Haonan Hu
2863545
Oct 2, 2020

Fork.c

Question 1: Which process prints this line? What is printed?
Both process prints the line,
In Child Process, It prints: After fork, Process id = 0 (The child process's pid is never zero. fork returns zero to the child to tell it that it is the child. The child process's pid, however, is the value that fork returns to the parent.)
In Parent Process, it prints: After fork, Process id = 8356(child pid)

Question 2: What will be printed if this like is commented?
After commented out that line, here is what printed on terminal:
After fork, Process id = 8989
In Parent: 8988, Child id: 8989
Final statement from Process: 8988
After fork, Process id = 0
print after execlp
Final statement from Process: 8989

Question 3: When is this line reached/printed?
In this file, when execlp function fails or missing

Question 4: What happens if the parent process is killed first? Uncomment the next two lines.
When the parent process dies, the child's parent process pid becomes 1.

Mfork.c

Question 1: How many processes are created? Explain.
7 processes are created

```
                  P0
           /       |       \
         P1        P2     P5
       /    \         \
     P3    P6      P4
    /
  P7
```

Pipe-sync.c

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

 int p1[2], p2[2];
 /* create pipe */
 pipe(p1);
 pipe(p2);

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

   /* child process. */
   printf("Child line 1\n");
   write(p1[1],buf,sizeof(buf));
   read(p2[0],buf,sizeof(buf));
   printf("Child line 2\n");
   write(p1[1],buf,sizeof(buf));
 } else {

   /* parent process */
   read(p1[0],buf,sizeof(buf));
   printf("Parent line 1\n");
   write(p2[1],buf,sizeof(buf));
   read(p1[0],buf,sizeof(buf));
   printf("Parent line 2\n");

   wait(&stat);
 }
}
```

fifo_producer.c and fifo_consumer.c

a. What happens if you only launch a producer (but no consumer)?
It is not doing anything until you open a consumer at the same time on the other terminal.

b. What happens if you only launch a consumer (but no producer)?

It is not doing anything until you open a producer at the same time on the other terminal.

c. If one producer and multiple consumers, then who gets the message sent?
Only for the first consumer you launched

d. Does the producer continue writing messages into the fifo, if there are no consumers?
No, you can still type in anything, but it terminates once you hit enter

e. What happens to the consumers, if all the producers are killed?
It terminates with message: consumer: read 0 bytes

shared_memory3.c
Question-1: Explain the output

The out is:

```
shared_buf before fork: First String          (1)
unshared_buf before fork: First String        (2)
shared_buf in child: First String             (3)
unshared_buf in child: First String           (4)
shared_buf after fork: Second String          (5)
unshared_buf after fork: First String         (6)
```

So, line 1 and 2 initialize shared and unshared regions, as they are initialized as STR1, so it prints First String. Then in the child process, it calls do_child function which first prints out the original value in the buffer, so that is why the line 3 and 4 are the same as 1 and 2. However, do_child function also makes shared and unshared buffers to STR2, but change of unshared memory in the child process will not bring to the parent process because they, that is why line 5 prints out the second String and line 6 prints out the first string.

thread-1.c

Question 1: Are changes made to the local or global variables by the child process reflected in the parent process? Explain.
No, because processes are isolated, it is very expensive to share data between processes.

Question 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.

Yes, since threads are not isolated and they can share data with the parent process, thus, the function call child_fn changes global and local will reflect and update global and val at the same time on the parent process.