**操作系统实验报告**

班级： 22计科3班 姓名：张雅瑞 学号：2022334323029

**实验一：熟悉Linux命令及进程管理**

1. 实验目的

熟悉Linux命令及进程管理

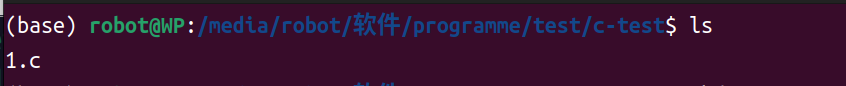
1. 实验内容和步骤

2.1 进入和退出系统

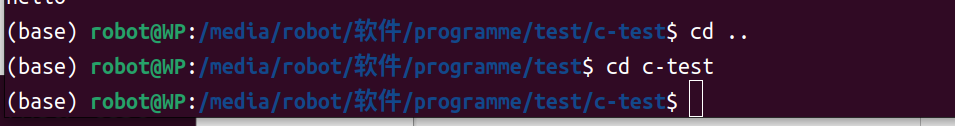
本次实验基于ubtunu双系统，未采用虚拟机，进入和退出系统在登录时便已经实现，此处省略。

2.2 文件与目录操作

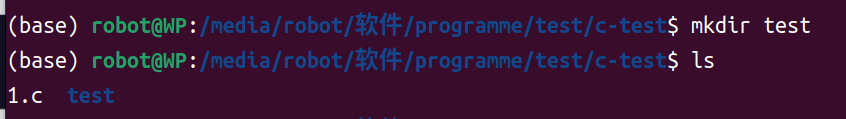
显示文件目录命令ls



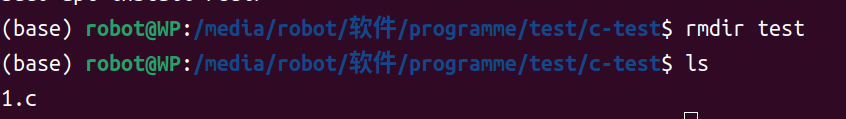
改变当前目录命令cd



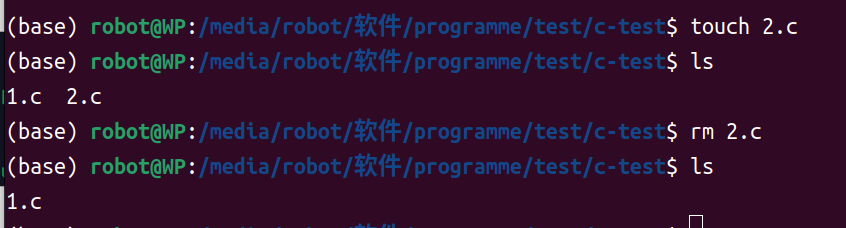
建立子目录mkdir



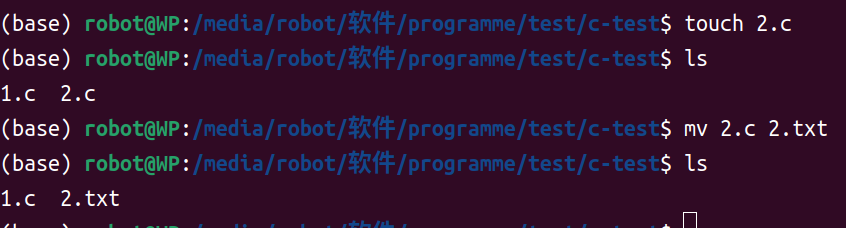
删除子目录命令rmdir



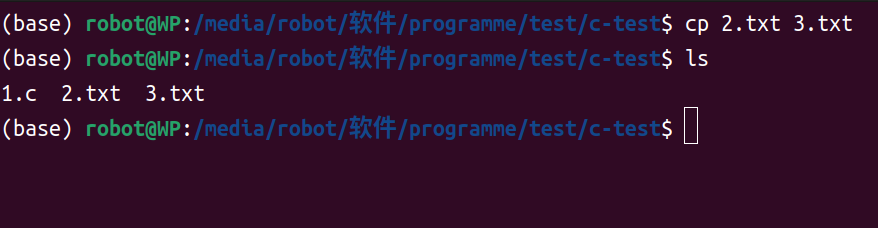
删除文件命令rm



文件改名命令mv



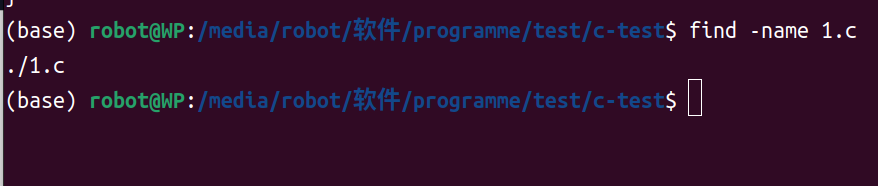
文件复制命令cp



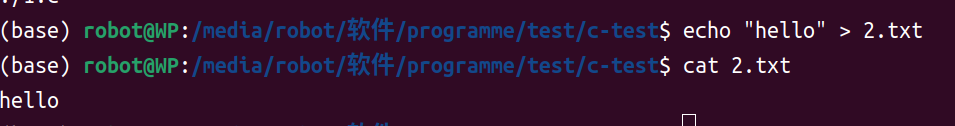
显示文件的内容more或者less



查找文件find



