

Dean Cosanella  
Flavio dos Santos-Ross  
CSE 460  
Dr. Tong Lai Yu  
Lab 10 Report

# 1. Introduction

## Exercise 0

dup2	read	write	pipe
fprintf	atoi	getpid	wait

Explain briefly in your own words what each of the above functions does.

### **dup2**

makes newfd be the copy of oldfd, closing newfd first if necessary, but note the following:

- \* If oldfd is not a valid file descriptor, then the call fails, and newfd is not closed.
- \* If oldfd is a valid file descriptor, and newfd has the same value as oldfd, then dup2() does nothing, and returns newfd.

### **fprintf**

write output to stdout, the standard output stream

### **read**

attempts to read up to count bytes from file descriptor fd into the buffer starting at buf.

### **atoi**

converts the initial portion of the string pointed to by nptr to int. The behavior is the same as strtol(nptr, NULL, 10); except that it does not detect errors.

### **write**

utility allows you to communicate with other users, by copying lines from your terminal to theirs.

### **getpid**

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

### **pipe**

creates a pipe, a unidirectional data channel that can be used for interprocess communication.

Dean Cosanella  
Flavio dos Santos – Ross  
CSE 460

## wait

used to wait for state changes in a child of the calling process, and obtain information about the child whose state has changed.

## 2. Forming a Ring

### Exercise 1

What happens if, after connecting standard output to standard input via a pipe, the process of code executes the following code segment? Explain.

```
int n = 98;
for (int i = 0; i < 10; i++)
{
    write (STDOUT_FILENO, &i, sizeof(i));
    read (STDIN_FILENO, &n, sizeof(n));
    fprintf(stderr, "%d\n", n);
}
```



The screenshot shows a terminal window titled "Terminal" with the following commands and output:

```
dean@dean-pc ~/460/lab10 $ g++ -o ring ring.cpp
dean@dean-pc ~/460/lab10 $ ./ring
0
1
2
3
4
5
6
7
8
9
dean@dean-pc ~/460/lab10 $
```

The program outputs the numbers 0 through 9, one per line. The prompt for the next line is visible, indicating the program has finished execution.

Dean Cosanella  
Flavio dos Santos – Ross  
CSE 460

The screenshot above is the output for the code given in exercise 1. We believe that the code outputs the number 0 through 9 because the write function reads the current value of “i” and the read function reads that value and stores it in “n”. Then the fprintf function prints out the value of “n”.

### Exercise 2a

What happens if the code in Exercise 1 is replaced by the following code? Explain.

```
int n = 98;
for (int i = 0; i < 10; i++)
{
    read (STDIN_FILENO, &n, sizeof(n));
    write (STDOUT_FILENO, &i, sizeof(i));
    fprintf(stderr, "%d\n", n);
}
```

Hint: The program hangs on the first read. Why?

**When this code is executed, it gets stuck on the first read command. The reason why this happens is because the program is trying to read from a file that is empty. In order for this code to work properly, the file must be written to before it is read from.**

### Exercise 2b

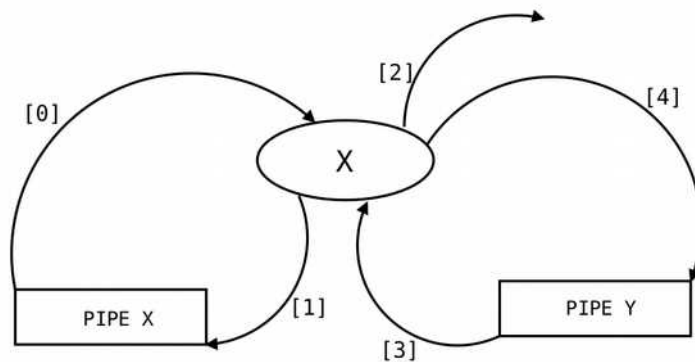
What happens if the code in Exercise 1 is replaced by the following code? Explain.

```
int n = 98;
for (int i = 0; i < 10; i++)
{
    printf("%d\n", i);
    scanf ("%d\n", &n);
    fprintf(stderr, "%d\n", n);
}
```

