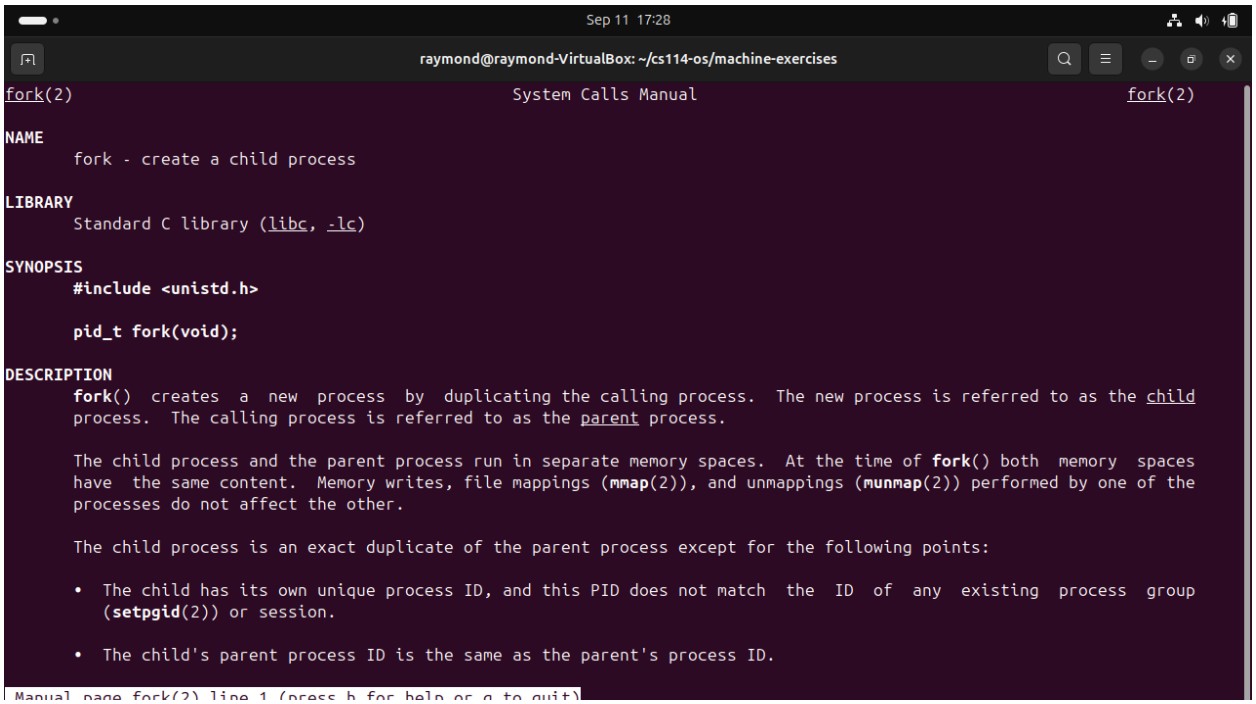
