Homework 2

As part of this homework assignment, you will get a better understanding of how to create processes and wait for processes to complete. These basic capabilities allow you to run multiple tasks in parallel and take advantage of the multicore capabilities of modern computers.

Manipulating Command Line Arguments

To get us started, you will have to write a simple program in print_args.c which prints the command line arguments that it is invoked with. You have already learned how to pass and process command line arguments in the previous homework.

Manipulating the Environment

Next, you will have to write a program print_env.c which prints the names and values of each environment variable. The output should be formatted such that on each line the name and value of a single variable are printed. An example of this format is shown below:

```
TERM_PROGRAM=Apple_Terminal
SHELL=/bin/zsh
TERM=xterm-256color
TMPDIR=/var/folders/8y/7qxbfzr90v79bk1b737tzb553ryznd/T/
TERM_PROGRAM_VERSION=443
TERM_SESSION_ID=CD96D8C1-BFBD-4889-A728-4891EF03C260
USER=ochipara
```

Hint: The following web page will be helpful https://man7.org/linux/man-pages/man7/environ.7.html.

Compile Script

You will have to write a script compile.sh (in bash or sh) that will compile the above two programs and run them if the compilation is successful. The script should not compile any of the subsequent programs.

Computing the CRC of files

In the following, you will have to write a program that is passed from the command line a list of files. As a starting point, I provide the parallel_crc.c file. For each file supplied as an argument, the (parent) program will create a child process that will compute and print a CRC checksum of the file. The CRC is computed based on the contents of a file. I provide you the gencrc(uint8_t *data, size_t len) function which computes the CRC of the bytes stored in the data buffer whose length len bytes. The child process will output the name of the file, an equal sign, and then the computed crc. For example, for a file a.txt with crc equal to 183, the child will output a.txt=183. The parent process waits for the completion of all child processes before completing its execution.

Note that it is common for hashes to be used to validate that the content of files remains intact. Obviously, a CRC8 would not be effective for large files. There are better hash functions that computed larger hashes. However, CRC8 is commonly used in embedded systems and networks to check if small packets/files are corrupted.

A simple shell

For this problem, you will write a simple shell. When the shell starts, the user can input from the keyboard the path of an executable followed by several arguments. It is safe to assume that the input supplied by the user is less than 1024 bytes. Your shell will then execute the program. When the executed program completes, then the user can run other programs. If the user types in “quit”, then the program terminates.

STRACE

I captured the followed trace using strace (https://man7.org/linux/man-pages/man1/strace.1.html). The strace program is a utility that intercepts and records the system calls made by a program. You will be doing a bit of detective work to understand what the program is doing.

```
execve("/bin/echo", ["echo", "hello world from strace"], 0x7ffdda37a1b8 /* 11 vars */) = 0
brk(NULL)                               = 0x55f1421f7000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=13948, ...}) = 0
mmap(NULL, 13948, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fd4c0174000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20\35\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=2030928, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fd4c0172000
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7fd4bfb5e000
mprotect(0x7fd4bfd45000, 2097152, PROT_NONE) = 0
mmap(0x7fd4bff45000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7fd4bff45000
mmap(0x7fd4bff4b000, 15072, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fd4bff4b000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7fd4c0173540) = 0
mprotect(0x7fd4bff45000, 16384, PROT_READ) = 0
```

```
mprotect(0x55f141807000, 4096, PROT_READ) = 0
mprotect(0x7fd4c0178000, 4096, PROT_READ) = 0
munmap(0x7fd4c0174000, 13948)           = 0
brk(NULL)                               = 0x55f1421f7000
brk(0x55f142218000)                     = 0x55f142218000
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
write(1, "hello world from strace\n", 24hello world from strace
) = 24
close(1)                                = 0
close(2)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Given the above trace, answer the following questions:

- What was the program that was executed?
- What arguments were passed to the program?
- What are the file descriptors 1 and 2 that are used by the program?
- What happens when the program writes to file descriptor 1?
- Did the program output anything? If so, what was the output?