

# 《Linux 实验指导书》

班 级：

学 号：

姓 名： B a n b a n

指 导 教 师：

2023 年 5 月 19 日

# 实验一 脚本程序设计

## 1. 目的和要求

加强对脚本程序设计的理解。

## 2. 实验内容

参数列表，循环。把下面程序的参数列表显示用 for 循环实现。

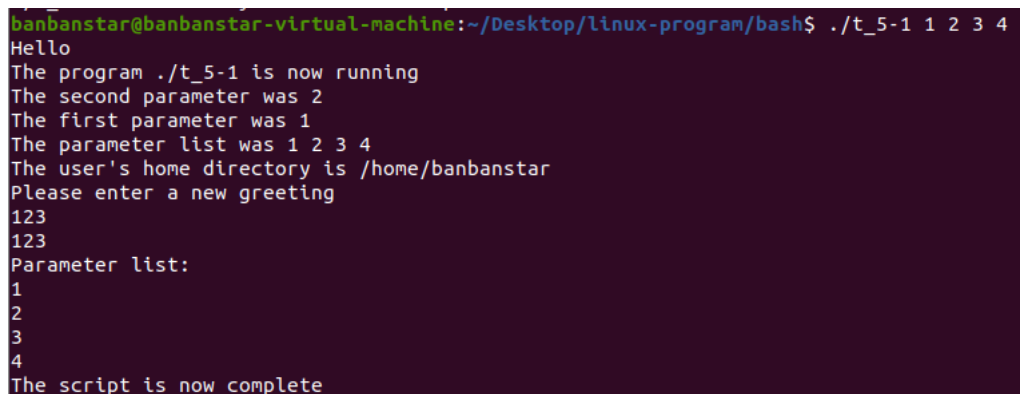
## 3. 实验提示

```
#!/bin/bash
salutation="Hello"
echo $salutation
echo "The program $0 is now running"
echo "The second parameter was $2"
echo "The first parameter was $1"
echo "The parameter list was $*"
echo "The user's home directory is $HOME"
echo "Please enter a new greeting"
read salutation
echo $salutation

# 将参数列表循环打印
for i in "$@";do
    echo "$i"
done

echo "The script is now complete"
exit 0
```

## 4. 实验运行结果



```
banbanstar@banbanstar-virtual-machine:~/Desktop/linux-program/bash$ ./t_5-1 1 2 3 4
Hello
The program ./t_5-1 is now running
The second parameter was 2
The first parameter was 1
The parameter list was 1 2 3 4
The user's home directory is /home/banbanstar
Please enter a new greeting
123
123
Parameter list:
1
2
3
4
The script is now complete
```

## 实验二 文件操作

### 1. 目的和要求

熟悉文件操作的系统调用和库函数

### 2. 实验内容

比较使用系统调用和库函数的文件拷贝程序的速度性能

### 3. 实验提示

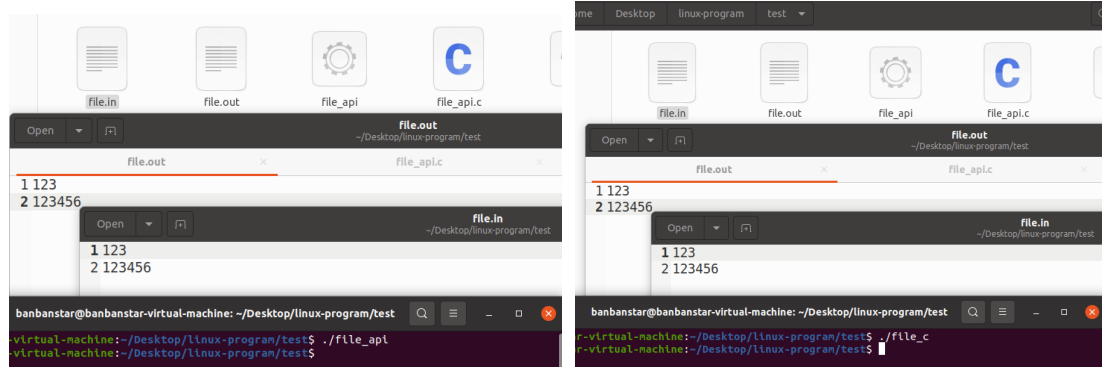
```
// ----- LINUX-API -----
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>

int main() {
    char c;
    int in, out;
    in = open("file.in", O_RDONLY);
    out = open("file.out", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
    while (read(in, &c, 1) == 1)
        write(out, &c, 1);
    exit(0);
}

// ----- STDIO -----
#include <stdio.h>
#include <stdlib.h>
int main() {
    int c;
    FILE *in, *out;
    in = fopen("file.in", "r");
    out = fopen("file.out", "w");
    while ((c = fgetc(in)) != EOF)
        fputc(c, out);
    exit(0);
}
```

