计算机科学与工程学院实验报告（首页）

课程名称 Linux 操作系统　　班级 17 软件 2 班

实验名称 实验 6 Linux 下 C 编程　　教导教师 于海洋

姓名 陈庆辉　　学号 1714080902201　　日期 2020/06/07

一、实验目的:

1、熟悉并掌握 linux 下 C 语言程序设计基本方法；
2、熟悉并掌握 gcc 及常用参数编译 C/C++源程序；
3、熟悉并掌握利用 GDB 调试工具调试 C/C++程序；
4、熟悉并掌握利用 MAKE 工具维护及自动编译应用程序。
5、掌握进程相关系统调用；
6、掌握线程相关系统调用；
7、掌握文件相关系统调用。

二、实验要求:

1、熟练使用 GCC 编译工具；
2、熟练使用 GDB 调试工具；
3、熟练使用 MAKE 工具；
4、了解 linux 进程相关概念和明确进程的含义；
5、认识并了解并发执行的实质并掌握 fork()系统调用的相关编程；
6、了解 exec()，system()，wait()，waitpid()，kill()等相关系统调用的功能；
7、熟练文件系统调用的相关编程。

三、实验步骤及结果：

1、 Linux 上的 **C/C++** 编译器：位置**/usr/bin** 或**/usr/local/bin** 可编译 **C/C++**源程序。通常在编译两个或少数几个 **C/C++** 源文件时，利用 **GCC** 编译、连接并生成可执行文件。请验证下面简单 **C** 程序的编译：

设两个源文件 main.c 和 factorial.c 两个源文件，现在要编译生成一个计算阶乘的程序。

```
//------------ factorial.c  ------------------------------
int factorial (int n)
{ if (n <= 1)
        return 1;
    else
        return factorial (n-1) * n;
}

//---------- main.c ----------------------
#include   <stdio.h>
int factorial (int n);
int main (int argc, char **argv)
{ int n;
    if (argc < 2) {
        printf ("Usage: %s n\n", argv [0]);
        return -1;
    }
```

```
    else {
        n = atoi (argv[1]);
        printf ("Factorial of %d is %d.\n", n, factorial (n));
    }
    return 0;
}
```

利用如下的命令可编译生成可执行文件，并执行程序：

```
$ gcc -o factorial main.c factorial.c     //-o FILE：生成指定的输出文件。用在生成可执行文件时
$ ./factorial 5
Factorial of 5 is 120.
```

```
[root@criclehotarux ~]# gcc -o factorial main.c factorial.c
[root@criclehotarux ~]# ./factorial 5
Factorial of 5 is 120.
```

2、利用 GDB 调试有错误程序：利用 gdb 调试有错误的 C/C++程序使用 gdb 调试程序之前，必须使用 -g 选项编译源文件。

```
//-----------------一个有错误的 C 源程序 bugging.c ----------------------
#include<stdio.h>
#include<string.h>
static char buff [256];
static char*  string;
int main ()
{ printf ("Please input a string: ");
    gets (string);
    printf ("\nYour string is: %s\n", string);
}
```

程序非常简单，其目的是接受用户的输入，然后将用户的输入打印出来。该程序使用了一个未经过初始化的字符串地址 string，因此，编译并运行之后，将出现 Segment Fault 错误：

```
$ gcc -o bugging -g bugging.c
$ ./bugging
Please input a string: asfd
Segmentation fault (core dumped)
```

```
[root@criclehotarux ~]# nano bugging.c
[root@criclehotarux ~]# gcc -o bugging -g bugging.c
bugging.c: 在函数 'main'中:
bugging.c:10:3: 警告：不建议使用 'gets'(声明于 /usr/include/stdio.h:638) [-Wdeprecated-declarations]
   gets (string);
   ^
/tmp/ccBj1PPs.o：在函数 'main'中：
/root/bugging.c:10: 警告：the `gets' function is dangerous and should not be used.
[root@criclehotarux ~]# ./bugging
Please input a string:asfd
段错误(吐核)
[root@criclehotarux ~]#
```

1. 运行 gdb bugging 命令，装入 bugging 可执行文件；
2. 执行装入的 bugging 命令；(run)

3．使用 where 命令查看程序出错的地方；(显示第10行有错误)
4．利用 list 命令查看调用 gets 函数附近的代码；（list）
5．唯一能够导致 gets 函数出错的因素就是变量 string。用 print 命令查看 string 的值；（p string）
6．在 gdb 中，可以直接修改变量值，只要将 string 取一个合法的指针值就可以了，为此在第行处设置断点；（br 10）
7．程序重新运行到第10行处停止，这时可以用 set variable 命令修改 string 的取值；（set string=”init”）
8．然后继续运行，将看到正确的程序运行结果。(c)

```
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
Reading symbols from /root/bugging...done.
(gdb) run
Starting program: /root/bugging
Please input a string:asfd

Program received signal SIGSEGV, Segmentation fault.
0x00007ffff7a7cdee in gets () from /lib64/libc.so.6
Missing separate debuginfos, use: debuginfo-install glibc-2.17-307.el7.1.x86_64
(gdb) where
#0  0x00007ffff7a7cdee in gets () from /lib64/libc.so.6
#1  0x000000000040059f in main () at bugging.c:10
(gdb) list
1       #include<stdio.h>
2       #include<string.h>
3
4       static char buff[256];
5       static char* string;
6
7       int main()
8       {
9         printf ("Please input a string:");
10        gets (string);
(gdb) p string
$1 = 0x0
(gdb) br 10
Breakpoint 1 at 0x400590: file bugging.c, line 10.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /root/bugging

Breakpoint 1, main () at bugging.c:10
10        gets (string);
(gdb) set string="init"
(gdb) c
Continuing.
Please input a string:asfd

Your string is: asfd
[Inferior 1 (process 16293) exited normally]
(gdb)
```

