Operating Systems

Lab 1

by Romain Brisse

due on the 27th of September 2018

Contents

1 Compiling under Linux

1.1 Question 1:

This command: " gcc -o execName program.c" is used to compile a program in a Linux terminal, using gcc (GNU Compiler Collection).

1.2 Question 2:

This command is used to execute a program on a Linux terminal.

1.3 Question 3:

Using the "-g" option when compiling with gcc indicates that you desire to add to your compiled program all the necessary symbols for debugging it in the future. You will be able to debug your program by using gdb for example.

1.4 Question 4:

- **list: this function of gdb is used to see up to the next ten lines of code of the file you are working on, directly in the console.**
- **break: this function is used in gdb to set a breakpoint at a specific line in the code. It means that when you will execute the program, execution will hold and wait further instructions when it reaches this line.**
- **run arglist: The run command is used to start debugging the program. The "arglist" keyword allows you to add instructions to debug. You can set these arguments by using the "set args" commmand.**
- **help: there is a help command, you can use it at any time during your use of the gdb interface. It gives you indications about all the tasks gdb can execute.**
- **the help command allowed me to discover all the available commands within gdb. (see following screenshot).**

```
(gdb) help
List of classes of commands:

aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
stack -- Examining the stack
status -- Status inquiries
support -- Support facilities
tracepoints -- Tracing of program execution without stopping the program
user-defined -- User-defined commands

Type "help" followed by a class name for a list of commands in that class.
Type "help all" for the list of all commands.
Type "help" followed by command name for full documentation.
Type "apropos word" to search for commands related to "word".
Command name abbreviations are allowed if unambiguous.
(gdb)
```

2 Creating and Running a Process (1) - fork()

2.1 Question 1:

After reading the fork, getpid and getppid manuals, here is one sentence about each to summarize their use:

- fork: This function is used to create a child process from a parent process. Both process have unique IDs and memory locations.
- getpid: This function is used to get the ID of the calling process.
- getppid: this function is used to get the ID of calling process's parent.

2.2 Question 2:

After a fork call, which is done by the parent process, there are two running processes, one parent one child. They both hold the same information but their IDs and allocated memory space are different. All resources were duplicated.

2.3 Question 3:

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

void forkThis()
{
        if(fork()==0)
                printf("this is the child!\n");
        else
                printf("this is the parent!\n");
}

int main()
{
        forkThis();
        printf("my ID: %d\n",getpid());
        printf("my Parent's ID: %d\n",getppid());
return 0;
}
```

Figure 1: Here is the program I wrote

```
romain@Romain:~/ece/ing4/os/lab1$ gcc -g -o forkTest forkTest.c
romain@Romain:~/ece/ing4/os/lab1$ ./forkTest
this is the parent!
this is the child!
my ID: 53
my ID: 54
my Parent's ID: 35
my Parent's ID: 53
```

Figure 2: Here is the result I got

From this result I can observe that the parent and child are well differentiated processes, because the child and his parent have different process IDs, respectively 53 and 54. This result also proves that process 54 is indeed the child of process 53 because when I make him call the getppid() function the returned ID is 53, ID of our parent process.

2.4 Question 4:

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>

int main()
{
        int i = 5;

        if(fork()==0)
        {
                //I'm the Child!
                i++;
        }
        else
        {
                //I'm the Parent!
                sleep(3);
                printf("%d\n",i);
        }

return 0;
}
```

Figure 3: Here is the program I wrote

```
romain@Romain:~/ece/ing4/os/lab1$ ./sharedData
5
romain@Romain:~/ece/ing4/os/lab1$
```

Figure 4: Here is the program I wrote

The manual for the fork function precises that the allocated memory spaces for parent and child processes are different. That would indicated that data is not shared between parent and child, only duplicated. The two preceding images show a program I used to test that hypothesis. We would expect the result to be i=6 if data was shared between parent and child but the result is five. That is because we only incremented i within the child process and displayed it on screen within the parent process. This result proves that data is not shared between parent and child.

2.5 Question 5:

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
        if(fork()==0)
        {
                printf("I am a first generation child!\n");
                if(fork()==0)
                {
                        printf("I am a second generation child!\n");
                }
        }
        else
        {
                printf("I am a first generation parent!\n");
                if(fork()==0)
                {
                        printf("I am a first generation child!\n");
                }
                sleep(3);

        }

return 0;
```

Figure 5: Here is the program I wrote

```
romain@Romain:~/ece/ing4/os/lab1$ ./fork3
I am a first generation parent!
I am a first generation child!
I am a first generation child!
I am a second generation child!
romain@Romain:~/ece/ing4/os/lab1$
```

Figure 6: Here is the program I wrote

Looking at this program and its outcome, it appears that creating more than one child process is possible. This may prove useful on the field of parallel computing for example.

3 Creating and Running a Process (2) - exec

3.1 Question 1:

Reading the manual of the "exec" function, its use is in the execution of a file. It has many different variations whose differences mainly lie in the arguments you give to the function.

3.2 Question 2:

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
        printf("this is my id: %d",getpid());
        char* args[] = {"firefox",NULL};
        execvp(args[0],args);
        return 0;
}
```

Figure 7: Here is one way of testing the exec function

when executing this code, the obtained result is that the process ID displayed in the terminal and the process ID of the newly opened Firefox application are identical. I was able to obtain the process ID of Firefox by using the top command in a terminal which displays the current running processes.

3.3 Question 3:

You could say that at the exact moment the exec call is executed, data between the calling and executed processes is shared. But, as soon as the call has been completed, the process called with the "exec" function takes over every resource his calling process had. I know that because their IDs are the same. In the end I would rather say the the data is identical but not shared since there is no process to share it with when the exec call has been completed.

3.4 Question 4:

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>

int main()
{
        int i=5;
        char* args[] = {"firefox",NULL};

        if(fork()==0)
        {
                execvp(args[],args);
                i++;
                printf("this is the value of i: %d",i);
        }
        else
        {
                printf("this is the parent process's ID: %d",getpid());
        }

return 0;
}
```

Figure 8: I wrote the following program to find the answer

This program has the following behavior:

- creation of a child process using fork()
- said child process dos an exec call on the Firefox application. the following instructions are ignored since Firefox has taken over every resource
- the parent process prints its ID, which is different from the now ID of the Firefox app

4 Conclusion: What did I understand?

Using fork() and exec(), managing different processes has become a lot easier. the fork() function will be used when I have a need for doing operations in parallel or when I need to test different outcomes with the same resources as a basis. On the other hand, I will use the exec call when I have done everything I need with a process and so I just use it to open a new program that takes everything over.