CSCI 4061: Introduction to Operating Systems (Spring 2014)
Programming Assignment 1 (100 pts.)

Released: Feb 4, 2014 Due: Feb 19, 2014

Welcome to CSCI4061. Hope you will have fun cracking this assignment!

1 Pre-requisites

It is assumed that you have **good** knowledge of C. Defining good: you can write, compile, run and debug C code. Also, you should have basic knowledge of *make* and structure of a *MakeFile*.

2 Introduction

This assignment will give you a practical exposure on spawning and maintaining Processes. You will get to code in C, and learn a few system calls: fork, exec, wait.

3 Problem

You will develop a simple custom *make* utility similar to GNU *make* [1]. GNU *make* is mature and has many interesting features, implementing each is out of scope and time limits of this assignment. For your as well as our simplicity let us unanimously call this utility *custmake*.

The problem is simple, your program reads a file named ***custMakefile*** and executes commands inside that file. The catch is, commands are grouped into targets and each target ***may*** depend on few other targets which should be executed first. An example ***custMakefile***:

```
target1: dependentTarget1 dependentTarget2 ...
         command1 <args>
         command2 <args>
end

target2: ...
end

dependentTarget1: ...
end

dependentTarget2: ...
end
```

This file is explained in detail in section 3.1 below.

Input:

```
$ custmake [options] [target]
```

> *custmake* is the name of the executable. For our simplicy make sure name of the executable is *custmake*, a template Makefile is provided (see section 4 below), please do not change the output executable name in that.
>
> At most one target that should be executed may be specified. If zero, first target in *custfile* should be executed.

```
options:
  -f filename: Use filename instead of custMakefile, to read targets from.
  -n: Only displays the commands that would be run, don't actually execute them.
```

Output:
The program should execute the targets as described in section 3.2 below and print the command line statements that will be executed in order. Also print warnings or error messages as appropriate.

3.1 *custMakefile* file explained

The file contains multiple targets. First line of each target contains a list of target dependencies separated by one or more horizontal white space characters(U+0020 [2]). Next lines contain multiple commands. Each command is seperated by line break(s) (Windows: CR-LF, Unix: LF and Macintosh: CR). A command contains the first word as executable name, followed by arguments all separated by white space characters.

Rules:

1. Each line in *custMakefile* is either ***a*** target, command, empty line or contains only whitespaces.
2. Empty lines and whitespaces should be ignored.
3. Target names can only contain alphanumeric characters. Command and argument names should not contain white spaces and should be equivalent to Unix command line.
4. "end" should mark the end of target.
5. Some targets may not have any commands associated with them.
6. Dependencies in the file can be represented as a DAG (Directed Acyclic Graph). Pay particular attention to the term Acyclic. Let us define a relationship ***depends on*** represented symbolically as $\rightarrow$. To give an example, **Target1** $\rightarrow$ **Target2** $\rightarrow$ **Target3** $\rightarrow$ **...** $\rightarrow$ **Targetn** $\rightarrow$ **Target1** will never be an input. In nutshell, there will be no cycles.
7. Each line in file cannot be more than 160 characters.
8. Some targets may not contain any dependencies. In this case, keep the line empty after colon. If you follow rule 5 above, there will be atleast one target with no dependencies.
9. Each vertex(i.e., target) in the DAG created at **runtime** should have at most one parent. In this case, to be more specific representation will be a tree. It can be safely assumed that the input file and input target argument will follow this rule.
10. Special characters such as "|, <, >, &" are not allowed in the commands. As one of the consequences, you can safely assume that there will be no background process executions.
11. Path names in arguments to command lines should be absolute if not relative to the directory from which you are executing the executable.

Parsing Hints:

1. For simplicity assume there is no separator between target name and colon.
2. Parse whole file and represent dependecies in some appropriate data structure before you start executing targets.
3. Consider using *strtok()* function in C to tokenize a line. Also, few commands in this link [3] can help.
4. Parsing may not need to be extensive, shoot an email if you have any questions that are not answered in this document.

3.2 Target Execution

This section aims at explaining target execution. Your code should execute targets by ordering them. There is only one initial target: the target provided as argument at command line or the first target in the file. This target can have dependencies which can further have their own dependencies and so on. This can be thought of as nested execution and should stop when a target with no dependencies is reached. Before executing each target, dependent targets should be executed. Executing any target is a two-step process:

1. Read all target dependencies and execute them in order from left to right in the file.
2. Execute all commands in order from top to below in the file until "end" is found.

