

实验一 进程管理

1. 实验目的:

- (1) 加深对进程概念的理解, 明确进程和程序的区别;
- (2) 进一步认识并发执行的实质;
- (3) 分析进程争用资源的现象, 学习解决进程互斥的方法;
- (4) 了解Linux系统中进程通信的基本原理。

2. 实验预备内容

- (1) 阅读Linux的sched.h源码文件, 加深对进程管理概念的理解;
- (2) 阅读Linux的fork()源码文件, 分析进程的创建过程。

3. 实验内容

(1) 进程的创建:

编写一段程序, 使用系统调用fork() 创建两个子进程。当此程序运行时, 在系统中有一个父进程和两个子进程活动。让每一个进程在屏幕上显示一个字符: 父进程显示字符“a”, 子进程分别显示字符“b”和“c”。试观察记录屏幕上的显示结果, 并分析原因。

三进程并发运行, 代码如下:

```
#include <sys/types.h>

#include <stdio.h>

#include <unistd.h>

void main(int argc, char* argv[])
{
    pid_t pid1, pid2;

    pid1 = fork();

    if(pid1 < 0){
        fprintf(stderr, "Fork failed");
    }
    else if (pid1 == 0){
        printf("b");
    }
    else{
        pid2 = fork();
        if(pid2 < 0){
```

```

        fprintf(stderr, "Fork failed");
    }
    else if (pid2 == 0){
        printf("c");
    }
    else{
        printf("a");
    }
}
}

```

```

→ /Users/crarc/Code/homework > ./task1
abc
→ /Users/crarc/Code/homework > ./task1
acb
→ /Users/crarc/Code/homework > ./task1
abc
→ /Users/crarc/Code/homework > ./task1
abc
→ /Users/crarc/Code/homework > ./task1
abc

```

因为进程是并发运行的，可以看到输出并不是固定的。

(2) 进程的控制

修改已经编写的程序，将每个进程输出一个字符改为每个进程输出一句话，再观察程序执行时屏幕上出现的现象，并分析原因。

如果在程序中使用系统调用lockf () 来给每一个进程加锁，可以实现进程之间的互斥，观察并分析出现的现象。

代码如下：

```

#include <sys/types.h>

#include <stdio.h>

#include <unistd.h>

void main(int argc, char* argv[])
{

```

```

pid_t pid1, pid2;

pid1 = fork();

if(pid1 < 0){
    fprintf(stderr, "Fork failed");
}
else if (pid1 == 0){
    printf("One got executed then there were nine.");
}
else{
    pid2 = fork();
    if(pid2 < 0){
        fprintf(stderr, "Fork failed");
    }
    else if (pid2 == 0){
        printf("Nine little Indians haven't long to wait.");
    }
    else{
        printf("Ten little Indians standing in a line.");
    }
}
}

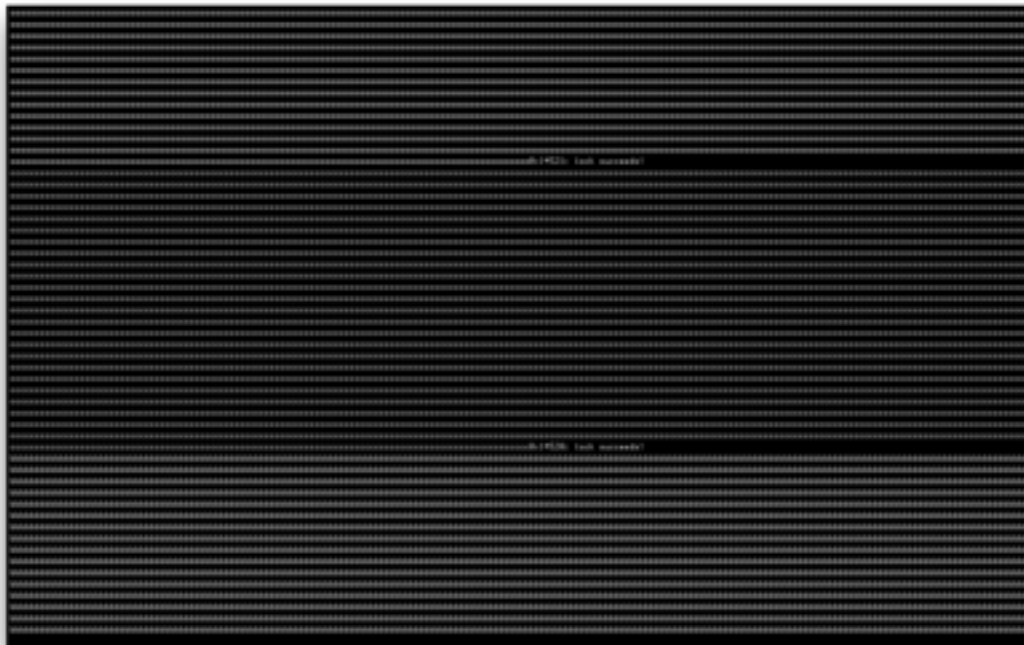
```

```

→/Users/crarc/Code/homework > ./task2
Ten little Indians standing in a line.One got executed then there were nine.Nine
  little Indians haven't long to wait.
→/Users/crarc/Code/homework > ./task2
Ten little Indians standing in a line.One got executed then there were nine.Nine
  little Indians haven't long to wait.
→/Users/crarc/Code/homework > ./task2
Ten little Indians standing in a line.One got executed then there were nine.Nine
  little Indians haven't long to wait.
→/Users/crarc/Code/homework > ./task2
One got executed then there were nine.Ten little Indians standing in a line.Nine
  little Indians haven't long to wait.
→/Users/crarc/Code/homework >

