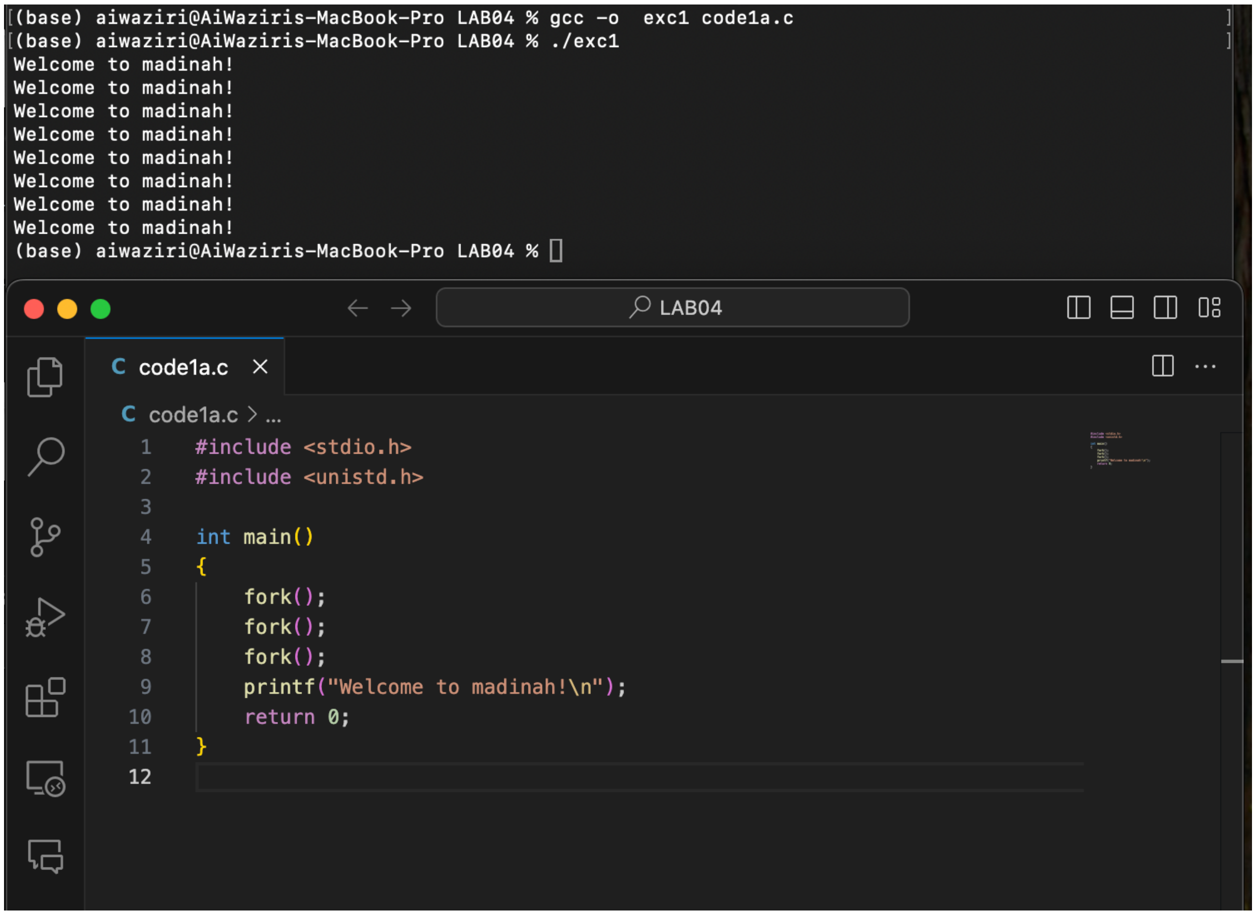
Name: Abubakar Waziri

ID: 4220056

Lab04

Q1a



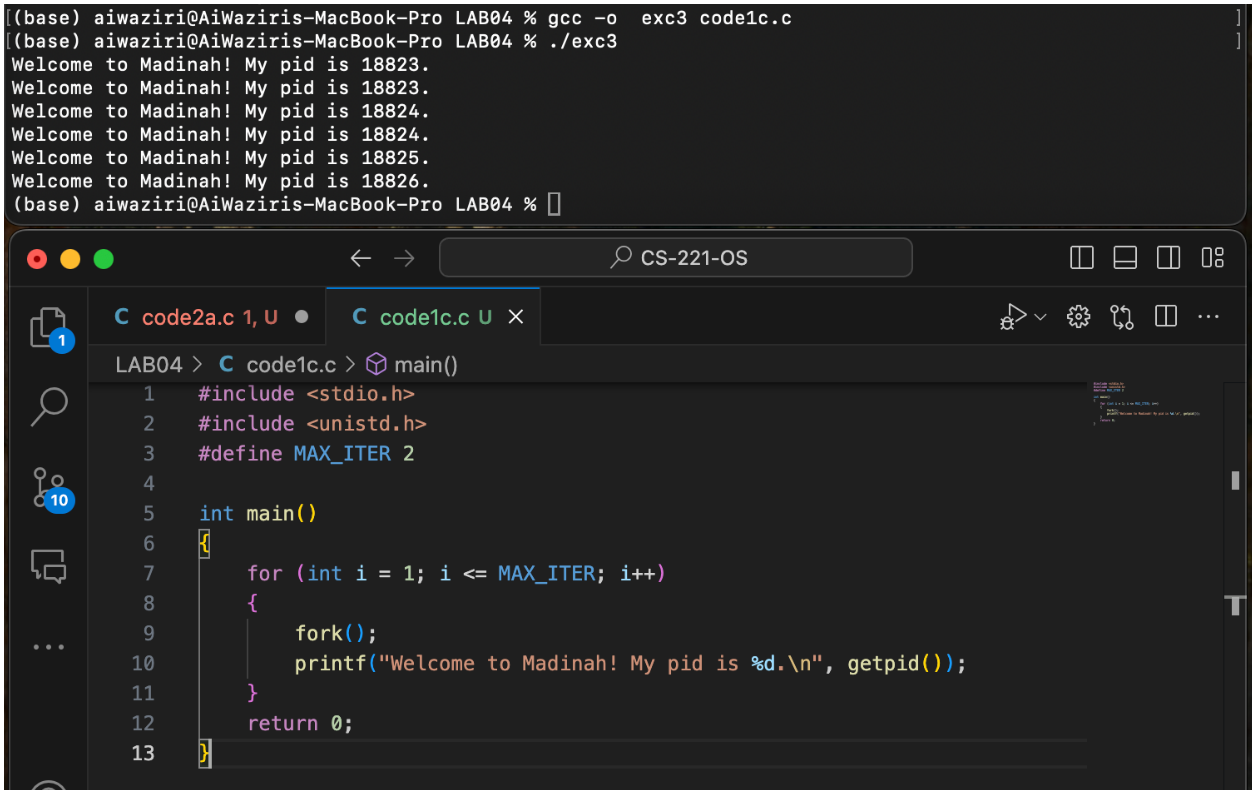
```
(base) aiwaziri@AiWaziris-MacBook-Pro LAB04 % gcc -o  exc1 code1a.c
(base) aiwaziri@AiWaziris-MacBook-Pro LAB04 % ./exc1
Welcome to madinah!
Welcome to madinah!
Welcome to madinah!
Welcome to madinah!
Welcome to madinah!
Welcome to madinah!
Welcome to madinah!
Welcome to madinah!
(base) aiwaziri@AiWaziris-MacBook-Pro LAB04 %
```

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
    fork();
    fork();
    fork();
    printf("Welcome to madinah!\n");
    return 0;
}
```

Q1b

4 processes

Q1c:



```
(base) aiwaziri@AiWaziris-MacBook-Pro LAB04 % gcc -o  exc3 code1c.c
(base) aiwaziri@AiWaziris-MacBook-Pro LAB04 % ./exc3
Welcome to Madinah! My pid is 18823.
Welcome to Madinah! My pid is 18823.
Welcome to Madinah! My pid is 18824.
Welcome to Madinah! My pid is 18824.
Welcome to Madinah! My pid is 18825.
Welcome to Madinah! My pid is 18826.
(base) aiwaziri@AiWaziris-MacBook-Pro LAB04 %
```

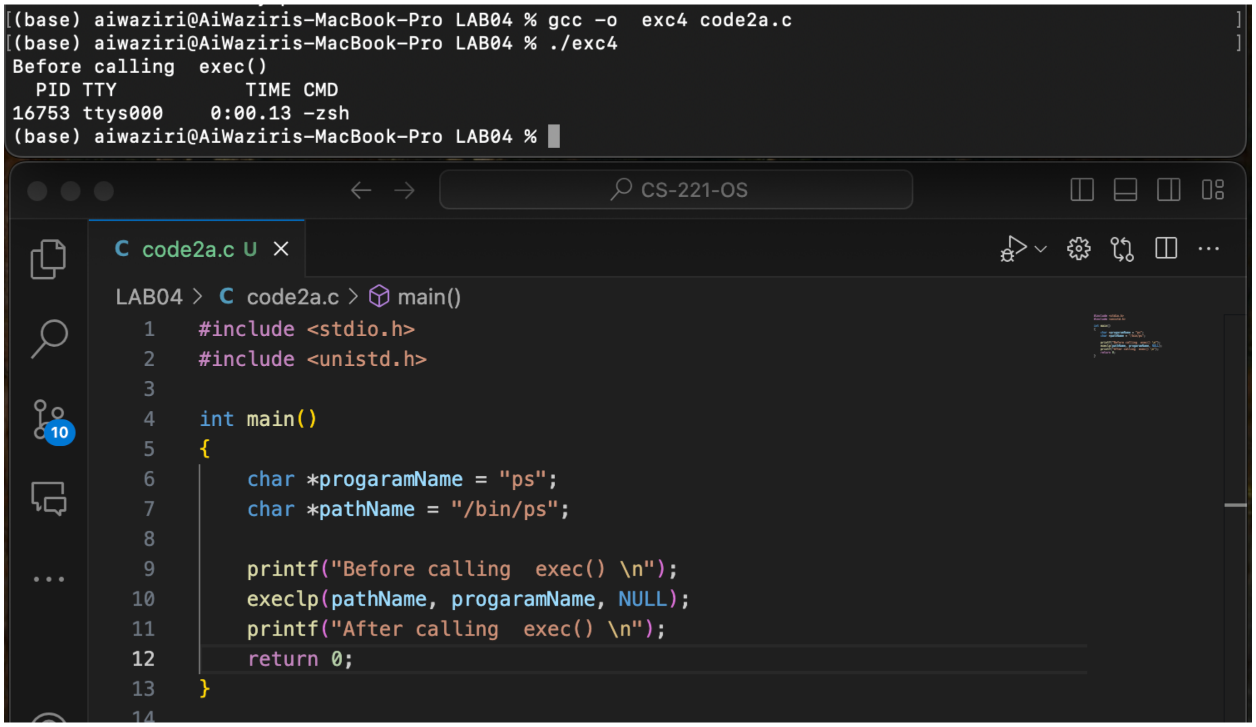
```c
#include <stdio.h>
#include <unistd.h>
#define MAX_ITER 2