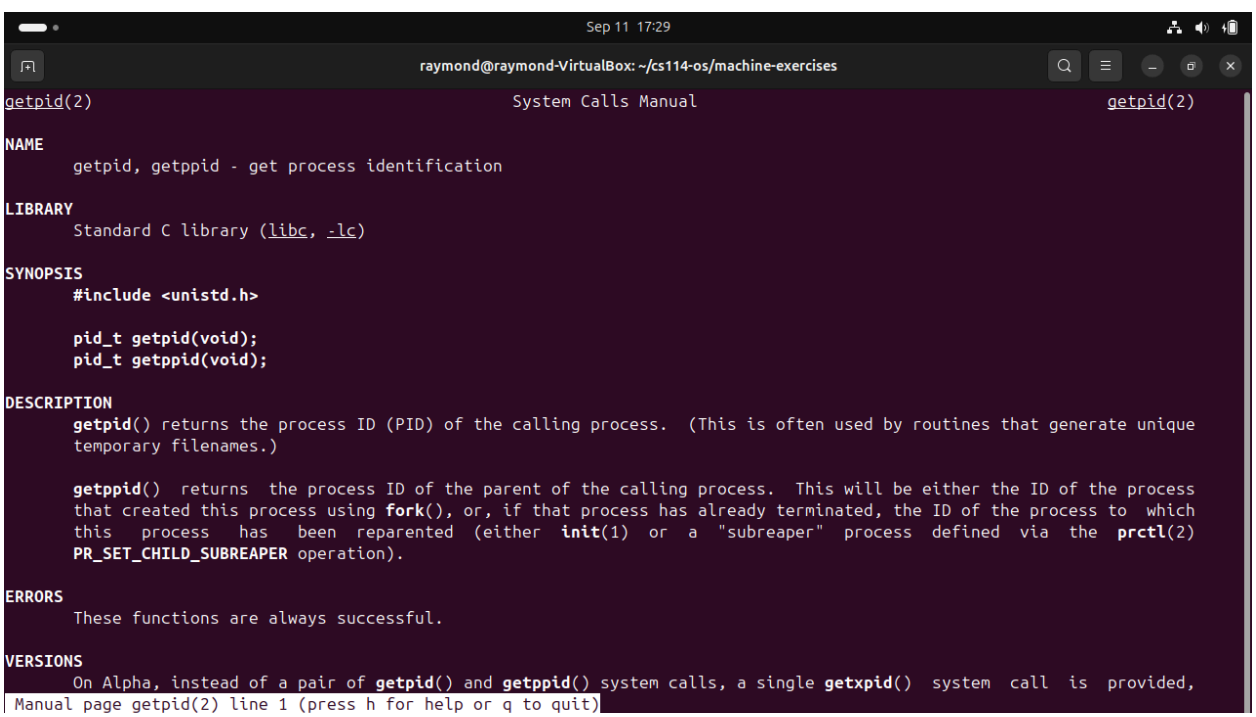
Machine Exercise #2

