


深圳大学 计算机与软件学院

College of Computer Science and Software Engineering of Shenzhen University

系统编程

基于TaiShan服务器/openEuler OS 的实践

第一讲：课程介绍 & 进程控制 （中）

fork() - 例4

```
int main(int argc, char **argv) {
        int remaining = 4;
        int child_pid;

        /* spawn off children */
        while(remaining > 0) {
            child_pid = fork();
            if(child_pid == 0) break;
           remaining--;
        }

        printf("P");
        return 0;
}
```

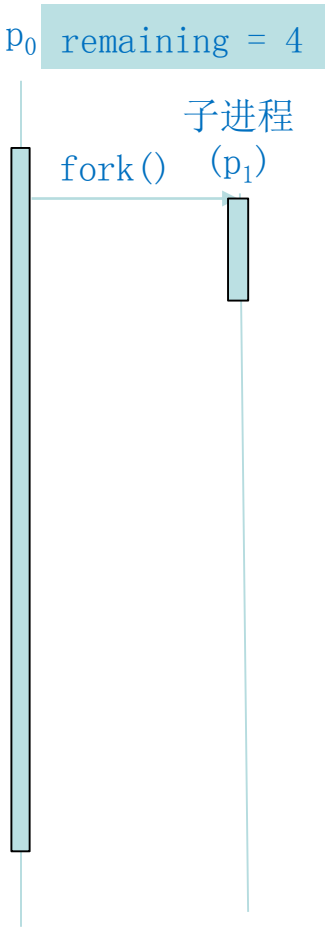
程序的输出是什么?

fork() – 例4

```
int main(int argc, char **argv) {
        int remaining = 4;
        int child_pid;

        /* spawn off children */
        while(remaining > 0) {
            child_pid = fork();
            if(child_pid == 0) break;
           remaining--;
       }

       printf("P");
       return 0;
}
```



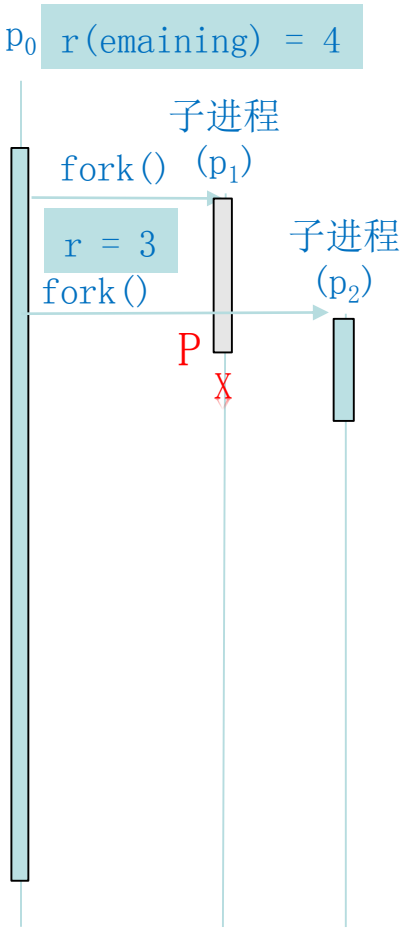
程序的输出是什么?

fork() – 例4

```
int main(int argc, char **argv) {
        int remaining = 4;
        int child_pid;

        /* spawn off children */
        while(remaining > 0) {
            child_pid = fork();
            if(child_pid == 0) break;
           remaining--;
        }

        printf("P");
        return 0;
}
```