**When the code above is executed, it gets stuck on the scanf command because the pipe reading and writing are fully buffered and the function printf does not write anything to the pipe until the buffer is full. This is a similar problem in the previous exercise where the code gets stuck on the first read command.**

### Exercise 3

Draw a diagram like those of Figure 2-4 to show the connections of the processes after the second pipe (fd) has been executed. Your diagrams should show pipe x, pipe y, the two processes (parent and child) and the connections with file descriptors labeled along the edges.



	Process X fd Table
[0]	pipe x read
[1]	pipe x write
[2]	standard error
[3]	pipe y read
[4]	pipe y write

### Exercise 4

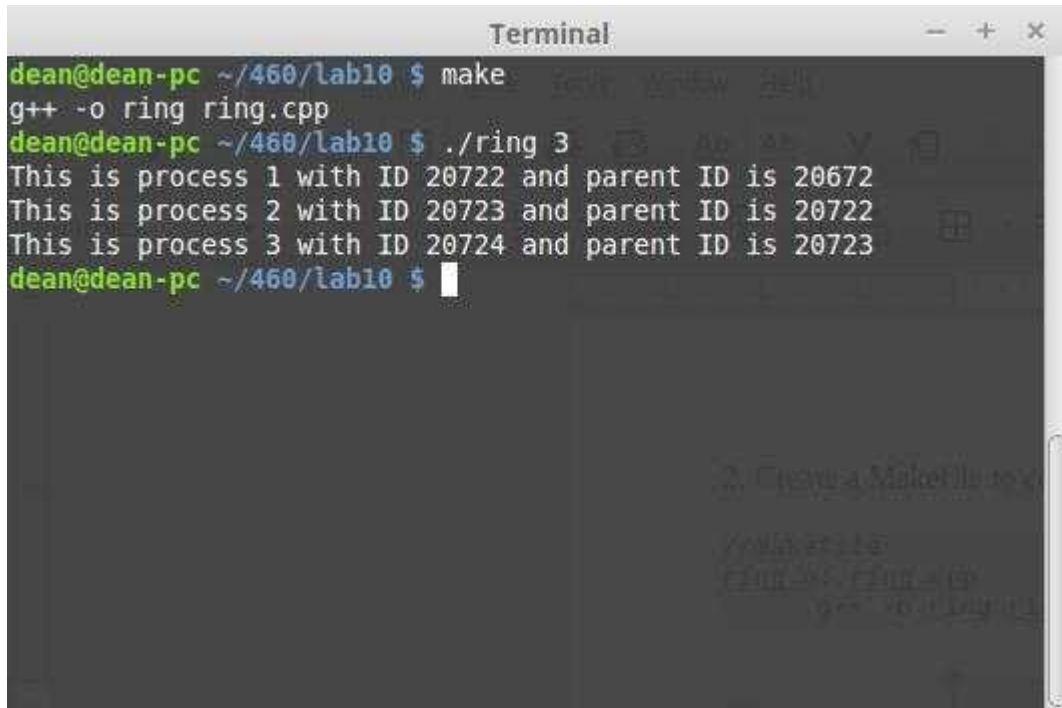
1. Compile and run the program “ring.cpp” shown in Listing 1.

```
Terminal
dean@dean-pc ~/460/lab10 $ g++ -o ring ring.cpp
dean@dean-pc ~/460/lab10 $ ./ring 3
This is process 1 with ID 22399 and parent ID is 22352
This is process 2 with ID 22400 and parent ID is 22399
This is process 3 with ID 22401 and parent ID is 22400
dean@dean-pc ~/460/lab10 $
```

