深 圳 大 学 实 验 报 告

课程名称：操作系统

实验项目名称：并发程序设计

学院：计算机与软件学院

专业：软件工程

指导教师：张 滇

报告人：郑彦薇 学号：2020151022 班级： 软件工程 01 班

实验时间： 2023 年 02 月 23 日

实验报告提交时间： 2023/3/14

教务处制

一、实验目的与要求：

通过进程的创建、撤销和运行加深对进程概念和进程并发执行的理解，明确进程与程序之间的区别。

二、方法、步骤：

1. 知识内容：

进程并发执行是指两个及两个以上的进程在同一时间段内同时执行，在不同时间片刻交替执行。在操作系统中，指一个时间段中有几个程序都处于已启动运行到运行完毕之间，且这几个程序都是在同一处理机上运行的。

2. 程序设计思路和方法：

该实验主要是通过编程实现子进程的创建，并执行相关语句分析子进程并发执行的过程。根据实验内容，可以分解出以下几个重要的步骤：

2.1. 如何根据例程的学习结果，创建两个子进程而不产生孙进程：

创建子进程时，可以对 pid 进行判断，设置条件语句判断当前进程为父进程时，才调用 fork 函数创建进程，确保创建的两个进程都为子进程，即这两个进程的父进程为同一个。

2.2. 创建多个子进程时，如何设计观察多个子进程的交替执行：

观察多个子进程的交替执行，可以通过 sleep 函数使当前子进程暂时休眠，而执行并发进程中的其他进程，即首先输出另一个进程的相关信息，休眠结束后再输出该进程的相关信息，实现进程相关信息的交替输出。

三、实验过程及内容：

1. 运行例程，分析例程中关键代码的功能，给出运行结果并对运行结果进行分析说明。

例程 1：测试开发环境正常

打开虚拟机终端 terminal，使用 vim 命令创建文件 helloworld.c

```
zhengyanwwei_2020151022@ubuntu:~/Desktop$ cd OS_test1
zhengyanwwei_2020151022@ubuntu:~/Desktop/OS_test1$ vim helloworld.c
```

文件 helloworld.c 内容如下

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

使用 gcc 命令编译该文件，并执行该文件。得到运行结果如下图所示，测试开发环境正常

```
zhengyanwwei_2020151022@ubuntu:~/Desktop/OS_test1$ gcc helloworld.c -o test
zhengyanwwei_2020151022@ubuntu:~/Desktop/OS_test1$ ./test
Hello World!!.
```

运行结果分析

在 main 函数中使用 printf 函数输出指定信息”Hello World!!.”

例程 2：单个子进程的创建

同样使用 vim 命令创建文件 ep_2.c，输入例程 2 所给的代码。

```
zhengyanwwei_2020151022@ubuntu:~/Desktop/OS_test1$ vim ep_2.c
```

对例程 2 文件代码中的关键代码进行解释：

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

int tprintf(const char*fmt,...);

int main(void)
{
        int i=0,j=0;
        pid_t pid;
        printf("Hello from Parent Process, PID is %d.\n", getpid());
        //getpid()函数，获取目前进程(Parent Process)的进程识别码
        pid = fork();
        //fork()函数，产生一个子进程，复制父进程的数据与堆栈空间，并继承父进程的用户代码。
        //但子进程有自己的进程编号，pid存放在不同的内存位置，交替执行。

        if(pid==0)//child process
        {
                sleep(1);
        //休眠函数，放弃当前线程执行的时间片，把自身放到等待队列之中。
        //等待相应时间后再进入就绪队列，直到获得时间片运行。
                for(i=0;i<3;i++)
                {
                        printf("Hello from Child Process %d.%d times\n", getpid(), i+1);
                        sleep(1);
                }

        }
        else if(pid!=-1) //Parent process
        {
                tprintf("Parent forked one child process--%d.\n", pid);
                tprintf("Parent is waiting for child to exit.\n");
                waitpid(pid,NULL,0);
        //waitpid()函数，等待子进程中断或结束。停止目前进程的执行，直到有信号来到或子进程结束。
        //对于fork()产生的子进程，一定要使用waitpid进行回收，否则就会产生僵尸进程。
                tprintf("Child Process has exited.\n");
                tprintf("Parent had exited.\n");
        }
        else tprintf("Everything was done without error.\n");

        return 0;