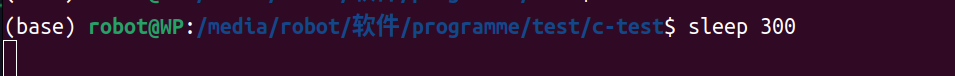
重定向与管道



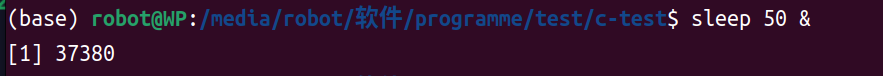
2.3进程管理及作业控制

启动进程

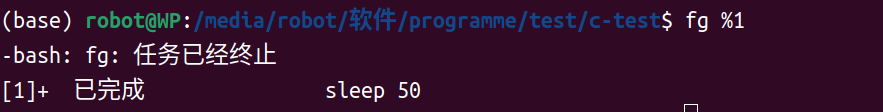
前台启动



后台启动

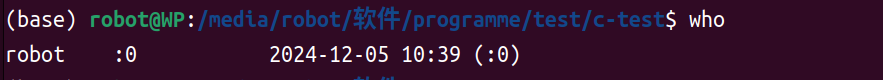


恢复到前台运行

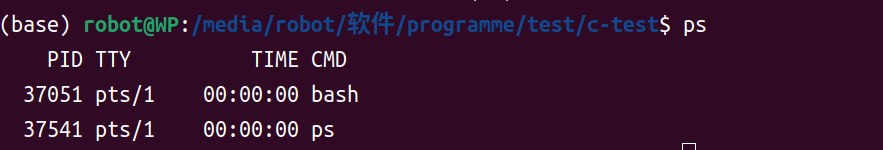


2.4 进程查看

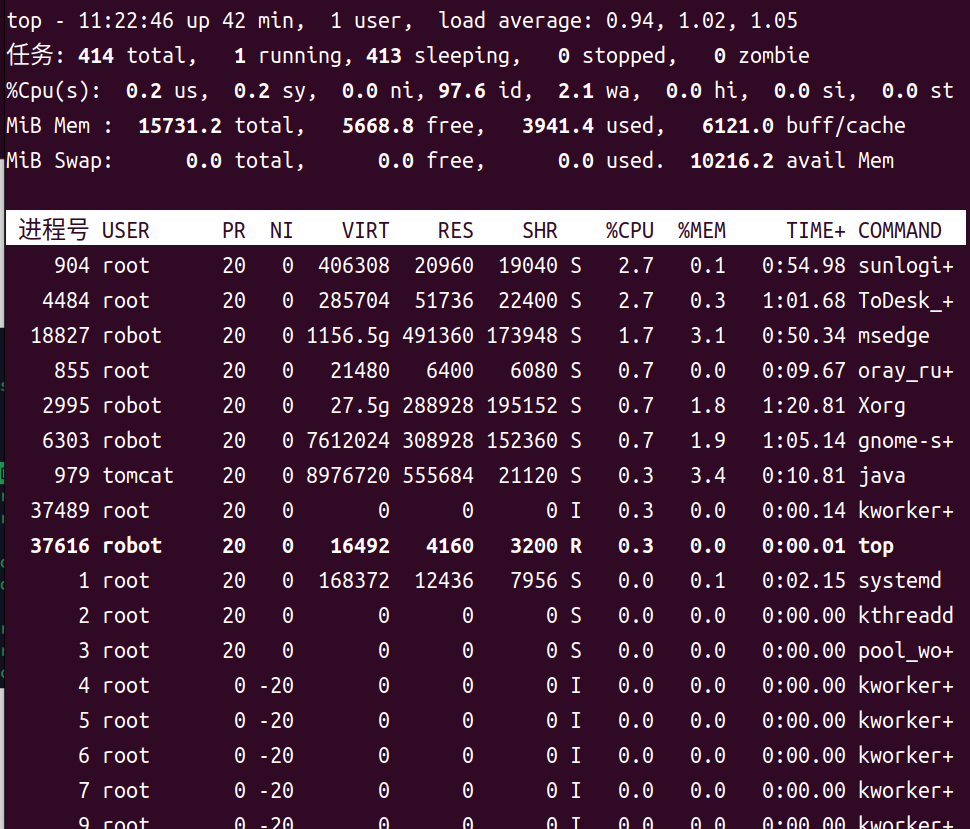
who命令



ps命令



top命令

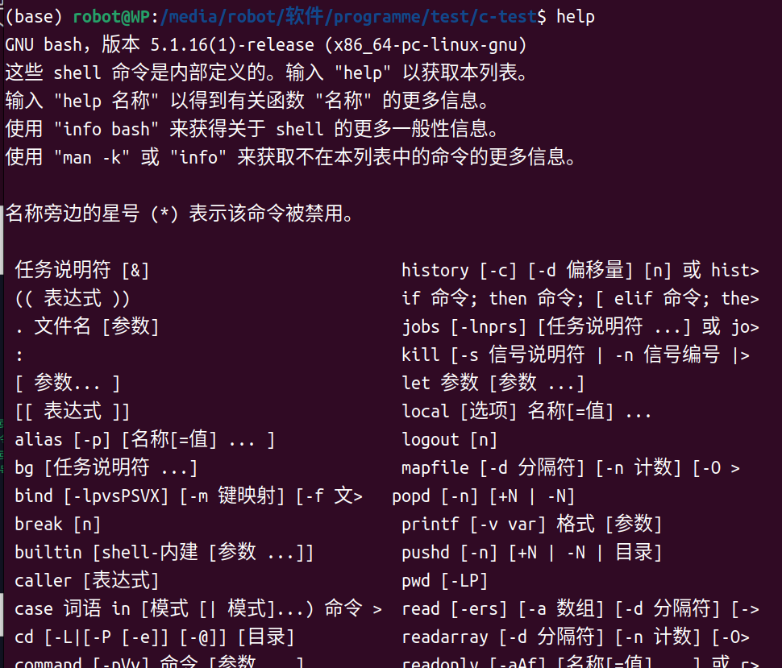


2.5 在线帮助

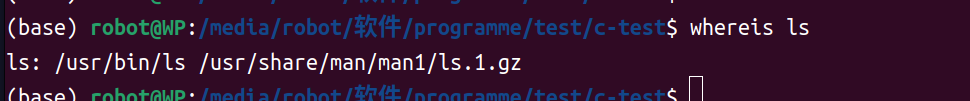
man命令



help命令



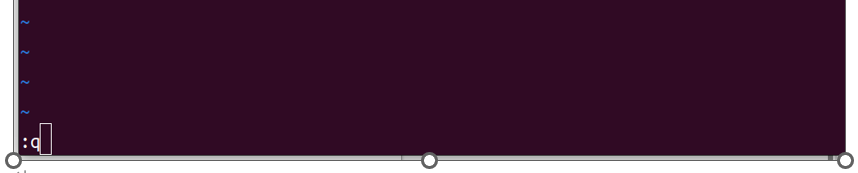
whereis命令



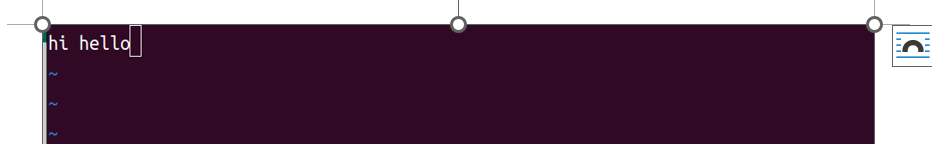
2.5 全屏幕文本编辑器Vim和Vi



命令行模式



文本输入模式

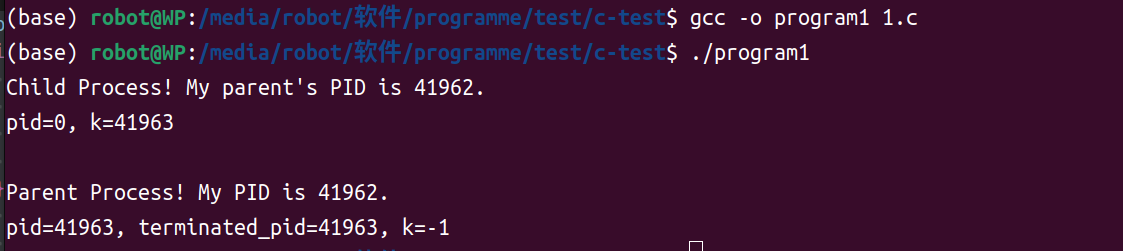


1. 代码及运行结果分析

**程序1**

|  |
| --- |
