操作系统实验

实验一 vi编辑器的使用

实验目的：

- 理解vi的运行模式及其切换方法。
- 学会使用vi的各种操作命令进行文本文件的编辑。
- 用vi编写Linux下C程序，会用gcc编译。

- vi是UNIX中最常用的编辑器。
- 对vi编辑器的全面而清晰的理解，对于后续内容的学习至关重要。

- **实验内容与要求:**

① 创建一个文件

② 保存退出一个文件 及 不保存退出一个文件

③ 在文本中使用不同的键进行光标的移动

④ 在一个文件中加入、删除与修改文本

⑤ 设定选项以自定义编辑环境

⑥ 调用命令行编辑功能

- 使用vi编辑器编写一个C语言程序Hello World

```
#include <stdio.h>
int main(void)
{
  printf ("Hello world, Linux   programming!\n");
  return 0;
}
```

- gcc编译：

  将程序存为hello.c。

  编译：**gcc hello.c -o hello.o**

- 运行：

  ./hello.o

实验二 Linux进程控制

- 实验目的

① 了解进程与程序的区别，加深对进程概念的理解；

② 进一步认识进程并发执行的原理，理解进程并发执行的特点，区别进程并发执行与顺序执行；

③ 分析进程争用临界资源的现象，学习解决进程互斥的方法。

④ 了解fork( )系统调用的返回值，掌握用fork()创建进程的方法；

⑤ 熟悉wait、exit、lockf等系统调用。

- 实验内容与要求

① 编写一C语言程序，实现在程序运行时通过系统调用fork( )创建两个子进程，使三进程并发执行，父亲进程执行时屏幕显示“I am father”，儿子进程执行时屏幕显示“I am son”，女儿进程执行时屏幕显示“I am daughter”。

或

编写一C语言程序，实现在程序运行时通过系统调用fork( )创建一个子进程，子进程再创建一个孙子进程。使三进程并发执行，父亲进程执行时屏幕显示“I am father”，儿子进程执行时屏幕显示“I am son”，孙子进程执行时屏幕显示“I am grandson”。

② 多次连续反复运行这个程序，观察屏幕显示结果的顺序，直至出现不一样的情况为止。记下这种情况，试简单分析其原因。

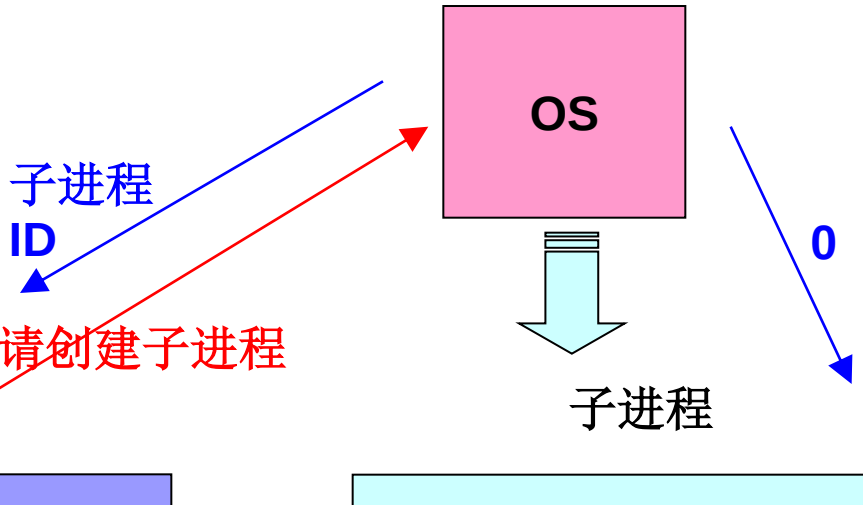
③ 修改程序，在三进程中分别使用**wait**、**exit**、**lockf**等系统调用“实现”其同步推进，多次反复运行改进后的程序，观察并记录运行结果。

