Operating System Architecture Course Project

Rahim Sharifov
Sara Shamilova
Sabina Hajimuradova

French - Azerbaijani University

March 22, 2019

Objective

Objective of this project is to create program named detecter to periodically launch a program and detect state modifications. Program is of following structure.

```
./detecter [-l limit] [-i interval] [-t format ] [-c] command arg1 arg2
```

-l: number of times that is given program should be executed.
-i: number of millisecond that program should sleep between two consecutive launch.
-t: time format to print local time after each execution
-c: to print exit code that is returned by program.

All options except -c expects argument. If no argument is given to those that requires argument, program will fail. All arguments has its range, limit must be non-negative integer, it cannot be zero too. Interval is positive integer.Time format should known by function strftime. One can find all known format options in manual page of strftime.

Giving option is not mandatory, if no option is given to the program, command will be executed infinitely, interval between two consecutive execution will be 10000 milliseconds (10 seconds), no time and no exit code will be printed.

Structure of giving command to the program to be executed is the same as writing it in command line. Command also can take arguments, options and etc.

Abstract

What program does is that it takes command and all options from command line. Executes command, prints its output in display (if there is any) , prints local time (if -t option is given) , prints exit code of command (if -c option is given), sleeps for given interval (or 10 seconds if no interval is given) ,then it executes command again, if the output is the same as previous output,it does not print it in display again, if output is different than previous one program prints in display. It goes like this, until it reaches the limit (Or goes infinitely if no limit is given).

Manipulation

Our project consist of 3 source file. Deteclib.h , Deteclib.c and detecter.c . DetecLib.h is header file that contains decleration of functions and constants. DetecLib.c is where those functions are defined. And detecter.c is main source file that contains main function.

MakeFile

Simple MakeFile is provided in tar file which simply compiles sources files with code coverage options

After compiling and executing program. Run "make gcov" to get code coverage files

Run "make clean" to remove temporary, log , gcov files, Also executable detecter program.

detecter.c

Contains main method that simply takes arguments from command line, arranges them, and call a method that is defined in Deteclib.c to manipulate them.

Getopt() function is used to get options and its arguments. As it is mentioned before that getopt() function is not POSIXLY_CORRECT that is, function will confuse if given command has the same option as program. For example:

```
./detecter  -l 4  -i 1000  -t "%H:%M:%S"  -c  ls -l
```

So as command ls also has -l option, getopt() will mix them up.

POSIX demands the following behavior: the first non-option stops option processing. This mode is selected by putting '+' sign at the beginning of the options argument string. So function will stop when encounters non-option argument which is "ls".

In a while loop getopt() goes through arguments to find options until non-options argument is encountered. When it finds an option, switch case checks with option it is and relevant manipulation is done. (Line 39-40)

Optarg, Optind that we used are globally defined variables used by getopt. Optarg indicates argument of given option that getopt has found. Optind is the index of the next element of argument list.

In the example above, firstly getopt takes -l option. As optarg indicates argument in front of this obtion , we assing to the variable ”limit” which is initilized to -1 by default. Argument that is taken from command line is character, atoi() function converts it to integer. Value of limit must be positive. So program checks whether it is positive or not, exits if it is negative. (Line 57-61)

Afterwards getopt takes -i option. Assigning value of interval is the same as limit.It cannot be negative or zero either. As usleep() function takes argument in microseconds and we are required to take it in milliseconds from command line,so we multiply it by 1000. (Line 45)

For -t option it simply assigns its argument to the variable ”format” which by default initialized to NULL. (Line 42)

-c option does not require any argument. By giving this option program simply changes state of variable EXIT which by default 0. Later on program will decide whether print exit code of command by checking state of this variable. (Line 66)

As we mentioned before getopt() function stops when it encounters non-option argument. But variable optind still indicates index of next argument. So by using this variable we assign command and all of its arguments to newly defined variable args as array of strings. (Line 81-82)

Function launch() that is defined in DetecLib.c, takes all data that is taken from command line and produces expected output.

DetecLib.c

DetecLib.c is a source file where all work is done. It defines all functions that is declared in header file DetecLib.h.

As it was mentioned,using intermediary file is restricted.Thus, we used array of characters to store output of execution of command to check later whether output has changed or not.

OLD_OTPT, NEW_OTPT is globally declared variable which we use to store output. NEW_OTPT is temporary buffer that stores current output, OLD_OTPT is stable buffer that contains last modified output.NEW_SIZE, OLD_SIZE are respectively sizes of these buffers. (Line 22-26)

launch()

Launch() function is called from main function.First, it allocates memory for globally declared variable OLD_OTPT with size MAX which is defined in header file as macro and expands to 128. To use memory more efficiently we firstly allocated buffer with this small size, later on with bigger size of output buffer will be reallocated again, this way memory usage is minimized.

