Lab 04: The exec Function

The `fork` function creates a copy of the calling process, but many applications require the child process to execute code that is different from that of the parent. The `exec` family of functions provides a facility for overlaying the process image of the calling process with a new image. The traditional way to use the `fork-exec` combination is for the child to execute (with an `exec` function) the new program while the parent continues to execute the original code.

SYNOPSIS

```
#include <unistd.h>

extern char **environ;
int execl(const char *path, const char *arg0, ... /*, char *(0) */);
int execle (const char *path, const char *arg0, ... /*, char *(0),
            char *const envp[] */);
int execlp (const char *file, const char *arg0,.../*, char *(0) */);
int execv(const char *path, char *const argv[]);
int execve (const char *path,char *const argv[],char *const envp[]);
int execvp (const char *file, char *const argv[]);
```

All `exec` functions return –1 and set `errno` if unsuccessful. In fact, if any of these functions return at all, the call was unsuccessful.

The six variations of the `exec` function differ in the way command-line arguments and the environment are passed. They also differ in whether a full pathname must be given for the executable. The `execl` (`execl`, `execlp` and `execle`) functions pass the command-line arguments in an explicit list and are useful if you know the number of command-line arguments at compile time. The `execv` (`execv`, `execvp` and `execve`) functions pass the command-line arguments in an argument array. The arg_i parameter represents a pointer to a string, and `argv` and `envp` represent `NULL`-terminated arrays of pointers to strings.

The `path` parameter to `execl` is the pathname of a process image file specified either as a fully qualified pathname or relative to the current directory. The individual command-line arguments are then listed, followed by a `(char *)0` pointer (a `NULL` pointer).

Tasks

Task 1: Write a program that takes N UNIX commands as arguments, creates N child processes, each of them implementing their respective commands.

Task 2: a. Write a program that takes integers as arguments and adds them.
b. Write a program that takes integers as arguments and multiplies them.
c. Write a program that takes integers as arguments & adds & multiplies them using the above two programs.

Task 3: Write a program "minmax.c" that takes an array as command line arguments. Program executes min.c and max.c programs in its two child processes. One child processes calculates and returns the min value and other calculates and returns the max value in the array. The program "minmax.c" shall receive the values returned by the child processes and display these values.