Dean Cosanella  
Flavio dos Santos – Ross  
CSE 460

2. Create a Makefile to compile the program.

```
//makefile  
ring.o: ring.cpp  
g++ -o ring ring.cpp
```

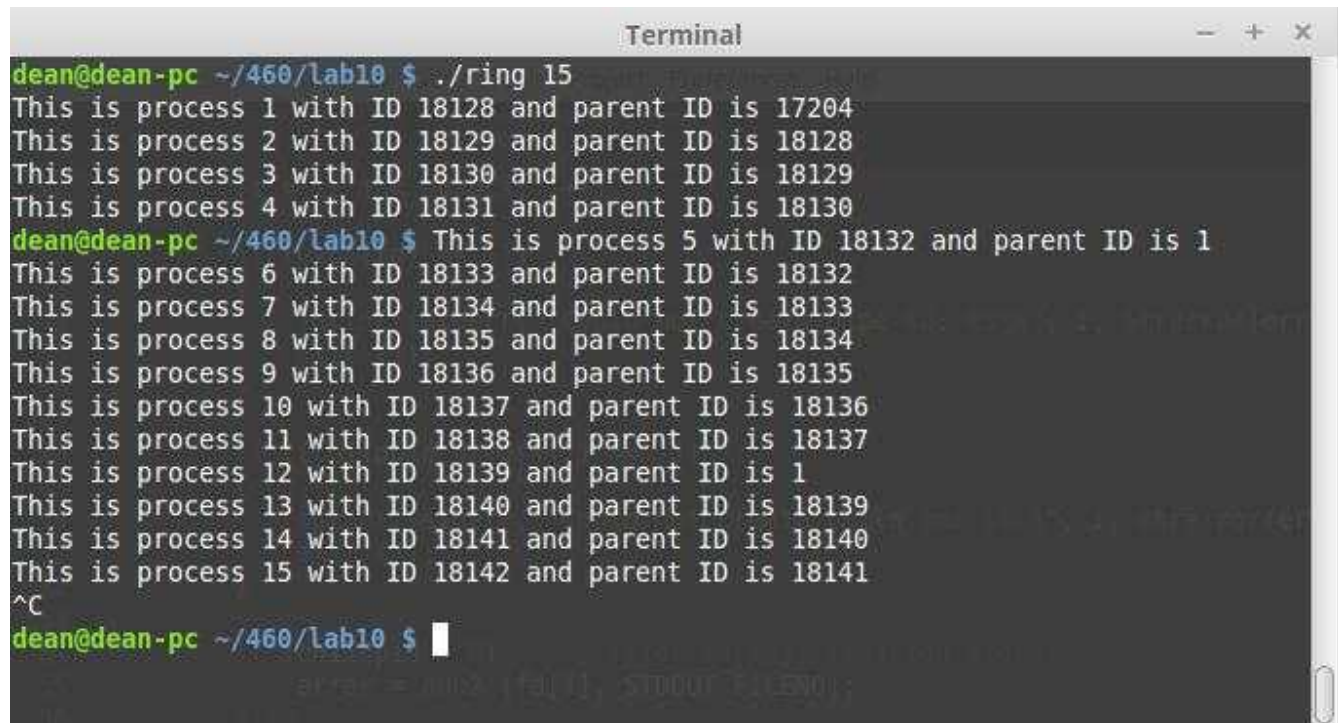


A terminal window titled "Terminal" with standard window controls (-, +, x). The prompt is "dean@dean-pc ~/460/lab10". The user enters "make", which outputs "g++ -o ring ring.cpp". Then, the user enters "./ring 3", which outputs three lines: "This is process 1 with ID 20722 and parent ID is 20672", "This is process 2 with ID 20723 and parent ID is 20722", and "This is process 3 with ID 20724 and parent ID is 20723". The prompt returns to "dean@dean-pc ~/460/lab10 \$".

```
dean@dean-pc ~/460/lab10 $ make  
g++ -o ring ring.cpp  
dean@dean-pc ~/460/lab10 $ ./ring 3  
This is process 1 with ID 20722 and parent ID is 20672  
This is process 2 with ID 20723 and parent ID is 20722  
This is process 3 with ID 20724 and parent ID is 20723  
dean@dean-pc ~/460/lab10 $
```

