Lab 1

1. Starting Code: fork_tb.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.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 processs */
         execlp("/bin/ls", "ls", NULL);
   }else{ /* parent process */
         /* parent will wait for the child to complete */
         wait(NULL);
         printf("Child Complete\n");
   }

   return 0;
}
```

Execution:
gcc fork_tb.c -o fork
./fork

Output: (it depends on your directory files)
fork forkQ1 forkQ2 forkQ3 forkstart fork_tb.c prac simpleshell text.c
fork.c forkQ1.c forkQ2.c forkQ3.c forkstart.c linuxSched.c prac1.c simpleshell.c text.txt
Child Complete ← Must be printed after executing the 'ls'

2. Lab Practices

- What does it happen if there is no the ***'wait***(NULL)'?
- How to execute "ls -l" using the ***'execlp"***?
- Test using ***execl***(), ***execv***() and ***execvp***() using linux manuals.
- Test using ***waitpid()*** instead of ***wait***().
- Test ***getpid***() and ***getppid***().

3. Lab Assignment: Submission date on the moodle (31/1, Friday 11 am) and Online Grading (31/1, 1 pm during lecture session or using the submitted files)

- Complete the below **simple_shell.c** file, which is a very simple shell that only handles **single word commands (no arguments)**; type "quit' to stop the program. (NOTE: don't have to support multi-word commands).
- And test your program using the output.

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

#define CMDLEN 80

int main(){
   int pid;    int status;
   char command[CMDLEN];

   printf("Program begins\n");
   for(;;){
         // user input
         printf("Please enter a command: ");
         TODO1: user input command preparation using fgets()
         if (strcmp(command, "quit") == 0) break;

         pid = fork();

         if (pid < 0){ // fork error

         }else if (pid > 0){ // parent
               TODO2: print pid of Child and the status (0 means okay)
         }else{ // child
               TODO3: print received command and execute the command.
         }
   }
   return 0;
   }
```

Output : (Example, the results depends on your machine)

```
jurn@ZBookG4:~/work/cs332/process$ ./simpleshell
Program begins
Please enter a command: ls
(received ls)
fork  forkQ1  forkQ2  forkQ3  forkstart  fork_tb.c  prac  simpleshell  text.c
fork.c  forkQ1.c  forkQ2.c  forkQ3.c  forkstart.c  linuxSched.c  prac1.c  simpleshell.c  text.txt
child pid=17276, parent pid=17275 (parent ppid =5833), status=0
-------------------------------------------------------------------- (just separation, don't have to print this line)
Please enter a command: pwd
(received pwd)
/home/jurn/work/cs332/process
child pid=17277, parent pid=17275 (parent ppid =5833), status=0
--------------------------------------------------------------------
Please enter a command: whoami
(received whoami)
jurn
child pid=17278, parent pid=17275 (parent ppid =5833), status=0
--------------------------------------------------------------------
Please enter a command: quit
jurn@ZBookG4:~/work/cs332/process$
```