

CS334 Assignment2

SID: 12012919

Name: 廖铭騫

1

(1)

三个方面分别是：Virtualization (虚拟化)、concurrency (并发)、persistence (持久性)
解释：

虚拟化 (virtualization) 意味着操作系统将实际的物理资源进行虚拟化，转换为更通用更强大，且更易于使用的虚拟形式。虚拟化具体的表现有虚拟化 CPU 以及 虚拟化内存，本质上都是提供了一层假象，使得应用程序认为自己独享着一块 CPU 或者一块物理内存。操作系统可以通过虚拟化提供更为安全的服务，比如程序的保护以及隔离。

并发 (concurrency) 意味着同时运行多个程序，几个程序同时处于已启动与等待完成的状态。并且在任一时刻，只有一个程序都是在同一个处理器上面运行。

持久性 (persistence) 意味着使用硬件以及软件来持久化地存储数据，防止数据因为断电或者系统崩溃或者其他原因而丢失。

(2)

虚拟化对应的章节为 3、5 章 以及 第 8、9 章。

并发对应的章节为第 4、6、7 章

持久化对应的章节为第 10 ~13 章

2

对于上下文切换，在时钟中断发生时，操作系统为了保存当前正在运行的进程的上下文，首先需要执行一些底层汇编代码来保存原进程的通用寄存器、程序计数器以及当前正在运行的进程的内核栈指针，然后进入内核模式，进入陷阱处理程序来处理陷阱。然后将寄存器 A 保存进进程结构 A，把将要切换到的目标进程的进程结构 B 恢复到寄存器 B，之后从陷阱返回。之后从内核栈恢复寄存器 B，再转向用户模式，再跳到 B 的程序计数器，之后便完成了上下文的切换，继续执行进程 B。

3

(1)

当内核处理 `fork()` 的系统调用的时候，对于系统调用机制（system call mechanism）来说，父进程会调用 `fork()` 系统调用来创建一个和父进程几乎完全一样的子进程，一样的部分包括用户空间（User space）的数据（PC、程序代码、内存以及打开的文件）。对于 PCB 来说，子进程的 PCB 将会和父进程完全一致，因为子进程的 PCB 由父进程复制而来。在 `fork()` 调用之后，子进程会被添加到系统的进程列表当中。复制完毕之后，父进程的 `fork()` 会返回子进程的 PID，而子进程的 `fork()` 会返回 0。在 `fork()` 返回之后，开始调度器的调度，由调度器(scheduler)来决定下一个运行的进程，如果下一个执行的进程不是当前的父进程，则需要进行上下文的切换，具体的切换流程见 题2 所示。

(2)

当一个进程执行 `exit()` 系统调用的时候，内核会释放所有分配给该进程的位于内核的内存空间(allocated kernel-space memory)，会关闭所有该进程打开的文件。之后，内核会释放所有与该进程相关的用户空间(User space)部分的内存，包括该进程的程序代码以及分配的内存空间，再使用 `SIGCHLD` 来通知该进程的父进程，告知该进程已经运行结束，终止。

但是进程号（Process ID）依然保留在内核的进程表当中，没被回收，此时进程进入僵尸状态（zombie's state）。并且僵尸进程的该表项依然被产生该僵尸进程的进程所需要，以读取该僵尸进程的退出状态（exit status）。在这之后，内核会通过 `SIGCHLD` 信号来通知该僵尸进程的父进程，告知父进程该僵尸进程已经运行结束。

`exit()` 与 `wait()` 的联系在于，有时候父进程会执行 `wait()`，此时 `wait()` 会阻塞父进程的运行，当子进程运行终止完成，调用 `exit()` 之后，父进程会收到 `exit()` 返回的 `SIGCHLD` 信号，之后 `wait()` 会返回，并且会执行操作：及时地回收子进程的进程号等相关资源。

在父进程执行 `wait()` 的情况下，当子进程运行结束之后产生的僵尸进程都会被及时回收处理。

在父进程没有执行 `wait()` 的情况下，如果父进程比子进程后结束，那么子进程会持续地以僵尸进程的状态存活。如果因为父进程不回收而导致出现僵尸进程，那么在父进程结束之后，僵尸进程会被过继给 "继父"，新的父进程会自动回收僵尸进程；如果父进程比子进程前结束，那么在父进程结束之后，子进程会被过继给新的父进程，由新的父进程来自动回收结束之后的子进程。

