Unit home

Lecture & Workshop recordings on LMS

help2002

Schedule

C textbooks

OS textbooks

Information resources

Extra reading

Past projects

Recent feedback

Working effectively

Project 1 2022

The goal of this project is to develop a system utility program, named *estimatecron*, which checks the correctness of a simplified *crontab-file*, and determines some simple metrics about the set of commands specified in the file.

Successful completion of the project will enhance your understanding of core features of the C11 programming language, functions from the C11 standard library, and your operating system's system-calls; and familiarise yourself with online systems' documentation.

Introduction

Most Unix-based systems, such as Linux and macOS, employ a command named *cron* which supports the scheduling and execution of *non-interactive*, often long-running, commands. The *cron* process 'wakes-up' every minute, reads from a text file termed a *crontab-file*, and commences the execution of any commands that should run at that time. *cron* is often used to schedule backups at times when a machine will otherwise be idle, and to support a list of task requests, a *batch queue*, on supercomputer and cloud-computing platforms.

(Do not go into too much depth, but you can read about the standard *cron* command from *section 8* of your system's online manual, and read about the standard *crontab* format from *section 5* of your system's online manual.)

For this project we will use the information in two text files, termed the *crontab-file* and the *estimates-file*, to anticipate the required computing load on a system, based on the execution time of previously executed commands.

A typical invocation of your project will be:

```
prompt> ./estimatecron  month  crontab-file  estimates-file
```

which specifies the month (of the current year) to be analysed (either an integer 0..11, or a 3-character name), and the name of the two required text files (you may choose *any* names for the files).

Your *estimatecron* program should check the validity of the information in its input files (reporting any errors), simulate (better word?) the execution of commands invoked during the requested month, and report the values of: the name of the most frequently executed command (a single word), the total number of commands invoked (a non-negative integer), and the maximum number of commands running at any time (a non-negative integer).

File formats

Your *estimatecron* program should read information from 2 text files - a simplified *crontab-file* which specifies at which times commands should be executed, and an *estimates-file* which specifies the average execution times (in minutes) of commands.

The *crontab-file* consists of 2 types of lines:

- *comment lines* are lines whose first non-whitespace (space ' ', or tab '\t') character is a '#' (the hash character). They are used to document the file or to 'comment out' other unwanted lines. Their contents is ignored.

- *command lines* consist of at least 6 'words' separated by one-or-more whitespace characters. The first 5 words contain, in order, the minute (0..59), the hour (0..23), the day of the month (1..31), the month (0..11), and the day of the week (0=sun...6=sat) when a command should be invoked. As a special case, any of the first 5 words may be just a '*' (an asterisk), indicating that all possible (valid) values are being specified. The words for the month and the day of the week, may be 3-letter lowercase abbreviations of their possible values, such as *jan*, *feb*, *mar*... and *sun*, *mon*, *tue*... The 6th word provides the name of the command to be executed.

  A representative *crontab-file* may include the lines:

```
# execute a command at 3AM every day
0 3 * * *      daily-backup
#
# execute a command at 4:15AM every Sunday
```

```
15 4 * * sun   weekly-backup
#
# start thinking about the project....
0 10 22 7 mon  deep-thought
#
# submit my project automatically, just in case I forget!
59 16 16 8 *   submit-project
#
# mail out a monthly newsletter
0 2 1 * *      send-monthly-newsletter
```

The *estimates-file* consists of 2 types of lines:

- *comment lines* are lines whose first non-whitespace character is a '#' (the hash character). They are used to document the file or to 'comment out' other unwanted lines. Their contents is ignored.

- *estimate lines* consist of exactly 2 'words' separated by one-or-more whitespace characters. The first word specifies the name of a command (that will possibly appear in the *crontab-file*). The second word specifies the *anticipated* number of minutes that the command will execute for. These estimates have been obtained from months of previous command invocations, and your project is not required to generate 'real' values for this file.

  A representative *estimates-file* may include the lines:

```
# format:  command-name   minutes
weekly-backup             6
daily-backup              2
send-monthly-newsletter   10
submit-project            1
deep-thought              600
```

Project requirements and fine-print

1. The project may be completed **individually or in teams of two** (but not teams of three). The choice of project partners is up to you - you will not be automatically assigned a project partner.

2. Your project should be written in the C11 programming language, in a single source-code file named *estimatecron.c*

3. The *crontab-file* and the *estimates-file* will each contain at most 20 non-comment lines. Each line will be at most 100 characters long, and each command name will be at most 40 characters long.

4. Your project should check for data errors in both the *crontab-file* and the *estimates-file*. If any errors are found, only the *first* error should be reported and the program should terminate.

5. Every command name appearing in the *crontab-file* will be also appear in the *estimates-file*.

6. Your project should simulate (better word?) the execution of the commands specified in the *crontab-file*, for the month specified on the command-line. *There is no need to consider leap-years (or leap seconds!)*.

7. At most 20 processes will ever be executing at the same time (during the same minute), and multiple instances of the same command may run at the same time.

8. Your project only needs to simulate the execution of processes for the requested month (of the current year). All processes will have terminated before the end of the requested month.

9. Your project should not attempt to execute any of the commands in the *crontab-file* (using, for example, *fork()* and *exec()*).

10. Your submitted project may produce *any* output you wish, typically debug output. It does not need to describe what it is during. Only the *very last line* will be considered during (automated) marking. It must provide 3 words separated by one-or-more whitespace characters:

    - the name of the most frequently executed command (a single word),
    - the total number of commands invoked (a non-negative integer), and
    - the maximum number of commands running at any time (a non-negative integer).

11. A command whose estimated running time is, say, 1 minute, will have terminated before new commands are considered at the beginning of the *next* minute (imagine they run for at most 59 seconds).

12. The project can be successfully completed without using any dynamic memory allocation in C (such as with *malloc()*). You may choose to use dynamic memory allocation, but will not receive additional marks for doing so.

13. Your project should not depend upon any libraries (such as 3rd-party downloaded from the internet) other than your system's standard library (providing OS, C, and POSIX functions).

Assessment

The project is due **5:00pm Friday 16th September (end of week 7)**. and is worth **25% of your final mark** for CITS2002. It will be marked out of 50.

- 25 of the possible 50 marks will come from the correctness of your solution (**automated marking**). Correctness includes checking the validity of input data, and reporting the correct final results.

- 25 of the possible 50 marks will come from assessing your programming style, including your use of meaningful comments; well chosen identifier names; appropriate choice of basic data-structures, data-types and functions; and appropriate choice of control-flow constructs (**manual marking**).

During the marking, attention will obviously be given to the correctness of your solution. However, a correct and efficient solution should not be considered as the perfect, nor necessarily desirable, form of solution. Preference will be given to well presented, well documented solutions that use the appropriate features of the language to complete tasks in an easy to understand and easy to follow manner. That is, do not expect to receive full marks for your project simply because it works correctly. Remember, a computer program should not only convey a message to the computer, but also to other human programmers.

Good luck!

Chris McDonald.

The University of Western Australia

Computer Science and Software Engineering

CRICOS Code: 00126G

Presented by Chris.McDonald@uwa.edu.au