- 程序参考：（仅供参考）

需要的头文件：

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

```
int main()
{
   int pid;
   if ((pid = fork())<0)
   {  printf("son fail create!\n");
       return;
   }
  else if (pid == 0)
  {   printf("I am son!\n");
      return;
  }
  else
  {  printf("I am father!\n");
  }
}
```




```
if ((pid = fork())<0)
 {  printf("son fail create!\n");
      return;
 }
else if (pid == 0)
{   printf("I am son!\n");
    return;
 }
 else
  {  printf("I am father!\n");
  }
```

```
if ((pid = fork())<0)
 {  printf("son fail create!\n");
      return;
 }
else if (pid == 0)
{   printf("I am son!\n");
    return;
 }
 else
  {  printf("I am father!\n");
  }
```

- **fork()的原理：**

① 当程序执行到下面的语句： **pid=fork();** **OS**创建子进程。

② 子进程和父进程的可执行程序是同一个程序；但它们是两个相互独立的进程！

③ 当前是父进程执行到**fork**调用即将返回（此时子进程不占有**CPU**）。

④ 父进程继续执行，**OS**对**fork**的实现，使这个系统调用在父进程中返回刚刚创建的子进程的**pid**（一个正整数），所以下面的**if**语句 **pid==0**的分支不会执行。所以输出**i am father**

⑤ 子进程在之后的某个时候得到调度，占用 **CPU**。

⑥ **OS**对**fork**的实现，使子进程中**fork**返回**0**。所以在子进程中，**pid=0**。

⑦ 子进程继续执行的过程中，**if**语句中 **pid==0**是**true**。

⑧ 所以输出**i am son**

- **wait 函数**
- **定义： pid_t wait (int * status);**
- **返回值：执行成功返回子进程的ID，否则返回-1。**
- **原理：**
  - **进程一旦调用wait，就立即阻塞自己，直到有信号来到或子进程结束。**
  - **如果有子进程退出，把该子进程彻底销毁后返回该子进程的状态信息；**
  - **如果没有子进程请求退出，wait会一直阻塞在这里，直到有一个出现为止。**
- **注意：**
  - **如果在调用wait时子进程已经结束，则wait会立即返回子进程的结束状态。**
  - **子进程的结束状态会由参数status 返回，子进程的ID也会一块返回。**
  - **如果不关心结束状态，参数status 可设成NULL**

- 范例

需要的头文件

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

```
int main()
{
   pid_t pid;
   int status,i;
   if(fork()= =0)
   {
      printf("This is the child process .pid =%d\n",getpid());
      exit(5);
   }
  else
  {   sleep(1);
      printf("This is the parent process ,wait for child...\n";
      pid = wait(&status);
      i = WEXITSTATUS(status);
      printf("child's pid =%d .exit status=%d\n",pid,i);
  }
}
```

- 执行示例：

  This is the child process .pid =1501
  This is the parent process ,wait for child...
  child’s pid =1501. exit status =5

实验三　进程控制Lockf()

- 实验目的
  - 学习进程控制机制，掌握Lockf()函数的使用和工作原理。
- 实验内容：
  - 根据参考程序，观察、记录并简单分析其运行结果。

- **lockf()函数**
  - 利用系统调用**lockf（fd，mode，size）**，对指定区域（有**size**指示）进行加锁或解锁，以实现进程的同步或互斥。
  - 其中，**fd**是文件描述字；
  - **mode**是锁定方式，**mode=1**表示加锁，**mode=0**表示解锁；
  - **size**是指定文件**fd**的指定区域，用**0**表示从当前位置到文件结尾
  - 注：有些**Linux**系统是**locking(fd，mode，size)**

- 范例

需要的头函数

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

```
int main(void)
{
    int    pid, i;
    if((pid=fork())<0)
    {      printf("child fail create\n");
           return;
    }
    else if(pid= =0)
    {       lockf(1,1,0); // 锁定标准输出设备(显示器)
             for(i=0;i<5;i++)
               printf("This is child (pid=%d) process:b\n",getpid());
           lockf(1,0,0); // 解锁标准输出设备(显示器)
           return;
    }
   else
   {  lockf(1,1,0);
     for(i=0;i<5;i++)
         printf("Parent process:a\n");
     lockf(1,0,0);
   }
```

实验四　管道通信

- 实验目的
  - 学习如何利用管道机制、消息缓冲队列、共享存储区机制进行进程间的通信，并加深对上述通信机制的理解。

- 实验内容：

① 编写一C语言程序，使其用管道来实现父子进程间通信。子进程向父进程发送字符串“**is sending a message to parent!**”；父进程则从管道中读出子进程发来的消息，并将其显示到屏幕上，然后终止。

