Practical 3 Exec family functions

Program 1

Zombie Process:

- It applies to processes that are dead but have not been removed from the process table.
- It is a process that has terminated but has not been cleaned up yet.
- It is the responsibility of the parent process to clean up its zombie process.
- The Wait function work for cleaning the zombie children.

Command Line Argument

Until now, the skeletons we have used for our C programs have looked something like this:

```
#include <stdio.h>
int main()
{
        return 0;
}
```

From now on, our examples may look a bit more like this:

```
#include <stdio.h>
int main (int argc, char *argv[])
{
          return 0;
}
```

As you can see, main now has arguments. The name of the variable argc stands for "argument count"; argc contains the number of arguments passed to the program.

The name of the variable argv stands for "argument vector". A vector is a one-dimensional array, and argv is a one-dimensional array of strings. Each string is one of the arguments that was passed to the program.

For example, the command line

gcc -o myprog myprog.c

would result in the following values internal to GCC:

argc 4
argv[0] gcc
argv[1] -o
argv[2] myprog
argv[3] myprog.c

```
//This program accepts numbers as arguments. Calculate sum of two
// numbers in child process and returned as exit status. Parent prints the result.

#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
int main(int argc, char *argv[])
{
		int x, y, exitstatus;
		if(argc != 3)
		{
			printf("Wrong usage of arguments\n");
			exit(1);
		 }
		 switch(fork())
		 {
			case -1: printf("Fork error\n");
					exit(2);
			case 0:		printf("Child process\n");
				x=atoi(argv[1]);
				y=atoi(argv[2]);
				exit(x+y);
			default:
				printf("Parent process\n");
				wait(&exitstatus);
				printf("Sum is %d\n", WEXITSTATUS(exitstatus));
				exit(0);
		}

}
```

Program 2:

Exec family functions:

- The exec functions replace the program running in a process with another program.
- When program calls an exec function, that process immediately ceases executing that program and begins executing a new program from the beginning, assuming that the exec call does not encounter any error.
- The library functions execl, execlp, execle, execv, and execvp are simply convenience functions that allow specifying the arguments in a different way, use the current environment instead of a new environment, and/or search the current path for the executable.
- Fuctions that contain the letter **p** in their names (execvp and execlp) accept a program name and search for a program by that name in the current execution program to be executed.
- Functions that contain the letter **v** in their names (execv, execvp, and execve) accept the argument list for the new program as a NULL terminated array of pointers to strings.
- Functions that contain the lette**r l** (execl, execlp, execle) accept the argument list using the C language vargs mechanism.
- Functions that contains the e in their names (execve and execle) accept an additional argument, an array of environment variables. The argument should be a NULL terminated array of pointers to character strings. Each character string should be of the form "Variable=value".
- **Exec replaces the calling program with another one, it never returns unless an error occurs.**
- Only rarely will you want to use these routines by themselves.

```
//This program shows use of execl() function
#include<stdio.h>
int main()
{
        execl("/usr/bin/wc","wc","-l","f1.txt",NULL);
        printf("Done\n");
         exit(0);
}
```

Exercise

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

```
int main()
{
                execl("/bin/ls", "ls", "-l", 0);
                printf("Can only get here on error\n");
}
```

Program 4:

execv : This is very similar to execvp() function in terms of syntax as well. The syntax of **execv()** is as shown below:

Syntax:

```
int execv(const char *path, char *const argv[]);
```

path: should point to the path of the file being executed.
argv[]: is a null terminated array of character pointers.

Let us see a small example to show how to use execv() function in C. We will have two .C files , **EXEC.c** and **execDemo.c** and we will replace the execDemo.c with EXEC.c by calling execv() function in execDemo.c

```
//EXEC.c

#include<stdio.h>
#include<unistd.h>
int main()
{
        int i;
        printf("I am EXEC.c called by execv() ");
        printf("\n");
        return 0;
}
```

Now, create an executable file of EXEC.c using command
gcc EXEC.c -o EXEC

```
//execDemo.c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
    //A null terminated array of character pointers
```

```
        char *args[]={"./EXEC",NULL};
        execvp(args[0],args);

        /*All statements are ignored after execvp() call as this whole
        process(execDemo.c) is replaced by another process (EXEC.c)
        */
    printf("Ending-----");
    return 0;
}
```

Now, create an executable file of execDemo.c using command
gcc execDemo.c -o execDemo

After running the executable file of execDemo.c by using command ./excDemo, we get the following output:

I AM EXEC.c called by execv()

Exercise

```
//This program shows use of execv()
#include<stdio.h>
int main()
{
        char *cmdargs[]={"wc","-l","f1.txt",NULL};
        execv("/usr/bin/wc",cmdargs);
        // OR execv("/usr/bin/wc",&cmdargs[0]);
        printf("Done\n");
}
```

Program 5:

//This program takes command and its argument from the user and executes it

```
#include<stdio.h>
#include<sys/wait.h>

int main(int argc, char *argv[])
{
          int exitstatus;
          switch(fork())
          {
                case -1: printf("Fork error\n");
```

```
                    exit(1);
        case 0:
                    execv(argv[1],&argv[2]);
                        printf("Done\n");
                    //exit(0);
        default:
            wait(&exitstatus);
            printf("proces exit status=%d\n",WEXITSTATUS(exitstatus));
        }

    }
```

Program 6:

```
//This program shows use of execlp()

#include<stdio.h>

int main()
{
        execlp("wc","wc","-l","f1.txt",NULL);
        printf("Done\n");
        //Similarly execvp("wc",cmdargs);
        //will also work
}
```

Exercise

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
main()
{
        char *temp,*temp1,*temp2;

        temp1="Funny";
        temp2="world";
        execlp("echo","echo",temp1,temp2,NULL);
        printf("Error");
}
```