| #include <stdio.h>  #include <sys/wait.h>  #include <unistd.h>  int main() {  int pid, status;  int k = -1;  pid = fork();  if (pid > 0) {  // 在父进程中  int terminated\_pid = wait(&status); // 返回结束子进程的PID  printf("Parent Process! My PID is %d.\n", getpid()); // 打印父进程的PID  printf("pid=%d, terminated\_pid=%d, k=%d\n", pid, terminated\_pid, k);  } else if (pid == 0) {  // 在子进程中  k = getpid();  printf("Child Process! My parent's PID is %d.\n", getppid()); // 打印父进程的PID  printf("pid=%d, k=%d\n\n", pid, k);  } else {  // fork失败  perror("fork failed");  return 1;  }  return 0;  } |

运行结果：



分析：

该代码演示了使用 `fork()` 创建子进程的基本过程：

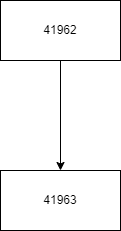
1. `fork()` 创建一个新进程。如果返回值 `pid` 大于 0，则在父进程中执行；如果返回值等于 0，则在子进程中执行；如果返回值小于 0，则表示 `fork()` 失败。

2. 在父进程中，使用 `wait()` 等待子进程结束，并打印父进程的 PID 和子进程的终止 PID。

3. 在子进程中，打印父进程的 PID 和子进程的 PID。

4. 如果 `fork()` 失败，打印错误信息并返回 1。

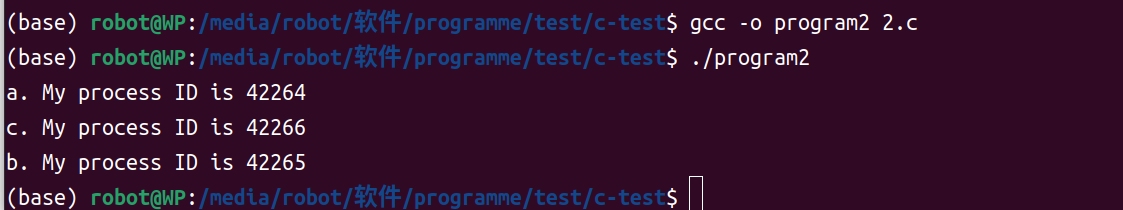
进程树：



**程序2**

|  |
| --- |
| #include <stdio.h>  #include <sys/wait.h>  #include <unistd.h>  int main() {  int p1, p2;  // 创建第一个子进程  while ((p1 = fork()) == -1)  ;  if (p1 == 0) {  // 在第一个子进程中  printf("b. My process ID is %d\n", getpid());  } else {  // 在父进程中，创建第二个子进程  while ((p2 = fork()) == -1)  ;  if (p2 == 0) {  // 在第二个子进程中  printf("c. My process ID is %d\n", getpid());  } else {  // 在父进程中  printf("a. My process ID is %d\n", getpid());  }  }  return 0;  } |