A.
Reading the Manuals

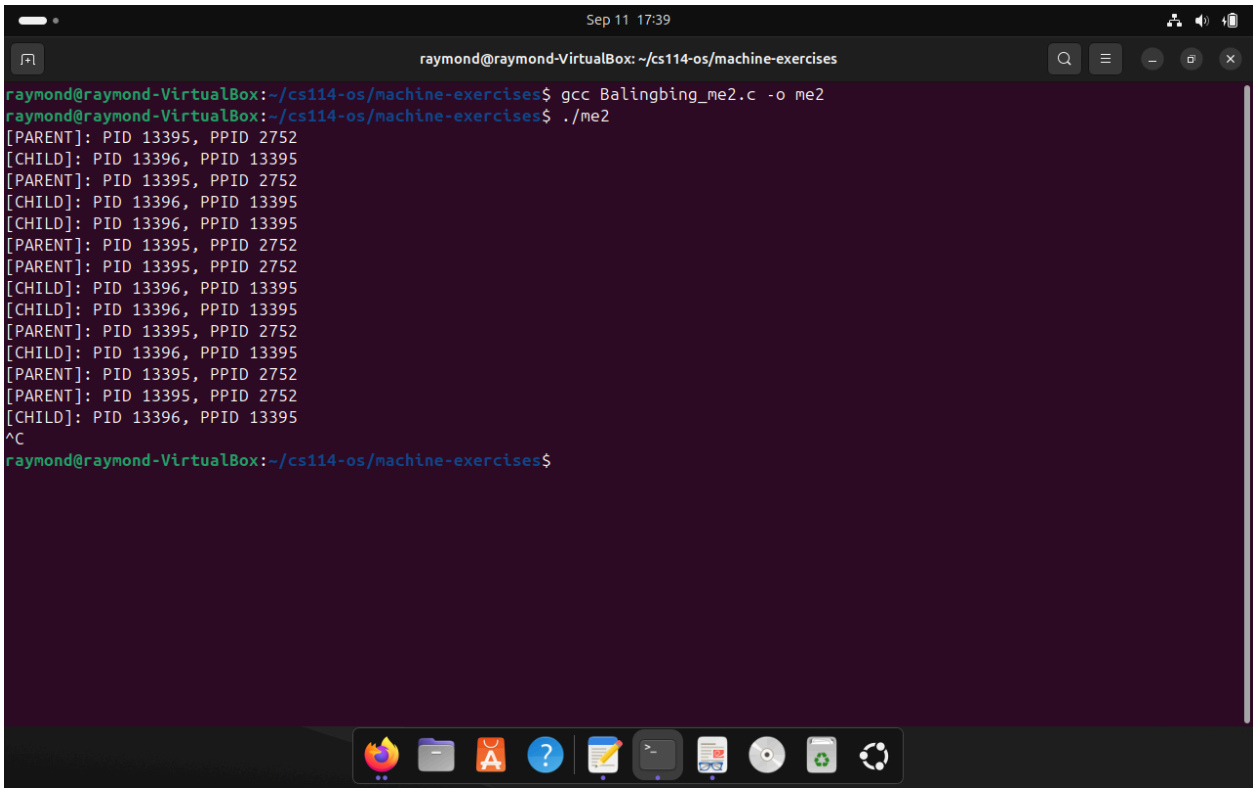


```
Sep 11 17:28
raymond@raymond-VirtualBox: ~/cs114-os/machine-exercises
fork(2)                    System Calls Manual                    fork(2)

NAME
       fork - create a child process

LIBRARY
       Standard C library (libc, -lc)

SYNOPSIS
       #include <unistd.h>

       pid_t fork(void);

DESCRIPTION
       fork() creates a new process by duplicating the calling process. The new process is referred to as the child
       process. The calling process is referred to as the parent process.

       The child process and the parent process run in separate memory spaces. At the time of fork() both memory spaces
       have the same content. Memory writes, file mappings (mmap(2)), and unmappings (munmap(2)) performed by one of the
       processes do not affect the other.

       The child process is an exact duplicate of the parent process except for the following points:

       • The child has its own unique process ID, and this PID does not match the ID of any existing process group
         (setpgid(2)) or session.

       • The child's parent process ID is the same as the parent's process ID.

 Manual page fork(2) line 1 (press h for help or q to quit)
```



```
Sep 11 17:29
raymond@raymond-VirtualBox: ~/cs114-os/machine-exercises
getpid(2)                  System Calls Manual                  getpid(2)

NAME
       getpid, getppid - get process identification

LIBRARY
       Standard C library (libc, -lc)

SYNOPSIS
       #include <unistd.h>

       pid_t getpid(void);
       pid_t getppid(void);

DESCRIPTION
       getpid() returns the process ID (PID) of the calling process. (This is often used by routines that generate unique
       temporary filenames.)

       getppid() returns the process ID of the parent of the calling process. This will be either the ID of the process
       that created this process using fork(), or, if that process has already terminated, the ID of the process to which
       this process has been reparented (either init(1) or a "subreaper" process defined via the prctl(2)
       PR_SET_CHILD_SUBREAPER operation).

ERRORS
       These functions are always successful.

VERSIONS
       On Alpha, instead of a pair of getpid() and getppid() system calls, a single getxpid() system call is provided,
 Manual page getpid(2) line 1 (press h for help or q to quit)
```

Running the C Program



Based on my observation, the CPU alternates between running the parent and child processes. In the terminal, the CPU alternates between in a quick manner making it looks like they are running at the same time, but in reality, the CPU is scheduling turn-by-turn.

