CS3377 Assignment 4

1. Assignment Learning Outcome:

- How to create a child process
- How to use stdin and stdout
- How to use pipes

2. Problem Description

Part1

In class, we have seen the following code, that uses pipes and fork() call to implement the shell pipe "|". The code is following:

```
// file: onepipe.cpp
// author: M. Amine Belkoura
// date: 03/04/2015
// purpose: CS3376
// description:
//   this program executes "ls -ltr | grep 3376", by dividing the two command
among the child and parent process

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

int main(int argc, char **argv){
  int status;
  int childpid;

  char *cat_args[] = {"ls", "-ltr", NULL};
  char *grep_args[] = {"grep", "3376", NULL};

  // create one pipe to send the output of "ls" process to "grep" process
  int pipes[2];
  pipe(pipes);


  // fork the first child (to execute cat)
  if((childpid = fork()) == -1){
               perror("Error creating a child process");
               exit(1);
  }
  if (childpid == 0) {
      // replace cat's stdout with write part of 1st pipe

      dup2(pipes[1], 1);
      // close all pipes (very important!); end we're using was safely copied

      close(pipes[0]);
```

```
      close(pipes[1]);
      execvp(*cat_args, cat_args);
      exit(0);
  }
  else {

     // replace grep's stdin with read end of 1st pipe
     dup2(pipes[0], 0);

     close(pipes[0]);
     close(pipes[1]);

     execvp(*grep_args, grep_args);
  }
  return(0);

}
```

Change code to execute the following double pipe command: "`ls -ltr | grep 3376 | wc -l`"

1- Use one parent and two children to do the work. Call the file `TwoPipesTwoChildren.cpp.`
2- Write another version where the parent create 3 children, and the children will execute the commands (parent will do nothing, just lay in the sofa ☺). Call the file `TwoPipesThreeChildren.cpp`

Part2

In part1, a double pipe is used to execute the command "`ls -ltr | grep 3376 | wc -l`". However, our program is static: it means that code need to be modified and rebuilt if we want to change the commands to execute. In this part, we make our program dynamic instead of static.

The following requirements should be implemented:

1- Source file will be called `DynPipe.cpp`, executable called `dynpipe`.
2- The piped commands to execute need to be passed as arguments to `dynpipe`, and not hardcoded.
3- The max number of arguments should not exceed 5, and not less than 2 (otherwise print an error message)
4- Each argument should be a unix/linux command with its parameters. The first argument will be the first to execute, followed by the second one, etc.. We will assume that only valid commands can be used, for simplicity

Example: the followings are possible command executions

Program execution	Shell equivalent
`dynpipe "ls -ltr" "grep 3376"`	`ls -ltr |grep 3376`
`dynpipe "ls -ltr" "grep 3376" "grep hi""wc -l"`	`ls -ltr |grep 3376| grep hi| wc-l`
`dynpipe "ls -ltr"`	`error`

3. WHAT TO SUBMIT TO ELEARNING:

1- Put both TwoPipeThreeChildren.cpp and DynPipe.cpp ina zip file called <firstname>_<lastname>_assign3.zip. The source files should compile using two-step building process:

```
g++ -c -Wall -o TwoPipeThreeChildren.o TwoPipeThreeChildren.cpp
g++ -c -Wall -o DynPipe.o DynPipe.cpp
g++ -o TwoPipeThreeChildren TwoPipeThreeChildren.o
g++ -o DynPipe DynPipe.o
```

2- Submit the <firstname>_<lastname>_assign3.zip to elearning.