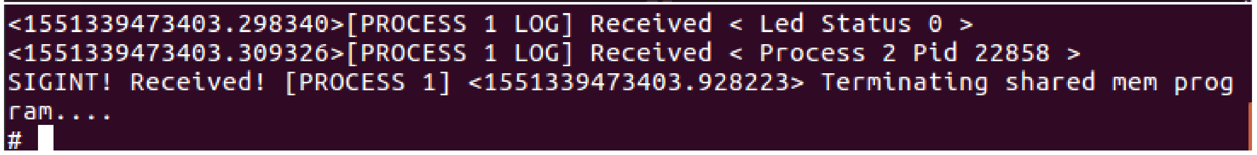


```
# ./process1
Process 1 PID: 22858
Posting sem
Waiting for sem
Hello from Process 2
Posting sem
Waiting for sem
Hello from Process 2
Posting sem
Waiting for sem
Hello from Process 2
Posting sem
Waiting for sem
Hello from Process 2
Posting sem
^CSIGINT!
Waiting for sem
Hello from Process 1
# cat log.txt
```

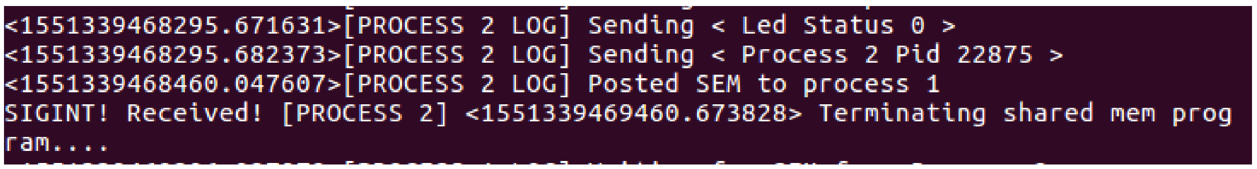
The entire file log.txt is included in the shared_mem folder of IPC in HW4 in my repository.To execute start process2 or process1 in any order .



```
# cat log.txt
------------IPC MECHANISMS -> SHARED MEMORY--------------
<1551339455275.213379>[PROCESS 2 LOG] Process ID 22875
<1551339455275.525391>[PROCESS 2 LOG] Named Semaphore Opened
<1551339455275.638916>[PROCESS 2 LOG] POSIX Shared Memory Opened
<1551339455278.380615>[PROCESS 2 LOG] Waiting for SEM from Process 1
<1551339456279.959717>[PROCESS 2 LOG] ------Recieved Data from Process 1-----
<1551339456280.037354>[PROCESS 2 LOG] Received < Message : Hello from Process 1
>
<1551339456280.051514>[PROCESS 2 LOG] Received < Message size :20 >
<1551339456280.068604>[PROCESS 2 LOG] Received < Message : IPC poxis shared memo
ry Programming >
<1551339456280.079834>[PROCESS 2 LOG] Received < Message size :35 >
<1551339456280.090820>[PROCESS 2 LOG] Received < Message : Process 1 string >
<1551339456280.101562>[PROCESS 2 LOG] Received < Message size :16 >
<1551339456280.113037>[PROCESS 2 LOG] Received < Message : Process 1 says bye >
<1551339456280.123779>[PROCESS 2 LOG] Received < Message size :18 >
<1551339456280.134521>[PROCESS 2 LOG] Received < Timestamp 1551339452356.208740
>
<1551339456280.151367>[PROCESS 2 LOG] Received < Led Status 0 >
<1551339456280.162354>[PROCESS 2 LOG] Received < Process 1 Pid 22858 >
<1551339456280.173828>[PROCESS 2 LOG] -----Copying Data by Process 2 to Shared M
```



