华北水利水电大学

North China University of Water Resources and Electric Power

《操作系统》实验报告

实验二
Linux 进程线程管理

院　　系　信息工程学院

专　　业　人工智能

学　　号　202018526

姓　　名　高树林

指 导 教 师　杨学颖

完成时间：2022-12-23

实验目的

1. 熟悉与掌握获取进程常见属性；
2. 熟悉与掌握进程的创建；
3. 熟悉与掌握进程的终止；
4. 熟悉与掌握线程的创建；
5. 熟悉与掌握线程的挂起；
6. 熟悉与掌握现成的终止；
7. 熟悉与掌握使用多线程。

实验要求

1. 学会使用 C 语言在 Linux 系统中获取进程的 pid 以及父进程的 pid。
2. 学会使用 C 语言在 Linux 系统中使用 fork 系统调用创建一个新的进程。
3. 学会使用 C 语言在 Linux 系统中使用 vfork 系统调用创建一个新的进程。
4. 学习终止进程的常见方法。
5. 学会使用 C 语言在 Linux 系统中使用 pthread_create 库函数创建一个新的线程。
6. 学会使用 C 语言在 Linux 系统中使用 pthread_join 库函数挂起当前线程，并等待指定的线程。
7. 学会使用 C 语言在 Linux 系统中终止一个线程。
8. 编程实现下列功能： 1.写一个实现如下功能的应用程序 01.c ：在一个进程中循环输出 100000 个字符串 0 ，在另一个进程中循环输出 100000 个字符串 1 。 2.用版本 0 内核启动 bochs 虚拟机，在该虚拟机中编译运行该程序，画面如下图所示。 3.修改 bochs 虚拟机的/etc/rc 文件，使该虚拟机启动时自动运行该程序。

实验步骤

步骤 1：获取进程的 pid 以及父进程的 pid 的核心代码为：

```
struct procIDInfo getProcInfo()
{
  struct procIDInfo ret;    //存放进程 ID 信息，并返回
    ret.pid=getpid();
    ret.ppid=getppid();
```

```
    return ret;
}
```

在上述代码中，每个进程都由一个唯一的标识符来表示，即进程 ID。通过 getpid 函数来获得获取进程本身的进程 ID，getppid 用于获取父进程的进程 ID。

步骤 2：使用 fork 系统调用创建一个新的进程的核心代码为：

```
void createProcess()
{

  pid_t pid=fork();
  if (pid==-1)  printf("创建进程失败！");
  else
  {
    if(pid==0) printf("Children");
    else printf("Parent");
  }
}
```

fork 函数调用将执行两次返回，它将从父进程和子进程中分别返回。从父进程返回时的返回值为子进程的 PID，,而从子进程返回时的返回值为 0，并且返回都将执行 fork 之后的语句。子进程和父进程都不是固定的执行顺序，由 fork 函数创建的子进程执行顺序是由操作系统调度器来选择执行的。

步骤 3：使用 vfork 系统调用创建一个新的进程的核心代码为：

```
void createProcess()
{
  pid_t pid=vfork();
  if(pid==-1) printf("创建进程失败！\n");
  else
  {
    if(pid==0) printf("Children\n");
    else printf("Parent\n");
  }
  exit(0);
}
```

vfork 创建进程与 fork 创建的进程主要是 vfork 创建的子进程与父进程共享所有的地址空间，而 fork 创建的子进程是采用 COW 技术为子进程创建地址空间。vfork 会使得父进程被挂起，直到子进程正确退出后父进程才会被继续执行，而 fork 创建的子进程与父进程的执行顺序是由操作系统调度来决定。

步骤 4：使用 pthread_create 库函数创建一个新的线程的核心代码为：

```
void createProcess()
```

```
{
  pid_t pid=vfork();
  if(pid==-1) printf("创建进程失败！\n");
  else
  {
    if(pid==0) printf("Children\n");
    else printf("Parent\n");
  }
  exit(0);
}
```

在上述式的代码中，先用 vfork 创建一个线程，之后判断它的返回值是否为-1，如果返回值为-1 的话，说明这个进程未被创建，或者该进程创建失败。如果进程创建返回值为 0，说明该进成为子进程，否则为父进程。