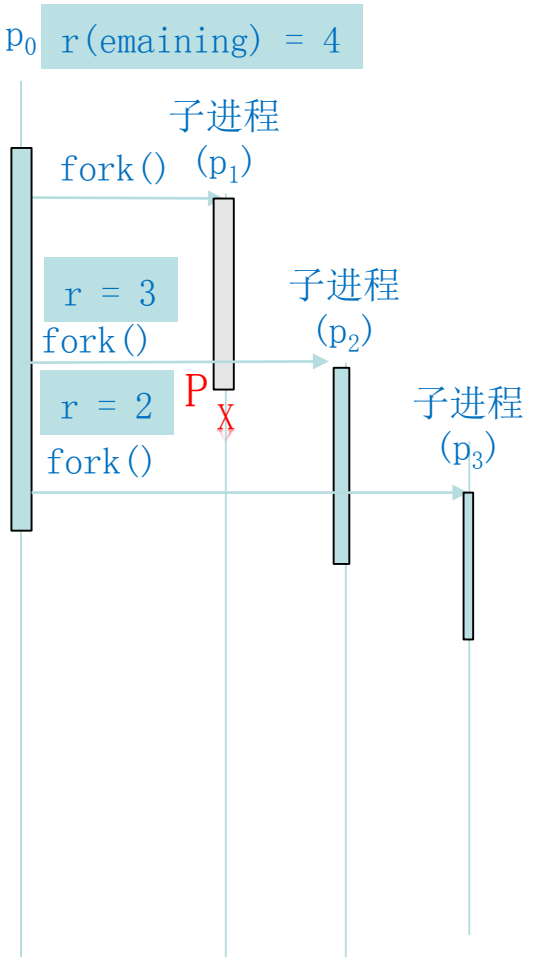
程序的输出是什么?

fork() – 例4

```
int main(int argc, char **argv) {
        int remaining = 4;
        int child_pid;

        /* spawn off children */
        while(remaining > 0) {
            child_pid = fork();
            if(child_pid == 0) break;
           remaining--;
       }

       printf("P");
       return 0;
}
```



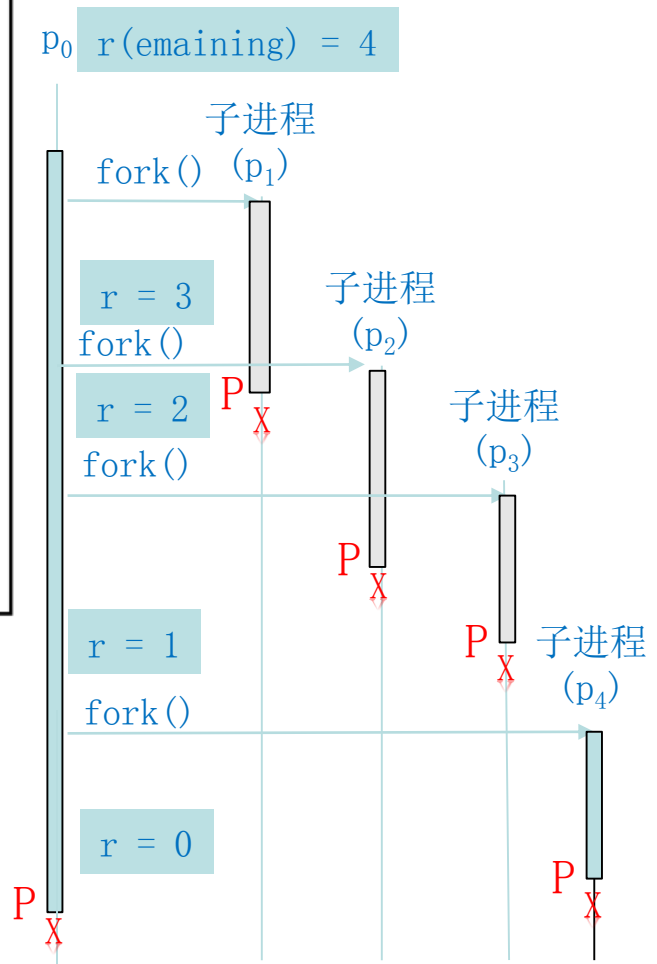
程序的输出是什么?

fork() – 例4

```
int main(int argc, char **argv) {
        int remaining = 4;
        int child_pid;

        /* spawn off children */
        while(remaining > 0) {
            child_pid = fork();
            if(child_pid == 0) break;
           remaining--;
       }

       printf("P");
       return 0;
}
```



程序的输出是什么?

fork() - 例5

```
int main(int argc, char **argv) {
        int remaining = 4;
        int child_pid;

        /* spawn off children */
        while(remaining > 0) {
            child_pid = fork();
        // if(child_pid == 0) break;
           remaining--;
       }

       printf("P");
       return 0;
}
```

