上海电力学院 LINUX/UNIX操作系统

实验报告说明：

实验报告的一二三四部分阅读后不做改动。

第五部分实验结果记录，主要是每道题的答案的展示，按照题目要求，

有每一题的代码截图和程序运行结果的截图，以及问答题的文字答案，

流程图等。注意截图时截取有效信息。

第六部分实验结果分析，主要是每道题的解题过程的简要说明，以及

对实验结果的简要分析。

第七部分实验心得是对该实验的总体感悟。

实验时，所有以学号 20150001为例的文本都要求改为你个人的学号。

Linux程序设计实验报告 6

——进程控制与线程

一、实验目的

1.掌握进程的常用终端命令；

2.掌握用 system、exec函数簇、fork函数创建进程;

3.掌握 waitpid函数的应用;

4.掌握守护进程过程;

5.掌握守护进程在各种监控中的应用。

6.了解 LINUX 下线程和进程的概念

7.了解多线程程序的基本原理

8.了解 PTHREAD 库，学会使用 PTHREAD 库中的函数编写多线程程序

二、实验内容

1.进程的常用终端命令；

2.用 execl函数创建进程;

3.应用 fork函数创建子进程;

4.应用 fork函数创建子进程,在父子进程中执行不同的任务;

5.waitpid函数的应用;

6.守护进程的编写与应用;

7.守护进程在邮件监控中的应用;

8.守护进程在文件监视中的应用。

9. 熟悉新线程的创建，以及和主线程之间的关系

三、实验设备

计算机 PC 机，Centos 操作系统或 Ubuntu 操作系统

四、实验步骤

**1.** 进程的常用终端命令及函数。

（1）学习 at 指令的使用。写出 at 指令的使用格式。在当前时间 2 分钟后，通过 at 指

令运行命令"ls

-l"

（2）下列是进程调度中常用的函数，通过网络资源查找这些函数的功能与用法。

函 数

功能与用法

kill

raise

alarm

pause

signal

sigemptyset

sigfillset

sigaddset

sigdelset

sigismember

sigprocmask

2． execl函数的应用。

程序设计，用 execl函数创造进程 ls -l，用 execvp函数创造进程 ps -ef。提示：

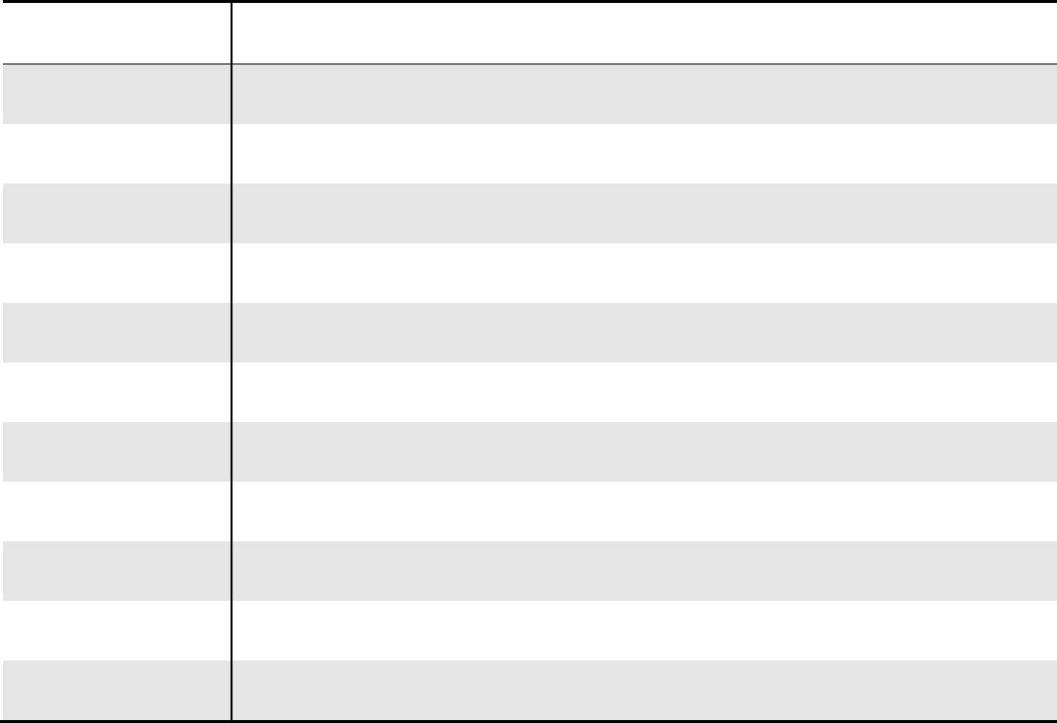
显示当前目录下的文件信息可执行以下语句：

(1) execl("/bin/ls", "ls", "-al", NULL) ;

