CS342 **Homework 1** Efe ACER 21602217

### 1)

The computer I use runs macOS Mojave 10.14.3 as its Operating System. Hence, I had to choose between allocating a portion of my hard-disk space to Ubuntu 18.04.1 LTS or to use it via a Virtual Machine. I chose the latter option and installed Oracle's VirtualBox 6.0.4. VirtualBox is essentially an efficient x86 and AMD64/Intel64 virtualization product. It is developed and regularly maintained by Oracle, so I found it reliable.

The installation process was very time-consuming and troublesome for me. I went through some video tutorials on the installation process; however, none of the recorded solutions worked in my computer. Thus, I tried the exact same steps in another macOS Mojave computer. Surprisingly, it worked.

The 10 Linux commands I learned from the Linux usage tutorials are as follows:

- `ls`: Shows all the major directories under a specified file system.
- `cd`: Lets the user to change between directories.
- `mv`: Lets the user to move a specified file to a particular directory.
- `mkdir`: Creates a new directory.
- `rmdir`: Removes an existing directory.
- `touch`: Creates a new (empty) file.
- `rm`: Removes an existing file.
- `clear`: Clears the screen and takes the user back to the initial prompt of the working directory.
- `locate`: Helps finding a particular file.
- `history`: Shows all commands used in the current session.

### 2)

(i) The kernel executable is named `vmlinuz` and is located under: `/boot`

(ii) Output of `uname -r` command (Version number): `4.15.0-29-generic`

### 3)

I downloaded the `4.14.99` version from `kernel.org`'s archieve, since it was the closest numbered version.

The subdirectories of the root directory of the source code (`/linux-4.14.99/`) are as follows:

- `arch`
- `block`
- `certs`
- `crypto`
- `Documentation`
- `drivers`
- `firmware`
- `fs`
- `include`
- `init`
- `ipc`
- `kernel`
- `lib`
- `mm`
- `net`
- `samples`
- `scripts`
- `security`
- `sound`
- `tools`
- `usr`
- `virt`

### 4)

The system call table definition in the `linux-4.14.99` source code has the path:

`/linux-4.14.99/arch/x86/entry/syscalls/syscall_64.tbl`

System call names corresponding to the asked system call numbers are given below:

- 5: `fstat`
- 43: `accept`
- 123: `setfsgid`

- 220: `semtimedop`

**5)**

`strace` records the names of the system calls that are called by the specified command until the command exits, together with the arguments and return values of the system calls.

