Computer Operating Systems, Practice Session 3

The `fork` and `exec` System Calls in Linux

Mustafa Ersen (ersenm@itu.edu.tr)

Istanbul Technical University
34469 Maslak, İstanbul

25 February 2015

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

```c
#include <stdio.h>      // printf
#include <unistd.h>     // fork
#include <stdlib.h>     // exit
#include <sys/wait.h>     // wait

int main(void){
  int f = fork();   // forking a child process
  if(f == -1){      // fork is not successful
    printf("Error\n");
    exit(1);
  }
  else if (f == 0){     // child process
    printf("  Child: Process ID: %d \n",getpid());
    // waiting for 10 seconds
    sleep(10);
    printf("  Child: Parent Process ID: %d \n",getppid());
  }
```

Example Program

```
    else{       // parent process
      printf("Parent: Process ID: %d \n", getpid());
      printf("Parent: Child Process ID: %d \n", f);
      printf("Parent: Parent Process ID: %d \n", getppid());
      // waiting until child process has exited
      wait(NULL);
      printf("Parent: Terminating... \n");
      exit(0);
    }
    return 0;
}
```

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

- `l` specifies that the argument pointers (arg_0, arg_1, ..., 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] ..., arg[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 *PATH_MAX*, or a path name is longer than *NAME_MAX*
EACCES	: Permission denied
E2BIG	: Argument list and environment list is greater than ARG_MAX
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)

```c
#include <stdio.h>      // printf
#include <unistd.h>     // fork, execlp
#include <stdlib.h>     // exit
#include <errno.h>      // errno
#include <string.h>     // strerror

int main(void){
   printf("\n Master is working: PID:%d \n",getpid());

   int f = fork();    // forking a child process

   if (f == 0) {      // child process
      printf("\n This is child... PID: %d \n", getpid());
      // execute the slave process
      execlp("./execSlave","./execSlave"
                  ,"test1","test2",(char*)NULL);
         // exec returns only when there is an error
         printf("\n %s\n", strerror(errno));
   }
```

Example Program (Master)

```
    else{       // parent process
            // waiting until child process has exited
            wait(NULL);
            exit(0);
    }
    return 0;
}
```

Example Program (Slave)

```c
#include <stdio.h>      // printf
#include <unistd.h>      // getpid

int main(int argc, char* argv[])
{
   printf("\nSlave started working ... PID: %d \n",getpid());
   printf("Name of the Program :%s \n",argv[0]);
   printf("The first parameter of the program:%s \n",argv[1]);
   printf("The second parameter of the program:%s \n",argv[2]);
   return 0;
}
```

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$
```