(2) char \*arg[] = {"ps", "-ef", NULL};

execvp("ps", arg);

3. 应用 fork函数创建子进程。



（1）调试下列程序，改正程序中少量的错误，写出程序的功能与程序的运行结果。

#include<stdio.h>

#include<stdlib.h>

#include<signal.h>

#include<sys/types.h>

#include<sys/wait.h>

#include<unistd.h>

int main ()

/\*文件预处理，包含标准输入输出库\*/

/\*文件预处理，包含 system、exit等函数库\*/

/\*文件预处理，包含 kill、raise等函数库\*/

/\*文件预处理，包含 waitpid、kill、raise等函数库\*/

/\*文件预处理，包含 waitpid函数库\*/

/\*文件预处理，包含进程控制函数库\*/

/\*C程序的主函数，开始入口\*/

{

pid\_t result;

result=fork();

int newret;

/\*调用 fork函数，复制进程,返回值存在变量 result中\*/

if(result<0) /\*通过 result的值来判断 fork函数的返回情况，这儿进行出错处理\*/

{

perror("创建子进程失败");

exit(1);

}

else if (result==0)

{

/\*返回值为 0代表子进程\*/

raise(SIGSTOP); /\*调用 raise函数，发送 SIGSTOP 使子进程暂停\*/

exit(0);

}

else

{

/\*返回值大于 0代表父进程\*/

printf("子进程的进程号(PID)是：%d\n",result);

if((waitpid(NULL,WNOHANG))==0)

{

if(ret=kill(result,SIGKILL)!=0)

/\*调用 kill函数，发送 SIGKILL信号结束子进程 result \*/

printf("用 kill 函 数 返 回 值 是 ： %d， 发 出 的 SIGKILL 信 号 结 束 的 进 程 进 程

号：%d\n",ret,result);

else{ perror("kill函数结束子进程失败");}

}

}

}

int ret;

（2）改写程序，父进程等待较长时间在作业的子进程，子进程退出后，父进程再退

出。

**4**．应用 fork函数创建子进程,在父子进程中执行不同的任务;

程序设计，实现：

（1）在父子进程中分别编写循环程序，应用函数 sleep 的不同参数等，体现程序中父子进

程的并发运行。

（2）。

**5.** wait、waitpid 函数的应用。

（1）调试下列程序。程序的功能是避免子进程成为僵尸进程

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/wait.h>

int main ()

{

pid\_t pid,wpid;

int status,i;

pid=fork();

if(pid==0)

{

printf("这是子进程,进程号(pid)是:%d\n",getpid());

sleep(5);

exit(6);

/\*子进程等待 5秒钟\*/

}

else

{

printf("这是父进程,正在等待子进程……\n");

wpid=wait(&status); /\*父进程调用 wait函数，消除僵尸进程\*/

i=WEXITSTATUS(status);

printf("等待的进程的进程号(pid)是:%d ,结束状态:%d\n",wpid,i);

}

}

（2）改写程序，在子程序中用函数 system 启动一个较长时间运行的任务，而在父进程中

执行完成任务后，应用 waitpid 函数等待子进程，子进程退出后父进程才退出。

在父子进程中分别执行不同的任务，例如在子进程中执行文件编辑任务，在父进程中

执行网络连通情况的测试任务，子进程退出后父进程才退出

**6.** 守护进程的编写与应用。

用C语言设计一个闹钟程序，用户输入时间，格式为小时:分钟，例如 9:18 表示设定的时间

为 9 时 18 分，到了设定时间后，发出蜂鸣声作为提示音，为保证检测时间的准确性，要求

使用守护进程。

#include <stdio.h>

#include <time.h>

#include <unistd.h>

#include <signal.h>

#include <sys/param.h>

#include <sys/types.h>

#include <sys/stat.h>

void init\_daemon(void);

int main()

{

int hour,min;

time\_t timep;

struct tm \*p;

time (&timep);

p=localtime(&timep);/\*获取系统时间\*/

printf("这是一个闹钟程序,输入你想要设定的时间:\n");

scanf("%d:%d",&hour,&min);

init\_daemon(); /\*调用守护进程\*/

while(1)