Dean Cosanella  
Flavio dos Santos – Ross  
CSE 460

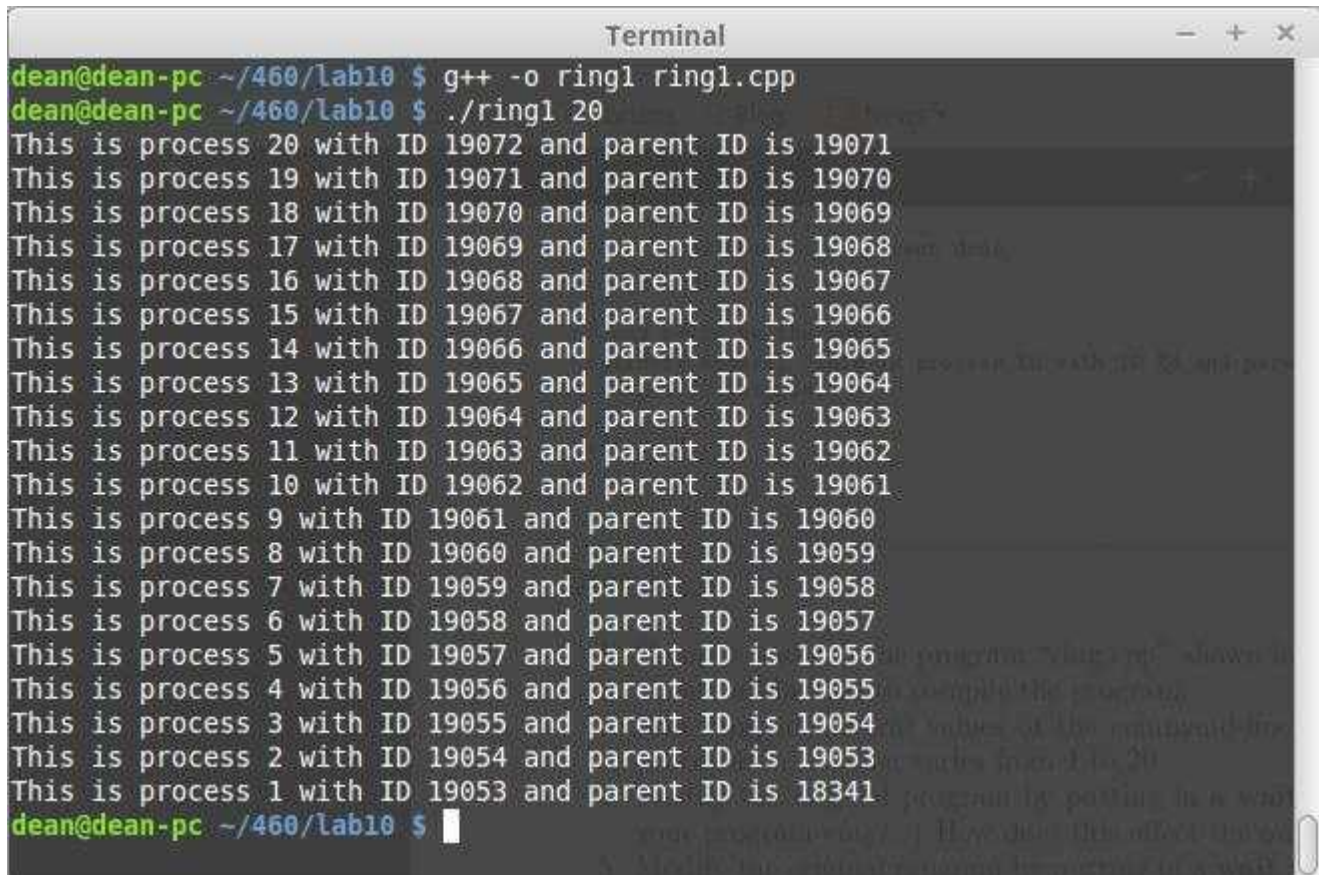
3. Run ring for several values of the command-line argument and observe what happens as the number of processes in the ring varies from 1 to 20.



```
Terminal
dean@dean-pc ~/460/lab10 $ ./ring 15
This is process 1 with ID 18128 and parent ID is 17204
This is process 2 with ID 18129 and parent ID is 18128
This is process 3 with ID 18130 and parent ID is 18129
This is process 4 with ID 18131 and parent ID is 18130
dean@dean-pc ~/460/lab10 $ This is process 5 with ID 18132 and parent ID is 1
This is process 6 with ID 18133 and parent ID is 18132
This is process 7 with ID 18134 and parent ID is 18133
This is process 8 with ID 18135 and parent ID is 18134
This is process 9 with ID 18136 and parent ID is 18135
This is process 10 with ID 18137 and parent ID is 18136
This is process 11 with ID 18138 and parent ID is 18137
This is process 12 with ID 18139 and parent ID is 1
This is process 13 with ID 18140 and parent ID is 18139
This is process 14 with ID 18141 and parent ID is 18140
This is process 15 with ID 18142 and parent ID is 18141
^C
dean@dean-pc ~/460/lab10 $
```