int main()
{
    for (int i = 1; i <= MAX_ITER; i++)
    {
        fork();
        printf("Welcome to Madinah! My pid is %d.\n", getpid());
    }
    return 0;
}
```

Explanation of Outputs:

- After the first iteration, there are 2 lines of output.
- After the second iteration, there are 4 lines of output.
- In total, there will be (2^2 = 4) lines of output for each iteration, resulting in (2 + 4 = 6) lines of output.

Each process prints its PID, and due to the fork() calls, the number of processes doubles with each iteration of the loop.

Q2a

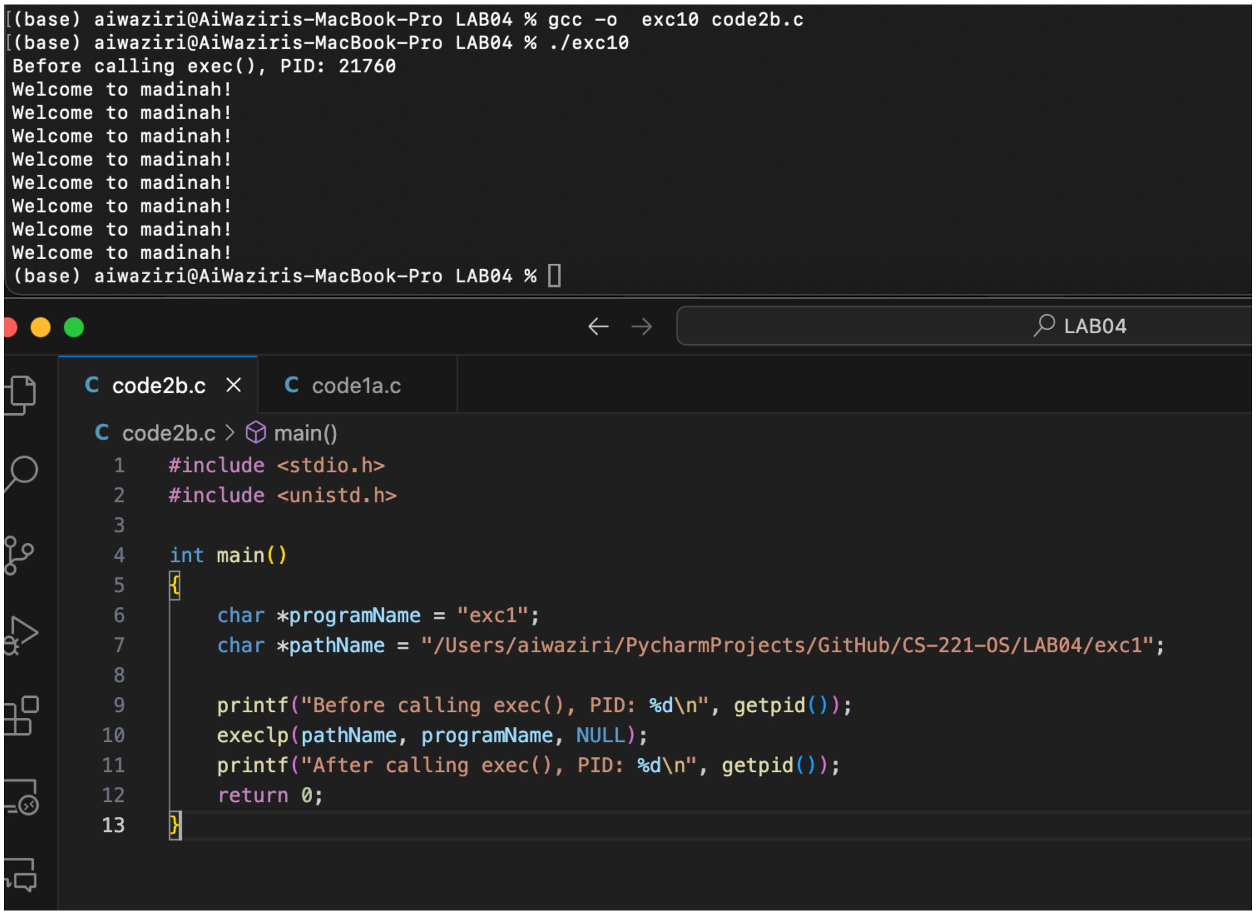


```
(base) aiwaziri@AiWaziris-MacBook-Pro LAB04 % gcc -o  exc4 code2a.c
(base) aiwaziri@AiWaziris-MacBook-Pro LAB04 % ./exc4
Before calling  exec()
  PID TTY           TIME CMD
16753 ttys000    0:00.13 -zsh
(base) aiwaziri@AiWaziris-MacBook-Pro LAB04 %
```

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
    char *prograamName = "ps";
    char *pathName = "/bin/ps";

    printf("Before calling  exec() \n");
    execlp(pathName, prograamName, NULL);
    printf("After calling  exec() \n");
    return 0;
}
```

The message "After calling exec()" is not displayed because ***execlp*** successfully replaces the current process with the ***ps*** command.