{

sleep(20);/\*每隔 20秒检查一下时间是否已到\*/

if(p->tm\_hour==hour &&p->tm\_min==min)

{

printf("时间到了!\n");

printf("\7\7\7\7\7");/\*到了发出 5声蜂鸣，作为提示\*/

}

}

}

void init\_daemon(void)/\*这是守护进程\*/

{

pid\_t child1,child2;

int i;

child1=fork();

if(child1>0)

exit(0);

else

if(child1< 0)

{

perror("创建子进程失败");

exit(1);

}

setsid();

umask(0);

for(i=0;i< NOFILE;++i)

close(i);

return;

}

7. 守护进程在邮件监控中的应用;

程序调试。设计三个并发的守护进程在后台运行，其中第一子进程写守护进程的运行

日志记录，第二子进程 child2 则监控进程中是否有 gedit 工具调用，第二子进程 child3

则检查自己是否有新邮件到达，若有则将邮件内容输出到一个主目录下文件。

#include <stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/wait.h>

#include <unistd.h>

#include <syslog.h>

#include <signal.h>

#include <sys/param.h>

#include <time.h>

#include <dirent.h>

int main()

{

pid\_t child1,child2,child3;

struct stat buf;

int i,check=0,j=0;

time\_t t;

#include <stdio.h>

DIR \* dir;

struct dirent \* ptr;

child1=fork();

if (child1>0)

exit(0); /\*父进程退出\*/

else if(child1<0)

{

perror("创建子进程失败");

exit(1);

}

/\*第一子进程\*/

setsid();

chdir("/");

umask(0);

for(i=0;i<NOFILE;++i)

close(i);

openlog("守护进程程序信息",LOG\_PID,LOG\_DAEMON);

child2=fork();

if (child2==-1)

{

perror("创建子进程失败");

exit(2);

}

else if (child2==0)/\*第二子进程中的 child2\*/

{

i=0;

while(i++<100){

system("ps -ef|grep gedit> /home/king/gedit.log");

stat("home/king/gedit.log",&buf);

if (buf.st\_size>180 && check==0)

{

t=time(0);

syslog(LOG\_INFO,"gedit开始时间为： %s\n",asctime(localtime(&t)));

check=1;

}

if (buf.st\_size<180 && buf.st\_size>0 && check==1)

{

}

t=time(0);

syslog(LOG\_INFO,"gedit结束时间为： %s\n",asctime(localtime(&t)));

check=0;

sleep(1);

}

}

else

{ /\*在第一子进程下继续创建进程\*/

child3=fork();

if (child3<0){

perror("创建子程序失败");

exit(3);

}

else if (child3==0)

{ /\*第二子进程 child3用来查看邮件\*/

j=0;

dir=opendir("/var/spool/mail/king");

while (j<6)

{

j++;

sleep(10);

if ((ptr=readdir(dir))!=NULL)

{

system("cat /var/spool/mail/king/\* > /home/king/mail.log");

}

}

closedir(dir);

}

else

{ /\*第一子进程写日志来记录守护进程的运行\*/

t=time(0);

syslog(LOG\_INFO,"守护进程开始时间为： %s\n",asctime(localtime(&t)));

waitpid(child2,NULL,0);

waitpid(child3,NULL,0);

t=time(0);

syslog(LOG\_INFO,"守护进程结束时间为： %s\n",asctime(localtime(&t)));

closelog();

while (1)

sleep(10);

}

}

}

8. 守护进程在文件监视中的应用。

程序设计。设计一个简单的文件监视器。该程序为守护进程，能够监视/etc/

filemonitor.conf内列出的文件以及文件夹的修改时间。当检测到某个文件的修改时间发

生变化时，程序在系统日志中输出该文件的修改时间。提示：

while (1)

{

fscanf(fp, "%s", str);

if (feof(fp)) break;

strcpy(list[count++], str);

}

fclose(fp);

for (i = 0; i < count; i++)

{

stat(list[i], &buf);

mtime[i] = buf.st\_mtime;

}

while (1)

{

for (i = 0; i < count; i++)

{

stat(list[i], &buf);

if (mtime[i] != buf.st\_mtime)

{

mtime[i] = buf.st\_mtime;

syslog(LOG\_INFO,"%swas modified at %s",list[i], asctime(gmtime(&buf.st\_mtime)));

}

}

sleep(5);

}

}

9. 任务 **1.**调试下列程序。程序中使用 pthread 线程库创建一个新线程，在父进程（也可以称

为主线程）和新线程中分别显示进程 id 和线程 id，并观察线程 id 数值。

程序代码如下：

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <pthread.h> /\*pthread\_create()函数的头文件\*/

