

# Computer Operating Systems, Practice Session 3

## The fork and exec System Calls in Linux

Resul Tugay (tugayr@itu.edu.tr)

February 22, 2017

Today

## Computer Operating Systems, PS 3

fork System Call

exec System Call

# fork Usage

- ▶ `int fork();`
- ▶ `fork()` is called once
- ▶ But it returns twice!!
  - ▶ Once in *parent process*
  - ▶ Once in *child process*
- ▶ How can we separate parent and child processes ??
  - ▶ Return value in child process = 0
  - ▶ Return value in parent process = process ID of the child process

## Example Program

```
1 #include <stdio.h>           // printf
2 #include <unistd.h>          // fork
3 #include <stdlib.h>          // exit
4 #include <sys/wait.h>        // wait
5
6 int main(void){
7     int f = fork();           // forking a child process
8     if(f == -1){              // fork is not successful
9         printf("Error\n");
10        exit(1);
11    }
12    else if (f == 0){          // child process
13        printf("  Child: Process ID: %d \n",getpid());
14        printf("  Child: Parent Process ID: %d \n",getppid());
15        // waiting for 10 seconds
16        sleep(10);
17    }
```

## Example Program

```
1  else{           // parent process
2      printf("Parent: Process ID: %d \n", getpid());
3      printf("Parent: Child Process ID: %d \n", f);
4      printf("Parent: Parent Process ID: %d \n", getppid());
5      // waiting until child process has exited
6      wait(NULL);
7      printf("Parent: Terminating... \n");
8      exit(0);
9  }
10 return 0;
11 }
```

## Output of the Example Program

```
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder/code$ gcc forkExample.c
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder/code$ ./a.out
Parent: Process ID: 1863
Parent: Child Process ID: 1864
Parent: Parent Process ID: 1745
      Child: Process ID: 1864
      Child: Parent Process ID: 1863
Parent: Terminating...
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder/code$ █
```

# exec Usage

```
int execlp(const char *filename, const char *arg0, const char *arg1,  
..., const char *argn, (char*)NULL);
```

- ▶ only way to run a program within a process in Unix
- ▶ PID value does NOT change
- ▶ is not a new process!

# exec Usage

There are 6 ways for calling exec function:

`execl`, `execv`, `execle`, `execve`, `execlp`, `execvp`

- ▶ `int execl(const char *path, const char *arg0, ...);`
- ▶ `int execle(const char *path, const char *arg0, ..., char *const envp[]);`
- ▶ `int execlp(const char *file, const char *arg0, ...);`
- ▶ `int execv(const char *path, char *const argv[]);`
- ▶ `int execve(const char *file, char *const argv[], ..., char *const envp[]);`
- ▶ `int execvp(const char *file, char *const argv[]);`



## exec Usage (suffixes)

execl, execl, execl, execl, execlp, execlp

- ▶ **l** specifies that the argument pointers ( $arg_0, arg_1, \dots, arg_n$ ) are passed as separate arguments. Typically, the **l** suffix is used when you know in advance the number of arguments to be passed.
- ▶ **v** specifies that the argument pointers ( $argv[0] \dots, argv[n]$ ) are passed as an array of pointers. Typically, the **v** suffix is used when a variable number of arguments is to be passed.
- ▶ **p** specifies that the function searches for the file in those directories specified by the PATH environment variable (without the **p**, the function searches only the current working directory). If the path parameter does not contain an explicit directory, the function searches first the current directory, then the directories set with the PATH environment variable.
- ▶ **e** allows the caller to specify the environment of the executed program via the argument **envp**. The **envp** argument is an array of pointers to null-terminated strings ( $name = value$ ) and must be terminated by a null pointer. Without the **e** suffix, child processes inherit the environment of the parent process.

## exec Usage - Return Value?

- ▶ exec replaces current process, can it return anything to original process ?? NO
- ▶ BUT we can use a special property: processes' exit value is collected by parent process ...

- ▶ `errno` can be checked:

Sample codes:

ENAMETOOLONG	: The length of path or file exceeds <code>PATH_MAX</code> , or a path name is longer than <code>NAME_MAX</code>
EACCES	: Permission denied
E2BIG	: Argument list and environment list is greater than <code>ARG_MAX</code>
ENOENT	: Path or file name not found
ENOEXEC	: Has the appropriate access permissions, but is not in the proper format
ENOMEM	: Not enough memory
ENOTDIR	: A component of the new process image file's path prefix is not a directory

## Example Program (Master)

```
1 #include <stdio.h>      // printf
2 #include <unistd.h>     // fork, execlp
3 #include <stdlib.h>     // exit
4 #include <errno.h>      // errno
5 #include <string.h>     // strerror
6
7 int main(void){
8     printf("\n Master is working: PID:%d \n",getpid());
9
10    int f = fork();      // forking a child process
11
12    if (f == 0) {        // child process
13        printf("\n This is child... PID: %d \n", getpid());
14        // execute the slave process
15        execlp("./execSlave", "./execSlave"
16              , "test1", "test2", (char*)NULL);
17        // exec returns only when there is an error
18        printf("\n %s\n", strerror(errno));
19    }
```

## Example Program (Master)

```
1  else{           // parent process
2                  // waiting until child process has exited
3                  wait(NULL);
4                  exit(0);
5              }
6  return 0;
7  }
```

## Example Program (Slave)

```
1 #include <stdio.h>      // printf
2 #include <unistd.h>     // getpid
3
4 int main(int argc, char* argv[])
5 {
6     printf("\nSlave started working ... PID: %d \n",getpid());
7     printf("Name of the Program :%s \n",argv[0]);
8     printf("The first parameter of the program:%s \n",argv[1]);
9     printf("The second parameter of the program:%s \n",argv[2]);
10    return 0;
11 }
```

## Output of the Example Program

```
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder/code$ gcc execSlave.c  
-o execSlave  
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder/code$ gcc execMaster.c  
-o execMaster  
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder/code$ ls  
execMaster  execMaster.c  execSlave  execSlave.c  forkExample.c  
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder/code$ ./execMaster
```

Master is working: PID:2088

This is child... PID: 2089

Slave started working ... PID: 2089

Name of the Program :./execSlave

The first parameter of the program:test1

The second parameter of the program:test2

```
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder/code$ █
```

## Output of the Example Program (Failure)

```
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder/code$ rm execSlave  
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder/code$ ./execMaster
```

```
Master is working: PID:2127
```

```
This is child... PID: 2128
```

```
No such file or directory
```

```
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder/code$ █
```