Bilgisayar İşletim Sistemleri, Uygulama 2

`fork` ve `exec` Linux Sistem Çağrıları

İstanbul Teknik Üniversitesi
34469 Maslak, İstanbul

22 Şubat 2017

Bugün

Bilgisayar İşletim Sistemleri, Uygulama 2

`fork` Sistem Çağrısı

`exec` Sistem Çağrısı

fork kullanımı

- `int fork();`
- `fork()` bir kere çağırılır
- Ancak iki kere değer döner!!
  - Birinde *anne proses*
  - Birinde *çocuk proses*
- Anne ve çocuk prosesleri birbirinden nasıl ayırabiliriz??
  - Çocuk proseste döndürülen değer = 0
  - Anne proseste döndürülen değer = çocuk prosesin proses ID bilgisi

```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());
  }
```

```
    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;
}
```

Örnek programın çıktısı

```
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 kullanımı

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

- exec fonksiyonları prosesin başka bir program olarak çalışmaya devam etmesini sağlamaktadır
- PID değeri değişmez
- yeni bir proses değildir!

exec Kullanımı

Tüm exec fonksiyonlarının prototipleri unistd.h dosyası içersindedir. exec fonksiyonunu çağırmak için 6 yol vardır:
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 Kullanımı (suffixes)

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

- `l` arguman pointer larının (arg_0, arg_1, ..., arg_n) farklı parametreler olarak verilmesini sağlar. Genelde, **l** kaç tane argumanın parametre olarak verileceğinin bilindiği durumlarda kullanılır.
- `v` arguman pointer larının (argv[0] ..., arg[n]) pointer dizisi olarak girileceği durumları belirtir. Genelde, **v** değişken sayıda arguman girileceğinde kullanılır.
- `p` fonksiyonun, PATH çevresel değişkeniyle belirtilen dizinlerdeki dosyaları ararmasını sağlar.(p olmazsa, fonksiyon sadece bulunduğu dizini arar). Eğer path parametresi açıkça bir dizin içermezse, fonksiyon ilk olarak bulunduğu dizini, sonra da PATH çevresel değişkeniyle belirtilen dizinlere bakar.
- exec fonksiyonlarının `e` li biçimleri çevre değişkenlerini (environment variables) de programcıdan istemektedir.

```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));
  }
```

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

```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;
}
```

Örnek programın çıktısı

```
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$
```

Örnek programın çıktısı

```
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$
```

Referans

- http://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/fork/create.html