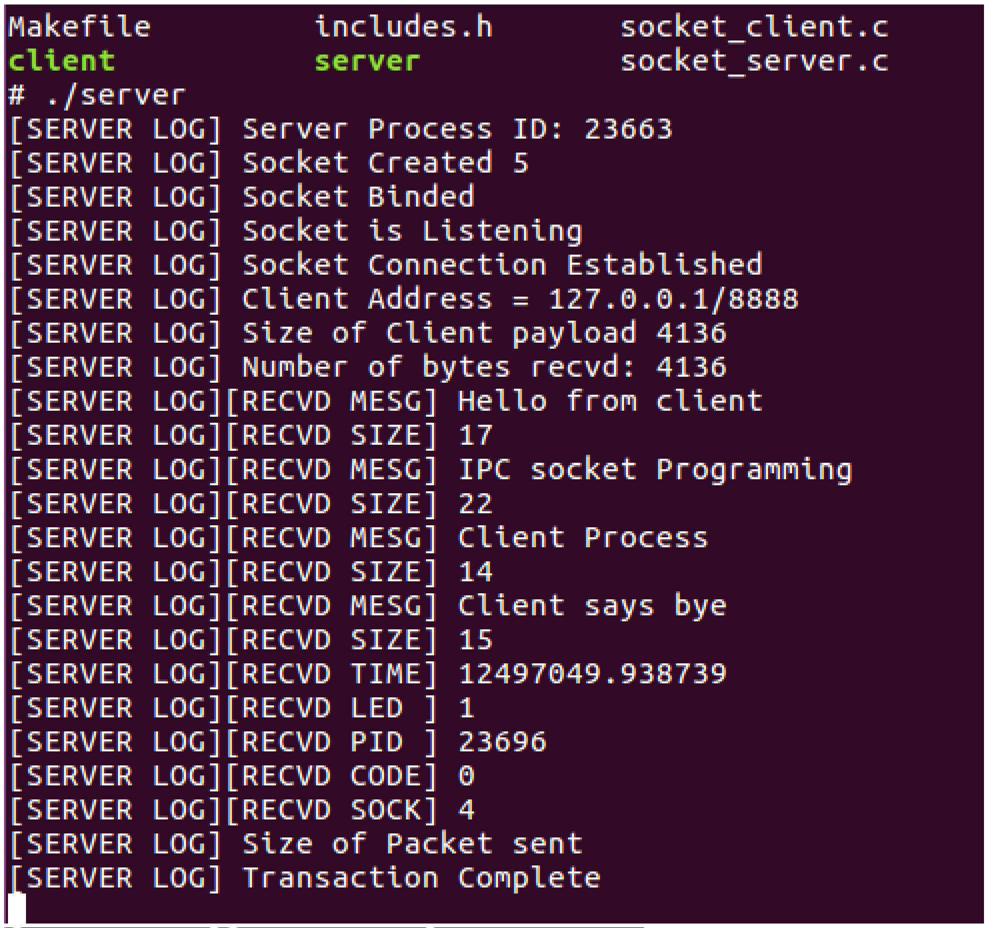
```
<1551339473403.298340>[PROCESS 1 LOG] Received < Led Status 0 >
<1551339473403.309326>[PROCESS 1 LOG] Received < Process 2 Pid 22858 >
SIGINT! Received! [PROCESS 1] <1551339473403.928223> Terminating shared mem prog
ram....
# 
```



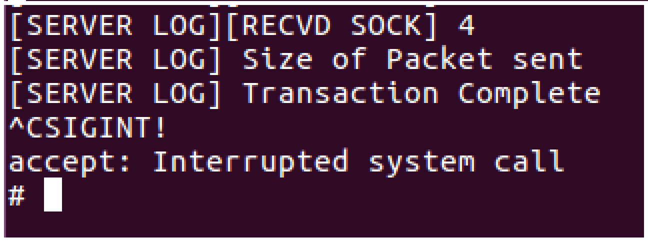
```
<1551339468295.671631>[PROCESS 2 LOG] Sending < Led Status 0 >
<1551339468295.682373>[PROCESS 2 LOG] Sending < Process 2 Pid 22875 >
<1551339468460.047607>[PROCESS 2 LOG] Posted SEM to process 1
SIGINT! Received! [PROCESS 2] <1551339469460.673828> Terminating shared mem prog
ram....
```

