Manvir Kaur

Lab 04 Questions

Question 1:

1. The parent process prints this line. “After fork, Process id = 31525” is printed

2. print after execlp
   Final statement from Process: 31549

3. It is reached only when line 35 containing execlp is commented out. This is because execlp overwrites the address space of the process in which it is called with a new executable.

4. The OS assigns the child process to a new parent. The OS assigned the child process to the init process with pid 1 resulting in the print statement:
   “In Child: 31572, Parent: 1”

Question 2:

1. There are 8 processes created because there are 3 fork system calls when the program is run. The total number of processes created is given by 2^n, where n is the number of forks. Therefore, for 3 forks, $2^n = 2^3 = 8$ meaning there are 8 processes created in total.

Question 3

1. In the main function:

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

        /* create pipe */
        int p1[2], p2[2], p3[2];        //file descriptors
        pipe(p1);
        pipe(p2);
        pipe(p3);
        ret = fork();
        if (ret == 0) {
                /* child process. */
                printf("Child line 1\n");
                write(p1[1],s,strlen(s));
```

```
        read(p2[0],buf,strlen(s));
        printf("Child line 2\n");
        write(p3[1],s,strlen(s)); }
else {
        /* parent process */
        read(p1[0],buf,strlen(s));
        printf("Parent line 1\n");
        write(p2[1], s, strlen(s));
        read(p3[0],buf,strlen(s));
        printf("Parent line 2\n");
        wait(&stat); } }
```

Question 4:

1. The program is blocking until a writer becomes available. It prints “waiting for readers . . .”

2. Similar to before, the program is blocking until a writer is available. It prints “waiting for writers . . .” to show this.

3. The consumers read the FIFO in a FIFO manner one after the other. This means that they start with the first consumer that was created, followed by the second, and so on. What was created first is read first.

4. No, the producer terminates if there are no more consumers left.

5. The consumers are automatically terminated as well.

Question 5:

1. The code provided in shared_memory3.c uses a child process to update a shared memory segment and an unshared memory segment in order to show the distinction between the two. First, the do_child() function is called. This results in the outputs
   “shared_buf before fork: First String” and “unshared_buf before fork: First String”
   These two lines were printed by the parent process before the fork() system call. This shows us that the contents of the shared and unshared are the same value that is used to initialize them later on in the code in lines 52 and 53. The parent process then has to wait for the child to finish and collect the return values in the variable status.
   The next output that is printed is:
   “shared_buf in child: First String” and “unshared_buf in child: First String”

Both of these lines are printed by the child process. This is because they contain “First String” and are passed as arguments to the do_child() routine, which is only accessible from the child.
The last output that is printed is:
“shared_buf after fork: Second String” and “unshared_buf after fork: First String”
These lines are printed after the parent process resumes. “Second String” from shared_buf is accessible from the parent process, even if it was written to by the child process. However, the content of unshared_buf in the parent process was unchanged meaning it was not accessible to the parent process.