程序的输出是什么？

Error-handling Wrappers

- We simplify the code we present to you even further by using Stevens[1]-style error-handling wrappers:

```
pid_t Fork(void)
{
    pid_t pid;

    if ((pid = fork()) < 0)
        unix_error("Fork error");
    return pid;
}
```

```
pid = Fork();
```

- NOT what you generally want to do in a real application

[1]e.g., in "UNIX Network Programming: The sockets networking API" W. Richard Stevens

更多查询进程特定信息的命令

假设进程id = x

- 进程属性和状态

```
$ps -ef|grep x
$ps  -axj | grep x
```

- 进程打开的文件

```
$lsof -p x
```

- 进程内存分配

```
$cat /proc/x/maps
```

学有余力的可以试下
有些需要安装特定的包

- 进程间关系

```
$pstree
```

```
[yuhong@taishan02-vm-10 ~]$ pstree
systemd─┬─NetworkManager─┬─dhclient
        │                └─2*[{NetworkManager}]
        ├─2*[agetty]
        ├─auditd───{auditd}
        ├─chronyd
        ├─crond
        ├─dbus-daemon───{dbus-daemon}
        ├─firewalld───{firewalld}
        ├─irqbalance───{irqbalance}
        ├─polkitd───7*[{polkitd}]
        ├─restorecond
        ├─rngd
        ├─rsyslogd───2*[{rsyslogd}]
        ├─sshd─┬─sshd───sshd───bash───pstree
```

- 进程堆栈

```
$pstack x
```

- 进程调用的系统调用

```
$strace -p x
```

- 进程调用的库函数

```
$ltrace -p x
```

孤儿进程

- 父进程退出,子进程还在运行
  - 子进程将成为孤儿进程
- 孤儿进程将特殊进程接管
  - init进程 (fedora)
  - init-user进程 (ubuntu)
  - systemd进程 (centos, openeuler)
- 孤儿进程不会导致资源浪费

Linux中的两个特殊的进程（跟系统有关）

- 0号进程：所有进程的祖先
  - swapper 进程（交换进程、调度进程）
  - 系统进程
  - 没有其他进程处于TASK_RUNNING状态
  - 内核会选择0号进程运行
    - 不执行任何磁盘上的程序
    - 执行cpu_idle()函数

- 1(x-系统相关)号进程
  - 创建和监控其他进程的活动
  - 接管孤儿进程

终止一个进程

- 显式的系统调用

```
#include <stdlib.h>
void exit(int status)
#include <unistd.h>
void _exit(int status)
```

- exit() vs. _exit()
  - exit()调用_exit()函数之前
    - 调用atexit()/on_exit()等注册的出口函数
    - 将文件缓冲区内容写回文件，释放资源
  - _exit()直接关闭进程，文件缓冲区中的数据会丢失

- 调用后，进程并非整体消逝---留下僵尸进程数据结构

终止一个进程

- 显式的系统调用
- 从程序的结尾离开
- 被信号终止
  - SIGTERM ： 可被阻塞和处理
  - SIGKILL：不可被忽略，立即处理

终止一个进程

- 显式的系统调用
- 从程序的结尾离开
- 被信号终止：SIGTERM, SIGKILL

  kill [-s <信号名称或编号>][程序]

  kill [-l <信号编号>]

  若不加<信息编号>选项，则-l参数会列出全部的信息名称。

  //强行中止（杀掉）一个进程pid为324的进程：

  #kill -9 324

  #free
- 被内核杀掉：Segmentation violation

终止一个进程

- 进程终止时,内核会传送SIGCHLD信号给其父进程

- 若子进程终止时整体消失,父进程将无法获取其运行信息
- 若子进程先于父进程终止
  - ① 内核让子进程进入一个特殊的进程状态
    - ◆僵死状态
    - ◆僵尸进程(zombie)
    - ◆等待父进程来打听它的状态
  - ② 父进程取得Zombie的信息
  - ③ Zombie正式结束