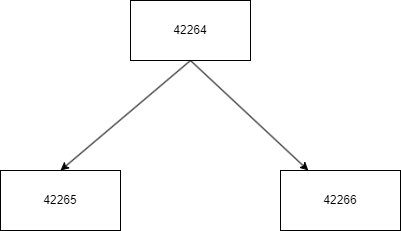
运行结果：



分析：

该程序演示了创建两个子进程的过程。首先，父进程通过 `fork()` 创建第一个子进程 `p1`，如果 `p1` 为 0，则表示在第一个子进程中，打印其进程 ID；否则，父进程继续通过 `fork()` 创建第二个子进程 `p2`，如果 `p2` 为 0，则表示在第二个子进程中，打印其进程 ID；否则，父进程打印其自身的进程 ID。最终，程序将输出三个进程的进程 ID，分别对应父进程和两个子进程。

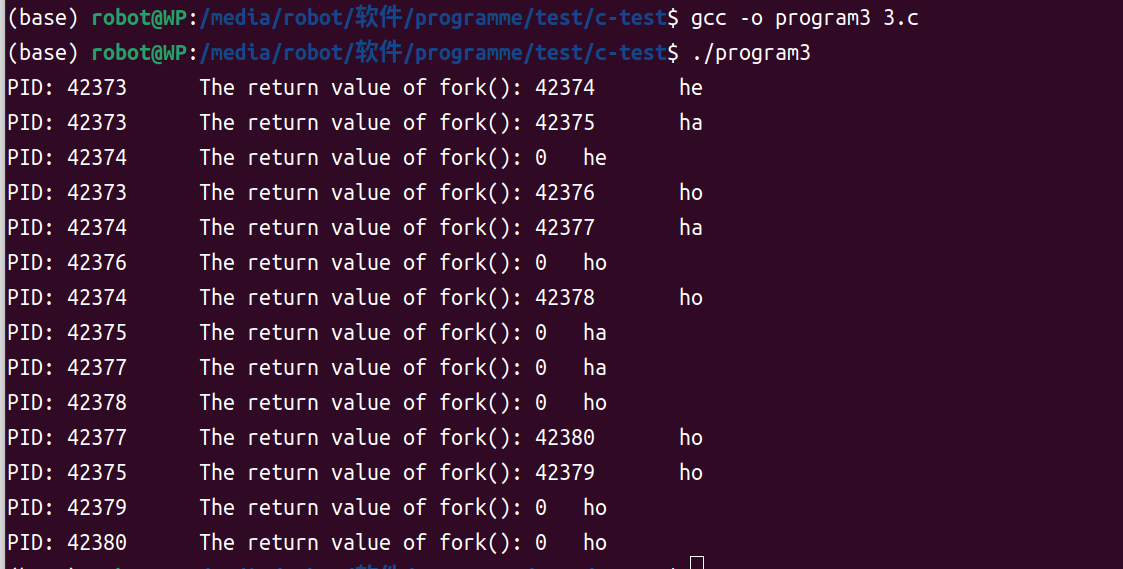
进程树：



**程序3**

|  |
| --- |
| #include <stdio.h>  #include <sys/wait.h>  #include <unistd.h>  int main() {  int m, n, k;  m = fork();  printf("PID: %d\t", getpid());  printf("The return value of fork(): %d\t", m);  printf("he\n");  n = fork();  printf("PID: %d\t", getpid());  printf("The return value of fork(): %d\t", n);  printf("ha\n");  k = fork();  printf("PID: %d\t", getpid());  printf("The return value of fork(): %d\t", k);  printf("ho\n");  return 0;  } |

