2022-2023 Spring

CS 342 – Operating Systems

Homework #1 – Report

Berk Çakar - 22003021 - Section 2

1) As assigned in the first step of the homework, I have read the first and second chapters of the textbook, which is Silberschatz's Operating System Concepts. In summary, Chapter 1 is named "Introduction," This chapter starts with reviewing concepts such as interrupts, storage and I/O structure, single/multi-processor systems, and more regarding computer organization and architecture. Then, the chapter discusses a selection of operating systems' operations and features like multitasking, memory management, and file system management. In addition to that, the chapter also covers some related concepts, such as security and protection, virtualization, and distributed systems. After that, the chapter talks about the data structures extensively used in operating systems, especially in the kernel. These include lists, stacks, queues, trees, maps, and bitmaps. Finally, the chapter ends with a brief introduction to the history of operating systems and the evolution of operating systems. On the other hand, Chapter 2 is named "Operating-System Structures." Unlike the first chapter, this chapter does not discuss hardware organization and architecture; instead, it focuses on the elements of operating systems. The chapter starts with introducing the ways for users to interact with the operating system, such as the command line, graphical user interface, and touch screen. Following that, system calls, services, linkers, and loaders are discussed in detail. Lastly, the chapter explains how the operating system is structured (monolithic, microkernel, hybrid, and layered approaches) and how the kernel and the user space are separated based on examples from operating systems, including iOS, macOS, Android, UNIX, and Linux. Lastly, the chapter ends with mentioning how an operating system is generated from scratch, how it is booted up, and some ways to debug an operating system.

2) Currently, I am using a MacBook, which has macOS Catalina 10.15.7 as its operating system. Rather than installing Ubuntu Desktop 64-bit 22.04 on my bare hardware, I preferred to use a virtual machine to install Ubuntu. To do that, I used VMWare Fusion software for macOS. VMWare Fusion is an enterprise-level virtualization software by the company VMWare Inc. One can create virtual machines on macOS using VMWare Fusion and run different operating systems on them. In VMWare Fusion, I created a new virtual machine with the following specifications: 4 GB RAM, 30 GB of storage space, and 4 CPU cores. Then, I downloaded the installation file of Ubuntu Desktop 64-bit 22.04 from the official website of Ubuntu using the link provided in the homework assignment. Then, I loaded the Ubuntu installation image into the virtual machine which I recently created. Since I was already experienced with Linux distributions, the installation process was straightforward. I followed the instructions on the screen and installed Ubuntu on the virtual machine by allocating all 30 GB of storage space to

Ubuntu. After completing the installation, I searched the Internet to learn basic Linux usage and some terminal commands. Then, I encountered a Linux usage guide document prepared at the University of Leicester [1]. Here are the 10 Linux commands that I learned from the guide:

- `ls`: Lists the contents of a directory.
- `mkdir`: Creates a new directory.
- `cd`: Changes to a different directory.
- `pwd`: Prints the path of the current working directory.
- `cp`: Copies a file or directory.
- `mv`: Moves a file or directory.
- `grep`: Prints lines matching specified words or patterns.
- `cat`: Concatenates files and prints on the standard output.
- `rmdir`: Deletes a directory if and only if the directory is empty.
- `clear`: Clears the terminal screen (prompt).

3) In this Linux installation of Ubuntu Desktop 64-bit 22.04, the kernel executable resides in the `/boot` directory. The kernel executable inside there is named `vmlinuz-5.15.0-58-generic` Therefore, the full path for the kernel executable is `/boot/vmlinuz-5.15.0-58-generic` `uname -r` command outputs `5.15.0-58-generic`, which is the version of the running kernel.

4) I downloaded the source code of the Linux kernel from the official website of the Linux kernel (i.e., https://www.kernel.org/). On the website, the version which is the closest to the version of my running kernel was 5.15.91. Therefore, I downloaded the source code of this version as a tarball (i.e., linux-5.15.91.tar.xz). Then, I extracted the tar package and changed the directory to the extracted directory. Names of the subdirectories in the root directory of the kernel source code are as follows:

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

5) For the 64-bit architecture, the definition of the system call table in the kernel source code root directory is located in the following file: `arch/x86/entry/syscalls/syscall_64.tbl` From this file, it is possible to find out the system call names corresponding to the requested system call numbers in the homework assignment:

- `0`: read
- `1`: write
- `2`: open
- `3`: close
- `4`: stat
- `5`: fstat
- `6`: lstat
- `39`: getpid
- `120`: getresgid
- `150`: munlock

6) According to the output of the `man strace` command (which provides the user manual for the `strace` program), `strace` is used for tracing system calls and signals.