## 4. 实验运行结果

```
banbanstar@banbanstar-virtual-machine:~/Desktop/linux-program/test$ TIMEFORMAT="" time ./file_c
0.00user 0.00system 0:00.00elapsed 100%CPU (0avgtext+0avgdata 1124maxresident)k
16inputs+352outputs (1major+59minor)pagefaults 0swaps
banbanstar@banbanstar-virtual-machine:~/Desktop/linux-program/test$ TIMEFORMAT="" time ./file_api
0.03user 0.40system 0:00.44elapsed 99%CPU (0avgtext+0avgdata 1028maxresident)k
40inputs+352outputs (1major+56minor)pagefaults 0swaps
banbanstar@banbanstar-virtual-machine:~/Desktop/linux-program/test$ ls -lh file*
-rwxrwxr-x 1 banbanstar banbanstar 17K 5月 17 21:12 file_api
-rw-rw-r-- 1 banbanstar banbanstar 295 5月 17 21:12 file_api.c
-rwxrwxr-x 1 banbanstar banbanstar 17K 5月 17 21:13 file_c
-rw-rw-r-- 1 banbanstar banbanstar 196 5月 17 21:13 file_c.c
-rw-r--r-- 1 banbanstar banbanstar 176K 5月 17 21:45 file.in
-rw-r--r-- 1 banbanstar banbanstar 176K 5月 17 21:46 file.out
```



这两个程序均用于将文件"file.in"中的内容拷贝到文件"file.out"中，但是采用了不同的实现方式，分别使用了 Unix/Linux 下的系统调用和 C 标准库函数。

第一个程序使用了系统调用，通过 `open()` 函数打开了"file.in"和"file.out"两个文件，"file.in"只读文件，"file.out" 读写，在打开"file.out"文件时，使用了 `O_CREAT` 选项来创建一个新文件，并通过 `S_IRUSR` 和 `S_IWUSR` 来设置文件的访问权限。之后使用 `read()` 函数和 `write()` 函数逐个字符地将"file.in"中的内容拷贝到"file.out"中。

第二个程序使用 C 标准库函数，首先通过 `fopen()` 打开了"file.in"和"file.out" 两个文件，"file.in"只读文件，"file.out"写文件，如"file.out"文件不存在，则使用"w"模式打开该文件会自动创建文件，并覆盖原有内容。之后使用 `fgetc()` 和 `fputc()` 逐个字符读取"file.in" 中的内容并写入"file.out"中。最后使用 `exit()` 退出程序，返回给操作系统一个退出状态码。

## 实验三 创建子进程

### 1. 目的和要求

熟悉创建子进程的系统调用 `fork()`

### 2. 实验内容

创建子进程并实现父子进程的同步

### 3. 实验提示

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>

int main() {
    pid_t pid;
    char *message;
    int n;
    int exit_code;
    printf("fork program starting\n");
    pid = fork();
    switch (pid) {
        case -1:
            perror("fork failed");
            exit(1);
        case 0:
            message = "This is the child";
            n = 5;
            exit_code = 37;
            break;
        default:
            message = "This is the parent";
            n = 3;
            exit_code = 0;
            break;
    }
    for (; n > 0; n--) {
        puts(message);
        sleep(1);
    }
    if (pid != 0) {
        int stat_val;
```

注意不要雷同

banban

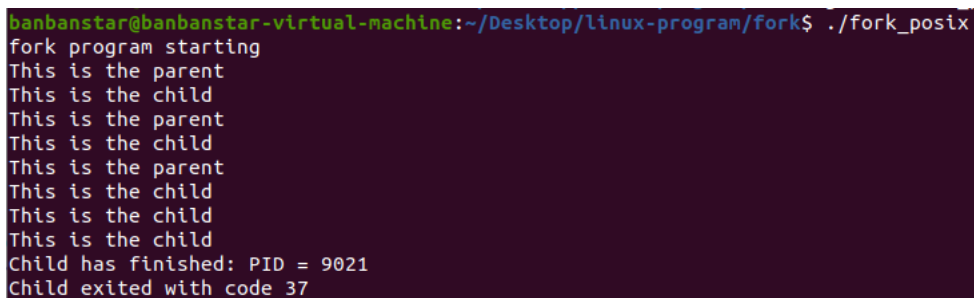
<https://github.com/dream4789/Computer-learning-resources.git>

```

    pid_t child_pid;
    child_pid = wait(&stat_val);
    printf("Child has finished: PID = %d\n", child_pid);
    if (WIFEXITED(stat_val))
        printf("Child exited with code %d\n", WEXITSTATUS(stat_val));
    else
        printf("Child terminated abnormally\n");
}
exit(exit_code);
}

```

## 4. 实验运行结果