僵尸进程的内核数据结构

- 进程处于僵死状态时，只保留最小的数据结构
  - 进程的pid
  - 退出状态

僵尸进程的避免

1. 父进程通过wait和waitpid等函数等待子进程结束
   - 导致父进程立即阻塞自己，直到有一个子进程退出

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

pid_t wait(int *status);
```

返回值

1. 结束的子进程pid
2. -1，如果没有子进程

status

1. 是否正常结束: 0, 正常退出；非0值：非正常退出
2. 正常退出
   1. 高字节：子进程exit时设置的代码，低字节为0
   2. 如果子进程的退出是因为收到信号，低字节为信号的编码

wait()的使用-例1

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main()
{
        pid_t pid;
        int status, exit_status;

        if ((pid = fork()) < 0) perror("fork failed");
        if (pid == 0){
                exit(EXIT_SUCCESS);
        } else {
                printf("Hello, I am parent process %d with child %d\n",getpid(),pid);
        }
        if ((pid = wait(&status)) == -1) {
                perror("wait failed");
                exit(EXIT_FAILURE);
        }

        if ((exit_status = WIFEXITED(status))){
                printf("exit status from %d is %d\n",pid, WEXITSTATUS(status));
        }
        exit(0);
}
```

```
[szu@taishan02-vm-10 wait]$ gcc -o exitstatus exitstatus.c
[szu@taishan02-vm-10 wait]$ ./exitstatus
Hello, I am parent process 1706562 with child 1706563
exit status from 1706563 was 0 is 0
[szu@taishan02-vm-10 wait]$
```

wait(NULL)

status>>8？

状态标志一览

The int *status* could be checked with the help of the following macros:

- WIFEXITED(status): returns true if the child terminated normally
- WEXITSTATUS(status): return
  the 8 bits of the *status* argume
  the argument for a return state
  if WIFEXITED returned true.
- WIFSIGNALED(status): ret
  signal.
- WTERMSIG(status): returns t
  process to terminate. This ma
  returned true.
- WCOREDUMP(status): retur
- WSTOPSIG(status): return
  stop.

首先正常退出，状态才有意义

练习1：模仿上一页的例子，编写代码，通过另一个终端发送信号终止子进程，父进程收集子进程退出前收到的信号并打印。

练习2：模仿上一页的例子，编写代码，子进程因缓冲区溢出，产生违例访问地址错误退出，父进程收集子进程退出前收到的信号并打印。

僵尸进程的避免

1. 父进程通过wait和waitpid等函数等待子进程结束

- 导致父进程立即阻塞自己，直到有一个子进程退出

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

pid_t wait(int *status);
```

- 等待指定子进程的退出

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

pid_t waitpid(pid_t pid, int *status, int options);
```

pid可以取以下值

① < -1： 等待进程组id为pid绝对值的子进程结束

② -1： 等待任意子进程结束

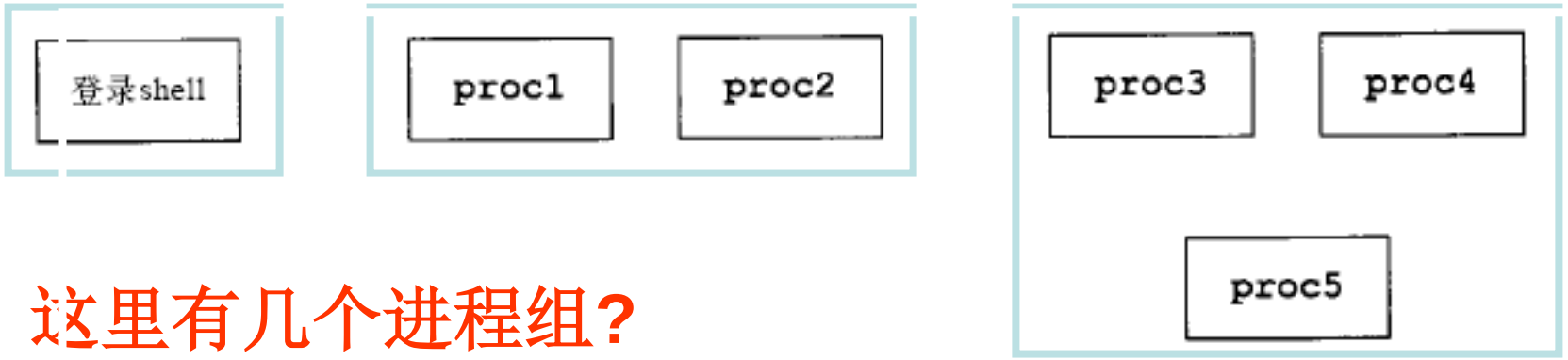
③ 0： 等待进程组id跟父进程进程组id相同的子进程结束

④ >0：等待进程id为pid的子进程结束

进程组

- 一个或多个进程的集合
- 作业控制
- getpgrp() & setpgid()

```
proc1 | proc2 &
proc3 | proc4 | proc5
```



