Advanced Study of Processes and Files in UNIX
Lab. #3

Arif A. Balik

Undergraduate Student
Sytstems Programming
Department of Computer Science
Arel University
Büyükçekmece, İstanbul 34537
Email: arifbalik@outlook.com

March 21, 2019

Abstract

This report include advance analysis of some system calls and files in UNIX environment based on programs which provided in System Programming course. First section explains theory of operation of Part 2 of lab ssheet , second section answers some questions asked in the experiment paper, and third section suggests and implements some improvements and optimisations on the code.

This report is powered by LaTeX

Theory of Operation

Part 1

Part 1 is already given in UML format.

Part 2

The program in part to starts with creating a child process to replace itself with the **echo** program along with a text message. At the same time parent process writes the current date by calling **date** command via **execl**

Q&A

Part 1

Q1. *Why has the child process the ability to copy file 1 that is opened by the parent process?*

Unix has a file description structure which keeps the current position in the file, path of the file, file descriptor etc. When the function **fork** called, it copies this structure to child, thus child can access the files those opened before the **fork**

Q2. *Explain why files 2 and 3 can have different size and contents (after copying)?*

Because of the amount of bytes are different that child and parent reads from the file.

Q3. *Which of the processes will terminate first?*

It depends on which process read the last byte. It is highly probable the child process finishes first because it reads more data but first they work on the same file so size does not matter too much, and parent always waits for the child to terminate.

Q4. *Assume that the block "Waiting for the child" in the parent part of the program put immediately after the block "Create new file 2". What will the result be?*

File 2 would be empty because parent would wait for the child to terminate, hence child wont terminate until it reads all the file. After termination of child, parent would try to read and get *END OF FILE* from **read** function.

Q5. *What is the effect of large value of N?*

Because of the uncertainty mentioned in the last lab sheets, it can cause the sizes of the *file 2* and *file 3* to be too different from each other.

Q6. *What is the effect of the small value N?*

Vice versa.

Part 2

Q1. *What is the purpose of this laboratory work?*

The purpose is to gain a complete understanding the mechanism of processes and multithreading behaviours of the programs during concurrent file access.

Q2. *What is the purpose of system calls of* ***exec*** *family?*

exec family of functions replaces the current program with a new program. It allows developer to switch between programs during execution.

Q3. *What is the purpose of system calls* ***wait*** *and* ***sleep****? Is there any difference between them?*

As the name calls, the function **wait**, keeps the program locked until a signal is given, which is the child process's exit function. The function **sleep** just locks the system for given amount of time in seconds.

Q4. *What is the difference between the functions* ***printf*** *and* ***perror****?*

perror prints a custom message along with error messages by looking at error flags. **printf** (print formatted) prints a text with a given format, printf has nothing to do with errors.

Q5. *What is the difference between system calls* ***exec*** *and* ***system****?*

exec overlays the program, replaces it with a program given as a parameter, **system** passes the given argument to the shell. **system** basically creates a child and calls **exec** functions.

Q6. *Suppose that the child process in the program at the step 3 terminated before the parent. What will be the state of the child and how long this state will exist?*

In such case the child can not terminate and called zombie and it will remain until the parent process terminates.

Q7. *Is it possible for a parent process in the program at the step 3 to terminate before child process? What will be the new parent for the child process.*

Such a child is called **orphan**, and **init** takes control over the process and becomes its new parent.

Q8. *Why the order of printing messages by the processes in step 3 is not always correct?*

Because parent does not wait for child to terminate to print its message, in some cases that causes parent to finish first and print the message before child.

Optimization

Part 2

It defeats the purpose of the program but, it is very pleasing to optimise entire program down to a single line of code.

```
system("echo Todays date is && date '+Date = %D Time = %H:%M'");
```

Appendix

Part 1 Source Code

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

#define HELP "./part1 [file1] [file2] [file3]"

#define OPEN 1
#define CREATE 0
#define OPEN_FILE(path, o_c, fd)        fd = ((o_c) ? open(path, O_RDONLY) :\
                                                open(path, O_CREAT|O_WRONLY|O_TRUNC, 0666));\
                                        if(fd == -1){\
                                                perror("Error while file creation/opening");\
                                                exit(1);\
                                        }



#define PARENT_READ_BLOCK 10
#define CHILD_READ_BLOCK 20

#define WAIT 1

int read_write(int fd_read, int fd_write, int size){
        char* block;
        int i;
        short bytes;

        block = (char*)malloc((char)size);

        bytes = read(fd_read, block, size);
        switch(bytes){
                case 0:
                        printf("%s File 1: EOF\n",
                                (size == PARENT_READ_BLOCK) ? "Parent" : "Child");
                        goto eof;
                break;
                case -1:
                        perror("Error during read from File 1");
                        goto error;
                break;
                default:
                        if(bytes == size){
                                i = WAIT;
                                while(i--);
                                if(write(fd_write, block, size) != size){
                                perror("Error during write into File 3");
                                goto error;
                                }
                                return 1; //ready for the next sequence
                        }else if(bytes > 0){
                                write(fd_write, block, bytes);
                                printf("%s File1: EOF with %d bytes\n",
                                (size == PARENT_READ_BLOCK) ? "Parent" : "Child", bytes);
                        }else{
                                printf("Error during read File 1. bytes: %d\n",bytes);
                                goto error;
                        }
                        goto eof;
                break;
        }

        error:
        free(block);
        exit(1);

        eof:
        free(block);
        return 0;
}

int main(int argc, char** argv)
{
        short fd1, fd2, fd3;

        if(argc != 4){
                printf("Invalid argument count. Help\n"HELP"\n" );
                exit(1);
        }

        OPEN_FILE(argv[1], OPEN, fd1);

        if(!fork()){
                OPEN_FILE(argv[3], CREATE, fd3);
                while(read_write(fd1, fd3, CHILD_READ_BLOCK));
                close(fd3);
                exit(1);
        }

        OPEN_FILE(argv[2], CREATE, fd2);

        while(read_write(fd1, fd2, PARENT_READ_BLOCK));

        close(fd2);
        wait(0);
        close(fd3);

}
```