I tested the `strace` utility to trace the system calls used by a copy operation (i.e., `strace cp test.txt test-copy.txt`), and the output is as follows:

```
1.  execve("/usr/bin/cp", ["cp", "test.txt", "test-copy.txt"], 0x7fff296a84e0 /* 53 vars */) = 0
2.  brk(NULL)                               = 0x563c2092f000
3.  arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc0c242e60) = -1 EINVAL (Invalid argument)
4.  mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff99e675000
5.  access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
6.  openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
7.  newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=51611, ...}, AT_EMPTY_PATH) = 0
8.  mmap(NULL, 51611, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ff99e668000
9.  close(3)                                = 0
10. openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libselinux.so.1", O_RDONLY|O_CLOEXEC) = 3
11. read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
12. newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=166280, ...}, AT_EMPTY_PATH) = 0
13. mmap(NULL, 177672, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ff99e63c000
14. mprotect(0x7ff99e642000, 139264, PROT_NONE) = 0
15. mmap(0x7ff99e642000, 106496, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
    0x6000) = 0x7ff99e642000
16. mmap(0x7ff99e65c000, 28672, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x20000) =
    0x7ff99e65c000
17. mmap(0x7ff99e664000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
    0x27000) = 0x7ff99e664000
18. mmap(0x7ff99e666000, 5640, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0)
    = 0x7ff99e666000
```

```
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libacl.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\0\0\0\0\0\0\0"..., 832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=34888, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 36896, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ff99e632000
mprotect(0x7ff99e634000, 24576, PROT_NONE) = 0
mmap(0x7ff99e634000, 16384, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7ff99e634000
mmap(0x7ff99e638000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x6000) = 0x7ff99e638000
mmap(0x7ff99e63a000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x7000) = 0x7ff99e63a000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libattr.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\0\0\0\0\0\0\0"..., 832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=26696, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 28696, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ff99e62a000
mmap(0x7ff99e62c000, 12288, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7ff99e62c000
mmap(0x7ff99e62f000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x5000) = 0x7ff99e62f000
mmap(0x7ff99e630000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x5000) = 0x7ff99e630000
close(3)                                = 0
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\0P\237\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0i8\235HZ\227\223\333\350s\360\352,\223\340."..., 68, 896) = 68
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=2216304, ...}, AT_EMPTY_PATH) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
mmap(NULL, 2260560, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ff99e402000
mmap(0x7ff99e42a000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7ff99e42a000
mmap(0x7ff99e5bf000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7ff99e5bf000
mmap(0x7ff99e617000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x214000) = 0x7ff99e617000