}

int tprintf(const char*fmt,...)
{
        va_list args;
        struct tm *tstruct;
        time_t tsec;
        tsec = time(NULL);
        tstruct = localtime(&tsec);
        printf("%02d:%02d:%02d: %5d|", tstruct->tm_hour,tstruct->tm_min,tstruct->tm_sec,getpid());
        va_start(args,fmt);
        return vprintf(fmt,args);
}
```

使用 gcc 命令对文件进行编译并执行。得到运行结果如下：

```
zhengyanwwei_2020151022@ubuntu:~/Desktop/OS_test1$ vim ep_2.c
zhengyanwwei_2020151022@ubuntu:~/Desktop/OS_test1$ gcc ep_2.c -o test
zhengyanwwei_2020151022@ubuntu:~/Desktop/OS_test1$ ./test
Hello from Parent Process, PID is 8070.
19:14:49:  8070|Parent forked one chile process--8071.
19:14:49:  8070|Parent is waiting for child to exit.
Hello from Child Process 8071.1 times
Hello from Child Process 8071.2 times
Hello from Child Process 8071.3 times
19:14:53:  8070|Child Process has exited.
19:14:53:  8070|Parent had exited.
```

运行结果分析：

在例程 2 中，首先定义一个 pid 值，并令该进程识别码即 8070 对应的进程为父进程，然后对该信息进行输出。

接着使用 fork()函数创建子进程，识别码 pid=8071。对 pid 值进行判断，判断出当前进程

为父进程，进入 else if 分支，执行该条件下的语句。在该分支中，父进程按顺序执行两句 tprintf 输出语句之后，调用 waitpid 函数对子进程进行回收，此时同样对 pid 值进行判断，得到当前进程为 fork 创建的子进程，进入 if 分支，执行该条件下的语句，调用 getpid 方法对进程识别码进行输出。

子进程被回收后，继续执行 else if 分支下 waitpid 之后的两句 tprintf 命令，对相关信息进行输出。

对于 tprintf 方法，调用该函数进行输出时，首先输出执行该语句的时间以及当前进程的识别码，再输出其余内容。

例程 3：在子进程中调用外部命令

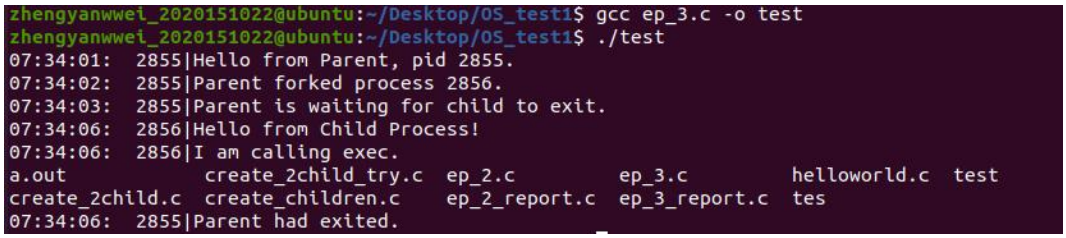
使用 vim 命令创建文件 ep_3.c，输入例程 3 所给代码。

```
zhengyanwwei_2020151022@ubuntu:~/Desktop/OS_test1$ vim ep_3.c
```

对例程 3 文件代码中的关键代码进行解释：

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

int tprintf(const char*fmt, ...);

int main(void)
{
        pid_t pid;
        pid = fork();

        if(pid==0) //Child Process
        {
                sleep(5);
                tprintf("Hello from Child Process!\n");
                tprintf("I am calling exec.\n");
                execl("/bin/ls","-a",NULL);
                //execl()函数，执行文件路径/bin/ps下的文件，-a为执行该文件时所传的参数，最后一个参数为NULL。
                //这里调用ls命令，显示例程3所在文件夹中的所有文件
                tprintf("You should never see this because the child is already gone.\n");
        }
        else if(pid != -1) //Parent Process
        {
                tprintf("Hello from Parent, pid %d.\n", getpid());
                sleep(1);
                tprintf("Parent forked process %d.\n", pid);
                sleep(1);
                tprintf("Parent is waiting for child to exit.\n");
                waitpid(pid, NULL, 0);
                tprintf("Parent had exited.\n");
        }
        else tprintf("Everything was done without error.\n");
        return 0;

}

int tprintf(const char*fmt,...)
{
        va_list args;
        struct tm *tstruct;
        time_t tsec;
        tsec = time(NULL);
        tstruct = localtime(&tsec);
        printf("%02d:%02d:%02d: %5d|", tstruct->tm_hour, tstruct->tm_min, tstruct->tm_sec, getpid());
        va_start(args,fmt);
        return vprintf(fmt,args);
}
```

