Module 4 Assignment (Homework Project #2)

System Development in the Unix Environment (605.614)

Overview

The second assignment for this class is to build a log manager library that can be used by multiple programs concurrently. Additionally, you will use the build hierarchy built in the first homework to build the log manager library and provided programs that make use of the log manager library.

Getting Started

Be sure to review the module content, particularly the 'log' example provided in a video and in the class notes directory. Then copy the Makefile hierarchy from your first homework assignment into ~you/614/homework2. Also, assure you have fixed the issues reported to you from the first homework assignment as soon as they are available.

Objectives

After completion of the project, you will be able to:

- Create files in a program that can be properly written to concurrently by multiple processes.
- Build functions using a variable number of arguments.
- Get current time and properly format it in a string that will be part of a log message.

Submission

Completion of the assignment involves completing, on or before the due date, the following materials:

1. The directory hierarchy and Makefiles along with the required log manager library and program source code.

2. A comment block at the beginning of each program and library source file shall include the following information: the student, the course, the name of the file, and a short description of the file's purpose.

3. The Assignment should be compiled and run on the (UNIX) system (dev4.jhuep.com) and Linux system (absaroka.jhuep.com). The executable programs must be installed in the structure mandated in the last section of this document.

The fundamental rules governing the assignment (see the Syllabus) are:

- The programs must be written in the 'C' or the 'C++' programming language.
- No copying or plagiarism. If copying or plagiarism occurs, the person or persons involved will receive no credit. They will be treated as not having submitted the assignment.
- Assignments are due by midnight of the due date stated above.
- Extensions are only granted if arrangements are made with the instructor well in advance of the due date of the assignment. No last minute extensions will be given; if a student has not completed the assignment by the due date, the student has the choice of receiving a grade based on the work completed to that point, or receiving a 5 point penalty for each day (or part of a day) it takes to complete the assignment. In these cases, the assignment is not considered complete until the instructor is notified by the student of the completion of the assignment.

To submit your assignment, you will run the Perl/Python script developed as part of the first assignment (with relevant fixes if needed). The output of the Perl/Python script will be used by the grader. The grader will copy those files to a separate directory for grading.

To submit your homework, you will run the Perl or Python script developed as part of this assignment. The output of the script will be used by the grader. You should ensure that the class group has at least read permission on the files. The grader will copy those files to a separate directory for grading.

NOTE: Your assignment is not considered complete until you submit the assignment in BlackBoard. You will not need to include any source or executables in BlackBoard, but this notifies the grader that your assignment tar files are complete and ready for grading.

Grading of the Assignment

The assignment will be graded according to the following criteria:

1. The executable programs must work correctly.

2. The source for the executables contained in your authored code must be well-written and follow the style guide.

3. The project hierarchy must be as specified.

4. Makefiles must exist in the appropriate directories and must function properly, supporting all targets as specified below.

The Assignment will be graded based on 100 points as follows:

The directory hierarchy/Makefiles/Perl or Python delivery script	20 points
The correctness of the programs	20 points
The log manager library	60 points

Assignment Requirements

Library Descriptions

In this assignment, a *log manager* library will be created. This log manager library will enable programs, perhaps running concurrently, to write time-tagged text messages to a default or specified log file. This library is described below.

The log manager library

The log manager library should be built from a single source file, log_mgr.c. This file will contain three functions, **log_event()**, **set_logfile()** and **close_logfile()**. The declarations of these functions should appear as follows:

int **log_event (*Levels l, const char *fmt, ...*);**
int **set_logfile (*const char *logfile_name*)**;
void **close_logfile (*void*);**

These declarations should be contained in an appropriate ``include" file (residing in ~/614/homework2/include) called log_mgr.h.

*int log_event (Levels l, const char *fmt, …)*

The first parameter of log_event, ***Levels l***, is a variable of the enumerated type *Levels*. (See K&R, 2nd Ed, page 39). The definition of *Levels* shall be as follows:

typedef enum {INFO, WARNING, FATAL} Levels;

and this definition should be included in the same include file as the function declarations above.

The second parameter of log_event, ***const char *fmt***, is a character pointer to a string containing a format specification for the log messages. This format string is similar in function to the format string in the **printf()** family of standard output routines.

The strange looking third parameter "...", is the ANSI-C syntax which indicates that the parameters following the ***fmt*** parameter may vary in both their type and number. See K&R, page 155-156 for a discussion of variable length argument lists. Additionally, it is

recommended to make use of the **vsprintf()** function which takes a variable of type va_list as its last argument. (See the manual page for vsprintf.)

The function **log_event( )** should return **OK** (0) upon successful return and **ERROR** (-1) otherwise.

The **log_event( )** function will take the argument information, format it into a time-tagged line of text, and append it the current log file. No text messages should appear garbled in the log_file, and regardless of the number of processes that are concurrently using the log library, no messages should be lost upon successful execution of the **log_event( )** function.