mmap(0x7ff99e61d000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7ff99e61d000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpcre2-8.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\0\0\0\0\0\0\0\0"..., 832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=613064, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 615184, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ff99e36b000
mmap(0x7ff99e36d000, 438272, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7ff99e36d000
mmap(0x7ff99e3d8000, 163840, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x6d000) = 0x7ff99e3d8000
mmap(0x7ff99e400000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x94000) = 0x7ff99e400000
close(3)                                = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff99e369000
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff99e366000
arch_prctl(ARCH_SET_FS, 0x7ff99e366800) = 0
set_tid_address(0x7ff99e366ad0)         = 42864
set_robust_list(0x7ff99e366ae0, 24)     = 0
rseq(0x7ff99e3671a0, 0x20, 0, 0x53053053) = 0
mprotect(0x7ff99e617000, 16384, PROT_READ) = 0
mprotect(0x7ff99e400000, 4096, PROT_READ) = 0
mprotect(0x7ff99e630000, 4096, PROT_READ) = 0
mprotect(0x7ff99e63a000, 4096, PROT_READ) = 0
```

```
68. mprotect(0x7ff99e664000, 4096, PROT_READ) = 0
69. mprotect(0x563c1f26a000, 4096, PROT_READ) = 0
70. mprotect(0x7ff99e6af000, 8192, PROT_READ) = 0
71. prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
72. munmap(0x7ff99e668000, 51611)           = 0
73. statfs("/sys/fs/selinux", 0x7ffc0c242ea0) = -1 ENOENT (No such file or directory)
74. statfs("/selinux", 0x7ffc0c242ea0)      = -1 ENOENT (No such file or directory)
75. getrandom("\x0b\x41\x7e\x90\x4a\x99\x49\xd0", 8, GRND_NONBLOCK) = 8
76. brk(NULL)                               = 0x563c2092f000
77. brk(0x563c20950000)                     = 0x563c20950000
78. openat(AT_FDCWD, "/proc/filesystems", O_RDONLY|O_CLOEXEC) = 3
79. newfstatat(3, "", {st_mode=S_IFREG|0444, st_size=0, ...}, AT_EMPTY_PATH) = 0
80. read(3, "nodev\tsysfs\nnodev\ttmpfs\nnodev\tbd"..., 1024) = 393
81. read(3, "", 1024)                       = 0
82. close(3)                                = 0
83. access("/etc/selinux/config", F_OK)     = -1 ENOENT (No such file or directory)
84. openat(AT_FDCWD, "/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
85. newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=17388592, ...}, AT_EMPTY_PATH) = 0
86. mmap(NULL, 17388592, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ff99d2d0000
87. close(3)                                = 0
88. geteuid()                               = 1000
89. newfstatat(AT_FDCWD, "test-copy.txt", 0x7ffc0c242cf0, 0) = -1 ENOENT (No such file or
    directory)
90. newfstatat(AT_FDCWD, "test.txt", {st_mode=S_IFREG|0664, st_size=0, ...}, 0) = 0
91. newfstatat(AT_FDCWD, "test-copy.txt", 0x7ffc0c242a70, 0) = -1 ENOENT (No such file or
    directory)
92. openat(AT_FDCWD, "test.txt", O_RDONLY)  = 3
93. newfstatat(3, "", {st_mode=S_IFREG|0664, st_size=0, ...}, AT_EMPTY_PATH) = 0
94. openat(AT_FDCWD, "test-copy.txt", O_WRONLY|O_CREAT|O_EXCL, 0664) = 4
95. newfstatat(4, "", {st_mode=S_IFREG|0664, st_size=0, ...}, AT_EMPTY_PATH) = 0
96. fadvise64(3, 0, 0, POSIX_FADV_SEQUENTIAL) = 0
97. mmap(NULL, 139264, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff99d2ae000
98. read(3, "", 131072)                     = 0
99. close(4)                                = 0
100. close(3)                               = 0
101. munmap(0x7ff99d2ae000, 139264)         = 0
102. lseek(0, 0, SEEK_CUR)                  = -1 ESPIPE (Illegal seek)
103. close(0)                               = 0
104. close(1)                               = 0
105. close(2)                               = 0
106. exit_group(0)                          = ?
107. +++ exited with 0 +++
```