这里有几个进程组?

僵尸进程的避免

1. 父进程通过wait和waitpid等函数等待子进程结束
   - 导致父进程立即阻塞自己，直到有一个子进程退出

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

pid_t wait(int *status);
```

   - 等待指定子进程的退出

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

pid_t waitpid(pid_t pid, int *status, int options);
```

Options可以是以下几个常数中的一个或多个

① WNOHANG：如果没有子进程执行或结束则返回

② WUNTRACED:调用进程阻塞，直到子进程结束或被停止

③ WCONTINUED:调用进程阻塞，直到已停止的子进程收到SIGCONT信号

waitpid()使用例子

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

int main(){
   pid_t  pid;
   int status, exit_status,i ;

   if ( (pid = fork()) < 0 )
      perror("fork failed");
   if ( pid == 0 ){
    printf("Still child %lu is sleeping... \n",(long)getpid());
        sleep(5); exit(57);
        }
   printf("reaching the father %lu process \n",(long)getpid());
   printf("PID is %lu \n", (long)pid);
   while( (waitpid(pid, &status, WNOHANG)) == 0 ){
        printf("Still waiting for child to return\n");
        sleep(1);
   }
   printf("reaching the father %lu process \n",(long)getpid());
   if (WIFEXITED(status)){
       exit_status = WEXITSTATUS(status);
       printf("Exit status from %lu was %d\n", (long)pid, exit_status);
       }
   exit(0);
}
```

waitpid()例子程序的结果

```
reaching the father 17321 process
PID is 17322
Still waiting for child to return
Still child 17322 is sleeping...
Still waiting for child to return
Still waiting for child to return
Still waiting for child to return
Still waiting for child to return
reaching the father 17321 process
Exit status from 17322 was 57
```

wait(...) and waitpid(...)的关系?

僵尸进程的避免

1. 父进程通过wait和waitpid等函数等待子进程结束
2. 如果父进程很忙,可以用signal函数为SIGCHLD安装handler,因为子进程结束后,父进程会收到该信号,可以在handler中调用wait回收

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

void handler(int num){
        int status;
        int pid = waitpid(-1,&status, WNOHANG);
        if (WIFEXITED(status)){
                printf("The child %d exit with code %d\n",pid,WEXITSTATUS(status));
        }
}

int main(){
        pid_t pid,pid1;
        int i;

        signal(SIGCHLD,handler);

        if ((pid = fork())){
                pid1 = pid;
                printf("The child process is %d\n",pid1);
                for (i=0; i<10; i++){
                        printf("Do parent things.\n");
                        sleep(1);
                }
                exit(0);
        } else {
                printf("I'm a child.\n");
                sleep(2);
                exit(0);
        }
}
```

```
The child process is 4252
Do parent things.
I'm a child.
Do parent things.
Do parent things.
The child 4252 exit with code 0
Do parent things.
Do parent things.
Do parent things.
Do parent things.
Do parent things.
Do parent things.
Do parent things.
```