Q2b



```
[(base) aiwaziri@AiWaziris-MacBook-Pro LAB04 % gcc -o  exc10 code2b.c
[(base) aiwaziri@AiWaziris-MacBook-Pro LAB04 % ./exc10
Before calling exec(), PID: 21760
Welcome to madinah!
Welcome to madinah!
Welcome to madinah!
Welcome to madinah!
Welcome to madinah!
Welcome to madinah!
Welcome to madinah!
Welcome to madinah!
(base) aiwaziri@AiWaziris-MacBook-Pro LAB04 %
```

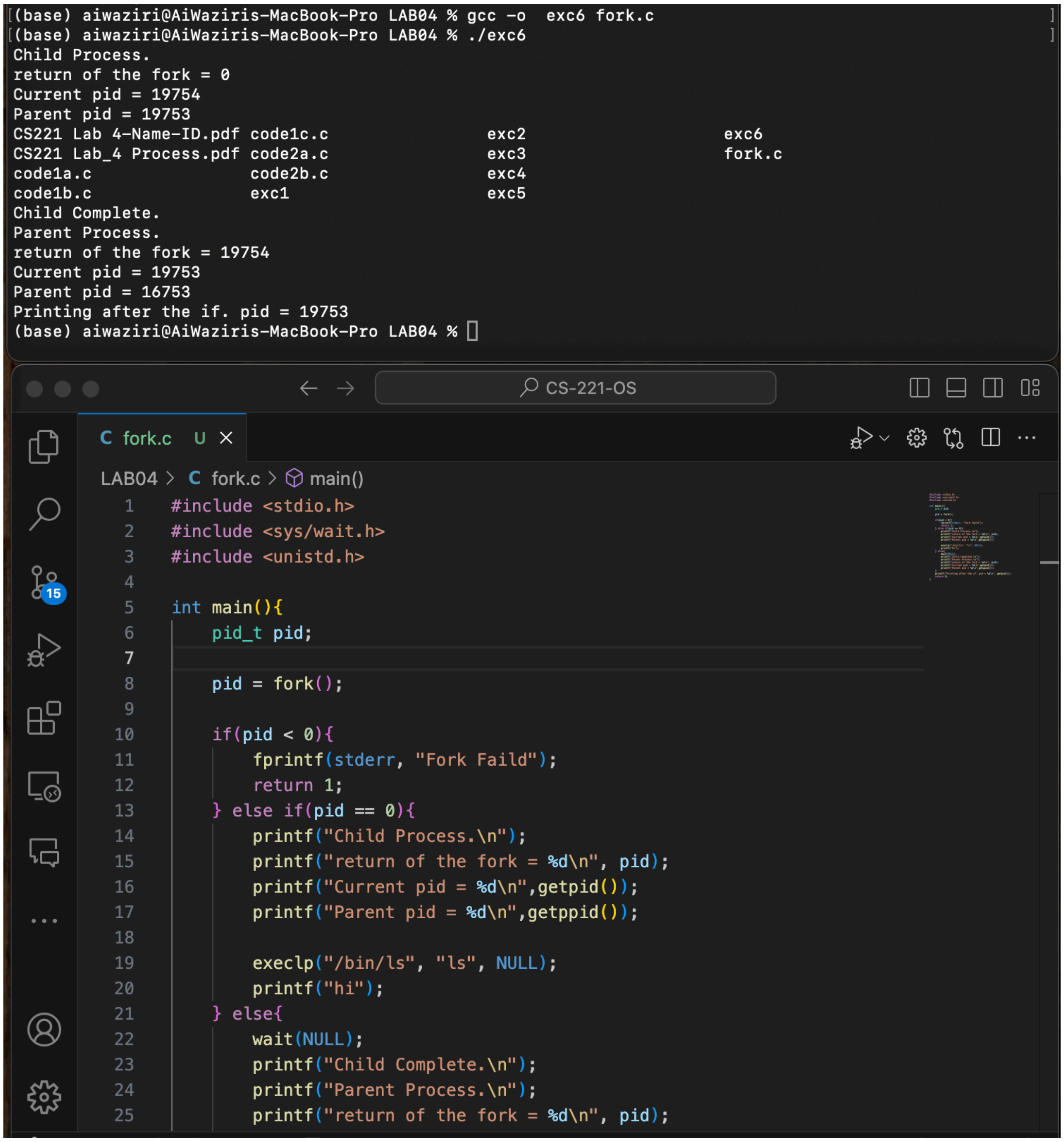
```c
#include <stdio.h>
#include <unistd.h>

int main()
{
    char *programName = "exc1";
    char *pathName = "/Users/aiwaziri/PycharmProjects/GitHub/CS-221-OS/LAB04/exc1";

    printf("Before calling exec(), PID: %d\n", getpid());
    execlp(pathName, programName, NULL);
    printf("After calling exec(), PID: %d\n", getpid());
    return 0;
}
```

Explanation

- The ***getpid()*** function is used to get the process ID of the current process.
- The ***execlp*** function replaces the current process image with a new process image specified by the given path *(./code2a)*.
- If ***execlp*** is successful, it does not return to the original program; instead, the new program *(./code2a)* starts executing.
- The process ID remains the same because ***execlp*** does not create a new process; it replaces the current process image with the new one.