4. Socket Programming (link)

Proof of execution of socket programming

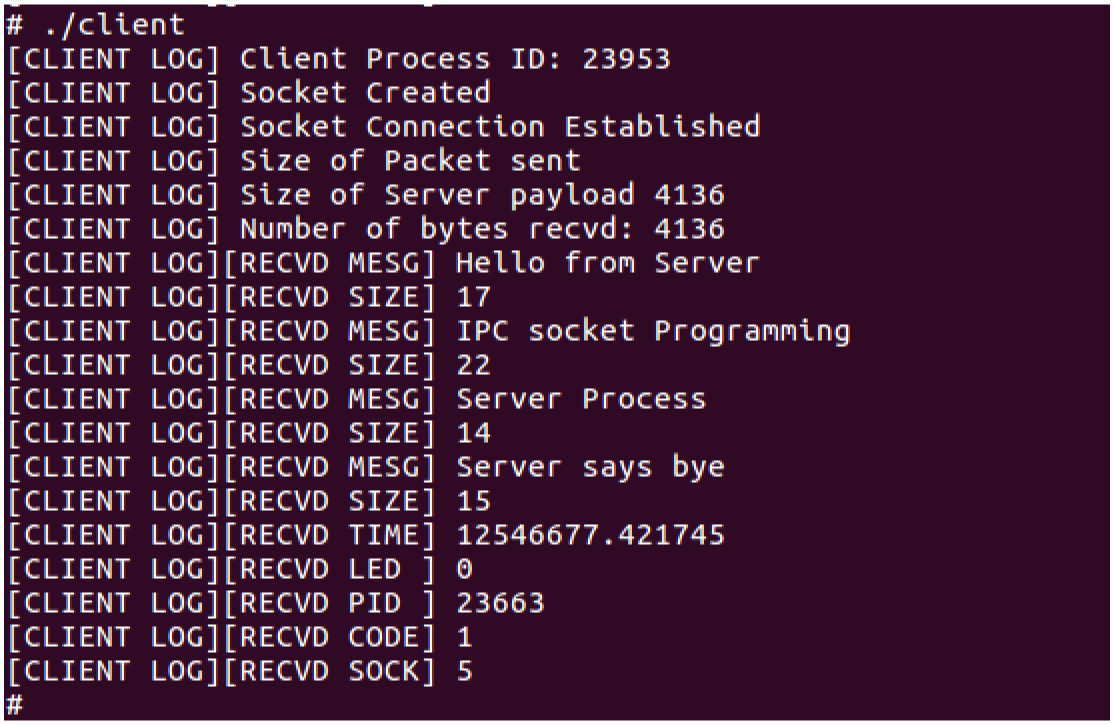


```
Makefile          includes.h        socket_client.c
client            server            socket_server.c
# ./server
[SERVER LOG] Server Process ID: 23663
[SERVER LOG] Socket Created 5
[SERVER LOG] Socket Binded
[SERVER LOG] Socket is Listening
[SERVER LOG] Socket Connection Established
[SERVER LOG] Client Address = 127.0.0.1/8888
[SERVER LOG] Size of Client payload 4136
[SERVER LOG] Number of bytes recvd: 4136
[SERVER LOG][RECVD MESG] Hello from client
[SERVER LOG][RECVD SIZE] 17
[SERVER LOG][RECVD MESG] IPC socket Programming
[SERVER LOG][RECVD SIZE] 22
[SERVER LOG][RECVD MESG] Client Process
[SERVER LOG][RECVD SIZE] 14
[SERVER LOG][RECVD MESG] Client says bye
[SERVER LOG][RECVD SIZE] 15
[SERVER LOG][RECVD TIME] 12497049.938739
[SERVER LOG][RECVD LED ] 1
[SERVER LOG][RECVD PID ] 23696
[SERVER LOG][RECVD CODE] 0
[SERVER LOG][RECVD SOCK] 4
[SERVER LOG] Size of Packet sent
[SERVER LOG] Transaction Complete
```



```
[SERVER LOG][RECVD SOCK] 4
[SERVER LOG] Size of Packet sent
[SERVER LOG] Transaction Complete
^CSIGINT!
accept: Interrupted system call
# 
```