```

我是应该看到混乱的输出流吗。。。。。。。。。。wait a minut。。



进程加锁后代码:

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

void print_something(char letter)
{
    int i, error;
    error = lockf(1, F_LOCK, 0);
    if (error == 0)
        printf("%#x: lock succeeds!\n", getpid());
    else
        perror("lockf");

    for(i=0;i<5000;i++){
        printf("%c",letter);
    }

    lockf(1, F_ULOCK, 0);
}

void main(int argc, char* argv[])
{
    pid_t pid1, pid2;

    pid1 = fork();

    if(pid1 < 0){
        fprintf(stderr, "Fork failed");
    }
    else if (pid1 == 0){
        print_something('b');
    }
    else{
        pid2 = fork();
        if(pid2 < 0){
```

```

        fprintf(stderr, "Fork failed");
    }
    else if (pid2 == 0){
        print_something('c');
    }
    else{
        print_something('a');
    }
}
}
}

```

因为一个进程对一个资源加锁后,其它进程就无法占用这个资源，只能挂起等待至该资源解锁。

(3)

a) 编写一段程序，使其实现进程的软中断通信。

要求：使用系统调用fork() 创建两个子进程，再用系统调用signal() 让父进程捕捉键盘上来的中断信号（即按DEL键）；当捕捉到中断信号后，父进程用系统调用Kill() 向两个子进程发出信号，子进程捕捉到信号后分别输出下列信息后终止：

```

Child Process 1 is killed by Parent!
Child Process 2 is killed by Parent!

```

父进程等待两个子进程终止后，输出如下的信息后终止：

```

Parent Process is killed!

```

代码如下：

```

#include <signal.h>
#include <unistd.h>
#include <sys/wait.h>
#include "stdio.h"
#include "stdlib.h"

#define CHECK(x) if(!(x)) { perror("#x " failed"); abort();}

pid_t pid1, pid2;

static void del_handler(int sig)
{
    printf("OUTH\n");
    kill(pid1, SIGUSR1);
}

```

```

    kill(pid2, SIGUSR1);
}

static void children_1st_handler(int sig)
{
    printf("Child Process 1 is killed by Parent!\n");
    exit(0);
}

static void children_2nd_handler(int sig)
{
    printf("Child Process 2 is killed by Parent!\n");
    exit(0);
}

int main(void)
{

    pid1 = fork();
    if(pid1 == 0){
        CHECK(setpgid(0, 0) == 0);
        if(signal(SIGUSR1, children_1st_handler) == SIG_ERR){
            exit(-1);
        }
        while( 1 ){
            pause();
        }
    }else if (pid1 == -1){
        exit(1);
    }else{
        pid2 = fork();
        if(pid2 == 0){
            CHECK(setpgid(0, 0) == 0);
            if(signal(SIGUSR1, children_2nd_handler) == SIG_ERR){
                exit(-1);
            }
        }
    }
}

```

```

while( 1 ){
    sleep(1);
}
} else if(pid2==1){
    exit(-1);
} else{
    if(signal(SIGINT, del_handler) == SIG_ERR){
        exit(-1);
    }
    waitpid(pid1, NULL, 0);

    waitpid(pid2, NULL, 0);

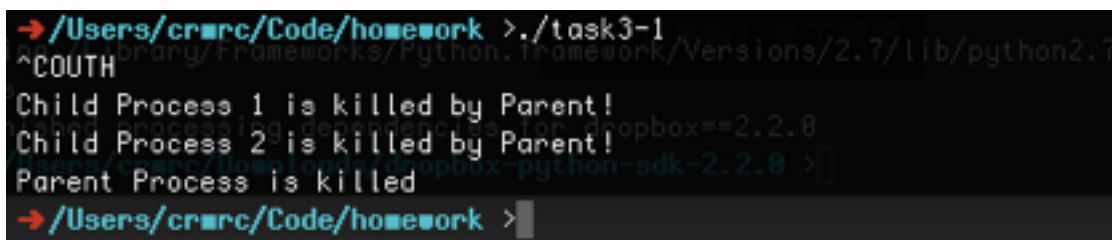
    printf("Parent Process is killed\n");
}
}
}

```

搜到的答案不对。。。。

SIGINT	当用户按中断键（一般采用DELETE或Ctrl+C）时，终端驱动程序产生此信号并送至前台进程组中的每一个进程（见图9-8）。当一个进程在运行时失控，特别是它正在屏幕上产生大量不需要的输出时，常用此信号终止它。
---------------	--

所以要改子进程的组id。。。。