4

三种方法分别是：

- System call（系统调用）
- Interrupt（中断）
- Trap or Exception（异常）

比较：

中断与异常相比较：中断通常是由外部异步事件引起的上下文切换引起的，而异常通常是由内部的同步事件引起的上下文切换引起的

系统调用是进程自身调用的，主动请求系统的服务，发生在进程之外。与中断的区别在于系统调用会使能中断，也就是在系统调用的过程中依然会被其他中断所中断；而中断发生的时候会禁用中断。且中断是外设引起，异常是应用程序意想不到的行为引起，而系统调用是应用程序自身为请求操作系统提供的服务而主动发起。且系统调用有同步也有异步，而中断是异步，异常是同步，这些都是三种方法的不同对比之处。

5

进程首先进入初始态（Start），初始化内存空间；在这之后，为了能在处理器上运行，进程进入准备状态（Ready），等待操作系统为其分配处理器来运行，还有一种进入准备状态的情况是，当进程在 CPU 上运行的时候，遇到了中断，且操作系统决定 CPU 需要交给另一个进程运行的时候，当前进程从运行状态（Running）或者等待状态（Waiting）进入准备状态（Ready）；当进程被分配了处理器的时候，进程从准备状态（Ready）进入运行状态（Running），在运行状态（Running）当中，处理器运行进程的相关指令；当进程需要等待资源响应（如用户的输入或者文件的读取）的时候，进程将会从运行状态（Running）进入等待状态（Waiting）；当进程运行完毕或者操作系统决定终止该进程的时候，进程将会进入终止状态（Terminated/ Exit）。

6

Simple Shell

在这次的操作系统 Assignment 2 当中，实现了一个简易版的Shell，以下是实现的功能介绍。

基本输出

通过使用 `get_user_name()` 以及 `gethostname()` 获取用户的用户名以及主机名，再通过 `getcwd()` 获取当前所在的路径。

再通过 `printf("\033[44;37;5m %s%\033[0m", prefix)` 的类似方式，调整控制台输出，来模拟真实的 shell 输出效果。

输出结果截图如下所示。

```
(base) PS C:\大二课程\操作系统\Assignment> wsl
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ ./shell
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$
```

wsl 运行结果

```
[ 75%] Building C object character/shell.o
[100%] Linking C executable shell
[100%] Built target shell
lmq@lmq-virtual-machine:~/Desktop/shell$ ./shell
lmq@lmq-virtual-machine:/home/lmq/Desktop/shell$
```

