实验1 Linux 命令行的基本使用及编程调试

1.1 实验目的

- 熟悉 Linux 操作系统在命令行的基本操作
- 掌握在 Linux 下 C 程序的编辑、编译、调试的基本方法：
  - 掌握 VIM 的基本操作及在 Vim 中 C 源程序的编辑技巧
  - 掌握使用 gcc 编译器进行 C 程序编译的基本方法和技巧
  - 掌握使用 gdb 进行调试的基本方法与技巧

1.1.1 实验环境要求

- 硬件
  - 主机：Pentium III 以上；
  - 内存：128MB 以上；
  - 显示器：VGA 或更高；
  - 硬盘空间：至少 100MB 以上剩余空间。
- 软件

Linux 操作系统，内核 2.4.26 以上，预装有 X-Window、vim、gcc、gdb 和任意 web 浏览器。

1.1.2 学生实验前的准备工作

阅读与 Linux 操作、vi 使用、gcc 和 gdb 有关的参考资料，理解以下内容：

1. Linux 命令行界面与图形界面有何不同；
2. vi 的功能和操作特点；
3. gcc 的功能和特色；
4. gdb 的功能和操作特点；
5. vi、gcc 和 gdb 组合后的功能

1.2 实验内容

1.2.1 Linux 的基本操作

通过一个 Linux 下的应用程序安装示例，学习部分 Shell 的基本操作命令，主要包括：

1. 系统的启动与关闭
2. 系统登录
3. 目录管理
4. 文件管理、
5. 压缩与解压缩

1.2.2 vi 的基本操作

vimtutor 是 vi 自带的一个入门教程，它直接在 vi 的真实环境下一步步地教授 vi 的基本操作命令。它有多种语言版本，通过命令行参数选择不同的语言。执行“vimtutor zh”进入简体中文教程，有些发行商的版本缺省语言就是简体中文，直接运行 vimtutor 即可。

我们将学习 vimtutor 中的前五讲，内容包括：

1. 新文件、编辑旧文件、保存和退出；
2. 光标移动、文本编辑之删除、插入、替换、更改；
3. 对象的概念和撤销操作；
4. 复制和粘贴；
5. 定位和搜索；
6. 执行外部命令的方法。

1.2.3 使用 gcc 和 gdb 编译和调试 C 语言程序

阅读由陈皓编写的教程《用 GDB 调试程序》，用 vi 建立示例程序，按其中的步骤操作，学习如下内容：

1. 用 gcc 编译器编译可执行文件和附带调试信息的方法；
2. 用 gdb 跟踪调试程序的基本命令，包括查看源代码、设断点、查看变量值、查看函数堆栈、单步运行和连续运行等。

1.3 实验报告

回答下列问题，写入实验报告。

1. 你认为命令行界面和图形界面各有什么优缺点？你更喜欢哪种界面？为什么？
2. 如果要执行当前目录下的 configure 文件，应该输入什么命令？
3. vi 中一次删除一行、一词和一个字符的命令分别是什么？
4. 如果用 gdb 调试未加入调试信息的目标代码会出现什么现象？
5. gcc 和 gdb 都属于一个叫做 GNU 的组织。这个组织的全称是什么？它有什么特色？

实验2 进程

2.1 实验目的

- 通过观察、分析实验现象，深入理解进程及进程在调度执行和内存空间等方面的特点
- 掌握在 POSIX 规范中 `fork` 和 `kill` 系统调用的功能和使用

2.2 实验要求

2.2.1 实验环境要求

1. 硬件
   (1) 主机：Pentium III 以上；
   (2) 内存：128MB 以上；
   (3) 显示器：VGA 或更高；
   (4) 硬盘空间：至少 100MB 以上剩余空间。
2. 软件

Linux 操作系统，内核 2.4.26 以上，预装有 X-Window、vi、gcc、gdb 和任意 web 浏览器。

2.2.2 学生实验前的准备工作

学习 man 命令的用法，通过它查看 fork 和 kill 系统调用的在线帮助，并阅读参考资料，学会 fork 与 kill 的用法。

复习 C 语言的相关内容。

2.3 实验内容

通读下列代码：

```
/*
 * POSIX下进程控制的实验程序残缺版 1.2
 * 作者：Sunner Sun
 * 最后修改时间：2005-3-10 23:59
 */

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <ctype.h>

/* 允许建立的子进程个数最大值 */
#define MAX_CHILD_NUMBER   10

/* 子进程睡眠时间 */
#define SLEEP_INTERVAL           2

int proc_number=0;  /* 子进程的自编号，从0开始 */
void do_something();

main(int argc, char* argv[])
{
    int child_proc_number = MAX_CHILD_NUMBER;   /* 子进程个数 */
    int i, ch;
    pid_t child_pid;
    pid_t pid[10]={0};      /* 存放每个子进程的id */

    if (argc > 1)
    {
        /* 命令行参数中的第一个参数表示建立几个子进程，最多10个 */
        child_proc_number = atoi(argv[1]);
        child_proc_number
            = (child_proc_number > 10) ? 10 : child_proc_number;
    }

    for (i=0; i<child_proc_number; i++)
    {
        /* 在这里填写代码，建立child_proc_number个子进程
         * 子进程要执行
         *  proc_number = i;
         *  do_something();
         * 父进程把子进程的id保存到pid[i] */
    }

    /* 让用户选择杀死哪个进程。输入数字(自编号)表示杀死该进程
     * 输入q退出 */
    while ((ch = getchar()) != 'q')
    {
        if (isdigit(ch))
        {
            /* 在这里填写代码，向pid[ch-'0']发信号SIGTERM，
             * 杀死该子进程 */
```

```
        }
    }

    /* 在这里填写代码，杀死本组的所有进程 */

    return;
}

void do_something()
{
    for(;;)
    {
        /* 打印子进程自编号。为清晰，在每个号码前加“号码+3”个空格
         * 比如号码是1，就打印"    1" */
        printf("This is process No.%*d\n",
                proc_number+3,
                proc_number);
        sleep(2); /* 主动阻塞两秒钟 */
    }
}
```

先猜想一下这个程序的运行结果。假如运行“`./process 20`”，输出会是什么样？

然后按照注释里的要求把代码补充完整，运行程序。可以多运行一会儿，并在此期间启动、关闭一些其它进程，看 process 的输出结果有什么特点，记录下这个结果。

开另一个终端窗口，运行“`ps aux|grep process`”命令，看看 process 究竟启动了多少个进程。

回到程序执行窗口，按“数字键+回车”尝试杀掉一两个进程，再到另一个窗口看进程状况。

按 q 退出程序再看进程情况。

2.4 实验报告

回答下列问题，写入实验报告。

1. 你最初认为运行结果会怎么样？
2. 实际的结果什么样？有什么特点？试对产生该现象的原因进行分析。
3. proc_number 这个全局变量在各个子进程里的值相同吗？为什么？
4. kill 命令在程序中使用了几次？每次的作用是什么？执行后的现象是什么？
5. 使用 kill 命令可以在进程的外部杀死进程。进程怎样能主动退出？这两种退出方式哪种更好一些？

6. 把你的程序源代码附到实验报告后。