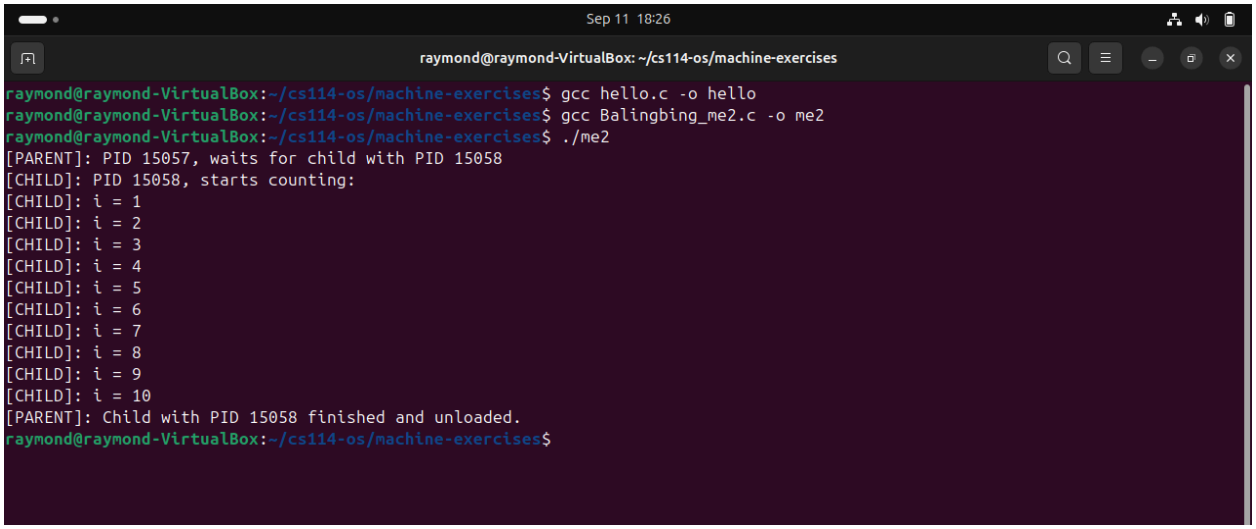
When I press Ctrl+C, both of the program and its processes terminated immediately since they received the interrupt signal from the terminal

B.
Reading the Manuals

```
Sep 11 17:58
raymond@raymond-VirtualBox: ~/cs114-os/machine-exercises
wait(2)                                System Calls Manual                                wait(2)

NAME
       wait, waitpid, waitid - wait for process to change state

LIBRARY
       Standard C library (libc, -lc)

SYNOPSIS
       #include <sys/wait.h>

       pid_t wait(int *_Nullable wstatus);
       pid_t waitpid(pid_t pid, int *_Nullable wstatus, int options);

       int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);
                       /* This is the glibc and POSIX interface; see
                          NOTES for information on the raw system call. */

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       waitid():
           Since glibc 2.26:
               _XOPEN_SOURCE >= 500 || _POSIX_C_SOURCE >= 200809L
           glibc 2.25 and earlier:
               _XOPEN_SOURCE
                   || /* Since glibc 2.12: */ _POSIX_C_SOURCE >= 200809L
                   || /* glibc <= 2.19: */ _BSD_SOURCE
Manual page wait(2) line 1 (press h for help or q to quit)
```

```
Sep 11 17:59
raymond@raymond-VirtualBox: ~/cs114-os/machine-exercises
exec(3)                                Library Functions Manual                                exec(3)

NAME
       execl, execlp, execle, execv, execvp, execvpe - execute a file

LIBRARY
       Standard C library (libc, -lc)

SYNOPSIS
       #include <unistd.h>

       extern char **environ;

       int execl(const char *pathname, const char *arg, ...
                       /*, (char *) NULL */);
       int execlp(const char *file, const char *arg, ...
                       /*, (char *) NULL */);
       int execle(const char *pathname, const char *arg, ...
                       /*, (char *) NULL, char *const envp[] */);
       int execv(const char *pathname, char *const argv[]);
       int execvp(const char *file, char *const argv[]);
       int execvpe(const char *file, char *const argv[], char *const envp[]);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       execvpe():
           _GNU_SOURCE
Manual page exec(3) line 1 (press h for help or q to quit)
```

Running the C Program



I created a new C program named **hello.c** and compiled it with an output file named **hello**.