虚拟机内运行结果

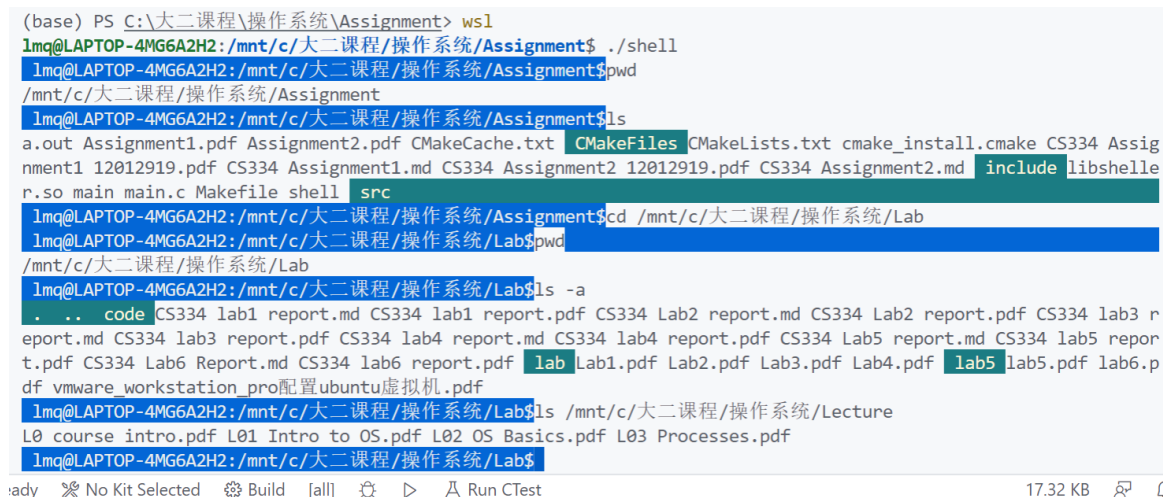
不失一般性，以下的测试将会使用 wsl 进行测试。

pwd

通过使用 `getpwd()` 函数，可以得到当前的工作路径。对

```
// pwd 获取当前工作路径
int pwd_function(char **tokens)
{
    char *path = getcwd(NULL, 0);
    if (path == NULL)
    {
        perror("Fault happens in getcwd()\n");
        return 0;
    }
    printf("%s\n", path);
    free(path); // 释放内存
    return 1;
}
```

下面是功能截图



```
(base) PS C:\大二课程\操作系统\Assignment> wsl
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ ./shell
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ pwd
/mnt/c/大二课程/操作系统/Assignment
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ ls
a.out Assignment1.pdf Assignment2.pdf CMakeCache.txt CMakeFiles CMakeLists.txt cmake_install.cmake CS334 Assignment1 12012919.pdf CS334 Assignment1.md CS334 Assignment2 12012919.pdf CS334 Assignment2.md include libshelle
r.so main main.c Makefile shell src
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ cd /mnt/c/大二课程/操作系统/Lab
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Lab$ pwd
/mnt/c/大二课程/操作系统/Lab
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Lab$ ls -a
. .. code CS334 lab1 report.md CS334 lab1 report.pdf CS334 Lab2 report.md CS334 Lab2 report.pdf CS334 lab3 r
eport.md CS334 lab3 report.pdf CS334 lab4 report.md CS334 lab4 report.pdf CS334 Lab5 report.md CS334 lab5 repor
t.pdf CS334 Lab6 Report.md CS334 lab6 report.pdf lab Lab1.pdf Lab2.pdf Lab3.pdf Lab4.pdf lab5 lab5.pdf lab6.p
df vmware_workstation_pro配置ubuntu虚拟机.pdf
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Lab$ ls /mnt/c/大二课程/操作系统/Lecture
L0 course intro.pdf L01 Intro to OS.pdf L02 OS Basics.pdf L03 Processes.pdf
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Lab$
```

ls

对函数的解释见行间注释

```
// ls 查看当前目录下所有文件
int ls_function(char **tokens)
{
    if (tokens == NULL) // 对传入参数进行检查
    {
        perror("Invalid NULL input\n");
        return 1;
    }
    int cnt = 0;

    int read_hidden_flag = 0;
    char *path = (char*)malloc(sizeof(char) * 512);
```

```

if(!path){
    perror("Failue in allocating for path\n");

    return 1;
}

memset(path, 0, sizeof(char) * 512);

while (tokens[cnt] != NULL) // 对参数个数计数
{
    cnt++;
}
if (cnt > 1 && tokens[1][0] != '-') // 指定路径使用 ls 的情况
{
    strcpy(path, tokens[1]);
}
else if (cnt > 1 && strcmp(tokens[1], "-a") == 0) // -a 输出的情况
{
    path = "./";
    read_hidden_flag = 1;
}
else // 默认路径
{
    path = "./";
}
DIR *dir = opendir(path);

if (dir == NULL)
{
    perror("Failure in opening dir\n");
    return 1;
}
else
{
    struct dirent *files = NULL;
    while (files = readdir(dir)) // 遍历目录下所有文件
    {
        if (!read_hidden_flag && files->d_name[0] == '.') // 如果不需要显式隐藏文件则忽略
            continue;
        if (files->d_type == DT_DIR) // 对文件夹进行特别显示
        {
            printf("\033[33;46;5m %s \033[0m", files->d_name);
        }
        else
        {
            char fullpath[NAME_MAX + 1] = {0};

```

```

        sprintf(fullpath, "%s %s", path, files->d_name); //将
        目录下文件连接到目录路径下
        struct stat st = {0};
        stat(fullpath, &st);
        if (st.st_mode & (S_IXUSR | S_IXGRP | S_IXOTH))
        { //如果目录下文件有执行权限
            printf("%s ", files->d_name);
        }
        else
        {
            printf("%s ", files->d_name);
        }
    }
    printf("\n");
    closedir(dir); //关闭目录
    return 0;
}
}