Dean Cosanella  
Flavio dos Santos – Ross  
CSE 460

4. Modify the original program by putting in a wait system call before the final fprintf statement. (Name your program ring1.) How does this affect the output of the program?



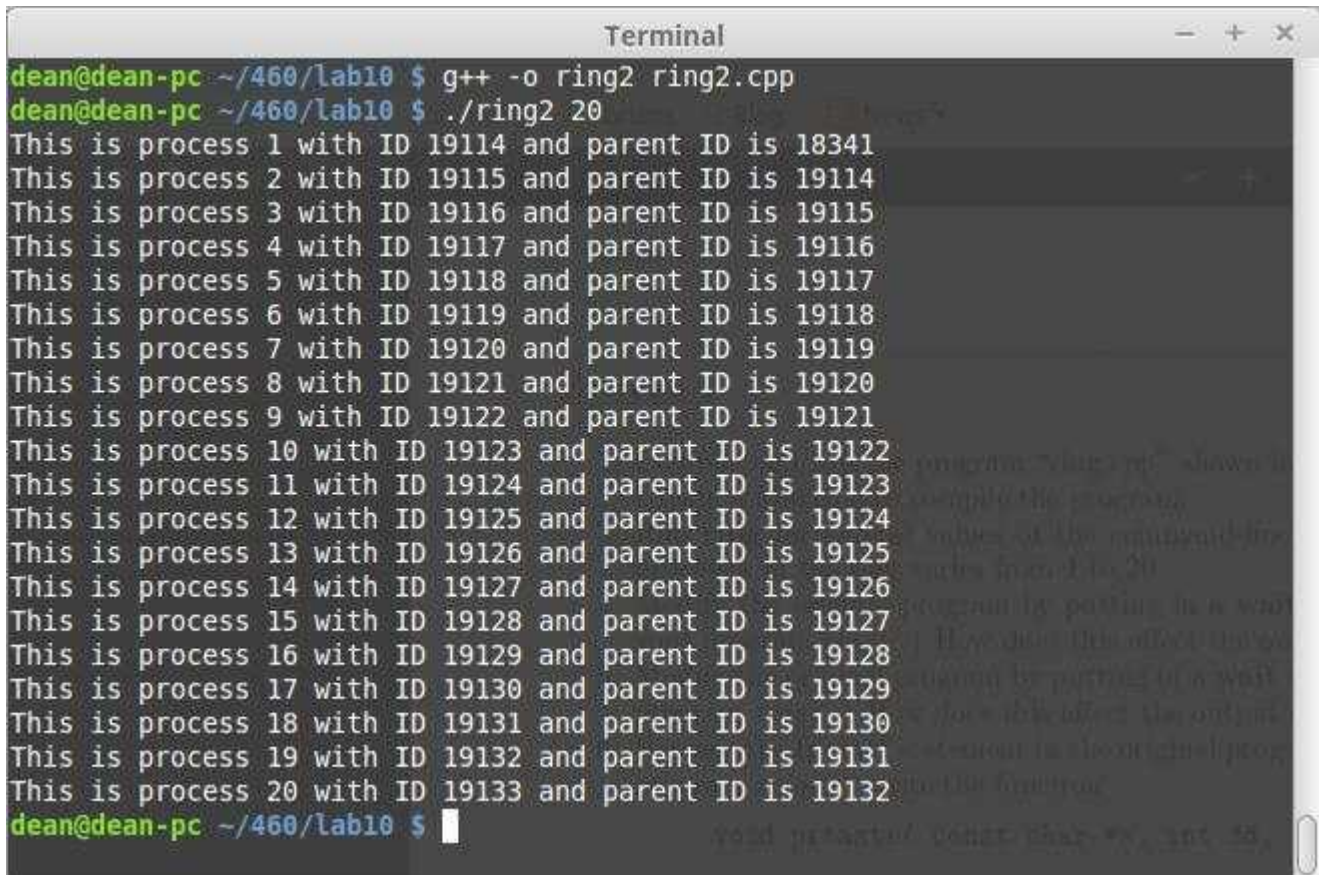
```
Terminal
dean@dean-pc ~/460/lab10 $ g++ -o ring1 ring1.cpp
dean@dean-pc ~/460/lab10 $ ./ring1 20
This is process 20 with ID 19072 and parent ID is 19071
This is process 19 with ID 19071 and parent ID is 19070
This is process 18 with ID 19070 and parent ID is 19069
This is process 17 with ID 19069 and parent ID is 19068
This is process 16 with ID 19068 and parent ID is 19067
This is process 15 with ID 19067 and parent ID is 19066
This is process 14 with ID 19066 and parent ID is 19065
This is process 13 with ID 19065 and parent ID is 19064
This is process 12 with ID 19064 and parent ID is 19063
This is process 11 with ID 19063 and parent ID is 19062
This is process 10 with ID 19062 and parent ID is 19061
This is process 9 with ID 19061 and parent ID is 19060
This is process 8 with ID 19060 and parent ID is 19059
This is process 7 with ID 19059 and parent ID is 19058
This is process 6 with ID 19058 and parent ID is 19057
This is process 5 with ID 19057 and parent ID is 19056
This is process 4 with ID 19056 and parent ID is 19055
This is process 3 with ID 19055 and parent ID is 19054
This is process 2 with ID 19054 and parent ID is 19053
This is process 1 with ID 19053 and parent ID is 18341
dean@dean-pc ~/460/lab10 $
```

When the wait system call is placed before the last fprintf statement, it appears that the processes are listed in reversed order. So the first process that is listed is process 20 and lists the pid and ppid and goes down to the first process and exits.



Dean Cosanella  
Flavio dos Santos – Ross  
CSE 460

5. Modify the original program by putting in a wait system call after the final fprintf statement. (Name your program ring2.) How does this affect the output of the program?