僵尸进程的避免

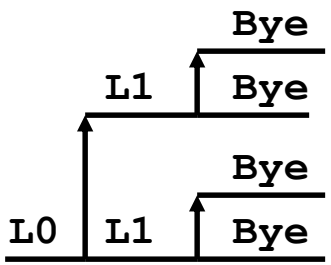
1. 父进程通过wait和waitpid等函数等待子进程结束
2. 如果父进程很忙,可以用signal函数为SIGCHLD安装handler,因为子进程结束后,父进程会收到该信号,可以在handler中调用wait回收
3. 如果父进程不关心进程什么时候结束,那么可以用signal(SIGCHLD,SIG_IGN)通知内核,内核会回收,并不再给父进程发送信号
4. Stevens的两次fork避免僵尸进程:就是fork两次，父进程fork一个子进程，然后继续工作，子进程fork一 个孙进程后退出，那么孙进程被init接管，孙进程结束后，init会回收。不过子进程的回收还要自己做

```
#include <stdio.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <unistd.h>
int main(void)
{
   pid_t pid;
    if ((pid = fork()) < 0) {  fprintf(stderr,"Fork error!\n");  exit(-1); }
   else if (pid == 0) /* first child */
   {
       if ((pid = fork()) < 0)  { fprintf(stderr,"Fork error!\n");  exit(-1); }
      else if (pid > 0)  exit(0); /* parent from second fork == first child */
      sleep(2);
      printf("Second child, parent pid = %d\n", getppid());
      exit(0);
   }
  if (waitpid(pid, NULL, 0) != pid) /* wait for first child */
  {
     fprintf(stderr,"Waitpid error!\n");  exit(-1);
  }
   exit(0);
}
```

关于fork的一些问题

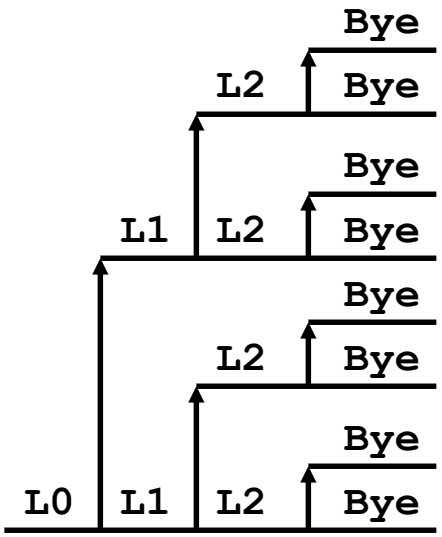
- 父亲和儿子都可以继续fork…

```
void fork2()
{
    printf("L0\n");
    fork();
    printf("L1\n");
    fork();
    printf("Bye\n");
}
```



这个应该没有问题了……

```
void fork3()
{
    printf("L0\n");
    fork();
    printf("L1\n");
    fork();
    printf("L2\n");
    fork();
    printf("Bye\n");
}
```



自问自答自验证- 以下程序运行结果是什么?

```
void fork4()
{
    printf("L0\n");
    if (fork() != 0) {
        printf("L1\n");
        if (fork() != 0) {
            printf("L2\n");
            fork();
        }
    }
    printf("Bye\n");
}
```

```
void fork5()
{
    printf("L0\n");
    if (fork() == 0) {
        printf("L1\n");
        if (fork() == 0) {
            printf("L2\n");
            fork();
        }
    }
    printf("Bye\n");
}
```

Kind of Guided Study…..

fork()函数

```
int main(int argc, char **argv) {
        int remaining = 4;
        int child_pid;

        /* spawn off children */
        while(remaining > 0) {
            child_pid = fork();
            if(child_pid == 0) break;
           remaining--;
       }

       printf("P");
       return 0;
}
```

请修改代码美化程序的输出

- 原来: [yuhong@taishan02-vm-10 Process]$./a.out
  PP[yuhong@taishan02-vm-10 Process]PPP
- 期望: [yuhong@taishan02-vm-10 Process]$./a.out
  PPPPP
  [yuhong@taishan02-vm-10 Process]

课堂讨论题:

1. 编例实现创建n个子进程P1，P2，…，Pn,其中，各进程之间的关系是：P1是调用进程的子进程，P(k+1)是Pk的子进程。请打印各进程本身的进程号、父进程号，子进程号。参考运行结果如下。要求：（1）每个父进程都要等待子进程退出后才能退出；（2）n通过命令行参数传入；（3）附上源代码截图和运行结果截图。（20分）