步骤 5：实现 8 功能的核心代码为：

```
  pid=fork();
  if(pid == 0)
  {
    for(i=0; i<100000; i++)
      {
        printf(" 0");
      }
  }
  pid1=fork();
    if(pid1 == 0)
  {
    for(i=0; i<100000; i++)
      {
        printf("1 ");
      }
  }
}
```

在代码中，定义了两个进程，分别是 pid 和 pid1。在 pai 进程中，运行的是输出 100000 个 0 的操作，在 pid1 进程中，运行的是输出 100000 个 1 的操作。这个程序写好放在 C 文件里面，然后上传文件并将该文件放入目录 linux-0.11-lab/b/下，然后用版本 0 内核启动虚拟机，并在虚拟机中用 mcopy 命令将该文件拷入虚拟机硬盘。可以使用命令 uemacs 编辑 rc 文件，再次启动虚拟机查看效果。

实验结果

结果 1：打印父进程 ID 和自身进程 ID 的结果如下图 1 所示。

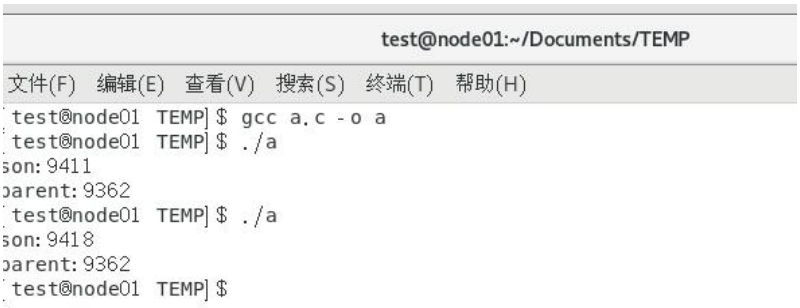

图 1 打印父进程 ID 和自身进程 ID

结果 2：用 fork 创建进程的结果图如下图 2 所示。

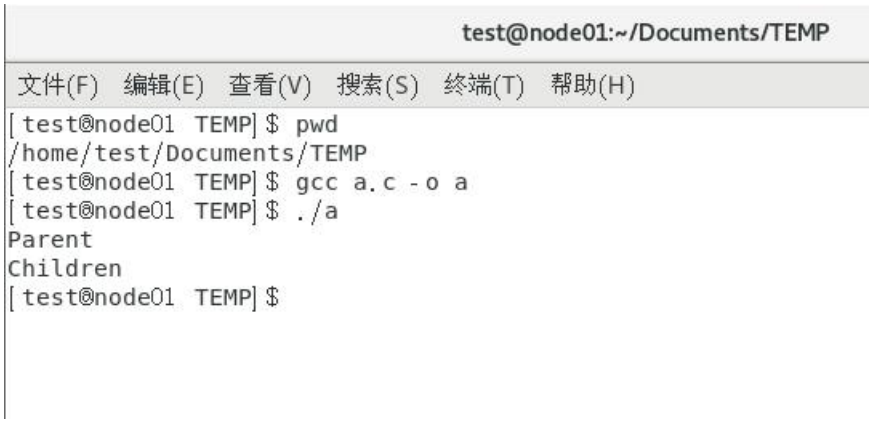

图 2 fork 创建进程并返回

结果 3：用 vfork 创建进程的结果图如下图 2 所示。

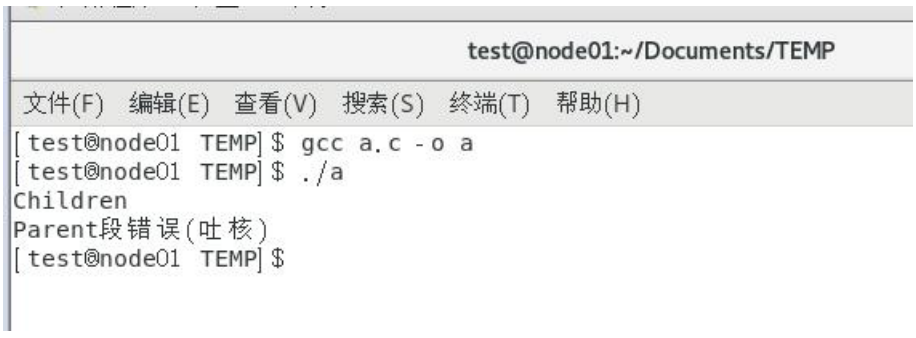

图 3 vfork 创建进程并返回

结果 4：打印 01 的结果如下图 4 所示。

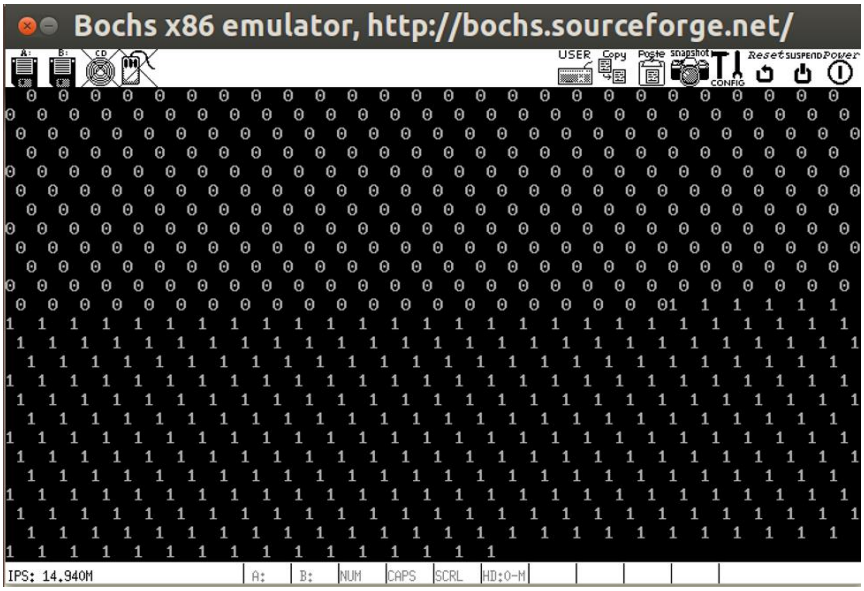

图 4　成功打印 01 的结果

实验总结