Then, I tried `strace` again with the `ls` command (i.e., `strace ls`) and got the following output:

```
1.  execve("/usr/bin/ls", ["ls"], 0x7ffcbc362c90 /* 53 vars */) = 0
2.  brk(NULL)                               = 0x559259544000
3.  arch_prctl(0x3001 /* ARCH_??? */, 0x7ffca0d9ae80) = -1 EINVAL (Invalid argument)
4.  mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f1ded1b6000
5.  access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
6.  openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
7.  newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=51611, ...}, AT_EMPTY_PATH) = 0
8.  mmap(NULL, 51611, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f1ded1a9000
9.  close(3)                                = 0
10. openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libselinux.so.1", O_RDONLY|O_CLOEXEC) = 3
11. read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
12. newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=166280, ...}, AT_EMPTY_PATH) = 0
13. mmap(NULL, 177672, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f1ded17d000
14. mprotect(0x7f1ded183000, 139264, PROT_NONE) = 0
15. mmap(0x7f1ded183000, 106496, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
    0x6000) = 0x7f1ded183000
16. mmap(0x7f1ded19d000, 28672, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x20000) =
    0x7f1ded19d000
17. mmap(0x7f1ded1a5000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
    0x27000) = 0x7f1ded1a5000
```

```
mmap(0x7f1ded1a7000, 5640, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f1ded1a7000
close(3)                                = 0
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\0P\237\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0i8\235HZ\227\223\333\350s\360\352,\223\340."..., 68, 896) = 68
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=2216304, ...}, AT_EMPTY_PATH) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
mmap(NULL, 2260560, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f1decf55000
mmap(0x7f1decf7d000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f1decf7d000
mmap(0x7f1ded112000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7f1ded112000
mmap(0x7f1ded16a000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x214000) = 0x7f1ded16a000
mmap(0x7f1ded170000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f1ded170000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpcre2-8.so.0", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=613064, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 615184, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f1decebe000
mmap(0x7f1decec0000, 438272, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f1decec0000
mmap(0x7f1decf2b000, 163840, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x6d000) = 0x7f1decf2b000
mmap(0x7f1decf53000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x94000) = 0x7f1decf53000
close(3)                                = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f1decebc000
arch_prctl(ARCH_SET_FS, 0x7f1decebcc40) = 0
set_tid_address(0x7f1decebcf10)         = 42873
set_robust_list(0x7f1decebcf20, 24)     = 0
rseq(0x7f1decebd5e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7f1ded16a000, 16384, PROT_READ) = 0
mprotect(0x7f1decf53000, 4096, PROT_READ) = 0
mprotect(0x7f1ded1a5000, 4096, PROT_READ) = 0
mprotect(0x55925892b000, 4096, PROT_READ) = 0
mprotect(0x7f1ded1f0000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f1ded1a9000, 51611)           = 0
statfs("/sys/fs/selinux", 0x7ffca0d9aec0) = -1 ENOENT (No such file or directory)
statfs("/selinux", 0x7ffca0d9aec0)      = -1 ENOENT (No such file or directory)
getrandom("\xb9\xd6\xc3\xdb\xc6\xec\x17\xcc", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x559259544000
brk(0x559259565000)                     = 0x559259565000
openat(AT_FDCWD, "/proc/filesystems", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0444, st_size=0, ...}, AT_EMPTY_PATH) = 0
read(3, "nodev\tsysfs\nnodev\ttmpfs\nnodev\tbd"..., 1024) = 393
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
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=17388592, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 17388592, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f1debe26000
close(3)                                = 0
ioctl(1, TCGETS, {B38400 opost isig icanon echo ...}) = 0
ioctl(1, TIOCGWINSZ, {ws_row=34, ws_col=158, ws_xpixel=0, ws_ypixel=0}) = 0
openat(AT_FDCWD, ".", O_RDONLY|O_NONBLOCK|O_CLOEXEC|O_DIRECTORY) = 3
newfstatat(3, "", {st_mode=S_IFDIR|0750, st_size=4096, ...}, AT_EMPTY_PATH) = 0
```

```
getdents64(3, 0x55925954bab0 /* 24 entries */, 32768) = 768
getdents64(3, 0x55925954bab0 /* 0 entries */, 32768) = 0
close(3)                                = 0
newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH) = 0
write(1,            "Desktop             Documents             Downloads             M"..., 102Desktop   Documents   Downloads   Music   Pictures   Public   snap   Templates   test-copy.txt  test.txt  Videos
) = 102
close(1)                                = 0
close(2)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

7) The `time` command reports the amount of time programs spend during their execution. However, it outputs the time statistics under three different headings: `real`, `user`, and `sys`. These three different time statistics have meanings explained below:

`real`: The real time is the actual time elapsed between the start and end of the program execution. This one is the most intuitive one, and it can be thought of as a regular stopwatch that measures the time elapsed between the start and end of the program execution.

`user`: The user time is the amount of CPU time spent by the program in user mode in which the program is executing instructions in user mode, and executing privileged instructions is not allowed. However, the time waited for I/O operations is not included in this time.

`sys`: The sys time is the amount of CPU time spent by the program in kernel mode in which the program is executing privileged instructions. Similar to the user time, the time waited for I/O operations is not included in this time.

Sample outputs of the `time` command are shown below:

```
$ time touch test-header.h
real    0m0,003s
user    0m0,001s
sys     0m0,002s

$ time strace mkdir myFiles
real    0m0,048s
user    0m0,008s
sys     0m0,019s
```

```
$ time gcc hello-world.c
real    0m0,147s
user    0m0,030s
sys     0m0,041s

$ time mv Music Videos Multimedia
real    0m0,009s
user    0m0,003s
sys     0m0,003s
```

8) As instructed in the homework assignment, I implemented a doubly linked list of integers in C, which keeps the integers in ascending order. Afterward, I inserted 10000 random integers into the list and printed the list. Then I measured the time in terms of milliseconds it took to insert 10000 random integers into the doubly linked list using the `gettimeofday()` system call.

The source code for the doubly linked list (`list.c`) is given below:

```c
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>

// Macro definitions

// #define DEBUG
#define dll_node struct doubly_linked_list_node
#define DLL_SIZE 10000

// Function prototypes (declarations)

dll_node* new_dll_node(int data);
void sorted_insert(dll_node **head, dll_node *node);
void print_dll(dll_node *head);
void delete_dll(dll_node *head);
int generate_random_int();
double get_time_diff();

// Struct definitions

struct doubly_linked_list_node {
    int data;
    struct doubly_linked_list_node *next;
    struct doubly_linked_list_node *prev;
};

struct timeval start, end;

// Function definitions

int generate_random_int() {
    return rand();
}

dll_node* new_dll_node(int data) {
    dll_node *new_node = (dll_node*)malloc(sizeof(dll_node));
    new_node->data = data;
    new_node->next = NULL;
    new_node->prev = NULL;
    return new_node;
}

void sorted_insert(dll_node **head, dll_node *node) {
    if (*head == NULL || (*head)->data >= node->data) {
        node->next = *head;
        node->prev = NULL;
        if (*head != NULL) {
            (*head)->prev = node;
        }
        *head = node;
    } else {
        dll_node *current = *head;
```

```
54.         while (current->next != NULL && current->next->data < node->data) {
55.             current = current->next;
56.         }
57.         node->next = current->next;
58.         node->prev = current;
59.         if (current->next != NULL) {
60.             current->next->prev = node;
61.         }
62.         current->next = node;
63.     }
64. }
65.
66. void print_dll(dll_node *head) {
67.     dll_node *current = head;
68.     while (current != NULL) {
69.         printf("%d\n", current->data);
70.         current = current->next;
71.     }
72. }
73.
74. void delete_dll(dll_node *head) {
75.     dll_node *current = head;
76.     while (current != NULL) {
77.         dll_node *next = current->next;
78.         free(current);
79.         current = next;
80.     }
81. }
82.
83. // Return the time difference in milliseconds (1 milliseconds = 1000 microseconds)
84. double get_time_diff() {
85.     return ((end.tv_sec - start.tv_sec) * 1000.0) + ((end.tv_usec - start.tv_usec) / 1000.0);
86. }
87.
88. int main() {
89.     gettimeofday(&start, NULL);
90.
91.     dll_node *head = NULL;
92.     for (int i = 0; i < DLL_SIZE; i++) {
93.         sorted_insert(&head, new_dll_node(generate_random_int()));
94.     }
95.
96.     gettimeofday(&end, NULL);
97.
98.     printf("Time taken to insert %d integers into a doubly linked list in ascending sorted
    order: %f milliseconds\n", DLL_SIZE, get_time_diff());
99.
100.    #ifdef DEBUG
101.    print_dll(head);
102.    #endif
103.
104.    delete_dll(head);
105.
106.    return 0;
107. }
```

The source code for the `Makefile` is given below:

```
1.  LIST_FILE := list.c
2.
3.  CXX        := gcc
4.  CXXFLAGS   := -Wall -g
5.  SRC        := .
6.  EXECUTABLE := list
7.
```

```
all: clean list

list:
    @-$(CXX) $(CXXFLAGS) -o $(SRC)/$(EXECUTABLE) $(SRC)/$(LIST_FILE)

run: all
    @-$(SRC)/$(EXECUTABLE)

clean:
    @-rm -rf $(SRC)/$(EXECUTABLE) $(SRC)/$(EXECUTABLE).o *~
```

The output of the program is the following:

```
berk-cakar@berk-ubuntu:~/hw1$ make
berk-cakar@berk-ubuntu:~/hw1$ ./list
Time taken to insert 10000 integers into a doubly linked list in ascending sorted order: 126.527000 milliseconds
```

References

[1] J. Wakelin, L. Gretton, G. Gilchrist, and T. Forey, “Linux Tutorial.” University of Leicester [Online]. Available: https://web.njit.edu/~alexg/courses/cs332/OLD/S2020/s20hand3/Linux-Tutorial.pdf. [Accessed: Feb. 02, 2023]