1. Program1
    a) basic ideas of the program

        The program is to run a process under user mode. The main process will fork a child process to execute a test program and wait for its returning signal. After receiving the terminated signal, the parent process will print out the related information of the signal.

    b) implementation

        Firstly, the program will use fork( ) function to fork a child process at the beginning. Then, the program will check the pid of the process and do the corresponding operations. If pid equals to 0, which means the process is the child process, then the program will let it execute the test program. If pid not equals to 0 or -1, then it is the parent process and it will wait for the terminated signal of the child process (using waitpid( )) and print out the information of the signal.

    c) environment

        version of OS: Ubuntu 16.04.2

        version of kernel: 4.10.14

    d) steps to execute the program

        i. go to the folder of the program (program1).

        ii. type the terminal “sudo make” in the terminal

        iii. type the terminal “./program1 ./testProgram” in the terminal. Notice that the “./testProgram” in the terminal need to be replaced by the name of the test program, for example, replace it with “./abort” if the test program is “abort”.

    e) outputs for the program

        1. normal

```
[10/08/20]seed@VM:~/.../program1$ ./program1 ./normal
Process start to fork
I'm the Parent Process, my pid = 5908
I'm the Child Process, my pid = 5909
Child process start to execute the program
------------CHILD PROCESS START------------
This is the normal program

------------CHILD PROCESS END------------
Parent process receiving the SIGCHLD signal
Normal termination with EXIT STATUS = 0
```

2. abort

```
[10/08/20]seed@VM:~/.../program1$ ./program1 ./abort
Process start to fork
I'm the Parent Process, my pid = 5915
I'm the Child Process, my pid = 5916
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGABRT program

Parent process receiving the SIGCHLD signal
child process get SIGABRT signal
child process is abort by abort signal
CHILD EXECUTION FAILED!!
```

3. alarm

```
[10/08/20]seed@VM:~/.../program1$ ./program1 ./alarm
Process start to fork
I'm the Parent Process, my pid = 6007
I'm the Child Process, my pid = 6008
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGALRM program

Parent process receiving the SIGCHLD signal
child process get SIGALRM signal
child process is terminated by alarm signal
CHILD EXECUTION FAILED!!
```

4. bus

```
[10/08/20]seed@VM:~/.../program1$ ./program1 ./bus
Process start to fork
I'm the Parent Process, my pid = 6010
I'm the Child Process, my pid = 6011
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGBUS program

Parent process receiving the SIGCHLD signal
child process get SIGBUS signal
child process gets bus error
CHILD EXECUTION FAILED!!
```

5. floating

```
[10/08/20]seed@VM:~/.../program1$ ./program1 ./floating
Process start to fork
I'm the Parent Process, my pid = 5931
I'm the Child Process, my pid = 5932
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGFPE program

Parent process receiving the SIGCHLD signal
child process get SIGFPE signal
child process gets floating point exception
CHILD EXECUTION FAILED!!
```

6. hangup

```
[10/08/20]seed@VM:~/.../program1$ ./program1 ./hangup
Process start to fork
I'm the Parent Process, my pid = 6023
I'm the Child Process, my pid = 6024
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGHUP program

Parent process receiving the SIGCHLD signal
child process get SIGHUP signal
child process is hang up by hangup signal
CHILD EXECUTION FAILED!!
```

7. illegal_instr

```
[10/08/20]seed@VM:~/.../program1$ ./program1 ./illegal_instr
Process start to fork
I'm the Parent Process, my pid = 6038
I'm the Child Process, my pid = 6039
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGILL program

Parent process receiving the SIGCHLD signal
child process get SIGILL signal
child process gets illegal instruction
CHILD EXECUTION FAILED!!
```

8. interrupt

```
[10/08/20]seed@VM:~/.../program1$ ./program1 ./interrupt
Process start to fork
I'm the Parent Process, my pid = 5938
I'm the Child Process, my pid = 5939
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGINT program

Parent process receiving the SIGCHLD signal
child process get SIGINT signal
child process is interrupted by interrupt signal
CHILD EXECUTION FAILED!!
```

9. kill

```
[10/08/20]seed@VM:~/.../program1$ ./program1 ./kill
Process start to fork
I'm the Parent Process, my pid = 6052
I'm the Child Process, my pid = 6053
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGKILL program

Parent process receiving the SIGCHLD signal
child process get SIGKILL signal
child process is killed by kill signal
CHILD EXECUTION FAILED!!
```