程序一

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

int main(int argc,char *argv[])
{
        pid_t childpid = 0;
        int i, n;

        if (argc !=2){
                fprintf(stderr, "Usage: %s processes \n",argv[0]);
                return 1;
        }

        n = atoi(argv[1]);

        for (i = 0; i < n; i++){
                //printf("Process Id: %d, parent Id: %d\n", getpid(), getppid());
                childpid = fork();
                if (childpid > 0){
                        printf("Process Id: %d, parent Id: %d", getpid(), getppid());
                        printf("Child Id: %d\n",childpid);
                        break;
                } else if (childpid < 0){
                        printf("fork error\n");
                        exit(-1);
                }
        }
        if (i==n) {
                        printf("Process Id: %d, parent Id: %d ", getpid(), getppid());
                        printf("Child Id:0\n");
        }
        if (childpid>0) waitpid(childpid,NULL,0);
}
```

```
Process Id: 10939, parent Id: 5043Child Id: 10940
Process Id: 10940, parent Id: 10939Child Id: 10941
Process Id: 10941, parent Id: 10940 Child Id:0
```

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

int main(int argc,char *argv[])
{
        pid_t childpid = 0;
        int i, n;

        if (argc !=2){
                fprintf(stderr, "Usage: %s processes \n",argv[0]);
                return 1;
        }

        n = atoi(argv[1]);

        for (i = 0; i < n; i++){
                printf("Process Id: %d, parent Id: %d", getpid(), getppid());
                childpid = fork();
                if (childpid > 0){
                //      printf("Process Id: %d, parent Id: %d", getpid(), getppid());
                        printf("Child Id: %d\n",childpid);
                        break;
                } else if (childpid < 0){
                        printf("fork error\n");
                        exit(-1);
                }
        }
        if (i==n) {
                        printf("Process Id: %d, parent Id: %d ", getpid(), getppid());
                        printf("Child Id:0\n");
        }
        if (childpid>0) waitpid(childpid,NULL,0);
}
```

```
Process Id: 10909, parent Id: 5043Child Id: 10910
Process Id: 10909, parent Id: 5043Process Id: 10910, parent Id: 10909Child Id: 10911
Process Id: 10909, parent Id: 5043Process Id: 10910, parent Id: 10909Process Id: 10911, parent Id: 10910 Child Id:0
```

程序二

```
Process Id: 10959, parent Id: 5043
Child Id: 10960
Process Id: 10960, parent Id: 10959
Child Id: 10961
Process Id: 10961, parent Id: 10960 Child Id:0
```

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

int main(int argc,char *argv[])
{
        pid_t childpid = 0;
        int i, n;

        if (argc !=2){
                fprintf(stderr, "Usage: %s processes \n",argv[0]);
                return 1;
        }

        n = atoi(argv[1]);

        for (i = 0; i < n; i++){
                printf("Process Id: %d, parent Id: %d\n", getpid(), getppid());
                childpid = fork();
                if (childpid > 0){
                        //printf("Process Id: %d, parent Id: %d", getpid(), getppid());
                        printf("Child Id: %d\n",childpid);
                        break;
                } else if (childpid < 0){
                        printf("fork error\n");
                        exit(-1);
                }
        }
        if (i==n) {
                        printf("Process Id: %d, parent Id: %d ", getpid(), getppid());
                        printf("Child Id:0\n");
        }
        if (childpid>0) waitpid(childpid,NULL,0);
}
```

程序三

明白了吗?

还有问题吗?

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

int main()
{
    for (int i = 0; i < 3; i++)
    {
        fork();
        printf("*\n");
    }

    return 0;
}
```