Then, in a loop program calls:
launch_process() which executes command and prints exit code if needed.
get_time() which prints local time relevant to time format.
usleep(), program sleeps for given interval.
The same process is done until limit is reached. By default limit is -1 , so if no limit is given in command line loop goes infinitely. And after loop ended program frees memory allocation. (Line 130-138)

launch_process()

Function launch_process() is called first in function launch() which takes array of strings that are command and its arguments, and variable EXIT whether print exit code or not. To execute command we used function execvp(). As process ends right after execvp() executes the command, another process is needed to continue program and to print its output.Thus, we used fork() to create two process and pipe for communication between processes.

First, temporary buffer NEW_OTPT is allocated with size MAX (128) to store output.This buffer stores current output and though it has initialized with small size but it will be reallocated as bigger output comes. At the end of function, it will be freed. (Line 79)

Program creates two processes using fork(). Child process to execute given command and parent process to do rest. (Line 81)

Child process executes command and ends right after. And pipe is used to send its output to parent process.As child process only sends output,so we close input of pipe in child process, as we dont need it. And we duplicate standart output which is 1 to output side of pipe to make sure that output goes to parent process. execvp() executes the command and after execution output goes to output side of pipe as we duplicate standart output to it. (Line 85-90)

Parent process receives output from pipe and manipulates it. So we close output side of pipe as parent process only receives input. And duplicate standart input to input side of pipe , so both them refers to the same place.So reading from standart input is the same as reading from pipe.

Program reads output of execution of command from pipe in while loop. Each time it reads MAX size of bytes from pipe and adds number of bytes it read to NEW_SIZE. If number of bytes it read is equal to MAX , that means there is more to read in pipe,and program reallocates buffer with NEW_SIZE, and program reads again the same number of bytes and stores these bytes to the end of buffer (NEW_OTPT+NEW_SIZE). So it goes like this until all bytes is read from pipe. So output is stored in buffer NEW_OTPT and variable NEW_SIZE represent its size. (Line 99-103)

Afterwards, program compares old output with new one using function check().
What check() function does is that it firstly compares sizes of these buffers. If size of output has changed, output obviously has changed.If size has changed, buffer for old output is reallocated with new size, as program is going to equalize new output to old one, because it has changed.Then it will print it.(Line 55-60)
If not , there is a small chance that size has not change but content has changed, so function also checks it character by character. If any difference is detected function returns 1, if not function returns 0. (Line 60-67)

So if check() function returns 0, that means no difference is detected,thus program skips 'if' block and just frees temporary buffer and sets its size to zero.But if function returns 1, that means new output and old output is not the same.So function equalize() simply copies content of new output to old) output. (Line 108-110)

What equalize() function does is that it simply takes two array of characters and copies it character by character,because they will always have the same size. Then, write function prints content of buffer OLD_OTPT to standart output. (Line 30-33)

As new output is displayed, program also checks state of variable EXIT whether print exit code or not. If condition is true, program prints exit code using using function WEXITSTATUS.

At the end after printing output, function frees temporary array and sets its size to zero.

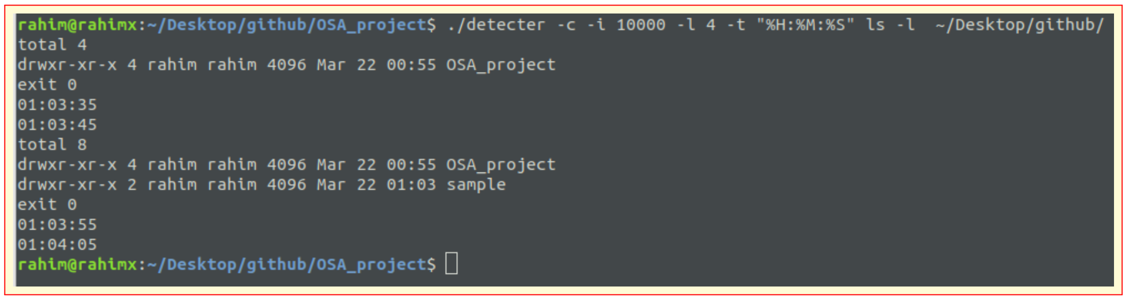
get_time()

After calling function launch_process(), function launch checks time format if it is null or not. If not it calls function get_time() which is supposed to print local time time in given format.

It simply calls function time() to get seconds since epoch (1970). And function localtime() takes these seconds and returns local time inside type struct tm. Function strftime() simply takes this local time, returns string of local time in given format. (Line 38-50)

Conclusion

In conclusion, program detecter launches given command for given number of times, detects modification of output, and print local time and exit code according to given options. It fulfills all the requirements and meets all demands of test cases. Simple example of program is below:



```
rahim@rahimx:~/Desktop/github/OSA_project$ ./detecter -c -i 10000 -l 4 -t "%H:%M:%S" ls -l  ~/Desktop/github/
total 4
drwxr-xr-x 4 rahim rahim 4096 Mar 22 00:55 OSA_project
exit 0
01:03:35
01:03:45
total 8
drwxr-xr-x 4 rahim rahim 4096 Mar 22 00:55 OSA_project
drwxr-xr-x 2 rahim rahim 4096 Mar 22 01:03 sample
exit 0
01:03:55
01:04:05
rahim@rahimx:~/Desktop/github/OSA_project$
```

Figure 1: Example Output.