**北京邮电大学软件学院**

**2017-2018学年第1学期实验报告**

**课程名称： 操作系统**

**实验名称： 实验一 进程管理**

**实验完成人：**

**姓名：**\_\_王子妍\_**学号：**\_2016211980**成绩：**\_\_\_\_\_\_\_\_

**指导教师：**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**日 期： 2018 年 3 月 19 日**

1. **实验目的**

（1）理解进程的概念，明确进程和程序的区别。

（2）理解并发执行的实质。

（3）掌握进程的睡眠、同步、撤销等进程控制方法。

1. **实验内容**

2.1. 进程的创建。

① 编写一段源程序，使系统调用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”和字符“c”。试观察记录屏幕上的显示结果，并分析原 因。

② 修改已编写的程序，将每个进程输出一个字符改为每个进程输出一句话，在观察程序执行时屏幕出现的现象，并分析原因。

2.2. 进程的控制 ① 用fork()创建一个进程，再调用exec()用新的程序替换该子进程的内容。 ② 利用wait()来控制进程执行顺序。

1. **实验环境**

在VMware虚拟机下安装Ubuntu系统作为Linux实验环境。采用以Vi +GCC+GDB为开发环境的C语言实验环境。其中Vi 作为编辑器，GCC作为编译器，GDB作为调试器。

1. **实验过程描述**
2. 2.1. 进程的创建。
3. 编写一段源程序，使系统调用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”和字符“c”。试观察记录屏幕上的显示结果，并分析原因。

第一步，分析阶段：在父进程中创建子进程1，在子进程1中输出b，如果创建成功，那么pid1的值将为0，再创建进程2，同理，创建成功后pid2的值将为0。

第二步，用代码实现分析部分。

首先先寻找fork需要的头文件，头文件如下：

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

然后实现代码，代码的实现在附录里。

第三步，debug和运行。

第一次使用在linux上运行c

方法如下：

1. Ctrl+Alt+T调出命令行界面
2. 在命令行中输入命令gcc project1.cpp,如果没有错误就是编译成功了，有错误就debug一下。
3. gcc project1.cpp编译完成后，会出现一个a.out. 这个是编译project1.cpp文件后生成的一个可执行文件。
4. 通过命令./a.out运行该文件

第四步，查看结果并且进行分析：



输出为：b,a,c

创建进程1就输出了’b’,进程2的创建需要时间，输出也需要时间，所以进程2与主程序的输出偶尔也有随机性。

在这里分别以文字和流程图的方式画出你的实验的几个阶段。

1. 修改已编写的程序，将每个进程输出一个字符改为每个进程输出一句话，在观察程序执行时屏幕出现的现象，并分析原因。

第一步，分析阶段：在父进程中创建子进程1，在子进程1中输出child1 has been called，如果创建成功，那么pid1的值将为0，再创建进程2，同理，创建成功后pid2的值将为0，在子进程2中输出child2 has been called。

第二步，用代码实现分析部分。

首先先寻找fork需要的头文件，头文件如下：

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

然后实现代码，代码的实现在附录里。

第三步，debug和运行。

方法如下：

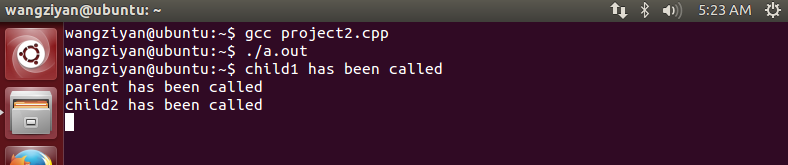
1.Ctrl+Alt+T调出命令行界面

2.在命令行中输入命令gcc project1.cpp,如果没有错误就是编译成功了，有错误就debug一下。

3.gcc project1.cpp编译完成后，会出现一个a.out. 这个是编译project1.cpp文件后生成的一个可执行文件。

4.通过命令./a.out运行该文件

第四步，查看结果并且进行分析：



输出为：child1 has been called

Parent has been called

Child2 has been called

分析：在父进程中创建子进程1，在子进程1中输出child1 has been called，在子进程2中输出child2 has been called，在父进程中输出parent has been called. 进程2的创建需要时间，输出也需要时间，所以进程2与主程序的输出偶尔也有随机性。

1. 2.2. 进程的控制 ① 用fork()创建一个进程，再调用exec()用新的程序替换该子进程的内容。 ② 利用wait()来控制进程执行顺序。

第一步，分析阶段：在子进程中调用 execlp("/bin/ls","ls",NULL);让子进程展示目录，在父进程中使用wait(NULL)来等待子进程结束，在子进程结束以后输出Child Complete.

