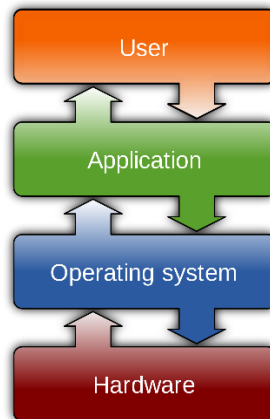


# Operating Systems

## Lab assignment 2: Programming in C and use of Systems Calls



### Objectives

1. To develop sample C programs
2. To develop programs with two or more processes using `fork( )`, `exit( )`, `wait( )`, and `exec( )` system calls

### Guidelines

For this course you need to develop a good knowledge of C in a Linux development environment. You are highly encouraged to use command line tools in developing, compiling, and running your programs.

Please pay attention to your coding style and good programming practices, if your program is not worth documenting, it will not worth running.

Skills in developing multi-processing and multi-threading applications is required for synchronization as a way to create parallelism. A process is defined as a program in execution. Multiple processes can be executing the same program, with each process owning its own copy of the program within its own address space and executes it independently of the other processes.

A `fork()` system call is used in a Linux to create a child process. The `fork()` takes no arguments and returns 0 for the child process and returns the process ID (PID) for the parent process.

After a new child process is created, the parent and the child processes execute the next instruction following the `fork()` system call. Therefore, the returned value of the `fork()` is used to distinguish the parent from the child.

- If `fork()` returns a negative value, system call failure
- If `fork()` returns a zero, a newly created child process
- If `fork()` returns a positive value, a process ID of the child process to the parent.

Include the following libraries for defining the process `pid_t` type and using `fork()`.

```
#include <sys/types.h>
```

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

Note, the child process inherits exact copy of the parent's address space and both have separate address spaces. There are two types of processes:

**Zombie Process:** A process is Zombie, i.e. inactive, as long as it maintains its entry in the parent's process table. Therefore, it is recommended that a child process uses `exit( )` system call at the end of its execution. The exit status of the child process is then passed to the parent process to remove its entry.

Orphan Process: A process is Orphan if its parent process no more exists. This is either, because the parent process is either has finished or terminated without waiting for its child process to terminate. Therefore, it is recommended that a parent process uses wait() system call to wait until the child process exits, i.e. terminates.

### C Program with two processes

Demonstrate each of the following steps to get a grade on this part of the lab assignment

Step 1. Please write the following C program in a Linux environment using vi, emacs, or an editor of your choice

```
/*Sample C program for Lab assignment 1*/
#include <stdio.h> /* printf, stderr */
#include <sys/types.h> /* pid_t */
#include <unistd.h> /* fork */
#include <stdlib.h> /* atoi */
#include <errno.h> /* errno */
/* main function with command-line arguments to pass */
int main(int argc, char *argv[]) {
    pid_t pid;
    int i, n = atoi(argv[1]); // n microseconds to input from keyboard for delay
    printf("\n Before forking.\n");
    pid = fork();
    if (pid == -1) {
        fprintf(stderr, "can't fork, error %d\n", errno);
    }
    if (pid){
        // Parent process
        for (i=0;i<100;i++) {
            printf("\t\t\t Parent Process %d \n",i);
            usleep(n);
        }
    }
    else{
        // Child process
        for (i=0;i<100;i++) {
            printf("Child process %d\n",i);
            usleep(n);
        }
    }
    return 0;
}
```

Step 2. Compile the program using gcc compiler by typing gcc YourProgram.c – o ExecutableName. When it compiles without errors or warnings, make a copy of the source file then go to step 3.

Step 3. Run the program by typing ./ExecutableName and take a note of your observation.

Step 4. Re-run the program by typing ./ExecutableName 3000. Note that the delay in the loop depends on the command line argument you give, here the delay is 3000 microseconds.

a. Enter delays of 500 and 5000, what happens?

### Changing the context of the process

Processes are often created to run separate programs. In this case, a process context is changed by causing it to replace its execution image by with an execution image of a new program, `exec( )` system call is used. Although the process loses its code space, data space, stack, and heap, it retains its process ID, parent, child processes, and open file descriptors. Six versions of `exec( )` exist. The simplest and widely used:

- `execlp(char *filename, char *arg0, char arg1,..... , char *argn, (char *) 0);`
- e.g. `execlp( "sort", "sort", "-n", "foo", 0);` ◇ \$ sort -n, foo

Step 5. Rewrite the program in Step 1., so that the child process runs the `ls` command, and that the parent process waits until the child process terminates before it exits. Demonstrate your code. You may use the following code snippet.

```
else if(pid == 0)
{
    execlp("/bin/ls", "ls", NULL);
}
else
{
    wait(NULL);
    printf("Child Complete");
    exit(0);
}
```

#### Requirements to complete the lab

1. Show the correct execution of the C programs.
2. Submit your answers to questions.
3. Submit the source code for all your programs as .c file(s).

Be sure to retain copies of your .c and .txt files. You will want these for study purposes and to resolve any grading questions (should they arise)

Please start each program/ text with a descriptive block that includes minimally the following information:

```
# Name: <your name>
# Date: <date> (the day you have lab)
# Title: Lab 2 - task
# Description: This program computes ... <you should
# complete an appropriate description here.>
```