```

banbanstar@banbanstar-virtual-machine:~/Desktop/linux-program/fork$ ./fork_posix
fork program starting
This is the parent
This is the child
This is the parent
This is the child
This is the parent
This is the child
This is the child
This is the child
This is the child
Child has finished: PID = 9021
Child exited with code 37

```

这个程序演示了在 Unix/Linux 系统中创建子进程的方法，通过调用 `fork()` 函数来创建一个新的进程，然后在父进程和子进程中分别执行不同的代码。

当 `fork()` 调用成功后，返回值分为三种情况：若返回值为 -1，表示 `fork()` 调用失败（新进程无法创建），程序就会退出并打印错误信息；若返回值为 0，表示当前进程为子进程，执行子进程的代码，这里将 `message` 设为 "This is the child"，`n` 设为 5，`exit_code` 设为 37；若返回值大于 0，表示当前进程为父进程。

接下来，在一个 `for` 循环中，打印 `message` 的值，并休眠 1 秒。当 `for` 循环执行完毕后，如果 `pid` 不等于 0，表示当前进程为父进程，使用 `wait` 函数收集子进程的终止状态，并打印出子进程的 PID 和退出状态码。`WIFEXITED()` 和 `WEXITSTATUS()` 这两个宏函数分别用于判断子进程是否正常终止，以及打印子进程的退出状态码。最后使用 `exit()` 函数退出程序，并返回给操作系统一个退出状态码。

## 实验四 SOCKET

### 1. 目的和要求

理解 socket 的基本原理。

### 2. 实验内容

利用 socket 完成本地通信

### 3. 实验提示

```
// ----- SERVER -----
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <sys/un.h>
#include <unistd.h>

int main() {
    int server_sockfd, client_sockfd;
    int server_len, client_len;
    struct sockaddr_un server_address;
    struct sockaddr_un client_address;

    unlink("server_socket");
    server_sockfd = socket(AF_UNIX, SOCK_STREAM, 0);
    server_address.sun_family = AF_UNIX;
    strcpy(server_address.sun_path, "server_socket");
    server_len = sizeof(server_address);
    bind(server_sockfd, (struct sockaddr *) &server_address, server_len);

    listen(server_sockfd, 5);
    while (1) {
        char ch;
        printf("server waiting\n");
        client_len = sizeof(client_address);
        client_sockfd = accept(server_sockfd, (struct sockaddr *)
&client_address, &client_len);
        read(client_sockfd, &ch, 1);
        ch++;
        write(client_sockfd, &ch, 1);
        close(client_sockfd);
    }
}
```

```
// ----- CLIENT -----
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/un.h>
#include <unistd.h>

int main() {
    int sockfd;
    int len;
    struct sockaddr_un address;
    int result;
    char ch = 'A';

    sockfd = socket(AF_UNIX, SOCK_STREAM, 0);
    if (sockfd == -1) {
        perror("socket");
        exit(EXIT_FAILURE);
    }

    address.sun_family = AF_UNIX;
    strcpy(address.sun_path, "server_socket");
    len = sizeof(address);

    result = connect(sockfd, (struct sockaddr *) &address, len);
    if (result == -1) {
        perror("oops:client1");
        close(sockfd);
        exit(EXIT_FAILURE);
    }

    result = write(sockfd, &ch, 1);
    if (result == -1) {
        perror("write");
        close(sockfd);
        exit(EXIT_FAILURE);
    }

    result = read(sockfd, &ch, 1);
    if (result == -1) {
        perror("read");
        close(sockfd);
        exit(EXIT_FAILURE);
    }
}
```



```

}
printf("char from server = %c\n", ch);
close(sockfd);
exit(EXIT_SUCCESS);
}

```

## 4. 实验运行结果

The screenshot displays a Linux environment where a Unix socket program is being tested. The terminal window shows the following sequence of events:

- The server program (`./socket_s`) is executed, and it prints "server waiting" three times, indicating it is ready to accept connections.
- The client program (`./socket_c`) is executed, and it prints "char from server = B", showing it has successfully received data from the server.
- The client program is executed again, and it prints "char from server = B", showing it has successfully received data from the server.
- The `ls -l` command is used to list the files in the current directory. The output shows several files, including `server_socket`, which is highlighted with a red box, indicating it is the file created by the server program.

The file manager window below the terminal shows the files in the `linux-program/test` directory. The files `server_socket`, `socket_c`, `socket_c.c`, and `socket_s` are highlighted with a red box, indicating they are the files created by the program.

这是一个基于 Unix socket 的通信程序。

服务端程序创建了一个名称为 `server_socket` 的 UNIX 域套接字，并通过 `bind` 函数将其绑定到该套接字上，然后通过 `listen` 函数将其设置为被动模式，以等待客户端的连接请求。

客户端实现了向一个名为 `server_socket` 的 UNIX 域套接字（UNIX domain socket）发起连接，并向其中写入一个字符，然后从服务器读取一个字符，并将其打印在终端上。在和服务器完成读写操作后，关闭套接字并退出程序。