```

实现功能截图如下：

```

(base) PS C:\大二课程\操作系统\Assignment> wsl
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ ./shell
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ pwd
/mnt/c/大二课程/操作系统/Assignment
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ ls
a.out Assignment1.pdf Assignment2.pdf CMakeCache.txt CMakeFiles CMakeLists.txt cmake_install.cmake CS334 Assignment1 12012919.pdf CS334 Assignment1.md CS334 Assignment2 12012919.pdf CS334 Assignment2.md include libshell.r.so main main.c Makefile shell src
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ cd /mnt/c/大二课程/操作系统/Lab
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Lab$ pwd
/mnt/c/大二课程/操作系统/Lab
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Lab$ ls -a
. .. code CS334 lab1 report.md CS334 lab1 report.pdf CS334 Lab2 report.md CS334 Lab2 report.pdf CS334 lab3 report.md CS334 lab3 report.pdf CS334 lab4 report.md CS334 lab4 report.pdf CS334 Lab5 report.md CS334 lab5 report.pdf CS334 Lab6 Report.md CS334 lab6 report.pdf lab Lab1.pdf Lab2.pdf Lab3.pdf Lab4.pdf lab5 lab5.pdf lab6.pdf vmware_workstation_pro配置ubuntu虚拟机.pdf
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Lab$ ls /mnt/c/大二课程/操作系统/Lecture
L0 course intro.pdf L01 Intro to OS.pdf L02 OS Basics.pdf L03 Processes.pdf
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Lab$

```

cd

对于 cd 切换目录指令的支持，是通过 `chdir()` 函数实现的，如果成功个修改了路径，返回值是0，否则，将会提示用户系统找不到这样的路径。

```

// cd 切换工作路径
int cd_function(char **tokens)
{
    if (tokens == NULL)
    {
        perror("Invalid input\n");
        return 1;
    }
    if (chdir(tokens[1]) == 0)

```

```

    { // 成功
        return 0;
    }
    else
    { // 失败
        printf("-bash: cd: %s: No such file or directory\n",
tokens[1]);
        return 1;
    }
}
}

```

```

(base) PS C:\大二课程\操作系统\Assignment> wsl
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ ./shell
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ pwd
/mnt/c/大二课程/操作系统/Assignment
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ ls
a.out Assignment1.pdf Assignment2.pdf CMakeCache.txt CMakeFiles CMakeLists.txt cmake_install.cmake CS334 Assignment1 12012919.pdf CS334 Assignment2 12012919.pdf CS334 Assignment2.md include libshelle
r.so main main.c Makefile shell src
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ cd /mnt/c/大二课程/操作系统/Lab
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Lab$ pwd
/mnt/c/大二课程/操作系统/Lab
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Lab$ ls -a
. .. code CS334 lab1 report.md CS334 lab1 report.pdf CS334 Lab2 report.md CS334 Lab2 report.pdf CS334 lab3 r
eport.md CS334 lab3 report.pdf CS334 lab4 report.md CS334 lab4 report.pdf CS334 Lab5 report.md CS334 lab5 repor
t.pdf CS334 Lab6 Report.md CS334 lab6 report.pdf lab Lab1.pdf Lab2.pdf Lab3.pdf Lab4.pdf lab5 lab5.pdf lab6.p
df vmware_workstation_pro配置ubuntu虚拟机.pdf
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Lab$ ls /mnt/c/大二课程/操作系统/Lecture
L0 course intro.pdf L01 Intro to OS.pdf L02 OS Basics.pdf L03 Processes.pdf
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Lab$

```

```

lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Lab$ cd /mnt/c/大二课程/操作系统/L
-bash: cd: /mnt/c/大二课程/操作系统/L: No such file or directory
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Lab$

```

```

lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Lab$ cd ..
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统$ cd ./
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统$

```

date

对于 date 的支持，我使用了 `time()`，对得到的时间信息进行处理，拼接，得到一个更为清晰的输出。

```

// date 获取当前系统时间
int date_function(char **tokens)
{
    time_t time_ptr;
    struct tm *tmp_ptr = NULL;

    time(&time_ptr);

    tmp_ptr = localtime(&time_ptr);

    char year[5], month[2], day[3], hour[3], minute[3], second[3];
    sprintf(year, "%d", (tmp_ptr->tm_year + 1900));
    sprintf(month, "%d", (tmp_ptr->tm_mon + 1));