10. pipe

```
[10/08/20]seed@VM:~/.../program1$ ./program1 ./pipe
Process start to fork
I'm the Parent Process, my pid = 6062
I'm the Child Process, my pid = 6063
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGPIPE program

Parent process receiving the SIGCHLD signal
child process get SIGPIPE signal
child process writes to pipe with no readers
CHILD EXECUTION FAILED!!
```

11. quit

```
[10/08/20]seed@VM:~/.../program1$ ./program1 ./quit
Process start to fork
I'm the Parent Process, my pid = 6073
I'm the Child Process, my pid = 6074
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGQUIT program

Parent process receiving the SIGCHLD signal
child process get SIGQUIT signal
child process is quited by quit signal
CHILD EXECUTION FAILED!!
```

12. stop

```
[10/08/20]seed@VM:~/.../program1$ ./program1 ./stop
Process start to fork
I'm the Parent Process, my pid = 6082
I'm the Child Process, my pid = 6083
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGSTOP program

Parent process receiving the SIGCHLD signal
child process get SIGSTOP signal
child process stopped
CHILD PROCESS STOPPED
```

13. terminate

```
[10/08/20]seed@VM:~/.../program1$ ./program1 ./terminate
Process start to fork
I'm the Parent Process, my pid = 6091
I'm the Child Process, my pid = 6092
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGTERM program

Parent process receiving the SIGCHLD signal
child process get SIGTERM signal
child process is terminated by termaniation signal
CHILD EXECUTION FAILED!!
```

14. trap

```
[10/08/20]seed@VM:~/.../program1$ ./program1 ./trap
Process start to fork
I'm the Parent Process, my pid = 6103
I'm the Child Process, my pid = 6104
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGTRAP program

Parent process receiving the SIGCHLD signal
child process get SIGTRAP signal
child process is terminated by trap signal
CHILD EXECUTION FAILED!!
```

f) what I learnt from the project

First, I learnt how to create child processes and how to use them to do some task. I also learnt how to execute other programs in the process using execve( ) function. Then, I also got some feelings about the terminated signals and know some basic knowledge of them.

2. Program2

a) basic ideas of the program

This program will create a kernel thread. In the thread, the program will fork a child process and make it to execute another program. The parent process will wait for the child's terminated signal and print out related information.

b) implementation

i. Firstly, since we need to use these 4 functions: "_do_fork"

(/kernel/fork.c) , “do_execve” (/fs/exec.c), “getname” (/fs/namei.c) and “do_wait”(/kernel/exit.c), we need to find their corresponding files in kernel and export them using EXPORT_SYMBOL. After modifications, we need to recompile the kernel. Then we just need to use extern to import them in program2.c.

ii. Then, when initiating the module, we need to create a kernel thread using kthread_create( ) to run my_fork( ) function, where the program will fork a child process to execute another program.

iii. To implement my_fork( ), several functions need to be implemented in advance.

a) my_exec( ):

This function will first get the information (using getname( )) of the test program using the address set inside it. After that, it will use do_execve( ) to execute the file.

b) my_wait( ):

this function will get the terminated signal of the child process using do_wait( ).

With these 2 functions, my_fork( ) is implemented in the following: First, my_fork( ) will use _do_fork to generate a child process to run my_exec( ) function. Then it will use my_wait( ) to get the terminated signal of the child process and print out corresponding information.

iv. After creating the kernel thread, the program will wake up the program if there kthread_create( ) create a thread successfully.

c) environment

version of OS: Ubuntu 16.04.2

version of kernel: 4.10.14

d) steps to execute the program

i. modified the kernel files of “_do_fork”, “do_execve” , “getname” and “do_wait”. After modification, rebuild the module using the following commands: “make bzImage”, “make modules”, “make modules_install”,

"make install", "reboot".

ii. go to the folder of program2 and type "sudo make" in the terminal.

iii. then type "sudo insmod program2.ko" and "sudo rmmod program2.ko" to insert and remove the module.

iv. use "gcc test.c -o test" to compile the test file.

v. using "dmesg" in the terminal to display the corresponding message.

e) sample outputs for the program

1. test.c (SIGBUS):

```
[20529.800497] [program2] : module_init
[20529.800499] [program2] : module_init create kthread start
[20529.804759] [program2] : module_init Kthread starts
[20529.808581] [program2] : The Child process has pid = 7101
[20529.808582] [program2] : This is the parent process, pid = 7099
[20529.808584] [program2] : child process
[20529.809001] [program2] : CHILD EXECUTION FAILED!!
[20529.809003] [program2] : child process get SIGBUS signal
[20529.809003] [program2] : The return signal is 7
[20537.624459] [program2] : module_exit
```