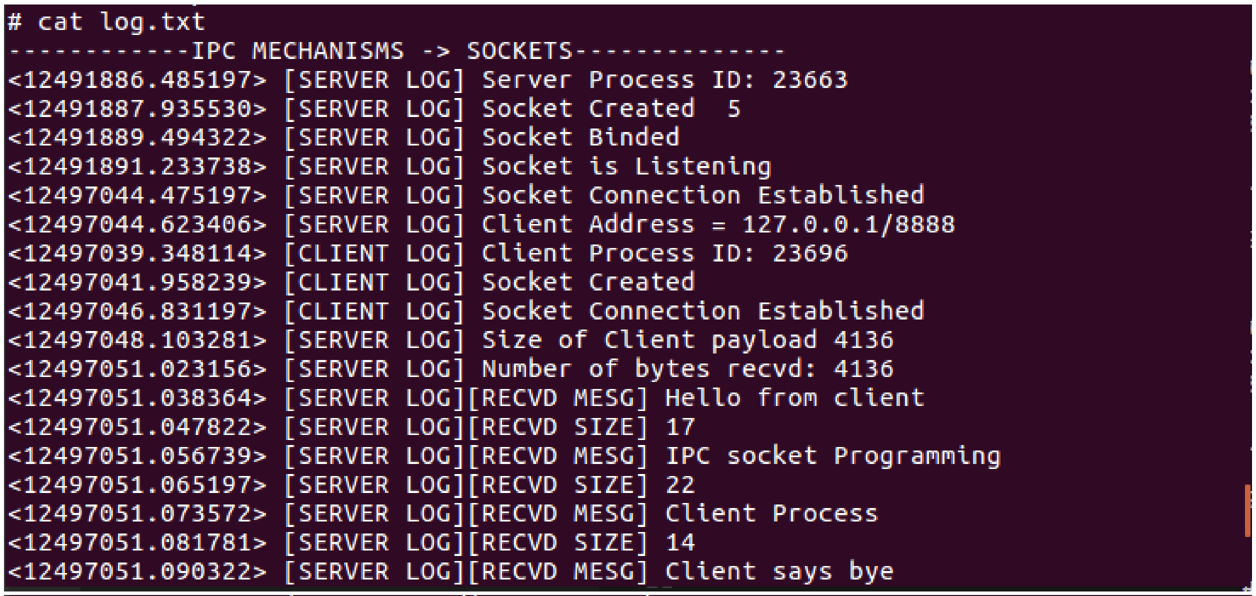
Server is always on and client is terminated each time it sends message. In socket programming multiple clients can connect to the server at the at various instances. Client requests for data or sends data and terminates when its job is done. Hence Client being always on or in a while loop is not is not considered useful.



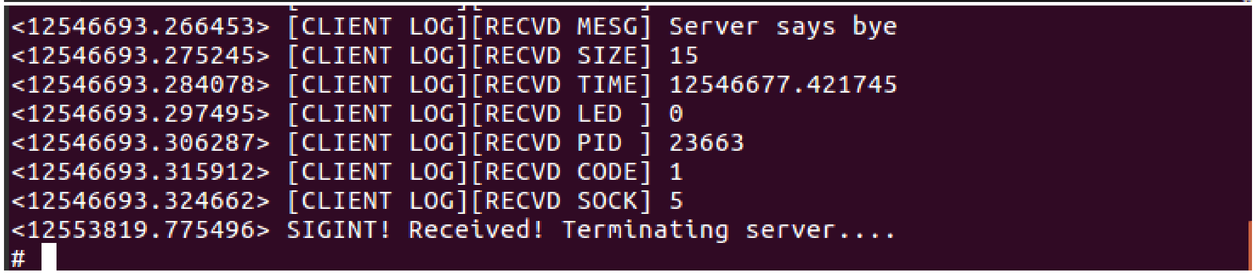
```
# ./client
[CLIENT LOG] Client Process ID: 23953
[CLIENT LOG] Socket Created
[CLIENT LOG] Socket Connection Established
[CLIENT LOG] Size of Packet sent
[CLIENT LOG] Size of Server payload 4136
[CLIENT LOG] Number of bytes recvd: 4136
[CLIENT LOG][RECVD MESG] Hello from Server
[CLIENT LOG][RECVD SIZE] 17
[CLIENT LOG][RECVD MESG] IPC socket Programming
[CLIENT LOG][RECVD SIZE] 22
[CLIENT LOG][RECVD MESG] Server Process
[CLIENT LOG][RECVD SIZE] 14
[CLIENT LOG][RECVD MESG] Server says bye
[CLIENT LOG][RECVD SIZE] 15
[CLIENT LOG][RECVD TIME] 12546677.421745
[CLIENT LOG][RECVD LED ] 0
[CLIENT LOG][RECVD PID ] 23663
[CLIENT LOG][RECVD CODE] 1
[CLIENT LOG][RECVD SOCK] 5
#
```