#include <unistd.h>

pthread\_t ntid;

void printids(const char \*s) /\*各线程共享的函数\*/

{

pid\_t

pid;

pthread\_t tid;

pid = getpid();

tid = pthread\_self();

printf("%s pid= %u tid= %u （0x%x）\n", s, (unsigned int)pid,

(unsigned int)tid, (unsigned int)tid);

}

void \*thread\_fun(void \*arg) /\*新线程执行代码\*/

{

printids(arg);

return NULL;

}

int main(void)

{

int err;

/\*下列函数创建线程\*/

err = pthread\_create(&ntid, NULL, thread\_fun, "我是新线程: ");

if (err != 0) {

fprintf(stderr, "创建线程失败: %s\n", strerror(err));

exit(1);

}

printids("我是父进程:");

sleep(2);/\*挂起 2秒，等待新线程运行结束\*/

return 0;

}

问题

（1）进程在一个全局变量 ntid 中保存了新创建的线程的 id，如果新创建的线程不调用

pthread\_self 而是直接打印这个 ntid，能不能达到同样的效果？

（2）在本题中，如果没有 sleep(5)函数，会出现什么结果？

（3）运行以下程序，能得出什么结论？

#include<stdio.h>

#include<pthread.h>

void \*print\_thread\_id(void \*arg)

{

/\* 打印当前线程的线程号\*/

printf("Current thread id is %u\n", (unsigned)pthread\_self());

}

int main(int argc, char \*argv[])

{

pthread\_t thread;

/\*保存线程号\*/

pthread\_create(&thread, NULL, print\_thread\_id, NULL);

/\*创建一个线程

\*/

sleep(10);

/\*休眠 1s\*/

/\*打印进程号 \*/

printf("Main thread id is %u\n", (unsigned)pthread\_self());

return 0;

}

任务 **2**．应用 pthread\_create 函数，启动一个线程，在线程中执行一个循环输出，在主进程中

执行一个较长时间执行的任务。

**五、实验结果记录**

1.

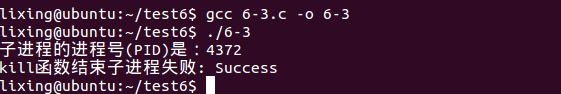
at命令用于在未来的某一个特定时刻进行一项任务，格式一般有两种

绝对时刻：at 12：46 9/7/2017

相对时刻：at now+5 min 或 at now+5 hours;

2.

3.

