山东大学 软件 学院

**操作系统** 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202100300157 | 姓名：孙荣骏 | | 班级：21.3 |
| 实验编号：实验1 | | | |
| 实验题目：进程控制实验 | | | |
| 实验学时：2 | | 实验日期：2022.4.14 | |
| 实验目的：  加深对于进程并发执行概念的理解。实践并发进程的创建和控制方法。观察和体验进程的动态特性。进一步理解进程生命期期间创建、变换、撤销状态变换的过  程。掌握进程控制的方法，了解父子进程间的控制和协作关系。练习Linux 系统中  进程创建与控制有关的系统调用的编程和调试技术。 | | | |
| 硬件环境：  宿主机  机型：联想拯救者r7000p2021  CPU：AMD R7 5800H  内存：16G  虚拟机  RAM：4GB | | | |
| 软件环境：  虚拟机：Ubuntu 16.04  宿主机：win10 | | | |
| 实验步骤与内容：  **示例实验：**        上面是示例的运行，让我初步了解了怎么在linux上运行c程序。在过程中有个坑，指导书上的文件名一会是pctrl一会是pctl，需要改成一样的才可以运行，否则在输入gmake编译时会报错。  情况如下：      **独立实验：**          主要实验代码及注释如下：  /\*  \* Filename: os1.c  \* Copyright: 2023 by srj  \* Function: 父子协作进程,父进程创建一个子进程并控制它每隔 3 秒显示一次当前目录中的文件名列表。  \*/  #include "os1.h"  int main(int argc, char \*argv[])  {  int i;  int pid; // 存放子进程号  int status; // 存放子进程返回状态  int count; // 记录执行次数  count = 1;  char \*args[] = {"/bin/ls", "-l", NULL}; // 子进程需要执行的命令  signal(SIGCONT, (sighandler\_t)sigcat);    pid = fork(); // 建立子进程  if (pid < 0)  {  // 建立子进程失败  printf("子进程创建失败！\n");  exit(EXIT\_FAILURE);  }  else if (pid == 0)  {  while(1)  {  // 子进程执行代码段  printf("我是子进程%d，我的父进程是%d\n", getpid(), getppid());  printf("开始执行任务：显示当前目录中文件名列表\n");  pid = fork(); // 用新创建的子进程实现exec函数，否则当前子进程中的内容会被覆盖  if (pid == 0)  status = execve(args[0], args, NULL); // 装入并执行程序  else  {  waitpid(pid, &status, 0); // 等待新创建的子进程结束  kill(getppid(), SIGCONT); // 通知父进程继续执行  pause(); // 执行父进程，子进程等待父进程的开始信号  }  }  }  else  {    // 父进程执行代码段  printf("开始第%d次执行\n", count);  printf("我是父进程%d，接下来交给子进程%d\n", getpid(), pid);  pause(); // 执行子进程，等待子进程任务结束信号  printf("第%d次执行结束，休眠3秒\n", count++);  sleep(3);  printf("\n");  while (1)  {  printf("开始第%d次执行\n", count);  printf("我是父进程%d，接下来交给子进程%d\n", getpid(), pid);  kill(pid, SIGCONT); // 向子进程发送开始信号  pause(); // 等待子进程任务结束信号  printf("第%d次执行结束，休眠3秒\n", count++);  sleep(3);  printf("\n");  }  }    return EXIT\_SUCCESS;  }  #include <sys/types.h>  #include <wait.h>  #include <unistd.h>  #include <signal.h>  #include <stdio.h>  #include <stdlib.h>  // 进程自定义的时间信号  typedef void (\*sighandler\_t)(int);  void sigcat()  {  printf("父进程%d收到任务结束信号\n", getpid());  }  # srj  os1: os1.o  gcc os1.o -o os1  exp1.o: os1.c os1.h  gcc -g -c os1.c  .PHONY: clean  clean:  rm os1 \*.o | | | |
| 结论分析与体会：  在本次试验中，我了解到了进程在创建新进程时的情况，了解了父进程将会等待子进程直到子进程执行完；还知道了新进程的去向，可能子进程是父进程的复制品，还有可能子进程装入了另一个新程序。  学习中我还学习到了如何在linux虚拟机上与宿主机共享文件夹，以此把实验代码传到宿主机上。  通过这次实验，我收获了很多，对于fork()的运用和父子进程的关系的认识更加清晰。 | | | |