Joseph Pennington
2912079
EECS 678 Lab 05 Report

Fork.c

1. Which process prints this line? What is printed?
    a. The child process prints the line. “After fork, Process id = 2225” is printed.
2. What will be printed if this line is commented?

```
~Lab05$ ./fork
After fork, Process id = 2279
In Parent: 2278, Child id: 2279
Final statement from Process: 2278
~Lab05$ ./fork
After fork, Process id = 0
Print after excelp
Final statement from Process: 2279
```

3. When is this line reached/printed?
    a. The line is not printed if the execlp command is executed. If the execlp command is commented out, then it prints “Print after execlp”.
4. What happens if the parent process is killed first? Uncomment the next two lines.
    a. If the parent process is killed first, after the fork, the process ID of the parent changes.

Mfork.C

1. How many processes are created? Explain.
    a. Seven processes were created in addition to the parent process. After each fork command is executed, a new process is created, and these child processes continue executing the remaining fork commands in a tree-like pattern. Also, there were 7 lines printed to the screen.

Pipe-sync.c

1. Update the source code to guarantee the following sequence: child line 1, parent line 1, child line 2, parent line 2.

```
int main()
{
  char *s, buf[1024];
 int ret, stat;
  s  = "Use Pipe for Process Synchronization\n";

  /* create pipe */
  int pipe1[2];
  pipe(pipe1);
  int pipe2[2];
  pipe(pipe2);

  ret = fork();
  if (ret == 0) {

    /* child process. */

    printf("Child line 1\n");

    close(pipe1[0]);
    write(pipe1[1],buf,strlen(s));
    close(pipe2[1]);
    read(pipe2[0],buf,strlen(s));

    printf("Child line 2\n");

    close(pipe1[0]);
    write(pipe1[1],buf,strlen(s));

  } else {
    /* parent process */

    close(pipe1[1]);
    read(pipe1[0],buf,strlen(s));

    printf("Parent line 1\n");

    close(pipe2[0]);
    write(pipe2[1],buf,strlen(s));
    close(pipe1[1]);
    read(pipe1[0],buf,strlen(s));

    printf("Parent line 2\n");
    wait(&stat);
  }
}
```

Fifo_producer.c & Fifo_consumer.c

1. Updated Code

Fifo_producer.c updated code:

```
main()
{
  char str[MAX_LENGTH];
  int num, fd;

  /* create a FIFO special file with name FIFO_NAME */
  mkfifo(FIFO_NAME, 0666);
  /* open the FIFO file for writing. open() blocks for readers */
  printf("waiting for readers...");
  fflush(stdout);
  fd = open(FIFO_NAME, O_WRONLY);
  printf("got a reader !\n");
  printf("Enter text to write in the FIFO file: ");
  fgets(str, MAX_LENGTH, stdin);
  while(!(feof(stdin))){
    if ((num = write(fd, str, strlen(str))) == -1)
      perror("write");
    else
      printf("producer: wrote %d bytes\n", num);
    fgets(str, MAX_LENGTH, stdin);
  }
}
```

Fifo_consumer.c updated code:

```
main()
{
  char str[MAX_LENGTH];
  int num, fd;

  /* create a FIFO special file with name FIFO_NAME */
  mkfifo(FIFO_NAME, 0666);
  /* open the FIFO file for reading. open() blocks for writers. */
  printf("waiting for writers...");
  fflush(stdout);
  fd = open(FIFO_NAME, O_RDONLY);
  printf("got a writer !\n");
  do{
    if((num = read(fd, str, MAX_LENGTH)) == -1)
      perror("read");
    else{
      str[num] = '\0';
      printf("consumer: read %d bytes\n", num);
      printf("%s", str);
    }
  }while(num > 0);
}
```

2. Compile programs
    a. Gcc -o fifo_producer.exe fifo_producer.c
    b. Gcc -o fifo_consumer.exe fifo_consumer.c
3. Open 4 terminals and answer questions
    a. What happens if you only launch a producer (but no consumer)?
        i. If you only launch a producer, it will wait for readers
    b. What happens if you only launch a consumer (but no producers)?
        i. If you only launch a producer, it will wait for writers.
    c. If one producer and multiple consumers, then who gets the message sent?
        i. If there are multiple consumers, it is unknown who gets the message. However, the message will go to one of the open consumers.
    d. Does the producer continue writing messages into the fifo, if there are no consumers?
        i. No, the producer does not continue writing messages to the fifo if there are no consumers. It will continue to wait for a reader.
    e. What happens to the consumers, if all the producers are killed?
        i. All the consumers are killed as well.

Shared_memory3.c

1. Explain the output
    a. The shared memory is received by using the shmget() function, and it is attached to the process space as a character array. The memory is not altered before the fork is created so that explains why the first two outputs are “First String”. The fork creates a child process and calls the function “do_child” which prints the original values of the two buffers, “First String”. It then updates the two buffers with STR2. This explains the third and fourth outputs. After the child process ends, the parent prints out the contents of the shared and local memory. This explains why the shared memory is “Second String” because the child changed that memory space. It also explains why the unshared memory is still “First String” because that memory is unique to the parent process. These are the final two outputs of the program.

Thread-1.c

1. Are changes made to the local or global variables by the child process reflected in the parent process? Explain.
    a. No, the changes made are not reflected in the parent process. This is because the child process has separate copies of the global and local variables from the parent process.
2. Are changes made to the local or global variables by the child thread reflected in the parent process? Separately explain what happens for the local and global variables.
    a. Yes, the changes made by the child thread are reflected in the parent process. For the global variable, the thread calls the child_fn function and changes the global variable to 678. It also changes the local variable to 100. This is caused by the

thread sharing resources within the same process. Reading and writing to the same memory location is possible.