- Therefore….. the PID before and after calling ***execlp*** will be the same, but the "After calling exec()" message will not be displayed if ***execlp*** is successful.

Q3a



```
[(base) aiwaziri@AiWaziris-MacBook-Pro LAB04 % gcc -o  exc6 fork.c
[(base) aiwaziri@AiWaziris-MacBook-Pro LAB04 % ./exc6
Child Process.
return of the fork = 0
Current pid = 19754
Parent pid = 19753
CS221 Lab 4-Name-ID.pdf code1c.c                exc2                    exc6
CS221 Lab_4 Process.pdf code2a.c                exc3                    fork.c
code1a.c                code2b.c                exc4
code1b.c                exc1                    exc5
Child Complete.
Parent Process.
return of the fork = 19754
Current pid = 19753
Parent pid = 16753
Printing after the if. pid = 19753
(base) aiwaziri@AiWaziris-MacBook-Pro LAB04 % 
```

```c
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>

int main(){
    pid_t pid;

    pid = fork();

    if(pid < 0){
        fprintf(stderr, "Fork Faild");
        return 1;
    } else if(pid == 0){
        printf("Child Process.\n");
        printf("return of the fork = %d\n", pid);
        printf("Current pid = %d\n",getpid());
        printf("Parent pid = %d\n",getppid());

        execlp("/bin/ls", "ls", NULL);
        printf("hi");
    } else{
        wait(NULL);
        printf("Child Complete.\n");
        printf("Parent Process.\n");
        printf("return of the fork = %d\n", pid);
```

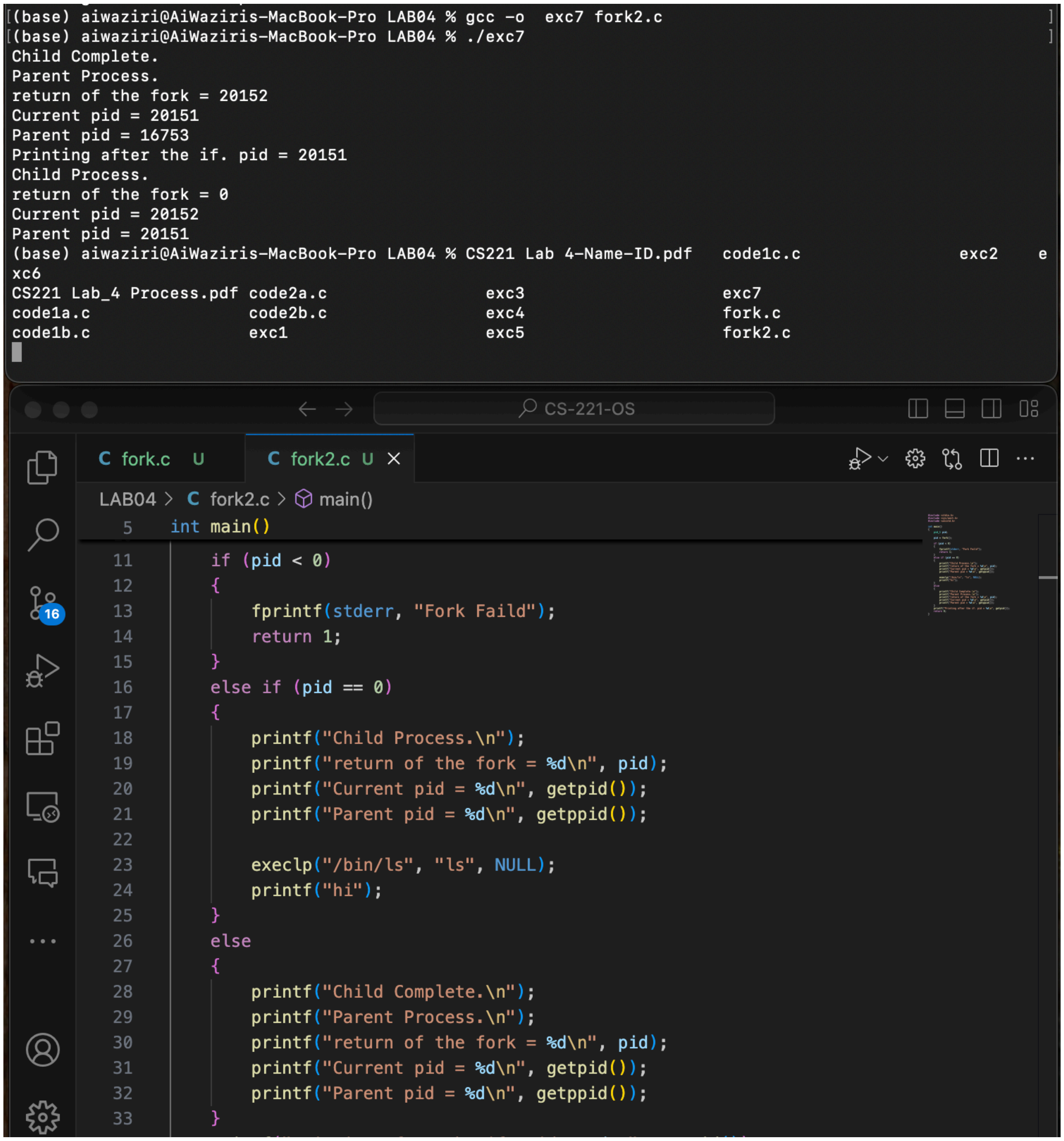
Explanation:

When fork() is called, it creates a new child process. If fork() returns 0, the child process executes, printing its process ID (PID) and its parent's PID, then replaces its memory space with the ls command
using execlp(), which lists the directory of the contents.

The printf("hi"); statement is never executed because execlp() replaces the process image. If fork() returns a positive value, the parent process waits for the child to complete using wait(NULL);, then prints a message indicating the child has completed, along with its own PID and the child's PID.

In general: The code shows how to create a child process, execute a command within it, and synchronize with the parent process.

Q3b



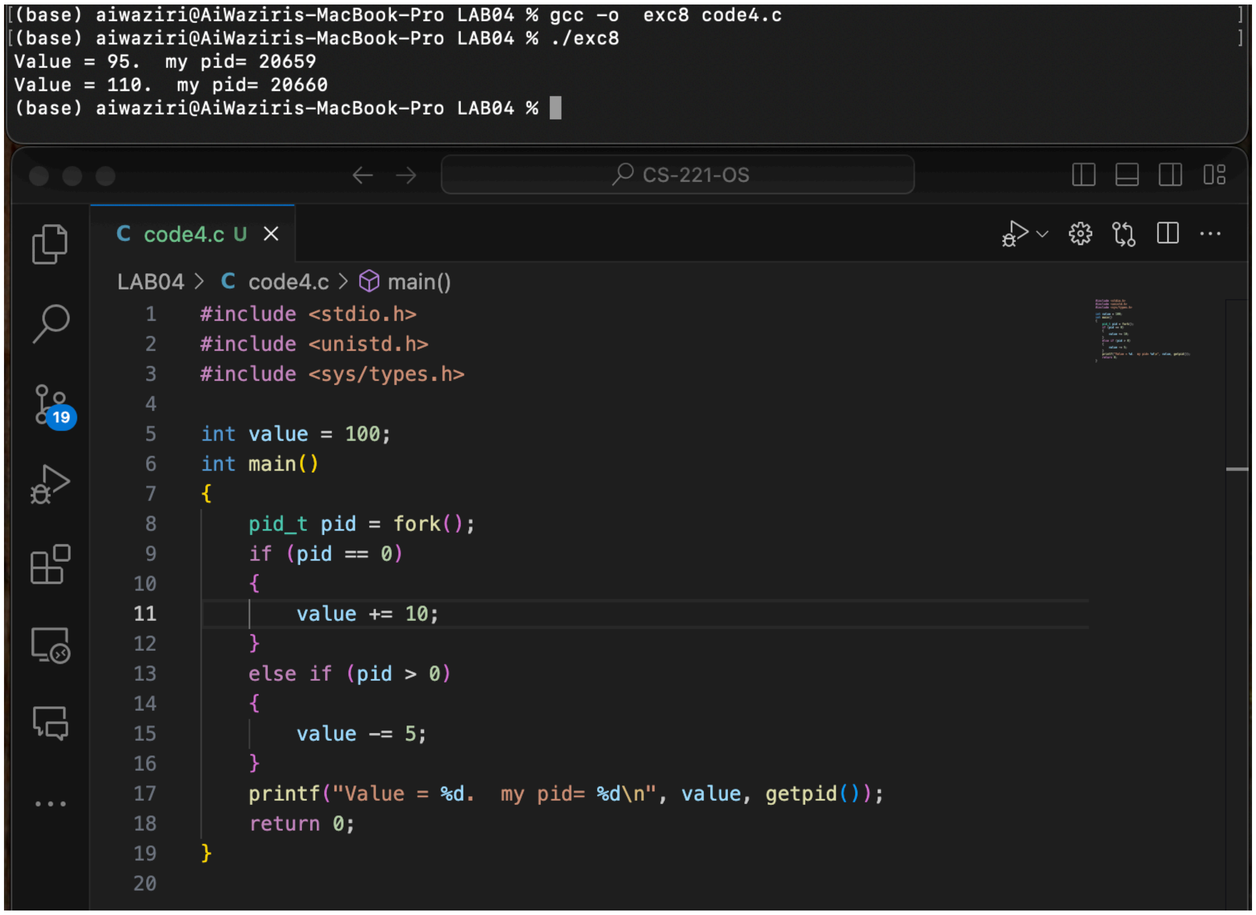
```
[(base) aiwaziri@AiWaziris-MacBook-Pro LAB04 % gcc -o  exc7 fork2.c
[(base) aiwaziri@AiWaziris-MacBook-Pro LAB04 % ./exc7
Child Complete.
Parent Process.
return of the fork = 20152
Current pid = 20151
Parent pid = 16753
Printing after the if. pid = 20151
Child Process.
return of the fork = 0
Current pid = 20152
Parent pid = 20151
(base) aiwaziri@AiWaziris-MacBook-Pro LAB04 % CS221 Lab 4-Name-ID.pdf   code1c.c                  exc2     e
xc6
CS221 Lab_4 Process.pdf code2a.c                 exc3                     exc7
code1a.c                code2b.c                 exc4                     fork.c
code1b.c                exc1                     exc5                     fork2.c
```

```c
int main()
        if (pid < 0)
        {
            fprintf(stderr, "Fork Faild");
            return 1;
        }
        else if (pid == 0)
