Lab 3: Multi-Processing and Signal Handling – mproc_crypt

Please read this entire assignment. Take some notes. Think about it some. Take some more notes. Look for answers to your questions in this document.

This document is a work-in-progress. The DRAFT watermark will be removed when done.

Due: As shown in Canvas. Submit a single tar.gz file.

Do not place ANY directories in your submitted tar file. I will not change into any sub-directories to hunt down your source files. When you create your `tar.gz` file to submit, do it within the directory where you created the source files, **NOT from a higher level directory. If I cannot find your source files in the same directory where I extract your submitted tar file, I will simply give you a zero on the assignment**. Submit a single `tar.gz` file to Canvas for this assignment. If you need help creating a tar.gz file, read the material about `Makefile` or look at the example `Makefile` in my `src` directory.

In this assignment, you will be working with UNIX fork() and crypt() calls. You'll also get better with signal handling. **I urge you to not delay beginning it**.

Part 1 – Makefile (50 points)

Write a **`Makefile`** to build the **`mproc_crypt`**.

You must have a single `Makefile` for this lab. Your `Makefile` must contain (at least) the following targets:

Target	Action
`all`	Builds all dependent C code modules for your applications in the directory. This should be the default target in your `Makefile`.
`mproc_crypt`	Builds all dependent C code modules for the `mproc_crypt` application in the directory. The dependency of `mproc_crypt` should be `mproc_crypt.o`.
`mproc_crypt.o`	Compiles the `mproc_crypt.c` module for the `mproc_crypt` application in the directory, based off of any changed dependent modules. The dependency for `mproc_crypt.o` should be `mproc_crypt.c`. and **`mproc_crypt.h`**.
`clean`	Deletes all executable programs, object files (files ending in `.o` produced by `gcc`), and any editor chaff (# files from `vi` and ~

	files from **emacs**). Make sure you use this before you bundle all your files together for submission.

When I build your assignment, I should be able to just type

```
make clean
make clean all
```

to have it completely clean the directory and build `mproc_crypt`.

I also recommend that you **use some form of revision control on your source files**. Not only does this reduce the possibility of catastrophic file loss, but it is a LOT better than making `.BAK1`, `.BAK17`, `.BAK4c` copies of your code. Putting this into your `Makefile`, as described in the notes about `make` would make your life better.

You must compile your program using the following flags for `gcc`. Putting these flags into the `Makefile` with the `CFLAGS` variable will make your life better. When compiled with the flags, the `gcc` compiler should emit no errors or warnings. **Any warnings from the compiler is an automatic 20% deduction**.

```
-Wall -Wextra -Wshadow -Wunreachable-code -Wredundant-decls
-Wmissing-declarations -Wold-style-definition
-Wmissing-prototypes -Wdeclaration-after-statement
-Wno-return-local-addr -Wunsafe-loop-optimizations
-Wuninitialized -Werror
```

Part 2 – `mproc_crypt` (350 points)

The `mproc_crypt` program is designed to use a dictionary attack to crack passwords. The real objectives for the class are to gain experience using `fork()` and the use of signal handlers. The mproc_crypt program is a multi-process application where each process is attempting to crack passwords using a dictionary attack.

When using the crypt(3) functions, some hashed passwords may take a very long time to test. Rather than wait for a very long time for each call to crypt(), we want to have the call timeout so that we can skip the very long running calls to crypt() and move on. However, the crypt() functions do not have a timeout parameter. In order to get the program behavior, we want (the timeout), we must set an alarm that goes off. The signal handler for the alarm exits the process with a specific exit value (EXIT_CHILD_TIMEOUT). We have the desired timeout behavior, though it is ugly.

The parent process receives the SIGCHLD signal for the exited child process with exit value of EXIT_CHILD_TIMEOUT and starts a new child process to continue working to crack remaining passwords.

Command line options:

Option	Short description	Longer description
`-p passwords file name`	hashed passwords file	**REQUIRED**: The name of the file **containing hashed passwords**.
`-d dictionary file name`	plain text passwords file	**REQUIRED**: The name of the file containing **plain text passwords**.
`-o output file name`	send output to the given file name	The name of the **output file**. This will contain a list of all the cracked, failed, or timeout hash passwords.
`-P #`	number of child processes to create	The number of child processes to maintain. **If this option is not given, the default is 1 child process**.
`-T #`	number seconds to wait for a timeout	The number of seconds to wait for the **crypt()** function to return. **If this option is not given, the default is 5 seconds.**
`-N #`	increment for **nice()**	A value to add to the nice value for the process. This should be between 1 and 19.
`-v`	verbose output, to stderr	Supplemental output to stderr.
`-h`	this marvelous help	This most helpful text.

An code example use of the crypt() function can be found in the Lab3 directory: uncrypt_example.c. It demonstrates how to make a call to crypt() with an existing hashed password and a potential dictionary word. To try out the code, run the following command:

```
./uncrypt_example < key10.txt
```

The `key10.txt` file contains 10 plain-text-password: hashed-password examples.

The parent process is responsible for starting the right number of child process (as given on the command line with the -P # option or the default of 1). If a child process exits with a timeout, the parent process creates a new child process to continue cracking passowrds.

Functions used in my code:

```
getopt()
sprintf()
sscanf()
fprintf()
exit()
strtol()
fopen()
nice()
signal() or sigaction()
sem_wait()
sem_post()
sem_init()
shm_open()
shm_unlink()
ftruncate()
mmap()
munmap()
open()
close()
fstat()
malloc()
free()
memset()
read()
strtok()
fork()
atexit()
alarm()
crypt()
strcmp()
kill()
wait()
waitpid()
```

Include files in my code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <time.h>
#include <unistd.h>
#include <stdbool.h>
#include <sys/wait.h>
#include <errno.h>
#include <libgen.h>
#include <crypt.h>
```

```
#include <semaphore.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "mproc_crypt.h"
```

Libraries used with my code:

```
-lcrypt
-lrt
```

Final note

The labs in this course are intended to give you basic skills. In later labs, we assume that you have mastered the skills introduced in earlier labs. **If you don't understand, ask questions.**