A terminal window titled "Terminal" with standard window controls (minimize, maximize, close). The prompt is "dean@dean-pc ~/460/lab10". The user enters "g++ -o ring2 ring2.cpp" and then "./ring2 20". The output consists of 20 lines, each stating "This is process X with ID Y and parent ID is Z", where X ranges from 1 to 20, Y ranges from 19114 to 19133, and Z is the ID of the parent process. The final prompt is "dean@dean-pc ~/460/lab10 \$".

```
dean@dean-pc ~/460/lab10 $ g++ -o ring2 ring2.cpp
dean@dean-pc ~/460/lab10 $ ./ring2 20
This is process 1 with ID 19114 and parent ID is 18341
This is process 2 with ID 19115 and parent ID is 19114
This is process 3 with ID 19116 and parent ID is 19115
This is process 4 with ID 19117 and parent ID is 19116
This is process 5 with ID 19118 and parent ID is 19117
This is process 6 with ID 19119 and parent ID is 19118
This is process 7 with ID 19120 and parent ID is 19119
This is process 8 with ID 19121 and parent ID is 19120
This is process 9 with ID 19122 and parent ID is 19121
This is process 10 with ID 19123 and parent ID is 19122
This is process 11 with ID 19124 and parent ID is 19123
This is process 12 with ID 19125 and parent ID is 19124
This is process 13 with ID 19126 and parent ID is 19125
This is process 14 with ID 19127 and parent ID is 19126
This is process 15 with ID 19128 and parent ID is 19127
This is process 16 with ID 19129 and parent ID is 19128
This is process 17 with ID 19130 and parent ID is 19129
This is process 18 with ID 19131 and parent ID is 19130
This is process 19 with ID 19132 and parent ID is 19131
This is process 20 with ID 19133 and parent ID is 19132
dean@dean-pc ~/460/lab10 $
```

When the wait system call is placed after the final fprintf statement, the processes are printed out in order and list the attributes of each process's pid and ppid. The key difference from this and the original ring.cpp program is that this version waits for each process to be completed before moving on to the next one.

Dean Cosanella  
Flavio dos Santos – Ross  
CSE 460

6. Replace the `fprintf` statement in the original program, `rin`, with calls to `sprintf` and `prtastr`. (name your program `ring3`.) write the function

```
void prtastr (const char *s, int fd, int n);
```

which prints the `s` string one character at a time to the file specified by the descriptor `fd` using `write`. After each character is output, `prtastr` executes the following loop:

```
for (i = 0; i < n; i++);
```

This loop just wastes some CPU time. Use `prtastr` to output the string to the standard error. The value of `n` is passed as an optional command-line argument to `ring3`. The default value for this parameter is 1. Run the program with a value of `n` that causes a small, but barely noticeable, delay between the output of characters.

**Below is our `ring3.cpp` code:**

```
//ring3.cpp
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <iostream>
#include <cstdio>
#include <sys/wait.h>

using namespace std;

void prtastr (const char *s, int fd, int n)
{
    int i = 0;
    int j = 0;
    while (s[i] != '\0')
    {
        if ((write (fd, &s[i++], 1)) == -1)
            fprintf(stderr, "write error\n");
        for (j = 0; j < n; j++);
    }
}

int main (int argc, char *argv[])
{
    int fd[2];
    int childpid;           //process id of child
    int nprocs;             //total number of processes in the ring
    int error;              //return value from dup2 call
    int i;
```