*int set_logfile(const char *logfile_name)*

The second function, **set_logfile(*const char *logfile_name*)**, allows the user to change the file used for the logging of messages for a particular process. It should not be required to call this function before calling **log_event()**; if **log_event()** is called before **set_logfile()** is invoked, a default log file, defined in the include file log_mgr.h (in ~/614/homework2/include), shall be used. The default log file shall be called *logfile* (in the current working directory). ***logfile_name*** is the name of the new logfile; if not an absolute pathname, the logfile will be opened in the current directory of the executing process. For example, **set_logfile (**"*newlog*"**)** would use a log file called *newlog* in the current working directory of the process; however, **set_logfile (**"*/tmp/logfile*"**)** would use the file *logfile* in the /tmp directory.

When **set_logfile()** is called, the argument ***logfile_name*** is first opened (or created if necessary); if that open is successful, the previous log file is closed, and the new log file is then used for subsequent **log_event( )** calls. If the open of the new log file fails, the previous log file should remain open. No information that was in the new logfile prior to the call to **set_logfile()** should be lost.

This function will return **OK** (0) if the new log file is opened successfully; **ERROR** (-1) otherwise.

void close_logfile(void)

This function shall be called whenever a logfile is to be closed. At a minimum, this function should close the file descriptor associated with an open logfile, if necessary.

The format of the log file

The log file shall consist of lines of text; each line will be formatted as follows:
<date & time>:<Level>:<Formatted line of text>
Each line of the log file will be terminated by a single newline. The date & time field shall contain, at a minimum, the month (Jan-Dec), the day of the month, and the time in

HH:MM:SS (hours, minutes and seconds). Day of the week, the year, and the timezone is optional. An example is provided below.

Program Descriptions

This assignment shall include some simple programs designed to exercise your log manager library. You are not to write these programs; you will find them in /home/unix_class/log_mgr_tests. You should copy these programs, build them, and test them to assure your library works as required.

test_log.c

This program is provided to assure log messages are being inserted into the log file in the proper order and format. When you test your library, make sure you run concurrent versions of test_log (e.g. test_log & test_log), and make sure all messages appear in the logfile. You should assure that this program compiles with no warnings or errors.

You should place a copy of this file in ~/614/homework2/src/bin/test_log, and provide a makefile to build it.

test_log2.c

This program is provided to test if the handling of the logfiles is done properly.

You should place a copy of this file in ~/614/homework2/src/bin/test_log2, and provide a makefile to build it.

lots_of_logs.c

This program is provided send lots of log messages quickly.

You should place a copy of this file in ~/614/homework2/src/bin/lots_of_logs, and provide a makefile to build it.

Other test programs

The student is encouraged to create other test programs to verify the correctness of the log message library. These programs will not be graded, but should assure the student that the log_mgr library works properly under concurrent use by multiple processes. Among other areas, the student should assure that messages are not lost and that messages are not garbled.

An Example

Suppose a program contains the following call to log_event:

log_event (WARNING, "File %s does not exist for process %d", filename, getpid());

Successful execution of this function would result in the following string to be appended to the current log file:

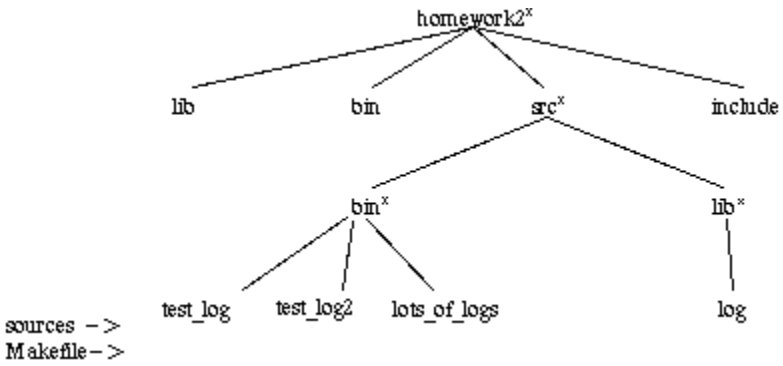
```
Sep 23 9:16:36 EST 2020:WARNING:File foobar does not exist for
process 7832
```

assuming that the filename pointer pointed to "foobar" and the process id of the executing process is 7832.

Assignment Implementation

The design and implementation of the assignment shall follow these guidelines:

- The assignment shall consist of the three programs and one library organized as described in this document. There shall be a minimum of one header (include) files that shall reside in 614/homework2/include in your home directory.
- The following directory structure shall be enforced:
- In the 614 directory, there will be a directory called "homework2."
- The directory hierarchy under this directory shall look as follows:



- The directory "homework2/lib" will contain project-specific libraries that you shall create (liblog_mgr.a).
- The directory "homework2/bin" will contain project-specific programs that you shall create or copy (test_log, test_log2, and lots_of_logs).
- The directory "homework2/src" will contain the source code for the project, organized as shown (Note: the directory 'log' above should be called log_mgr).
- The source for each program will be in the deepest directories of the above hierarchy. Each of these directories will contain a Makefile; the function of the

Makefile will allow the UNIX command ``make'' to make the executable (or the library) and install it in the appropriate installation directory (either homework2/bin or homework2/lib).

- In addition, each directory marked with an (*) above should contain a Makefile; these Makefiles should allow 'make' to be run in these directories, the effect being to run make with the specified argument in each of the directories underneath.

- All Makefiles should be called ``Makefile'' and should contain support for the following targets at a minimum:
    - it
    - install
    - depend
    - clean

- The directory ``homework2/include'' will contain project-specific include files (include files that need to be shared among distinct programs).