启动新进程(fork 和 exec 系列函数实现)

一、复制进程映像

1、fork 函数介绍

此系统调用主要复制当前进程，在进程表中创建一个新的表项，新表项中的许多属性与当前进程是相同的。新进程几乎与原进程一模一样，执行的代码也完全相同，但新进程有自己的数据空间、环境和文件描述符。

2、典型使用 fork 的代码片段：

```
pid_t pid;
pid = fork();

switch(pid)
{
case -1: // error occur
        perror("fork failed");
        exit(1);
case 0: // child
        break;
default: // parent
        break;
}
```

3、示例

示例代码：

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
        pid_t pid;
        char *message;
        int n;

        printf("fork program starting \n");
        pid = fork();

        switch(pid)
        {
        case -1:
                perror("fork failed");
                exit(1);
        case 0:
                message = "This is the child";
                n = 5;
                break;
        default:
                message = "This is the parent";
                n = 3;
```

```
                break;
        }
        for(;n>0;n--)
        {
                puts(message);
                sleep(1);
        }
        exit(0);
}
```

运行效果如下：

```
[root@host1 process2]# ./fork1
fork program starting
This is the parent
This is the child
This is the parent
This is the child
This is the parent
This is the child
[root@host1 process2]# This is the child
This is the child
```

E-Mail : Mike_Zhang@live.com

二、替换进程映像

1、exec 系列函数介绍

exec 系列函数可以把当前进程替换为一个新的进程，函数列表如下：

int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(const char *path, const char *arg,..., char * const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execvpe(const char *file, char *const argv[],char *const envp[]);

2、头文件：

#include <unistd.h>

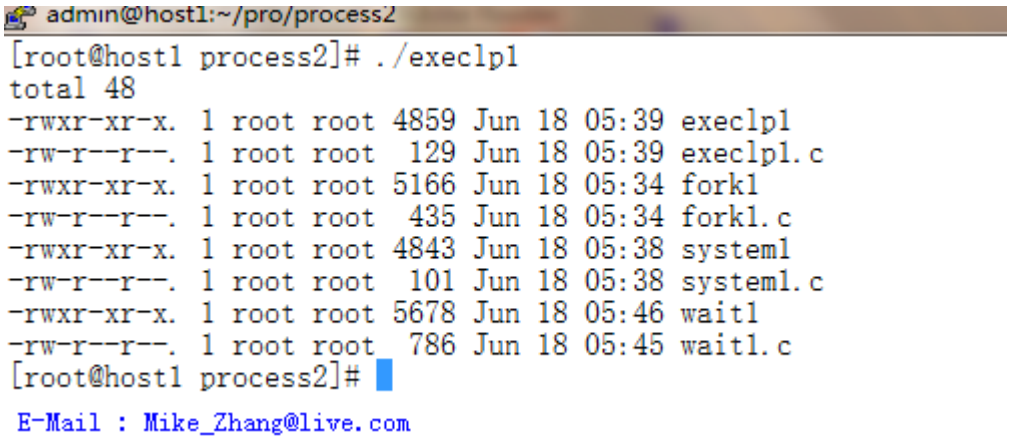
3、示例

示例代码：

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

int main()
{
        execlp("ls","ls","-l",0);
        printf("OK\n");
        exit(0);
}
```

运行效果：



三、启动新进程

1、system 函数实现

1.1 说明

system 函数可以在一个程序的内部启动另一个程序，从而创建一个新进程。

#include <stdlib.h>
int system(const char *string);

system 函数的作用是，运行以字符串参数的形式传递给它的命令并等待该命令的完成。命令的执行情况就如同在 shell 中执行如下命令：
$sh -c string

1.2 示例

示例代码：

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

int main()
{
        system("ls -l");
        printf("OK\n");
        exit(0);
}
```

运行效果：

```
admin@host1:~/pro/process2
[root@host1 process2]# ./system1
total 48
-rwxr-xr-x. 1 root root 4859 Jun 18 05:39 execlp1
-rw-r--r--. 1 root root  129 Jun 18 05:39 execlp1.c
-rwxr-xr-x. 1 root root 5166 Jun 18 05:34 fork1
-rw-r--r--. 1 root root  435 Jun 18 05:34 fork1.c
-rwxr-xr-x. 1 root root 4843 Jun 18 05:38 system1
-rw-r--r--. 1 root root  101 Jun 18 05:38 system1.c
-rwxr-xr-x. 1 root root 5678 Jun 18 05:46 wait1
-rw-r--r--. 1 root root  786 Jun 18 05:45 wait1.c
OK
[root@host1 process2]#
E-Mail : Mike_Zhang@live.com
```

2、fork 和 exec 系列函数实现

一般来说，使用 system 函数不是启动其它进程的理想手段，因为它必须用一个 shell 来启动需要的程序。由于启动之前需要先启动一个 shell，而且对 shell 的安装情况及使用的环境的依赖也很大，所以使用 system 函数效率不高。鉴于此种情况，可以考虑使用 fork 启动子进程，然后用 exec 系列函数替换当前子进程满足此种需求。

示例代码（wait1.c）：

```c
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
        pid_t pid;
        char *message;
        int exit_code;

        printf("fork program starting\n");
        pid = fork();
        switch(pid)
        {
        case -1:
                perror("fork failed");
                exit(1);
        case 0:
                message = "This is the child";
                sleep(3); //sleep
                execlp("ls","ls","-l",0);
                exit_code = 37;
                break;
        default:
                message = "This is the parent";
                exit_code = 0;
                break;
```

```
        }
        if(pid != 0) //wait child
        {
                int stat_val;
                pid_t child_pid;

                child_pid = wait(&stat_val);

                printf("Child has finished : PID = %d\n",child_pid);
                if(WIFEXITED(stat_val))
                        printf("Child exited with code %d\n",WEXITSTATUS(stat_val));
                else
                        printf("Child terminated abnormally\n");
        }
        exit(exit_code);
}
```

运行效果：