```

```

    sprintf(day, "%d", (tmp_ptr->tm_mday));

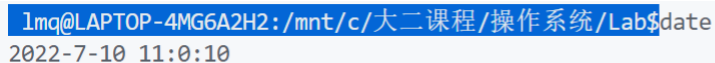
    sprintf(hour, "%d", (tmp_ptr->tm_hour));

    sprintf(minute, "%d", (tmp_ptr->tm_min));
    sprintf(second, "%d", (tmp_ptr->tm_sec));

    char *timeline = (char *)malloc(sizeof(char) * 30);
    memset(timeline, 0, sizeof(char) * 30);
    strcat(timeline, year);
    strcat(timeline, "-");
    strcat(timeline, month);
    strcat(timeline, "-");
    strcat(timeline, day);
    strcat(timeline, " ");
    strcat(timeline, hour);
    strcat(timeline, ":");
    strcat(timeline, minute);
    strcat(timeline, ":");
    strcat(timeline, second);
    printf("%s\n", timeline);
    free(timeline);
    return 1;
}

```

实现的功能截图如下所示:



```

lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Lab$date
2022-7-10 11:0:10

```

help

对于一个 shell 而言, help 也是基本的功能之一, 这里通过输出本程序实现的内部指令以及支持的外部指令样例, 告知用户本 shell 程序的使用方法。

```

// help 展示内部命令
int help(char **tokens)
{
    printf("GNU bash, version 5.0.17(1)-release (x86_64-pc-linux-
gnu)\nThese shell commands are defined internally.  Type `help' to see
this list.\n");
    printf("-----\n");
    printf("[pwd] show the current working path\n");
    printf("[echo] print the string again\n");
    printf("[cd] convert to a specified path\n");
    printf("[date] print the current time and date\n");
    printf("[ls] check all the files in currnet directory\n");
}

```



```

printf("-----\n");
printf("The above commands are implemented, and you can also try
other commands which system can support\n");
printf("Here are some example commands\n");
printf("[ps] have a look at all the running process\n");
printf("[ping] ping to some websites to test the network\n");
printf("Use the man command for information on other
programs.\n");
printf("Also, my shell supports for simple pipes and
redirects\n");
printf("-----\n");
printf("Have a good time ~\n");
return 1;
}

```

功能截图如下：

```

lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统$help
GNU bash, version 5.0.17(1)-release (x86_64-pc-linux-gnu)
These shell commands are defined internally. Type `help' to see this list.
-----
[pwd] show the current working path
[echo] print the string again
[cd] convert to a specified path
[date] print the current time and date
[ls] check all the files in currnet directory
-----
The above commands are implemented, and you can also try other commands which system can support
Here are some example commands
[ps] have a look at all the running process
[ping] ping to some websites to test the network
Use the man command for information on other programs.
Also, my shell supports for simple pipes and redirects
-----
Have a good time ~

```

echo

对于 echo 的支持，主要是通过对输入的字符串数组进行字符串拼接实现的。

```

// echo 回显
int echo_function(char **tokens)
{
    if (tokens == NULL)
    {
        perror("Input tokens is NULL\n");
        return 0;
    }

    else
    {
        char *res = (char *)malloc(sizeof(char) * 100);

        if (res == NULL)
        {

```

```

        perror("Allocation Fault\n");
        free(res);
        return 0;
    }
    memset(res, 0, sizeof(char) * 100);

    if (tokens[1] == NULL)
    {
        printf("\n");
        free(res);
        return 1;
    }

    int i = 1;

    while (tokens[i] != NULL)
    {
        strcat(res, tokens[i]);
        strcat(res, " ");
        i++;
    }

    printf("%s\n", res);
    free(res);
    return 1;
}
}

```

功能截图如下：

```

1mq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统$
1mq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统$
1mq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统$ echo hello world and I love os
hello world and I love os
1mq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统$ echo 沈姐姐无敌
沈姐姐无敌
1mq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统$ echo 五星好评
五星好评
1mq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统$