Sample output for `strace ls` is provided, starting from below:

```
efe@efe-VirtualBox:~$ strace ls
execve("/bin/ls", ["ls"], 0x7ffddbf811b0 /* 61 vars */) = 0
brk(NULL) = 0x55e0cae5d000
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=73553, ...}) = 0
mmap(NULL, 73553, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ff3f7088000
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/libselinux.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20b\0\0\0\0\0\0"..., 832)
    = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=154832, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0
    x7ff3f7086000
mmap(NULL, 2259152, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0
    x7ff3f6c4b000
mprotect(0x7ff3f6c70000, 2093056, PROT_NONE) = 0
mmap(0x7ff3f6e6f000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
    MAP_DENYWRITE, 3, 0x24000) = 0x7ff3f6e6f000
mmap(0x7ff3f6e71000, 6352, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
    MAP_ANONYMOUS, -1, 0) = 0x7ff3f6e71000
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\260\34\2\0\0\0\0\0"...,
    832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=2030544, ...}) = 0
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0
    x7ff3f685a000
mprotect(0x7ff3f6a41000, 2097152, PROT_NONE) = 0
mmap(0x7ff3f6c41000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
    MAP_DENYWRITE, 3, 0x1e7000) = 0x7ff3f6c41000
mmap(0x7ff3f6c47000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
    MAP_ANONYMOUS, -1, 0) = 0x7ff3f6c47000
```

```
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/libpcre.so.3", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0 \25\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=464824, ...}) = 0
mmap(NULL, 2560264, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ff3f65e8000
mprotect(0x7ff3f6658000, 2097152, PROT_NONE) = 0
mmap(0x7ff3f6858000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x70000) = 0x7ff3f6858000
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/libdl.so.2", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\16\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=14560, ...}) = 0
mmap(NULL, 2109712, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ff3f63e4000
mprotect(0x7ff3f63e7000, 2093056, PROT_NONE) = 0
mmap(0x7ff3f65e6000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7ff3f65e6000
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/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0000b\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=144976, ...}) = 0
mmap(NULL, 2221184, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ff3f61c5000
mprotect(0x7ff3f61df000, 2093056, PROT_NONE) = 0
mmap(0x7ff3f63de000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19000) = 0x7ff3f63de000
mmap(0x7ff3f63e0000, 13440, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7ff3f63e0000
close(3) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff3f7084000
arch_prctl(ARCH_SET_FS, 0x7ff3f7085040) = 0
mprotect(0x7ff3f6c41000, 16384, PROT_READ) = 0
mprotect(0x7ff3f63de000, 4096, PROT_READ) = 0
mprotect(0x7ff3f65e6000, 4096, PROT_READ) = 0
mprotect(0x7ff3f6858000, 4096, PROT_READ) = 0
mprotect(0x7ff3f6e6f000, 4096, PROT_READ) = 0
mprotect(0x55e0c98d5000, 8192, PROT_READ) = 0
mprotect(0x7ff3f709a000, 4096, PROT_READ) = 0
munmap(0x7ff3f7088000, 73553) = 0
set_tid_address(0x7ff3f7085310) = 2097
set_robust_list(0x7ff3f7085320, 24) = 0
rt_sigaction(SIGRTMIN, {sa_handler=0x7ff3f61cacb0, sa_mask=[], sa_flags=SA_RESTORER|SA_SIGINFO, sa_restorer=0x7ff3f61d7890}, NULL, 8) = 0
rt_sigaction(SIGRT_1, {sa_handler=0x7ff3f61cad50, sa_mask=[], sa_flags=SA_RESTORER|SA_RESTART|SA_SIGINFO, sa_restorer=0x7ff3f61d7890}, NULL, 8) = 0
rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
statfs("/sys/fs/selinux", 0x7ffd3bd51d50) = -1 ENOENT (No such file or directory)
statfs("/selinux", 0x7ffd3bd51d50) = -1 ENOENT (No such file or directory)
brk(NULL) = 0x55e0cae5d000
brk(0x55e0cae7e000) = 0x55e0cae7e000
openat(AT_FDCWD, "/proc/filesystems", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
read(3, "nodev\tsysfs\nnodev\trootfs\nnodev\tr"..., 1024) = 383
read(3, "", 1024) = 0
close(3) = 0
access("/etc/selinux/config", F_OK) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=11731760, ...}) = 0
mmap(NULL, 11731760, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ff3f5694000
close(3) = 0
ioctl(1, TCGETS, {B38400 opost isig icanon echo ...}) = 0
ioctl(1, TIOCGWINSZ, {ws_row=28, ws_col=79, ws_xpixel=0, ws_ypixel=0}) = 0
openat(AT_FDCWD, ".", O_RDONLY|O_NONBLOCK|O_CLOEXEC|O_DIRECTORY) = 3
fstat(3, {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0
getdents(3, /* 21 entries */, 32768) = 664
getdents(3, /* 0 entries */, 32768) = 0
close(3) = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
write(1, "Desktop Downloads\t Music "..., 52Desktop Downloads Music Public Videos
) = 52
write(1, "Documents examples.desktop Pic"..., 49Documents examples.desktop Pictures Templates
) = 49
close(1) = 0
close(2) = 0
exit_group(0) = ?
+++ exited with 0 +++
```

## 6)

`time` command provides timing statistics about the execution of a specified command, the different times the command outputs have the following meanings:

- `real`: The total time from start to finish of the call, ie. from the moment you press Enter until the moment the command returns.
- `user`: The amount of CPU time that is spent in the user mode, in which privileged instructions cannot be executed.
- `sys`: The amount of CPU time that is spent in the kernel mode, in which privileged instructions are allowed to be executed.