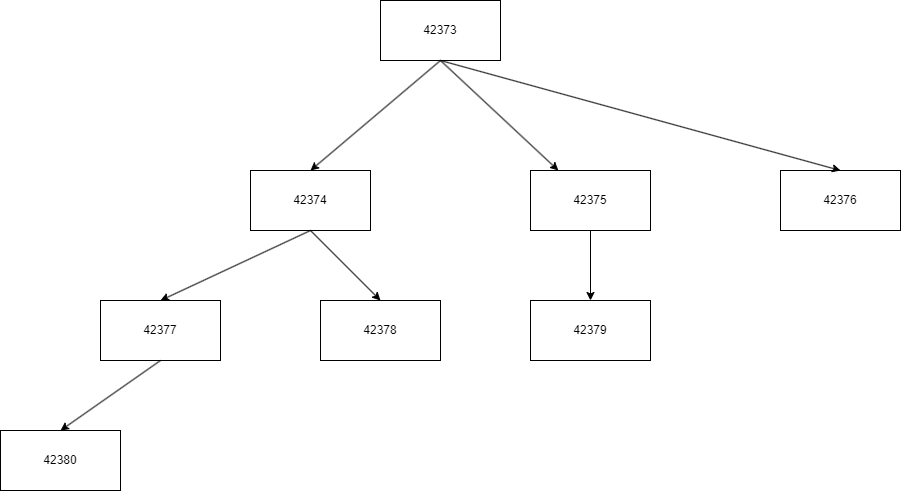
运行结果：



分析：

该程序演示了通过 `fork()` 系统调用创建多个子进程的过程。每次调用 `fork()` 都会创建一个新进程，并返回两次：一次在父进程中返回子进程的 PID，一次在子进程中返回 0。程序中总共调用了三次 `fork()`，因此会创建多个子进程，每个进程都会执行相同的代码并打印其进程 ID和 `fork()` 的返回值。最终输出将包含多个进程的 PID 和相应的消息，显示出进程的创建和执行顺序。有三个fork()调用，创建7个新的子进程，加上原始的父进程，共有8个进程。

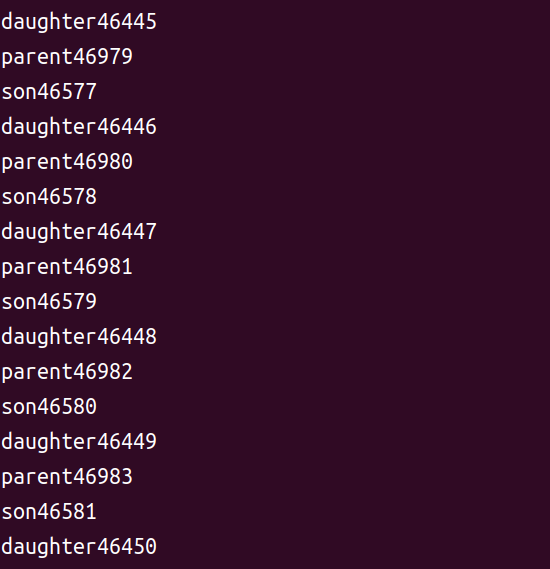
进程树：



**程序4**

|  |
| --- |
| #include <stdio.h>  #include <sys/wait.h>  #include <unistd.h>  main() {  int p1, p2, i;  while ((p1 = fork()) == -1)  ;  if (p1 == 0)  for (i = 0; i < 50000; i++) printf("son%d\n", i);  else {  while ((p2 = fork()) == -1)  ;  if (p2 == 0)  for (i = 0; i < 50000; i++) printf("daughter%d\n", i);  else  for (i = 0; i < 50000; i++) printf("parent%d\n", i);  }  } |

运行结果：



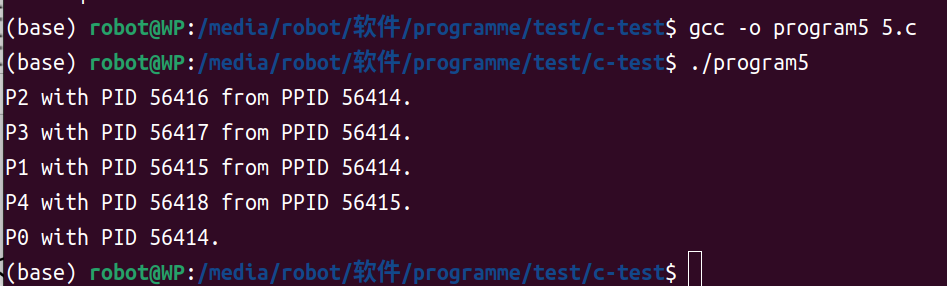
分析：

该程序通过两次调用 `fork()` 创建了两个子进程。第一个 `fork()` 创建了一个子进程 `p1`，如果 `p1` 为 0，则表示在第一个子进程中，循环打印 "son" 和计数器值；否则，父进程继续通过第二个 `fork()` 创建另一个子进程 `p2`，如果 `p2` 为 0，则表示在第二个子进程中，循环打印 "daughter" 和计数器值；否则，父进程循环打印 "parent" 和计数器值。最终，程序将并行输出三个进程的打印结果。

