

## Project No. 1: *Mymake*, a Simplified Make Program

### PURPOSE

- Practice programming with the OS interface API (UNIX system calls).

### DESCRIPTION

This is a group project for groups of two people. In this project, you will implement a simplified make program, *mymake*, that realizes a subset of the functionality supported by the UNIX make utility. *Mymake* organizes a project by performing operations based on the dependencies specified by target rules in a makefile. It only supports simplified target rules with one target and zero, one, or more prerequisites of the format:

```
Target : [prerequisite] [prerequisite] ...
<tab>command1
<tab>command2
...
```

*Mymake* should have exactly the same behavior as the UNIX make utility when the makefile only has such target rules.<sup>1</sup>

- Prerequisites must be made before the target can be made.
- Assuming that all prerequisites have been made, the commands associated with a rule must **only** be executed when at least one of the following conditions is met:
  1. There is no prerequisite in the rule.
  2. Some commands have been executed to make some prerequisites.
  3. The target is not a file.
  4. The last modified timestamp on the target (target is a file) is older than the last modified timestamps of some prerequisites or some prerequisites are not files.
- If there is no rule to make a target (or a prerequisite) and the target is not a file, an error message should be printed and the program stops (use the error message that *make* produces in this situation).
- If there is no rule to make a target, but the target is a file, *mymake* should keep going, trying to make other targets.
- If no command has been executed, but *mymake* completes normally, the program should report that “Nothing to be done for 'xxx'”, where xxx is the target specified in the command line.

*Mymake* displays each command before running it. The commands associated with a target rule to be supported by *mymake* are in one of the following forms:

- a simple command with/without command line arguments.

---

<sup>1</sup>If there are any conflicts between the project description and the behavior of the UNIX make utility on linprog, you should follow the behavior of the UNIX make utility, report the error in the description, and get 5 extra points per error reported (the first person who reports the error gets the extra points).

- multiple commands with/without arguments separated by ';'. These commands are to be executed sequentially in order.
- the 'cd' command (making sense only in multiple commands, see the behavior of 'cd' with the UNIX make utility).
- multiple piped commands: *cmd1* | *cmd2* | *cmd3* | *cmd4*.
- a command to be executed in background ('&').
- a command with redirected I/O ('>' and '<').

*Mymake* does not need to handle combinations of the forms. *Mymake* should also have the following features:

- The program should behave like *make*: checking all dependencies (and the timestamps of the related files) in the makefile and performing operations only when needed.
- The default makefiles are *mymake1*, *mymake2*, and *mymake3* in the current directory in order. The default can be overwritten with an "-f" flag (like *make*). You only need to support zero or one target in the command line. The default target to be made is the first rule in the makefile.
- The makefile should allow comments: everything that follows character '#' is a comment.
- The program should stop executing when the execution of some command failed (non zero exit code).
- The paths to search for the commands in a relative path are stored in an environment variable MYMAKEPATH, which has the same format as the *PATH* variable in *tcsch*.
- The program should allow circular dependencies in the makefile.
- You should use *exec* family system calls to run commands. The *system* routine should **not** be used at all in your final program.

## DEADLINES AND MATERIALS TO BE HANDED IN

Due date: **Feb. 5**. You should clean and tar your project directory and send it to Mr. Chi Zhang (czhang@cs.fsu.edu). Please send the tar file as an attachment. In the directory, you should have all the needed files to create the *mymake* executable. Typing 'make' in the directory should create the executable. You should also have a README file describing how to compile and run your program and any known bugs in your program.

You need to hand in a hard-copy of your programs, makefiles and the README file, and demo the project with the Mr. Zhang in the due date.

## GRADING POLICY

A program with compiling errors will get 0 point. A program that cannot run any command in the makefile will get 0 point. A program with any 'system' routine in it will at most get 50 points (the 'system' routine is NOT allowed in the project).

1. proper README file (5)
2. proper makefile file (for compiling your simplified make program) (5)
3. using the correct default makefile, the “-f” flag (5)
4. running one rule with one simple command (20)
5. allowing multiple commands in a line (';') (5)
6. supporting the 'cd' command (the directory is changed only for the execution of one command line) (5)
7. running multiple piped commands ('|') (10)
8. allowing background execution (5)
9. allowing I/O redirection in a command ('>' and '<') (10)
10. Searching each relative path with paths in MYMAKEPATH (10)
11. supporting comments (#) (5)
12. supporting simple dependencies with multiple rules (5)
13. supporting advanced dependencies with multiple rules (5)
14. Overall similar behavior as the UNIX make (5)
15. 15 points for the first unknown bug and 10 points for the first known bug/unimplemented feature.

## MISCELLANEOUS

- The implementation of this program will likely be around 1000 lines of dense code (500- for parsing and dependency checking and 500+ for executing commands in different forms). Please start the project as early as you can.
- It is normal for system calls to have unexpected behavior. You can either call it a bug or a feature. As a result, you might need to work your way around if that occurs to your program — this is part of the project.
- You can make any assumptions of the makefile format as long as you can handle all sample makefiles.
- There are mainly three modules in this program (1) the module to load the makefile into some internal data structures that can be easily manipulated, (2) the module to check the dependencies and act accordingly (this is likely a recursive routine), and (3) the execution module that runs one line of commands at a time. You can implement the system in the order (1) → (2) → (3) or (1) → (3) → (2). Using the order (1) → (3) → (2) will get you more points if you cannot finish the whole project.
- All programs submitted will be checked by a software plagiarism detection tool.