第二步，用代码实现分析部分。和之前的实验类似，也是先寻找一些需要的头文件，新使用的wait()需要添加新的头文件#include<sys/wait.h>。其余的头文件和之前的相同。#include<stdio.h> #include<stdlib.h> #include<unistd.h> #include<sys/types.h>。然后实现代码，代码在附录中。

第三步，debug和运行。

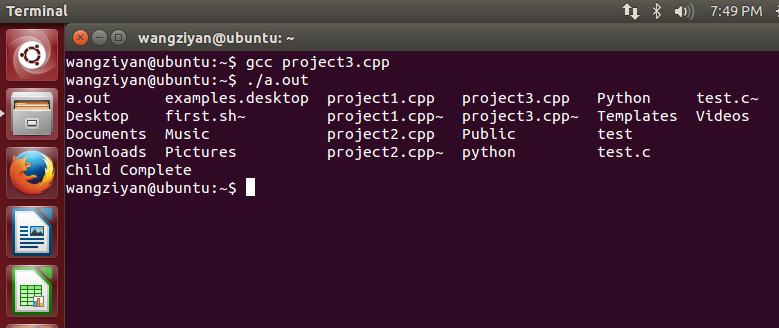
1.Ctrl+Alt+T调出命令行界面

2.在命令行中输入命令gcc project1.cpp,如果没有错误就是编译成功了，有错误就debug一下。

3.gcc project1.cpp编译完成后，会出现一个a.out. 这个是编译project1.cpp文件后生成的一个可执行文件。

4.通过命令./a.out运行该文件

第四步，查看结果并且进行分析：



先输出了所有的目录，然后再输出：Child Complete

分析：因为父进程在等待子进程结束wait(NULL)，所以父进程在子进程执行以后才开始执行，显示子进程输出所有的目录，在这以后才进行父进程的输出”Child Complete”。

1. **实验结果**

所有实验均已认真完成，思路与分析在实验过程描述中，代码添加在了附件中。

1. **附件**

**6.1 附件1：源代码**

编写一段源程序，使系统调用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”和字符“c”。

代码问题：

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

int main()

{

pid\_t pid1,pid2;

pid1=fork();

if(pid1<0)

{

fprintf(stderr,"Fork Failed");

exit(-1);

}

else if(pid1==0)

{

//child process

printf("b\n");

pid2=fork();

if(pid2<0)

{

fprintf(stderr,"Fork Failed");

exit(-1);

}

if(pid2==0)

{

printf("c\n");

}

else

{

printf("a\n");

}

}

return 0;

}

② 修改已编写的程序，将每个进程输出一个字符改为每个进程输出一句话

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

int main()

{

pid\_t pid1,pid2;

pid1=fork();

if(pid1<0)

{

fprintf(stderr,"Fork Failed");

exit(-1);

}

else if(pid1==0)

{

//child process

printf("child1 has been called\n");

pid2=fork();

if(pid2<0)

{

fprintf(stderr,"Fork Failed");

exit(-1);

}

if(pid2==0)

{

printf("child2 has been called\n");

}

else

{

printf("parent has been called\n");

}

}

return 0;

}

2.2. 进程的控制 ① 用fork()创建一个进程，再调用exec()用新的程序替换该子进程的内容。 ② 利用wait()来控制进程执行顺序。

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/wait.h>

int main()

{

pid\_t pid;

pid=fork();

if(pid<0)

{

fprintf(stderr,"Fork Failed");

exit(-1);

}

else if(pid==0)

{

//child process

execlp("/bin/ls","ls",NULL);

}

else

{

//parent process

wait(NULL);

printf("Child Complete\n");

exit(0);

}

return 0;

}

**6.2 附件2：XXX**