        {
            printf("Child Process.\n");
            printf("return of the fork = %d\n", pid);
            printf("Current pid = %d\n", getpid());
            printf("Parent pid = %d\n", getppid());

            execlp("/bin/ls", "ls", NULL);
            printf("hi");
        }
        else
        {
            printf("Child Complete.\n");
            printf("Parent Process.\n");
            printf("return of the fork = %d\n", pid);
            printf("Current pid = %d\n", getpid());
            printf("Parent pid = %d\n", getppid());
        }
```

YES:

because the parent process no longer waits for the child process to complete before continuing its execution. This results in the parent and child processes running concurrently. Consequently, the parent process may print its messages before the child process finishes executing the ls command, leading to interleaved output.

Q4



```
[(base) aiwaziri@AiWaziris-MacBook-Pro LAB04 % gcc -o  exc8 code4.c
[(base) aiwaziri@AiWaziris-MacBook-Pro LAB04 % ./exc8
Value = 95.  my pid= 20659
Value = 110.  my pid= 20660
(base) aiwaziri@AiWaziris-MacBook-Pro LAB04 %
```

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int value = 100;
int main()
{
    pid_t pid = fork();
    if (pid == 0)
    {
        value += 10;
    }
    else if (pid > 0)
    {
        value -= 5;
    }
    printf("Value = %d.  my pid= %d\n", value, getpid());
    return 0;
}

```

because the parent and child processes have separate copies of the global variable value, and each process modifies its own copy independently after the fork().