Consider the example file in Section 3 above, first target executed should be *target1* assuming no target was specified as command line argument. In step one of execution, your code should read all the dependent targets and execute each in the two-step process. So, first execute *dependentTarget1* and then *dependentTarget2* by reading those targets from same file. In step two, execute commands (*command1* and *command2*) using fork and exec. Make sure to finish execution of step one before proceeding to next step.Execution stops after this as "end" string is read.

Rules and Tips:

1. Execute all the commands in all the targets encountered.

2. Any error message printed should be prefixed by "ERROR: " (do not print quotes, in most programming languages it is a way to enclose a String). Similarly a warning should be prefixed by "WARN: ".
3. If any of the targets do not exist in file, print a warning and continue execution.
4. Consider using *fork, exec, execv* and/or *wait* system calls.

3.3 An Example

Here is a simple example of **custMakefile** and order of actions that should be performed.

```
all: main fake
end

main: util
      echo Compile main
      echo Stop
end

util: parseutil
      echo Compile util
end

parseutil:
      echo Start
      echo Compile parseutil
end

fake:
end

clean:
      echo Clean object files
      echo Clean executable
end
```

Sample Input1 (no argument):
$ *custmake*

Output1:
Targets should be executed in order: 1. all, 2. main 3. util. 4. parseutil 5. fake. Executions will finish in order: 1. parseutil 2. util 3. main 4. fake 5. all. Output will be:

```
Start
Compile parseutil
Compile util
Compile main
Stop
```

While executing each command, *fork* and *exec* should be used. For example above, first command to be executed is *echo* with argument *Start*. Next command should not be executed unless previous finishes.

Sample Input2 (with some specific target as argument):
$ *custmake clean*

Output2:
Only **clean** target will be executed. Output:

```
Clean object files
Clean executables
```

4 Extra Credit

If you are able to parse comma separated multiple commands in a single line and execute them concurrently, you will get extra credit points. Refer section 3.2 for execution details. It can be safely assumed that command and its arguments will not contain comma. Please note that this is optional, but will be graded if you do it. Example file:

```
parallel:
      echo Start
      echo Compile parseutil,echo Compile util,echo Compile main
      echo Stop
end
```

Sample Input3 (custom file name, extra credit):
Assume that the name of the file above is custMakeFile_extra. Input command:
$ *custmake -f custMakeFile_extra parallel*

Output3:

```
Start
```

Following three commands will be executed **concurrently**, so can be any order:

```
Compile parseutil
Compile util
Compile main

Stop
```

5 Code organization

Use the base template provided with this assignment on Moodle. Tar file should contain a dir named **csci4061pa1** which should contain following:

1. **main.c:** This is the main file that should contain the main() function. This is where you will parse input from files, build data structures and run targets.
2. **main.h:** Header file for main.c. Header files are used to contain function signatures, global constants and some preprocessor derivatives.
3. **util.h and util.c:** Put all the common code here which you think may be leveraged independent of the problem you are trying to solve.
4. **Makefile:** This file is used to compile the code.
5. **custMakefile:** The input file to the program. This will be tested.
6. **README:** This is the file that describes everything about your code that we should know apart from program logic. Include information like, author names, CSE Lab machine names on which this code was compiled and tested, code organization, how to compile, how to run and so on.

6 Few Do's

Please abide by these rules.

Technical:

1. Program must be written in C.
2. Make sure your program compiles using gcc compiler on any of the CSELab ubuntu machines. We will use the same development Environment. No points will be given if program does not compile.

Non-Technical

1. Assignment can be done in group of at most two head counts. Please include your names at the top in README file.
2. It is expected that you do not take any unfair advantage of any ambiguity in the problem statement. In that case, please confirm with Instructor or Teaching Asistants before you proceed with any approach. It might not make sense coming up with a solution that does not help hone your skills in the expected area.

7 Submission Guidelines

The code should be packaged and submitted as a tar(.gz) file. Name of the TAR file should be **csci4061pa1**. This TAR should contain a directory named **csci4061pa1**. This directory should contain files as described in section 4 above. Feel free to include any .c or .h files that you feel are necessary. Edit the Makefile accordingly.

8 Grading

Maximum you can get for this assignment is 110%. Point distribution is as follows:

1. Following Submission Instructions and a sensible README:	5 points
2. Documentation and style:	15 points
3. Functionality and Error Handling:	80 points
4. Extra Credit:	10 points

References

[1] GNU make, `http://www.gnu.org/software/make/`

[2] White space, `http://unicode.org/cldr/utility/character.jsp?a=0020`

[3] string.h, `http://www.cplusplus.com/reference/clibrary/cstring/`