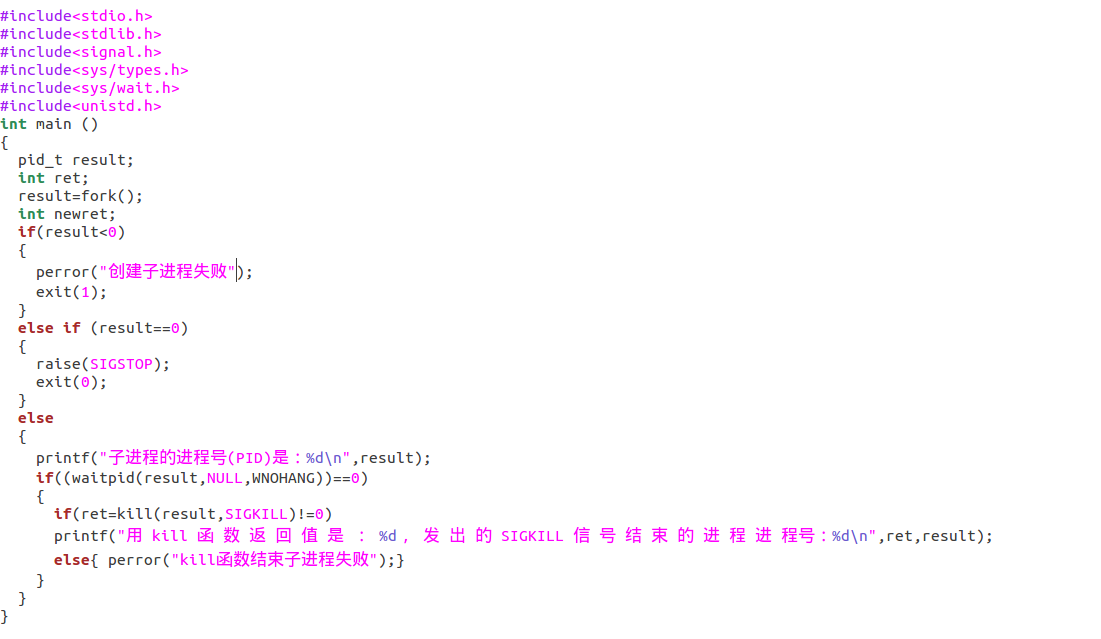


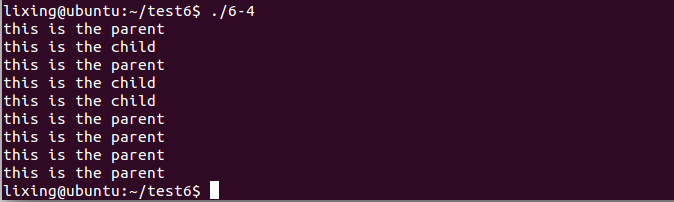
4.

5.

6.

7.

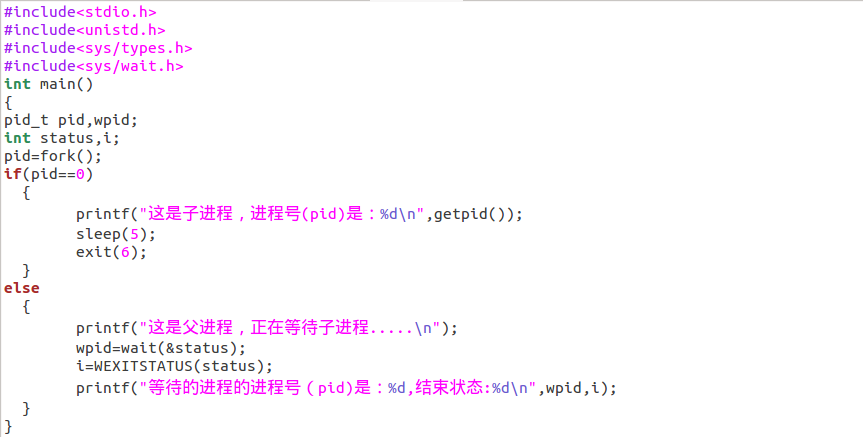




4.

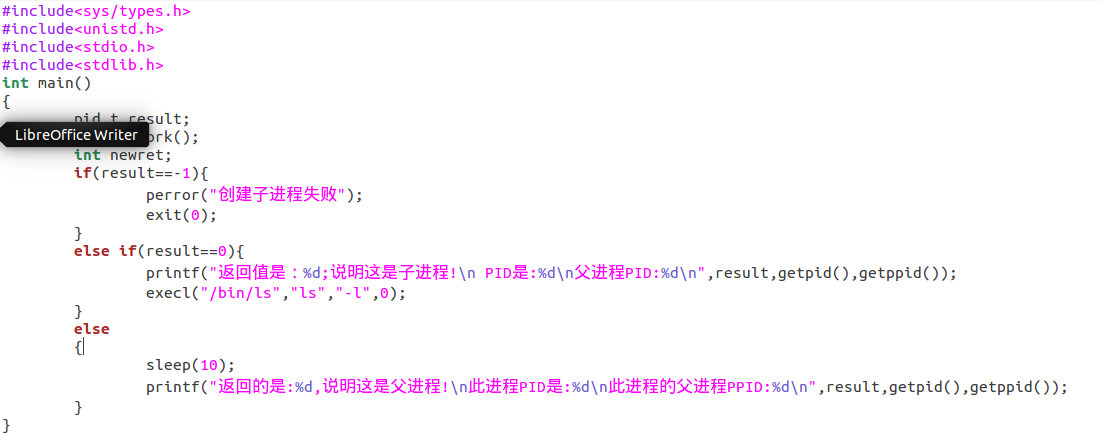


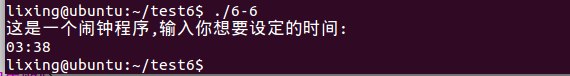
5.





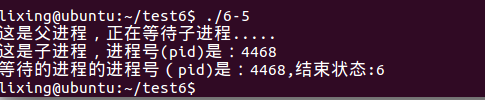
4.2





6.