The following table displays the timing statistics for five different commands:

| command (including arguments) | real | user | sys |
|---|---|---|---|
| cp Music | 0.016s | 0.002s | 0.000s |
| time strace ls | 0.007s | 0.002s | 0.003s |
| cd Desktop | 0.000s | 0.000s | 0.000s |
| ls | 0.003s | 0.001s | 0.000s |
| time time strace ls | 0.006s | 0.005s | 0.000s |

Table 1: Some commands and their timing statistics

## 7)

I measured the execution times for `getpid()`, `open()`, `read()`, `write()` and `mkdir()` system calls. `getpid()` has a simple job and it does not have any arguments, so it had a relatively short execution time as expected. I used `open()` system call to create a non-existing `.txt` file with the permissions necessary to perform read and write operations. The execution for this took relatively long, probably because the kernel had to allocate many resources under the hood to create the file. Then I measured the execution times for `write()` and `read()` system calls for various number of bytes. The results made sense, since the execution time increased with the number of bytes. The very first execution of `write()` took longer than I have expected, this may again be something about the kernel internals. I intentionally called `write()`s before `read()`s to fill the `.txt` file I had created with enough number of bytes. An implementation detail was to close and re-open the `.txt` file in between the write and read operations, otherwise the pointer used to read the file gets stuck in the end-of-file byte. Lastly, I made time measurements on `mkdir()` system call for two cases. In one case, a directory was created with full permissions; and in the other, a directory was created with only read permissions. The case with full permissions had a longer execution time as expected, since the kernel performs specific operations to grant those permissions.

The experiment is summarized in the table below:

| system call (arguments) | execution time |
|---|---|
| | (in microseconds) |
| getpid() | 2 |
| open("read.txt", O_CREAT \| O_RDWR \| O_APPEND, 00700) | 33 |
| write(fileDescriptor, &bytes, 100) | 13 |
| write(fileDescriptor, &bytes, 1000) | 3 |
| write(fileDescriptor, &bytes, 10000) | 7 |
| write(fileDescriptor, &bytes, 100000) | 42 |
| read(fileDescriptor, &buffer, 100) | 1 |
| read(fileDescriptor, &buffer, 1000) | 1 |
| read(fileDescriptor, &buffer, 10000) | 4 |
| read(fileDescriptor, &buffer, 100000) | 31 |
| mkdir("allAccessPermissionsGiven", ACCESSPERMS) | 25 |
| mkdir("onlyReadPermissionsGiven", S_IRUSR \| S_IRGRP \| S_IROTH) | 14 |

Table 2: Some system calls and their execution times

The C code and command line output follows:

The C code:

```c
/**
 * This program performs some timing experiments on some of the commonly
 * used Linux system calls. It executes the system calls with the specified
 * arguments and reports the execution time in microseconds.
 * @author Efe Acer
 * @version 1.0
 */

// Necessary imports to be able to run the system calls
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <stdio.h>

// Function decleration(s)
unsigned long getCurrentTime();

int main() {
    printf("\nTime measurements for different system calls in microseconds follows
        :\n");

    // Preparation for getpid() measurements
    unsigned long getpidStart;
    unsigned long getpidEnd;
    int processID;

    // Get process ID
    getpidStart = getCurrentTime();
    processID = getpid();
    getpidEnd = getCurrentTime();
    printf("\nTime to execute getpid(): %ld\n", getpidEnd - getpidStart);
    printf("Process ID is: %d\n", processID);

    // Preparation for open() measurements
    int fileDescriptor;
    unsigned long openStart;
    unsigned long openEnd;

    // Create a .txt file that will be used to read bytes from
    openStart = getCurrentTime();
    fileDescriptor = open("read.txt", O_CREAT | O_RDWR | O_APPEND, 00700); // 00700 is for file owner permissions
    openEnd = getCurrentTime();
    printf("\nTime to execute open() to create a new .txt file is: %ld\n", openEnd - openStart);
    printf("The value of fileDescriptor is %d:\n", fileDescriptor);

    // Preparation for write() measurements
    unsigned char bytes[100000];
    for (int i = 0; i < 100000; i++) {
        bytes[i] = '.';
    }
    unsigned long writeStart;
    unsigned long writeEnd;
    int numWritten;

    // Write 100 bytes to read.txt
    writeStart = getCurrentTime();
    numWritten = write(fileDescriptor, &bytes, 100);
    writeEnd = getCurrentTime();
    printf("\nTime to execute write() for %d bytes: %ld\n", numWritten, (writeEnd - writeStart));

    // Write 1000 bytes to read.txt
    writeStart = getCurrentTime();
    numWritten = write(fileDescriptor, &bytes, 1000);
    writeEnd = getCurrentTime();
    printf("Time to execute write() for %d bytes: %ld\n", numWritten, (writeEnd -
       writeStart));

    // Write 10000 bytes to read.txt
    writeStart = getCurrentTime();
    numWritten = write(fileDescriptor, &bytes, 10000);
    writeEnd = getCurrentTime();
    printf("Time to execute write() for %d bytes: %ld\n", numWritten, (writeEnd -
       writeStart));

    // Write 100000 bytes to read.txt
    writeStart= getCurrentTime();
    numWritten = write(fileDescriptor, &bytes, 100000);
    writeEnd = getCurrentTime();
    printf("Time to execute write() for %d bytes: %ld\n", numWritten, (writeEnd -
       writeStart));

    // Restore the pointer that is used to read and write to the file
    close(fileDescriptor);
    fileDescriptor = open("read.txt", O_RDWR, 00700); // 00700 is for file owner
       permissions

    // Preparation for read() measurements
    unsigned char buffer[111100];
    unsigned long readStart;
    unsigned long readEnd;
    int numRead;

    // Read 100 bytes from read.txt
    readStart = getCurrentTime();
    numRead = read(fileDescriptor, &buffer, 100);
    readEnd = getCurrentTime();
    printf("\nTime to execute read() for %d bytes: %ld\n", numRead, (readEnd -
       readStart));

    // Read 1000 bytes from read.txt
    readStart = getCurrentTime();
    numRead = read(fileDescriptor, &buffer, 1000);
    readEnd = getCurrentTime();
    printf("Time to execute read() for %d bytes: %ld\n", numRead, (readEnd -
        readStart));

    // Read 10000 bytes from read.txt
    readStart = getCurrentTime();
    numRead = read(fileDescriptor, &buffer, 10000);
    readEnd = getCurrentTime();
    printf("Time to execute read() for %d bytes: %ld\n", numRead, (readEnd - readStart));

    // Read 100000 bytes from read.txt
    readStart = getCurrentTime();
    numRead = read(fileDescriptor, &buffer, 100000);
    readEnd = getCurrentTime();
    printf("Time to execute read() for %d bytes: %ld\n", numRead, (readEnd - readStart));

    // Preparation for mkdir() measurements
    unsigned long mkdirStart;
    unsigned long mkdirEnd;
    int success;

    // Make a directory with all access permissions given
    mkdirStart = getCurrentTime();
    success = mkdir("allAccessPermissionsGiven", ACCESSPERMS);
    mkdirEnd = getCurrentTime();
    printf("\nTime to execute mkdir() giving all access permissions: %ld\n", mkdirEnd - mkdirStart);
    printf("Successful creation (0 is OK if -1 the directory probably exists): %d\n", success);

    // Make a directory with only read permissions given
    mkdirStart = getCurrentTime();
    success = mkdir("onlyReadPermissionsGiven", S_IRUSR | S_IRGRP | S_IROTH);
    mkdirEnd = getCurrentTime();
    printf("Time to execute mkdir() giving only read permissions: %ld\n", mkdirEnd - mkdirStart);
    printf("Successful creation (0 is OK if -1 the directory probably exists): %d\n", success);

    return 0;
}

/**
 * Returns the current time in microseconds. The current
 * time corresponds to the time elapsed from the starting
 * point used by the gettimeofday() function.
 * @return currentTime: The current time in microseconds
 */
unsigned long getCurrentTime() {
    struct timeval timeValue;
    gettimeofday(&timeValue, NULL);
    unsigned long currentTime = timeValue.tv_usec; // microseconds part of the struct
    currentTime += timeValue.tv_sec * 1e6; // add the seconds part of the struct
    return currentTime;
}
```

The command line output:

```
efe@efe-VirtualBox:~/Desktop$ make
gcc -Wall -g -o cost cost.c
efe@efe-VirtualBox:~/Desktop$ ./cost

Time measurements for different system calls in microseconds follows:

Time to execute getpid(): 2
Process ID is: 4160

Time to execute open() to create a new .txt file is: 33
The value of fileDescriptor is 3:

Time to execute write() for 100 bytes: 13
Time to execute write() for 1000 bytes: 3
Time to execute write() for 10000 bytes: 7
Time to execute write() for 100000 bytes: 42

Time to execute read() for 100 bytes: 1
Time to execute read() for 1000 bytes: 1
Time to execute read() for 10000 bytes: 4
Time to execute read() for 100000 bytes: 31

Time to execute mkdir() giving all access permissions: 25
Successful creation (0 is OK if -1 the directory probably exists): 0
Time to execute mkdir() giving only read permissions: 14
Successful creation (0 is OK if -1 the directory probably exists): 0
```

For the make command, The Makefile provided in the homework assignment sheet is used as it is.