使用 gcc 命令对文件进行编译并执行。得到运行结果如下：



运行结果分析：

同理于例程 2 的执行过程，程序首先定义一个 pid 值，令该识别码所对应的进程为父进程，然后使用 fork 函数创建一个子进程。接着设置条件语句对当前执行进程为子进程还是父进程进行判断。

根据输出可以知道 fork 之后执行进程仍为父进程，于是输出相应信息。之后父进程调用 waitpid 函数对子进程进行回收，当前执行进程则为子进程。于是执行子进程下的命令。首先输出子进程相关信息，在子进程分支中，调用 execl 函数执行文件“/bin/ls”，对例程 3 所在文件夹中的文件进行列举，得到上图所示执行结果。

子进程回收完成后，回到父进程分支执行剩下的 tpritf 输出语句输出相应信息。

2. 模仿例程，编写一段程序实现以下功能：

a) 使用系统调用 fork()创建两个子进程

b) 各个子进程显示和输出一些提示信息和自己的进程标识符

c) 父进程显示自己的进程 ID 和一些提示信息，然后调用 waitpid()等待多个子进程结束，并在子进程结束后输出提示信息表示进程已结束。

2.1. 编程思路

题目要求使用 fork 创建两个子进程，为防止两次 fork 产生孙进程，在编程中设置条件语句判断得当前进程为父进程，再在该分支下调用 fork()创建第二个子进程。

2.2. 代码及解释如下

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

int tprintf(const char*fmt,...);

int main(void)
{
        pid_t pid;
        tprintf("Hello from Parent Process, PID is %d.\n", getpid());

        pid = fork(); //create first child

        if(pid==0) //first child
        {
                tprintf("Hello from First Child Process,PID is %d.\n",getpid());
        }
        else if(pid!=-1) //parent Process
        {
                tprintf("Here is Parent Process, PID is %d, now I am going to do...\n",getpid());
                tprintf("Parent forked the first child process--%d.\n", pid);
                tprintf("Parent is waiting for the First Child to exit.\n");
                waitpid(pid, NULL,0);
                //recycle the first child
                tprintf("The First Child has exited.\n");

                pid_t pid1;
                pid1 = fork();//create second child

                if(pid1==0) //second child
                {
                        tprintf("Hello from Second Child Process, PID is %d.\n", getpid());
                }
                else if(pid1!=-1) //Parent Process
                {
                        tprintf("Here is Parent Process, PID is %d, now I am going to do...\n",getpid());
                        tprintf("Parent forked the second child process--%d.\n", pid1);

                        tprintf("Parent is waiting for the Second Child to exit.\n");
                        waitpid(pid1, NULL, 0);
                        //recycle the second child
                        tprintf("The Second Child Process has exited.\n");
                        tprintf("Parent had exited.\n");
                }
                else tprintf("Everything was done without error.\n");
        }
        else tprintf("Everything was done without error.\n");
        return 0;
}

int tprintf(const char*fmt,...)
{
        va_list args;
        struct tm *tstruct;
        time_t tsec;
        tsec = time(NULL);
        tstruct = localtime(&tsec);
        printf("%02d:%02d:%02d: %5d|", tstruct->tm_hour, tstruct->tm_min, tstruct->tm_sec, getpid());
        va_start(args, fmt);
        return vprintf(fmt, args);
}
```

2.3. 运行结果