通过本次实验，我对线程和进程的理解更加深刻了。因为我本人本身在 python 程序上研究颇深，但是 python 的弊端就是完全屏蔽掉底层逻辑，就好像是你知道得这个干，但不知道为什么得这么干，这是怎么干的一样，在接触这个实验之前，我始终是半瓢水晃荡。通过对本次实验的深究和溯源，我知道其实 linux 环境下的多线程、多进程任务也好还是 python 环境下的多线程、多进程任务也好，在操作系统的内核中，他们运行的机制是一模一样的，即：内核空间含有进程表，进程可由操作系统通过进/线程控制块来进行管理，进/线程控制块存放包括寄存器、程序计数器、状态字、栈指针、优先级、进程 ID、信号、创立时间、所耗 CPU 时间等等。进/线程表中包括进程基本信息指针，也包括进/线程家族树指针，子进程、父进程、孙子进程、祖父进程，及其其他需要的信息指针。实际情况是不同的操作系统维护的进程资料不尽相同，不同的商业系统的内核教程列有更为详细的信息，不过基本的所需信息基本包含。

操作系统设置进程管理的主要目的是让各个程序均有机会执行，为公平；任何程序不能无休止的阻挠其他程序的正常执行，为非阻塞；程序对于资源的利用需要程序设定合理的优先级。

在对操作系统进程管理这个章节进行了解学习之后，做应用程序的编程，会觉得就像是在做操作系统所实现的功能的排列组合。实现应用在编码初期是最大的成就感来源，然而其实我们都是站在巨人的肩膀之上。更为可贵的是，操作系

统的学习过程中，学习和体会前人在解决问题的思路， 再具体到细节时，会发现数据结构以及算法的实现，对这两者的学习和认知也会很有帮助。可以说，操作系统的学习，不同的层次，由浅及深，都会有不同层次的收获。

01.c 代码

```c
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main()
{
  int i;
  pid_t pid;
  pid_t pid1;
  pid=fork();
  if(pid == 0)
  {
    for(i=0; i<100000; i++)
      {
        printf(" 0");
      }
  }
  pid1=fork();
    if(pid1 == 0)
  {
    for(i=0; i<100000; i++)
      {
        printf("1 ");
      }
  }
}
```