The file log.txt is included in the sockets folder of IPC in HW4 in my repository

To execute start server first then followed by client.



```
# cat log.txt
------------IPC MECHANISMS -> SOCKETS--------------
<12491886.485197> [SERVER LOG] Server Process ID: 23663
<12491887.935530> [SERVER LOG] Socket Created  5
<12491889.494322> [SERVER LOG] Socket Binded
<12491891.233738> [SERVER LOG] Socket is Listening
<12497044.475197> [SERVER LOG] Socket Connection Established
<12497044.623406> [SERVER LOG] Client Address = 127.0.0.1/8888
<12497039.348114> [CLIENT LOG] Client Process ID: 23696
<12497041.958239> [CLIENT LOG] Socket Created
<12497046.831197> [CLIENT LOG] Socket Connection Established
<12497048.103281> [SERVER LOG] Size of Client payload 4136
<12497051.023156> [SERVER LOG] Number of bytes recvd: 4136
<12497051.038364> [SERVER LOG][RECVD MESG] Hello from client
<12497051.047822> [SERVER LOG][RECVD SIZE] 17
<12497051.056739> [SERVER LOG][RECVD MESG] IPC socket Programming
<12497051.065197> [SERVER LOG][RECVD SIZE] 22
<12497051.073572> [SERVER LOG][RECVD MESG] Client Process
<12497051.081781> [SERVER LOG][RECVD SIZE] 14
<12497051.090322> [SERVER LOG][RECVD MESG] Client says bye
```



```
<12546693.266453> [CLIENT LOG][RECVD MESG] Server says bye
<12546693.275245> [CLIENT LOG][RECVD SIZE] 15
<12546693.284078> [CLIENT LOG][RECVD TIME] 12546677.421745
<12546693.297495> [CLIENT LOG][RECVD LED ] 0
<12546693.306287> [CLIENT LOG][RECVD PID ] 23663
<12546693.315912> [CLIENT LOG][RECVD CODE] 1
<12546693.324662> [CLIENT LOG][RECVD SOCK] 5
<12553819.775496> SIGINT! Received! Terminating server....
#
```

References:

1. Sockets:
   https://stackoverflow.com/questions/33848558/catching-sigterm-in-c
   https://www.geeksforgeeks.org/socket-programming-cc/
2. pipes:
   https://www.geeksforgeeks.org/c-program-demonstrate-fork-and-pipe/
   https://www.tutorialspoint.com/inter_process_communication/inter_process_communication_pipes.htm
3. Posix queues:
   https://www.softprayog.in/programming/interprocess-communication-using-posix-message-queues-in-linux
4. Shared memory:
   https://www.geeksforgeeks.org/posix-shared-memory-api/
   https://stackoverflow.com/questions/8359322/how-to-share-semaphores-between-processes-using-shared-memory
5. Pthreads
   https://www.linuxquestions.org/questions/programming-9/c-timer_create-skeleton-922654/
   https://stackoverflow.com/questions/5740954/problem-in-timers-and-signal
   https://stackoverflow.com/questions/44787643/custom-signal-handler-in-c
   Makefile from RTES course
   https://stackoverflow.com/questions/9229333/how-to-get-overall-cpu-usage-e-g-57-on-linux
   https://stackoverflow.com/questions/166884/array-versus-linked-list