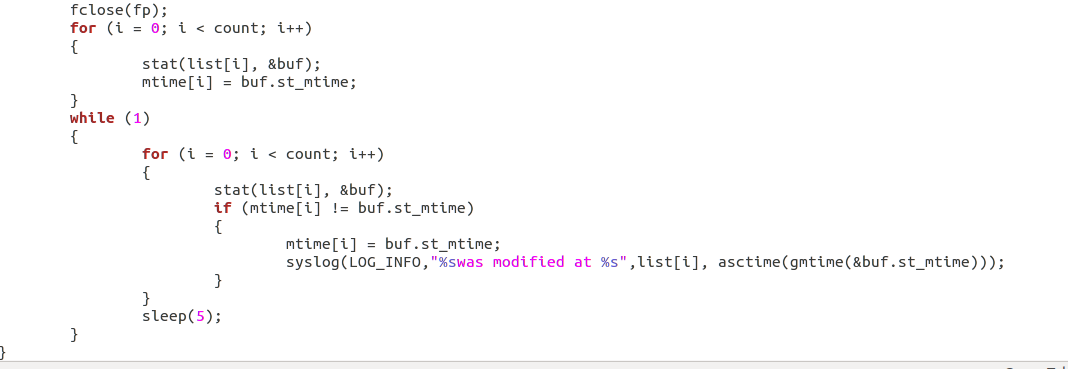


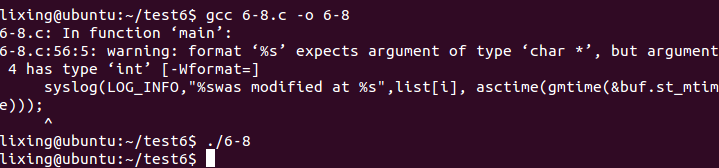




7.





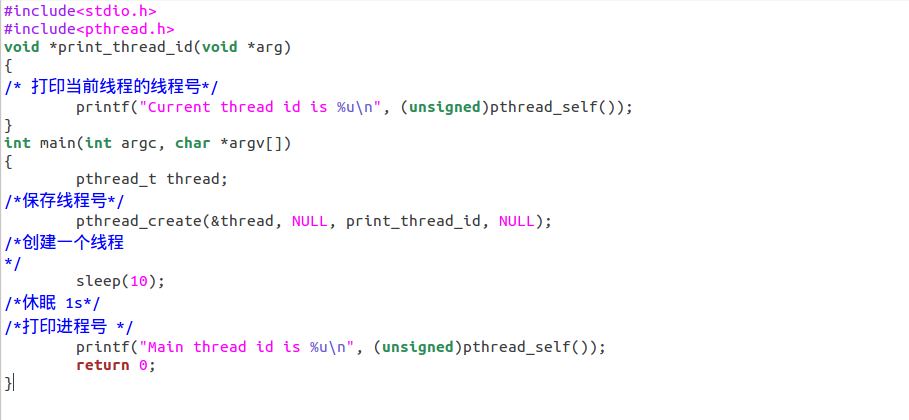


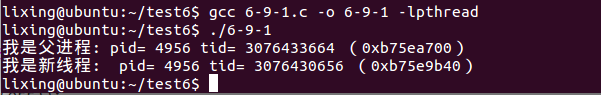
8.





9.3





9.



实验心得：

1. Fork函数从已存在的进程中复制一个进程，该函数被调用一次，但返回两次，子进程的返回值是0，而父进程的返回值则是新子进程的PID
2. Execl函数，函数语法为int execl(const char \*path,const char \*arg,....)例如，要在程序中执行ps -ef命令，命令ps在”/bin”目录下。在这一函数中，参数一表示命令或参数逐个列举，文件需给路径，因此此函数的调用格式为 execl(“/bin/ps”,”ps”,”-ef”,NULL);
3. exit和\_exit，exit函数，缓冲区中的记录能正常输出，而调用\_exit函数时，缓冲区的内容无法输出。
4. 僵尸进程。Wait函数用于使父进程阻塞，直到一个子进程终止或者该进程接到一个指令的信号为止。Waitpid 函数有三个参数，如(waitpid(result,NULL,WNOHANG))。
5. 守护进程。守护进程最重要的特点是后台运行。1.创建子进程，终止父进程。2.在子进程中创建会话。3.改变工作目录。4.重设文件创建掩码。5.关闭文件描述符。
6. undefined reference to `pthread\_create' 的解决方法，pthread 库不是 Linux 系统默认的库，连接时需要使用静态库 libpthread.a，所以在使用pthread\_create()创建线程，以及调用 pthread\_atfork()函数建立fork处理程序时，需要链接该库。在命令后面加上 -lpthread 即可，如gcc 6-9-1.c -o 6-9-1 -lpthread。