2. normal (normal in program1):

```
[20772.274390] [program2] : module_init
[20772.274391] [program2] : module_init create kthread start
[20772.274411] [program2] : module_init Kthread starts
[20772.279998] [program2] : The Child process has pid = 7876
[20772.280000] [program2] : This is the parent process, pid = 7874
[20772.280497] [program2] : child process
[20772.283174] [program2] : child process gets normal termination
[20772.283175] [program2] : The return signal is 0
[20780.359328] [program2] : module_exit
```

3. stop (stop in program1):

```
[20910.106605] [program2] : module_init
[20910.106606] [program2] : module_init create kthread start
[20910.106624] [program2] : module_init Kthread starts
[20910.110626] [program2] : The Child process has pid = 8304
[20910.110627] [program2] : This is the parent process, pid = 8302
[20910.114223] [program2] : child process
[20910.125070] [program2] : CHILD PROCESS STOPPED
[20910.125071] [program2] : child process get SIGSTOP signal
[20910.125072] [program2] : The return signal is 19
[20917.054198] [program2] : module_exit
```

f) what I learnt from the project

In this project, I learnt how to modify the kernel files and recompile the kernel to use it. I also learnt how to insert and remove modules to kernels.

3. bonus
    a) basic ideas of the program

        This program is quite similar to program1, however, it needs to execute the functions recursively, that is, it will create processes recursively to handle the test files. For example, if the input files are “test1 test2”, then the program would generate a child process executing “test1” and this child process will generate another child process to executing “test2” recursively.

    b) implementation
        i. In the main function, the program will first create 2 pointers mapping to 2 arrays in the memory. In this way, different child processed can modified and share the same arrays. One of the arrays holding the pids for different process and another array holds the file names of the testing files.
        ii. After that, the main function will record the test file names one shared array. Then, it will fork( ) a child process to run the recursiveProcess( ), which will run the test files recursively. The parent process will wait for child process and then print out all the information about the recursive processes.
        iii. For recursiveProcess( ) function, it will first generate a child process to execute next test program in the test file list if its test program is not the last one. And it will wait for its child process and store the return signal and pid in the share arrays. After that, it will generate its own test program and return to its parent process.

    c) environment

        version of OS: Ubuntu 16.04.2

        version of kernel: 4.10.14

    d) steps to execute the program
        i. go to the bonus folder and type “sudo make” in the terminal.
        ii. type “./myfork test1 test2 …” in the command to execute files. Notice that test1 and test2 are the name of the testing program.

e) sample outputs:

1. no test file:

```
[10/08/20]seed@VM:~/.../bonus$ ./myfork
the process tree : 8845
myfork(pid=8845) has normal execution
the exit status = 0
```

2. single test files:

```
[10/08/20]seed@VM:~/.../bonus$ ./myfork bus
------------CHILD PROCESS START------------
This is the SIGBUS program

the process tree : 8859->8860
The child process (pid=8860) of the parent process(pid=8859)is termiated by siga
nl
Its signal number = 7
child process get SIGBUS signal
child process gets bus error
CHILD EXECUTION FAILED!!

myfork(pid=8859) has normal execution
the exit status = 0
```

3. multiple test files:

```
[10/08/20]seed@VM:~/.../bonus$ ./myfork bus normal1 trap hangup
------------CHILD PROCESS START------------
This is the SIGHUP program

------------CHILD PROCESS START------------
This is the SIGTRAP program

This is normal1 program
------------CHILD PROCESS START------------
This is the SIGBUS program

the process tree : 9217->9218->9219->9220->9221
The child process (pid=9221) of the parent process(pid=9220)is termiated by siganl
Its signal number = 1
child process get SIGHUP signal
child process is hang up by hangup signal
CHILD EXECUTION FAILED!!

The child process (pid=9220) of the parent process(pid=9219)is termiated by siganl
Its signal number = 5
child process get SIGTRAP signal
child process is terminated by trap signal
CHILD EXECUTION FAILED!!

The child process (pid=9219) of the parent process(pid=9218)has normal execution
the exit status = 0

The child process (pid=9218) of the parent process(pid=9217)is termiated by siganl
Its signal number = 7
child process get SIGBUS signal
child process gets bus error
CHILD EXECUTION FAILED!!

myfork(pid=9217) has normal execution
the exit status = 0
```

f) what I learnt from the project

In this program, I learnt how to use memory map to allow different child processes to share the same memory. Also, I learnt how to use child processes to generate other processes recursively.