Dean Cosanella  
Flavio dos Santos – Ross  
CSE 460

```
int n;
char buffer [100];

if (argc < 2 || (nprocs = atoi(argv[1])) <= 0)
{
    cout << "Usage: " << argv[0] << " nprocesses" << endl;
    exit (1);
}

if (pipe (fd) == -1)
{
    perror("Could not create pipe");
    exit (1);
}

if (dup2 (fd[0], STDIN_FILENO) == -1 || dup2 (fd[1], STDOUT_FILENO) == -1)
{
    perror("Could not dup pipes");
    exit (1);
}
close (fd[0]);
close (fd[1]);

for (i = 1; i < nprocs; i++)
{
    if (pipe(fd) == -1)
    {
        fprintf(stderr, "Could not create pipe %d: %s\n", i,
strerror(errno));
        exit (1);
    }

    if ((childpid = fork()) == -1)
    {
        fprintf(stderr, "Could not create child %d: %s\n", i,
strerror(errno));
        exit (1);
    }

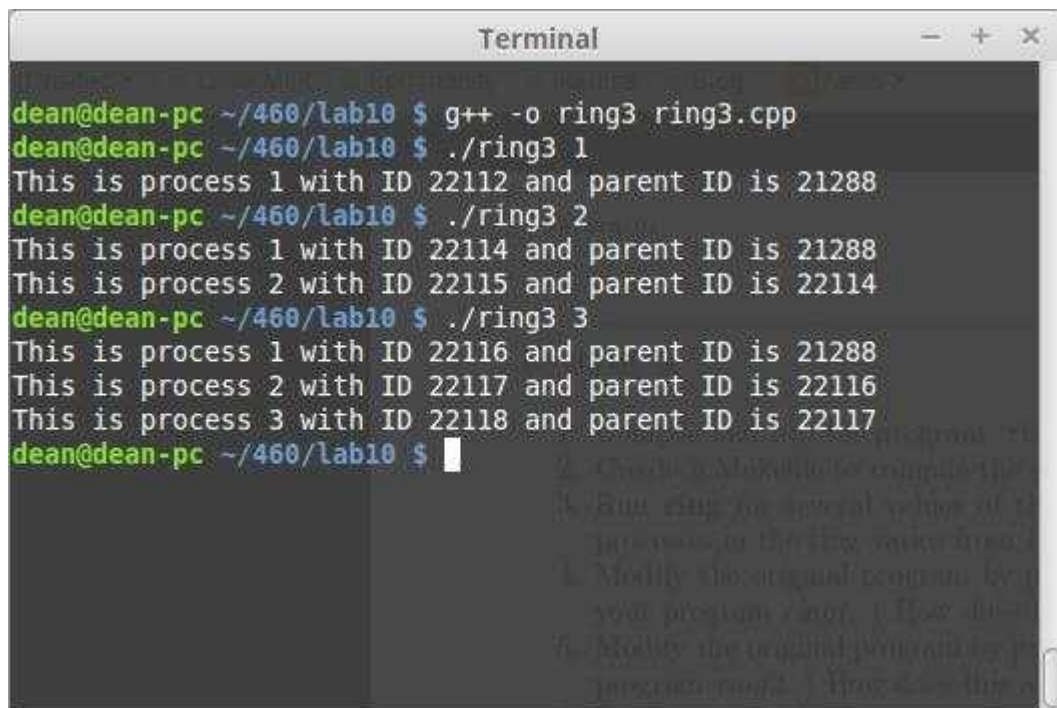
    if (childpid > 0)                //for parent, reassign stdout
        error = dup2 (fd[1], STDOUT_FILENO);
    else
        error = dup2 (fd[0], STDIN_FILENO);

    if (error == -1)
    {
        fprintf(stderr, "Could not dup pipes for iteration %d:
%s\n", i, strerror(errno));
        exit (1);
    }
    close (fd[0]);
    close (fd[1]);
    if (childpid)                    //parent done
        break;
}
```

Dean Cosanella  
Flavio dos Santos – Ross  
CSE 460

```
}  
    //say hello to the world  
  
    sprintf(buffer, "This is process %d with ID %d and parent ID is %d\n", i,  
(int)getpid(), (int)getppid());  
    prtrastr(buffer, STDERR_FILENO, n);  
    wait((int *) 0);  
  
    return 0;  
}
```

The following screenshot is our output for ring3.cpp:



```
Terminal  
dean@dean-pc ~/460/lab10 $ g++ -o ring3 ring3.cpp  
dean@dean-pc ~/460/lab10 $ ./ring3 1  
This is process 1 with ID 22112 and parent ID is 21288  
dean@dean-pc ~/460/lab10 $ ./ring3 2  
This is process 1 with ID 22114 and parent ID is 21288  
This is process 2 with ID 22115 and parent ID is 22114  
dean@dean-pc ~/460/lab10 $ ./ring3 3  
This is process 1 with ID 22116 and parent ID is 21288  
This is process 2 with ID 22117 and parent ID is 22116  
This is process 3 with ID 22118 and parent ID is 22117  
dean@dean-pc ~/460/lab10 $
```

When we executed the program by passing it an integer of 3, we noticed a slight delay between the output of characters.

In this lab, we learned a lot about creating rings as well as different functions that can be used in our own programs. These functions will be very helpful on future projects we might have. Because we were able to complete the entire lab, we give ourselves a score of 20/20.