② 运行该程序，观察、记录并简单分析其运行结果。

- **pipe()函数：**创建管道的方法

  函数原型为int pipe(int filedes[2]);

- **write()函数：**写入数据方法

  函数原型为:

  int write(int pipe_fd，char *buffer，size_t len);

- **read()函数：**读取数据的方法

  函数原型为:

  int read(int pipe_fd，char *buffer，size_t len);

- **pipe函数—— 建立管道**
  - 头文件：#include<unistd.h>
  - 定义函数：int pipe(int filedes[2]);
  - 函数说明：
    - 建立管道，并将文件描述符由数组filedes返回。
    - filedes[0]为管道里的读取端，用read从管道中读数据
    - filedes[1]则为管道的写入端，用write向管道中写数据。
    - 返回值：若成功则返回0，否则返回-1

- 基本步骤：

① 利用pipe创建管道:

```
int filedes [2];
pipe(filedes);
```

② 利用fork创建子进程

③ 子进程利用write向管道写入数据：

```
char *str="is sending a message to parent!";
 close(filedes[0]); //关闭读端
 write(filedes[1], str, strlen(str)); //写数据
```

④ 父进程利用**read**从管道读取数据：

close(filedes[1]); //关闭写端

read(filedes[0], buf, sizeof(buf)); //读数据

⑤ 父进程显示接收到的数据，及数据的长度

- 编译举例：

```
[root@localhost]# gcc pipe.c  -o pipe.o
[root@localhost]# ./pipe.o
```

- 显示结果：

```
in the parent process...
in the child process...
31 bytes of data received from child:
is sending a message to parent!
```

实验报告格式

- 分以下几个部分：

一、实验名称

二、实验目标

三、实验要求

四、实验内容 (写上你的程序)

五、实验结果 (打印程序运行的结果,贴在这)

六、实验结论 (写为什么会出现这种运行结果)

实验报告撰写注意事项

- 除“实验结果”外，其余部分必须手写
- 报告必须用标准的实验报告纸书写（如下图所示），否则实验成绩做0分处理。

中国海洋大学实验报告

____年___月___日

姓名_____系年级_____专业_____同组者_____

科目_____题 目_____

实验报告评分的基本标准

- 书写工整(字不一定好看，但要书写认真)
- 代码完整且正确
- 实验结果部分附有粘贴的执行结果
- 实验结论分析正确

课堂纪律基本要求

- 上课和下课时需要到研究生助教处签到、签退。(迟到、早退、未到都要扣分)
- 试验期间，做任何违反课堂要求的活动，比如玩游戏、看视频、上网聊天等，都要扣分。
- 课堂纪律得分最终将计入期末考试成绩。

作业一（4分）

- 进程同步
  - 甲、乙两个老师分别批改作业，批完的作业放在一张桌子上，但桌子每次只能放一本作业
  - 另有甲、乙两个同学
  - 甲同学从桌子上拿甲老师批完的作业
  - 乙同学从桌子上拿乙老师批完的作业
  - 请将甲乙老师和甲乙同学分别模拟成一个进程，并用信号量机制实现他们之间的同步。
- 要求：手写在报告纸上，实验课期间统一上交

作业二（4分）

- 用栈实现LRU算法（参考书P165页）
  - 设分配给进程5个物理块，初始物理块是空的
  - 所要访问页面的页号顺序为

    4, 7, 0, 7, 1, 0, 1, 2, 1, 2, 6
  - 利用栈实现LRU算法，在访问完页号6之后，要求显示以下内容：
    ① 被置换出去的页号是哪个
    ② 从栈顶到栈底，栈中各保存了哪些页号

- 要求：
  - 在程序中定义一个数组：**int visit[]={4, 7, 0, 7, 1, 0, 1, 2, 1, 2, 6}**，用于存储页号访问顺序。
  - 定义一个栈**int stack[5]**，用来实现**LRU**算法
  - 在**Linux**下用**C**语言实现
  - 实验课期间统一检查程序运行结果。