Operating System Experiment 1

September 6, 2023

1 `simple-fork.c`

1. Examine `simple-fork.c`. What does the call `getppid()` do? Based on Textbook page 2.4, what kind of system call does `getppid()` belong to?

2. Before you compile `simple-fork.c`, try working out what you are expecting as an output. Does it work as you expected? If not, what were you expecting and why the difference?

2 Man Pages are your friends!

Linux/Unix operating systems provide documentation for their commands and APIs (including both system calls and C library calls) in what is now known as man pages.
As a start, type `man ls` in a WSL or Linux prompt and try to read it. What is this document about? The man pages installed in your system may be quite limited. On the other hand, it is fortunate that most of these man pages can be found online. In the following sections, we will be referring to some man pages found on the Internet to help us understand the intricacies of some of the system calls we have.

3 `fork-and-wait.c`

1. Examine `fork-and-wait.c`. Try to understand what the code is doing. Compile the code using `gcc` and run the program.

2. Click here and read the man page of `wait`. What does line 23 of the code do?

3. Try to search for `WIFEXITED` and `WEXITSTATUS`. What do these do?

4. Try the following changes to the code and compare with the original output.

   (a) There are in fact 3 ways to exit a process. Calling `exit`, calling `_exit` and using `return` in the main function. Please note that calling `exit` and `_exit` anywhere in the code will cause the process to terminate. Try changing the comments in line 16-18 and see if there is any difference in the output. What is common to all these calls?

   (b) Read the man page of `wait` and look at `waitpid`. Are you able to change the code so that we can use `waitpid` instead for the same effect? What's the difference between `waitpid` and `wait`?

(c) Scan through the man pages of `exit` and `_exit`. Note that if we exit a process via using `return`, the process did not explicitly call the `exit` family of system calls. How did the return value of the main function gets passed along to the parent process then?

4 exit() and return

1. Examine `test_exit.cpp, test_exit_staticvar.cpp, and test_return.cpp`. Can you see the difference between calling exit() and return?
2. When the exit() is used, which destructor of objects is called?
3. When `return` is used, which destructor of objects is called?

5 _atexit.c

```
int atexit(void (*function)(void));
```

The function prototype of `atexit` is given above. It is a function that indicates what function can be called whenever the program calls `exit` function call.

1. Read the code and try to understand the code. Compile and run it using `gcc`.
2. Try changing the comments at lines 29-31 of `_atexit.c` to see if that makes a difference between the printouts. What is the difference between the output when using `_exit` or `exit`? What does this experiment tells us about the relationship between `exit` and `return` in the main function?

6 _exit_and_exit.c

This section of the experiment seeks to find out the effect of using `_exit` and `exit` on the stdio buffers of a process.

1. Compile the program using `gcc` and run. What's the difference in the output when using either `_exit` or `exit`?
2. What does this experiment tell us about the difference between `_exit` and `exit` ?

7 multiple-forks.c

1. Compile the program using `gcc` and run. You should see four "Hello World"s printed on the console.
2. Are you able to tweak the code so that you print 5 "Hello World"s exactly? (Hint: Obviously, doing stuff like changing the main function to call "Hello World" 5 times explicitly is not what I am looking for.)

8 Code Listings

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>   // standard POSIX header file
#include <sys/wait.h> // POSIX header file for 'wait' function

int main()
{
    int pid;

    pid = getpid(); /* Parent's PID */
    printf("Original Process's  PID = %d\n", pid);

    pid=fork();
    if(pid==0)
    {
        int ppid = getppid();
        printf("Parent's PID is %d\n", ppid);
        printf("Child process.\n");
    }
    else
    {
        printf("Parent process.\n");
    }

    return 0;
}
```

simple–fork.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>   // standard POSIX header file
#include <sys/wait.h> // POSIX header file for 'wait' function

int main()
{
    int pid;

    pid=fork();
    if(pid==0)
    {
        //Child Process Code
        printf("Child Process.");

        /*exit(19);
          _exit(19);*/
        return 19;
    }
    else
    {
        int status;
        wait(&status);

        if(WIFEXITED(status))
            printf("status = %d\n", WEXITSTATUS(status));
```

```
28          printf("Parent Process.\n");
        }
30
        return 0;
32  }
```

fork–and–wait.c

```
    // C++ example for exit() in main()
 2  #include <iostream>
    #include <stdio.h>
 4  #include <stdlib.h>

 6  using namespace std;

 8  class MyExitTest {
    public:
10      MyExitTest() { std::cout << "Inside MyExitTest's constructor\n" << std::
            endl; }

12      ~MyExitTest()
        {
14          std::cout << "Inside MyExitTest's Destructor" << std::endl;
        }
16  };

18  int main()
    {
20      MyExitTest m1;

22      exit(0);
    }
```

test_exit.cpp

```
 1  // C++ example for exit() in main()
    #include <iostream>
 3  #include <stdio.h>
    #include <stdlib.h>
 5
    using namespace std;
 7
    class MyExitTest {
 9  public:
        MyExitTest() { std::cout << "Inside MyExitTest's constructor\n" << std::
            endl; }
11
        ~MyExitTest()
13      {
            std::cout << "Inside MyExitTest's Destructor" << std::endl;
15      }
    };
17
    int main()
19  {
        static MyExitTest m1;
21
        exit(0);
23  }
```

test_exit_staticvar.cpp

```
// C++ example for return in main()
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

using namespace std;

class MyExitTest {
public:
    MyExitTest() { std::cout << "Inside MyExitTest's constructor\n" << std::
        endl; }

    ~MyExitTest()
    {
        std::cout << "Inside MyExitTest's Destructor" << std::endl;
    }
};

int main()
{
    static MyExitTest m1;
    MyExitTest m2;

    return 0;
}
```

test_return.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>   // standard POSIX header file
#include <sys/wait.h> // POSIX header file for 'wait' function


void my_last_call()
{
    printf(" This should be printed last for a process.\n");
}

int main()
{

    /*
       Informs the C library that my_last_call
       should be called before my process
       finally exit.
    */
    atexit(my_last_call);


    /*
      Exiting the process by either
      calling exit, _exit,
      or return.
    */
```

```
28
        exit(100);
30      /*_exit(100);*/
        /*return 100;*/
32  }
```

_atexit.c

```
    #include <stdio.h>
2   #include <stdlib.h>
    #include <unistd.h>   // standard POSIX header file
4   #include <sys/wait.h> // POSIX header file for 'wait' function

6
    int main()
8   {
        int pid;
10
        pid=fork();
12      if(pid==0)
        {
14          printf("Parent's PID is %d", ppid);
            printf(" Child process. ");
16
            /*
18            Exiting the process by either
              calling
20                exit,
                  _exit,
22                and return.
             */
24
            /*exit(100);*/
26          _exit(10);
        }
28      else
        {
30          int status;

32          printf("In parent process...\n");
            wait(&status);
34
            printf("Done.\n");
36      }

38      return 0;
    }
```

_exit_and_exit.c

```
1   #include <stdio.h>
    #include <stdlib.h>
3   #include <unistd.h>   // standard POSIX header file
    #include <sys/wait.h> // POSIX header file for 'wait' function
5
    int main()
7   {
        fork();
9       fork();
```

```
    printf("Hello World\n");
}
```

multiple–forks.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>   // standard POSIX header file
#include <sys/wait.h> // POSIX header file for 'wait' function

int main()
{
    fork();
    fork();
    printf("Hello World\n");
}
```

multiple–forks.c