3、利用 **make** 工具实现自动编译和工程管理。验证下面题目。
设计一个程序，程序运行时从三道题目中随机抽取一道，题存放在二维数组中。
（1）分析程序，组织文件

fun_main.c 从三道题目中随机抽取一道题。

head_random.h 包含把函数声明和用的的库函数。

fun_random.c 文件中定义随机函数，返回随机数，并返回主函数。

（2） 程序代码：

```
//------------fun_main.c-----------------
#include "head_random.h"
int main()
{
 int n;
 static char str[3][80]={"linux c 函数都可以分割成独立文件吗？","make 工具管理器的作用是？","makefile 文件是否可以使用变量？"};
 n=f_random();
 printf("随机抽取的题号是：%d\n",n+1);
 printf("第%d 题：",n+1);
 puts(str[n]);
}
//-----------head_random.h---------------
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int f_random();
//----------fun_random.c-----------------
#include "head_random.h"
int f_random()
{
 srand(time(NULL));
 int a;
 a=(rand() % 3);
 return a;
}
```

（3）编辑 makefile 文件

```
#---------makefile---------
fun_main : fun_main.o fun_random.o
 gcc -o fun_main fun_main.o fun_random.o
fun_main.o : fun_main.c head_random.h
 gcc -c fun_main.c
fun_random.o : fun_random.c
 gcc -c fun_random.c
```

或者

```
#----------makefilevar----------
CC=gcc
objects=fun_main.o fun_random.o
fun_main:$(objects)
 $(CC) $(objects) -o fun_main
fun_main.o:fun_main.c head_random.h
 $(CC) fun_main.c -c
fun_random.o:fun_random.c
 $(CC) fun_random.c -c
```

（4）用 make 命令编译程序

make 或者 make -f makefilevar

（5）运行程序

./fun_main

```
[root@criclehotarux ~]# make
gcc -c fun_random.c
gcc -o fun_main fun_main.o fun_random.o
[root@criclehotarux ~]# ./fun_main
随机抽取的题号是：2
第2 题：make 工具管理器的作用是？
```

4、编写一段程序，使用系统调用 fork( )创建两个子进程。当此程序运行时，在系统中有一个父进程和两个子进程活动。让每一个进程在屏幕上显示一个字符：父进程显示’a’，子进程分别显示字符’b’和字符’c’。试观察记录屏幕上的显示结果，并分析原因。

参考程序：

```
#include <stdio.h>
main( )
{
    int p1,p2;
    while((p1=fork( ))== -1);          /*创建子进
    p1*/
    if (p1==0)    putchar('b');
    else
        {
            while((p2=fork( ))== -1);    /*创建子进
            程 p2*/ if(p2==0)            putchar('c');
            else   putchar('a');
        }
}
```

结果：从进程并发执行来看，输出abc,bac,acb等都有可能。

原因：fork()创建进程所需的时间要多于输出一个字符的时间，因此在住进程创建进程2的同时，进程1就输出了“b”，而进程2和主程序的输出次序是有随机性的，所以会出现上述结果。

5、修改上述程序，每一个进程循环显示一句话。子进程显示'daughter …'及'son ……'，父进程显示 'parent ……'，观察结果，分析原因。

参考程序：

```
#include <stdio.h>
main( )
{
int p1,p2,i;
while((p1=fork( ))= = -1); /*创建子进程 p1*/
if (p1= =0)
  for(i=0;i<10;i++)
printf("daughter %d\n",i);
else
{
  while((p2=fork( ))= = -1); /*创建子进程 p2*/
if(p2= =0)
  for(i=0;i<10;i++)
  printf("son %d\n",i);
else
  for(i=0;i<10;i++)
  printf("parent %d\n",i);
}
}
```

```
son 9
[root@criclehotarux ~]# ./fun_fork_plus
parent 0
parent 1
parent 2
parent 3
parent 4
parent 5
parent 6
parent 7
parent 8
parent 9
daughter 0
daughter 1
daughter 2
daughter 3
daughter 4
daughter 5
daughter 6
daughter 7
daughter 8
daughter 9
son 0
son 1
son 2
son 3
son 4
son 5
son 6
son 7
son 8
son 9
[root@criclehotarux ~]#
```

分析同上。

四、感想心得

平时习惯了在 Win10 使用 IDE 进行编程，这次在 Linux 上编程多少有些别扭。不过 Linux 上也可以安装诸如 eclipse 等工具，对新手会更友好。