```

➔ /Users/crmrc/Code/homework > ./task3-1
^COUTH
Child Process 1 is killed by Parent!
Child Process 2 is killed by Parent!
Parent Process is killed
➔ /Users/crmrc/Code/homework >

```

b) 在上面的程序中增加语句 `signal(SIGINT, SIG_IGN)` 和 `signal(SIGQUIT, SIG_IGN)`，观察执行结果，并分析原因。


```
if(signal(SIGINT, SIG_IGN) == SIG_ERR){
    exit(-1);
}

if(signal(SIGQUIT, SIG_IGN) == SIG_ERR){
    exit(-1);
}

// if(signal(SIGINT, del_handler) == SIG_ERR){
//     exit(-1);
// }
```

9/12

Child 1 is sending a message!

Child 2 is sending a message!

而父进程则从管道中读出来自于两个子进程的信息，显示在屏幕上。

要求父进程先接收子进程P1发来的消息，然后再接收子进程P2发来的消息。

代码如下：

```
#include <sys/types.h>
```

```
#include <signal.h>
```

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

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

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

```
pid_t pid1, pid2;
```

```
void main(int argc, char* argv[])
```

```
{
```

```
    int fd[2];
```

```
    char OutPipe[100], InPipe[100];
```

```
    pipe(fd);
```

```
    pid1 = fork();
```

```
    if(pid1 < 0){
```

```
        fprintf(stderr, "Fork failed");
```

```
    }
```

```
    else if (pid1 == 0){
```

```
        lockf(fd[1], F_LOCK, 0);
```

```
        sprintf(InPipe, "child 1 process is sending message!");
```

```
        write(fd[1], InPipe, 50);
```

```
        lockf(fd[1], F_ULOCK, 0);
```

```
        exit(0);
```

```
    }
```

```

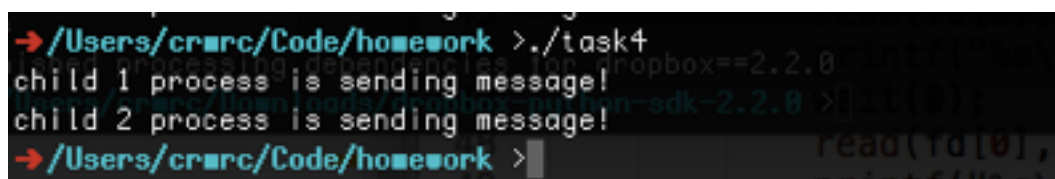
else{
    pid2 = fork();
    if(pid2 < 0){
        fprintf(stderr, "Fork failed");
    }
    else if (pid2 == 0){

        lockf(fd[1],F_LOCK,0);

        sprintf(InPipe,"child 2 process is sending message!");
        write(fd[1], InPipe, 50);
        lockf(fd[1],F_ULOCK,0);
        exit(0);

    }
    else{
        wait(0);
        read(fd[0], OutPipe, 50);
        printf("%s\n",OutPipe);
        wait(0);
        read(fd[0], OutPipe, 50);
        printf("%s\n",OutPipe);
        exit(0);
    }
}
}
}

```



```

→ /Users/crmrc/Code/homework > ./task4
child 1 process is sending message!
child 2 process is sending message!
→ /Users/crmrc/Code/homework >

```

要求父进程先接收子进程P1发来的消息，然后再接收子进程P2发来的消息。

不会做。要么多加一个进程，先管道到它那里但同时要在管道信息里说明自己是哪个进程，然后由该进程排完序后再管道到父进程。只改子进程自身没意义。

4. 思考

(1) 系统是怎样创建流程（进程？？？？？？？？？？？？？？）的？

进程由 `fork()` 系统调用创建子进程（复制父进程的代码段数据段以及环境）。Linux 中所有进程都是进程 0 的子进程，调用一次返回两次，如果调用成功，在父进程中返回子进程 `uid`，在子进程中返回 0。

(2) 可执行文件加载时进行了哪些处理？

先判断该文件是否是一个合法的可执行文件，如果是，操作系统将按照段表中的指示为可执行程序分配地址空间，加载运行。

(3) 当首次调用新创建进程时，其入口在哪里？

`if (fork == 0)` 之后

(4) 进程通信有什么特点？

同步与互斥是进程通信两部分。

进程同步源于进程合作，是进程间共同完成一项任务时直接发生相互作用的关系。为进程之间的直接制约关系。在多道环境下，这种进程间在执行次序上的协调是必不可少的。

进程互斥源于资源共享，是进程之间的间接制约关系。在多道系统中，每次只允许一个进程访问的资源称为临界资源，进程互斥就是保证每次只有一个进程使用临界资源。