**程序5**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <sys/wait.h>  #include <unistd.h>  int main() {  int pid1, pid2, pid3, pid4;  // P0 创建 P1  pid1 = fork();  if (pid1 < 0) {  perror("fork failed");  exit(1);  } else if (pid1 == 0) {  // 在 P1 中  printf("P1 with PID %d from PPID %d.\n", getpid(), getppid());  // P1 创建 P4  pid4 = fork();  if (pid4 < 0) {  perror("fork failed");  exit(1);  } else if (pid4 == 0) {  // 在 P4 中  printf("P4 with PID %d from PPID %d.\n", getpid(), getppid());  exit(0);  }  // P1 等待 P4 结束  wait(NULL);  exit(0);  } else {  // 在 P0 中，创建 P2  pid2 = fork();  if (pid2 < 0) {  perror("fork failed");  exit(1);  } else if (pid2 == 0) {  // 在 P2 中  printf("P2 with PID %d from PPID %d.\n", getpid(), getppid());  exit(0);  } else {  // 在 P0 中，创建 P3  pid3 = fork();  if (pid3 < 0) {  perror("fork failed");  exit(1);  } else if (pid3 == 0) {  // 在 P3 中  printf("P3 with PID %d from PPID %d.\n", getpid(), getppid());  exit(0);  }  }  }  // P0 等待所有子进程结束  wait(NULL);  wait(NULL);  wait(NULL);  printf("P0 with PID %d.\n", getpid());  return 0;  } |

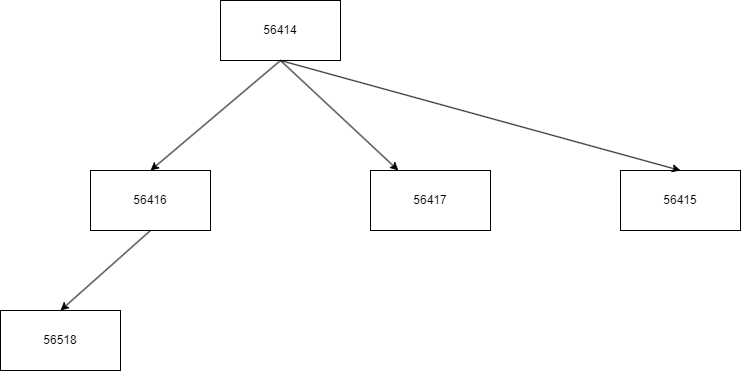
运行结果：



分析：

该程序演示了通过 `fork()` 创建多个子进程的过程。主进程（P0）首先创建一个子进程（P1），P1 再创建一个子进程（P4）。同时，P0 继续创建另外两个子进程（P2 和 P3）。每个子进程在创建后都会打印其进程 ID 和父进程 ID，然后退出。主进程（P0）在创建完所有子进程后，等待所有子进程结束，并打印其自身的进程 ID。最终，程序将输出所有进程的创建和退出信息。

进程图：



1. 心得体会

在本实验中，通过实操各种Linux环境下的命令，我不仅加深了对Linux操作系统的理解，还提升了实际操作技能。使用vim进行简单的文本输入和输出，使我熟悉了这一强大的文本编辑工具，掌握了基本的编辑命令和操作技巧，这对提高编程效率和代码管理起到了积极作用。之后，我使用gcc在命令行中进行编译，进一步理解了编译过程的基本原理和常用选项。通过这些操作，我不仅学会了如何在Linux环境下编写、编辑和编译代码，还熟悉了进程管理的各项工作，包括查看进程状态、终止进程等。这些实践经验使我对Linux系统有了更全面的认识，并为今后的开发工作打下了坚实的基础。