```

exit

通过告知用户退出的消息，然后正常退出程序

```

// exit 退出 shell 程序
int exit_function(char **tokens)
{
    printf("logout\n");
    free(tokens);
    exit(0);
    return 1;
}

```

```
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ echo 五星好评
五星好评
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ exit
logout
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$
```

其余 Shell 指令

其余的比较复杂的 Shell 指令可以通过使用 `fork()`、`exec()` 以及 `wait()` 加以支持。

一般是通过子进程调用 `exec()` 执行对应的外部指令，父进程使用 `wait()` 等待子进程执行。

```
// 调用外部进程
int outer_process(char **tokens)
{
    int cnt = 0; // tokens 长度
    char *args[10];
    for (int i = 0; i < 10; i++)
    {
        args[i] = NULL;
    }
    while (tokens[cnt] != NULL)
    {
        args[cnt] = tokens[cnt];
        cnt++;
    }

    cnt--;
    int pid = fork();
    if (pid < 0)
    {
        perror("Fork Failure\n");
        return 0;
    }
    else if (pid == 0)
    {
        int flag = execvp(args[0], args);

        if (!flag)
        {
            perror("Wrong command input\n");
            return 0;
        }
    }
    else
    {
        int wc = wait(NULL);
    }
    return 1;
}
```

```
}
```

实现之后，功能截图如下（进行了 `ps -al`，`wc`，`uname`，`ping`，`clear` 的测试）：

```
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ ./shell
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ ps -al
F S      UID        PID  PPID    C  PRI   NI     ADDR  SZ  WCHAN    TTY          TIME CMD
0 S      1000         73     72     0   80    0 -   4525  -          tty1        00:00:00 bash
0 S      1000        149     72     0   80    0 -   4525  -          tty2        00:00:00 bash
0 S      1000        166     73     1   80    0 -   2664  -          tty1        00:00:00 shell
0 R      1000        167    166     0   80    0 -   4645  -          tty1        00:00:00 ps
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ wc src/shell.c
653  1585 16445 src/shell.c
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ uname
Linux
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ man ls
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ ping www.baidu.com
PING www.a.shifen.com (14.215.177.39) 56(84) bytes of data.
64 bytes from 14.215.177.39 (14.215.177.39): icmp_seq=1 ttl=54 time=12.0 ms
64 bytes from 14.215.177.39 (14.215.177.39): icmp_seq=2 ttl=54 time=9.02 ms
64 bytes from 14.215.177.39 (14.215.177.39): icmp_seq=3 ttl=54 time=7.62 ms
```

```
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ vi test.c
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ gcc test.c -o test -O3
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$ ./test
Hello world!
No problem!
```

Pipe 管道

Pipe 管道的实现稍微复杂，主要原理是通过改变标准输入输出流的位置，将其设置为输入管道的位置。这部分通过 `pipe()` 函数以及 `dup2()` 函数进行实现。本程序中，使用子进程的输出作为父进程管道的输入，所以父进程调用 `wait` 等待子进程执行完毕。

在程序内部，为了保证程序的鲁棒性，需要不断地确认参数的合法性以及需要确认内存被正常分配，最后也要确认内存是否完全释放，避免内存泄露。

程序运行完毕之后，确认恢复标准输入输出流，否则会导致后续的 shell 程序无法读入命令。

```
// pipe 管道，返回 0代表没有管道，-1代表进程失败，1代表管道执行成功
int pipe_function(char *line){
    int size = strlen(line);
    int pipe_idx = -1;
    for(int i = 0; i < size; i++){
        if (line[i] == '|'){
            pipe_idx = i;
        }
    }
    if(pipe_idx == -1){ // 表示原指令当中没有管道标志
        return 0;
    }else{
        if(line[pipe_idx - 1] != ' ' || (pipe_idx == size - 1 ||
line[pipe_idx + 1] != ' ')){ // 表示管道不合法，直接退出程序
            perror("Invalid pipe, missing parameter\n");
            return -1;
        }
    }
}
```

```

}
int point[2];
int success = pipe(point);
if(success == -1){ // 创建失败
    return -1;
}
int pid = fork();

if(pid < 0){
    perror("Failure fork\n");
    return -1;
}else if(pid == 0){ // 子进程

    close(point[0]); // 关闭读

    int s_out = dup(STDOUT_FILENO);
    dup2(point[1], STDOUT_FILENO); // 直接使得标准输出流也指向
point[1]指向的句柄，也就是管道的另一头

    close(point[1]);
    // 将管道左边的指令切割下来
    char *left_com = (char *)malloc(sizeof(char) * 512);
    if(left_com == NULL){
        perror("Failure in allocating\n");
        return -1;
    }
    memset(left_com, 0, sizeof(char) * 512);
    for(int i = 0; i < pipe_idx - 1; i++){
        left_com[i] = line[i];
    }
    // printf("The left command is %s\n", left_com);
    char **tokens = split(left_com);
    if(execute(tokens) != 1){
        free(tokens);
        free(left_com);
        return -1;
    }
    free(left_com);
    free(tokens);
    dup2(s_out, STDOUT_FILENO); // 恢复标准输出缓冲区
    exit(0);
}else{// 父进程
    wait(NULL);

    close(point[1]);
    int s_in = dup(STDIN_FILENO);
    dup2(point[0], STDIN_FILENO); // 将管道0作为命令的输入
    close(point[0]);
    int cnt = 0;