```
zhengyanwwei_2020151022@ubuntu:~/Desktop/OS_test1$ gcc create_2child.c -o test
zhengyanwwei_2020151022@ubuntu:~/Desktop/OS_test1$ ./test
03:13:47:  2277|Hello from Parent Process, PID is 2277.
03:13:47:  2277|Here is Parent Process, PID is 2277, now I am going to do...
03:13:47:  2277|Parent forked the first child process--2278.
03:13:47:  2277|Parent is waiting for the First Child to exit.
03:13:47:  2278|Hello from First Child Process,PID is 2278.
03:13:47:  2277|The First Child has exited.
03:13:47:  2277|Here is Parent Process, PID is 2277, now I am going to do...
03:13:47:  2277|Parent forked the second child process--2279.
03:13:47:  2277|Parent is waiting for the Second Child to exit.
03:13:47:  2279|Hello from Second Child Process, PID is 2279.
03:13:47:  2277|The Second Child Process has exited.
03:13:47:  2277|Parent had exited.
```

可以看到程序首先创建了父进程 2277，然后创建了第一个子进程 2278 并在输出相关语句后进行回收；回收完成后创建第二个子进程 2279，同样输出相关语句后进行回收。

3. 创建多个进程并发运行，各个子进程输出自己的进程标识符和相关信息，对运行结果进行分析。

3.1. 编程思路

设置循环进行多个子进程的创建，在实验中，每 fork 一次都对当前进程进行判断，创建出平行的子进程（即所有子进程的父进程为同一个）。再次设置循环并发执行进程，为观察进程的交替执行，对于每个进程的执行过程，都通过 sleep 函数使当前进程暂时休眠，从而实现进程交替执行的展示。

3.2. 代码及解释如下

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

int tprintf(const char*fmt,...);

int main(void)
{
        int amount = 5;
        pid_t pid_father;
        printf("Hello from Parent Process, PID is %d\n", getpid());
        pid_t pid[5];
        //Create 5 children
        for(int i=0;i<amount;i++){
                pid[i] = fork();
                if(pid[i] == 0){ //child, I dont want to create grandson
                        break;
                }else if(pid[i] != -1){ //father, create the next child
                        tprintf("Parent foked child Process--%d\n", pid[i]);
                        continue;
                }
        }

        for(int i=0;i<amount;i++){
                if(pid[i] == 0){
                        sleep(1);
                        for(int j=0;j<3;j++){
                                printf("Hello from child Process--%d\n", getpid());
                                sleep(1);
                        }
                }
        }

        while(amount--){
                wait(NULL);
                //recycle all child process
        }

        return 0;

}

int tprintf(const char*fmt,...){
        va_list args;
        struct tm * tstruct;
        time_t tsec;
        tsec = time(NULL);
        tstruct = localtime(&tsec);
        printf("%02d:%02d:%02d: %5d|", tstruct->tm_hour, tstruct->tm_min, tstruct->tm_sec, getpid());
        va_start(args,fmt);
        return vprintf(fmt, args);
}
```

3.3. 运行结果

```
zhengyanwwei_2020151022@ubuntu:~/Desktop/OS_test1$ gcc create_children.c -o test
zhengyanwwei_2020151022@ubuntu:~/Desktop/OS_test1$ ./test
Hello from Parent Process, PID is 2372
03:21:03:  2372|Parent foked child Process--2373
03:21:03:  2372|Parent foked child Process--2374
03:21:03:  2372|Parent foked child Process--2375
03:21:03:  2372|Parent foked child Process--2376
03:21:03:  2372|Parent foked child Process--2377
Hello from child Process--2373
Hello from child Process--2374
Hello from child Process--2375
Hello from child Process--2376
Hello from child Process--2377
Hello from child Process--2375
Hello from child Process--2376
Hello from child Process--2374
Hello from child Process--2373
Hello from child Process--2377
Hello from child Process--2376
Hello from child Process--2374
Hello from child Process--2373
Hello from child Process--2375
Hello from child Process--2377
Hello from child Process--2373
Hello from child Process--2373
Hello from child Process--2373
```

通过上图所示运行结果的输出语句，可以看到父进程 2372 创建了 5 个平行的子进程。编程中设置每个子进程输出 3 行相关语句，在运行结果中可以看到一个子进程的相关语句并非连续输出，即 5 个子进程的相关信息交替输出。

四、实验结论：

1. 子进程的创建可以通过调用 fork()函数实现，对于程序中某个进程创建的子进程，一定要对其进行回收，避免出现僵尸进程。
2. 程序执行时相关信息的输出顺序与当前是哪个进程有关，于是在学习过程中可以通过输出进程的识别码等相关信息对进程的执行顺序进行观察和分析。

五、实验体会：

通过本次实验，学习和掌握了如何在 Linux 系统中创建子进程，以及如何通过编程观察多个子进程的并发交替执行过程，对进程并发执行的概念有了更深的认识。

指导教师批阅意见： 成绩评定： 指导教师签字： 年　　月　　日
备注：