1、程序将打印多少个“*”？

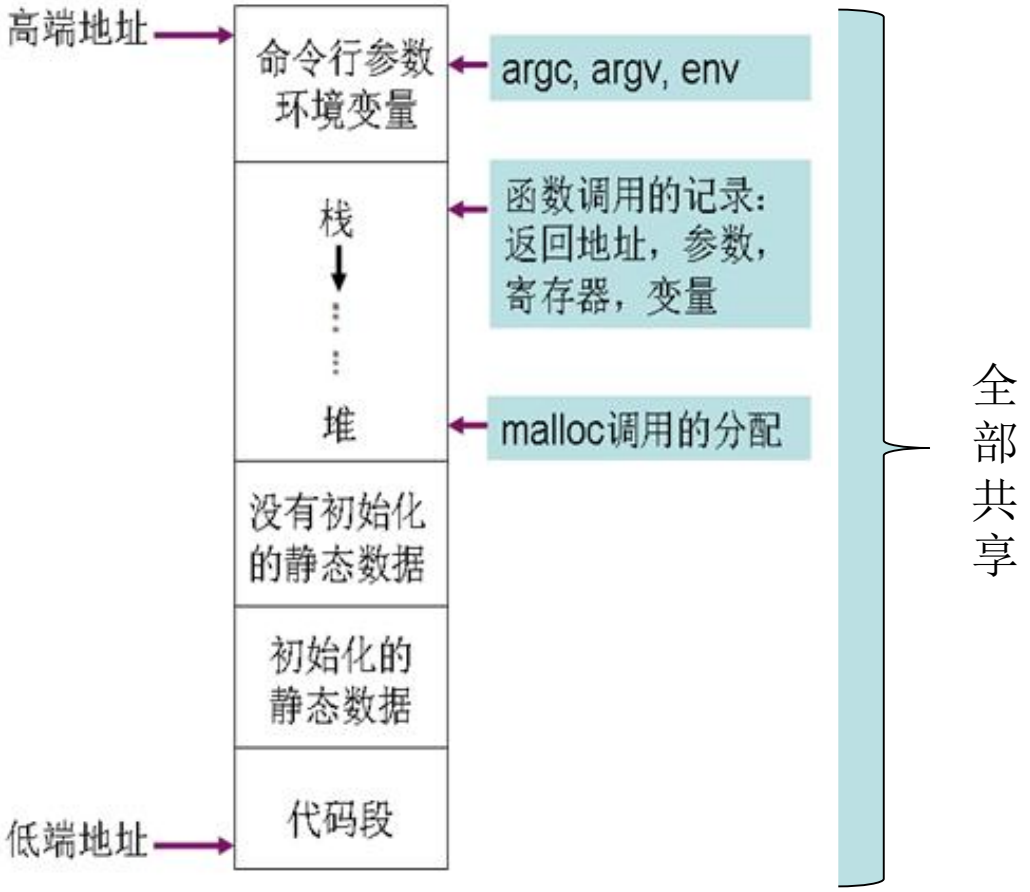
2、如果“*”后面去掉\n呢，程序将打印多少个*？

vfork()

- 父子进程运行顺序

  ① 父进程暂停
  ② 子进程先运行，直至显式调用_exit(0)或exec()族函数成功
  ③ 父进程重新被调度

- 父子进程共享地址空间和页表



下面的程序运行结果分别是什么？

(1)

```c
#include <unistd.h>
#include <stdio.h>
int main(void)
{
        pid_t pid;
        int count = 0;
        pid = fork();
        count++;
        printf("count=%d\n",count);
        return 0;
}
```

(2)

```c
#include <unistd.h>
#include <stdio.h>
int main(void)
{
        pid_t pid;
        int count = 0;
        pid = vfork();
        count++;
        printf("count=%d\n",count);
        return 0;
}
```

(3)

```c
#include <unistd.h>
#include <stdio.h>
int main(void)
{
        pid_t pid;
        int count = 0;
        pid = vfork();
        if (pid ==0){
                count++;
                printf("count=%d\n",count);
                _exit(0);
        } else {
                count++;
                printf("count=%d\n",count);
        }
        return 0;
}
```

思考题

当创建子进程是为了调用exec家族函数时，你选 ___?

A. fork()　　B. vfork()

- vfork is similar to fork but does not create a copy of the data of the parent process. Instead,it shares the data between the parent and child process. This saves a great deal of CPU time (and if one of the processes were to manipulate the shared data, the other would notice automatically).
- vfork is designed for the situation in which a child process just generated immediately executes an execve system call to load a new program. The kernel also guarantees that the parent process is blocked until the child process exits or starts a new program.
- Quoting the manual page vfork(2), it is ‘‘rather unfortunate that Linux revived this specter from the past.’’ Since fork uses copy-on-write, the speed argument for vfork does not really count anymore, and its use should therefore be avoided.

Wolfgang Mauerer, Linux Kernel Architecture, Wiley Publishing, Inc.