```

```

char* right_com = (char*)malloc(sizeof(char) * 512);
if(right_com == NULL){
    perror("Failure in allocating\n");
    return -1;
}
memset(right_com, 0, sizeof(char) * 512);

for(int i = pipe_idx + 2; i < size; i++){
    right_com[cnt++] = line[i];
}

char **tokens = split(right_com);
if(execute(tokens) != 1){
    free(tokens);
    free(right_com);
    return -1;
}

free(right_com);
free(tokens);
dup2(s_in, STDIN_FILENO); // 恢复标准输入缓冲区
}
return 1;
}

```

截图和 FIFO 重定向一并展示

FIFO 重定向

输入输出重定向原理和管道的实现类似，在重定向当中，使用 `freopen()` 函数改变输入输出流，定向到文件当中。

```

// redirect 重定向，-1 表示不合法，0 表示没有重定向，1 表示合法调用重定向
int redirect_function(char * line){

    int size = strlen(line);

    int output_flag = 0;

    int output_idx = size;

    char *output_file = (char*) malloc(sizeof(char) * 256);
    if(output_file == NULL){
        perror("Allocation failure\n");

        return -1;
    }
}

```

```

char *command = (char*) malloc(sizeof(char) * 32);

// 初始化
// memset(input_file, 0, sizeof(char) * 32);
memset(output_file, 0, sizeof(char) * 256);
memset(command, 0, sizeof(char) * 32);

for(int i = 0; i < size; i++){

    if(line[i] == '>'){
        output_flag = 1;
        output_idx = i;
        if(i == size - 1 || i == size - 2){
            perror("Missing parameter\n");

            free(output_file);
            return -1;
        }
    }

}

if(!output_flag){
    return 0;
}

for(int i = 0; i < output_idx - 1; i++){
    command[i] = line[i];
}

int cnt = 0;
for(int i = output_idx + 2; i < size; i++){
    output_file[cnt++] = line[i];
}

int rc = fork();

if(rc < 0){
    perror("Fork Failure\n");

    free(output_file);
    return -1;
}else if(rc == 0){ // 子进程

    if(output_flag){
        freopen(output_file, "w", stdout);
    }

    char ** tokens = split(command);
    int status = execute(tokens);
}

```

```

        if(status != 1){
            // free(input_file);
            free(output_file);
            free(tokens);
            exit(1);
            return -1;
        }
        // free(input_file);
        free(tokens);
        free(output_file);
        free(command);

        exit(0);

    }else{ // 父进程
        wait(NULL);

    }

    free(output_file);
    free(command);
    return 1;
}

```

和Pipe 管道的功能实现截图如下：

```

lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$wc src/shell.c > shell.txt
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$find include/shell.h > include.txt
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$echo hello world | xargs echo
hello world
lmq@LAPTOP-4MG6A2H2:/mnt/c/大二课程/操作系统/Assignment$

```

运行之后，两个文件内容截图如下：

```

include.txt
1  include/shell.h
2

shell.txt
1  653  1585 16445 src/shell.c
2

```

实现了基本的管道输出以及FIFO重定向功能。

以上就是本次 shell 实现的所有内容，感谢阅读。

