《操作系统》课第六次实验报告

学院:	软件学院
姓名:	郁万祥
学号:	2013852
邮箱:	yuwanxiang0114@163.com
时间:	2022.10.21

0. 开篇感言

在使用 C 语言进行多进程操作的过程中，对于我而言，如何操作多线程成为了最大的困难，最开始，我直接参考着课件，尝试着完成任务，但是对于多进程的操作，实在捉襟见肘，因此，经过几番尝试无果之后，还是通过查阅资料，对于系统多进程原理进行了学习，这才再次借助于实验课件的参考，逐步完成了实验的目标，所以，毋庸置疑，理论虽然读起来晦涩难懂，看起来也是不着边际，但是对于实操的过程而言，学习理论知识又是不可避免地重要一环。

1. 实验题目

多进程拷贝目录

2. 实验目标

1、编写 C 程序实现目录拷贝。

2、编写 C 程序实现多进程运行拷贝程序。

3、验证拷贝的正确性。

4、比较多进程拷贝和单进程拷贝的效率。

3. 原理方法

1、目录拷贝：实现过程基于实验二中的拷贝目录的 C 程序，不过此次实验，为了实现后续在多进程运行是对 execlp 函数的使用，我们需要手动输入并传递 main 函数的参数, 因此将之前无参的 main 函数修改为 main(int argc , int *argv[]) 的形式。

2、多进程并发：我们通过 fork 函数进行多进程的创建，然后通过多进程同时完成拷贝目录程序的运行从而达到并发的效果。

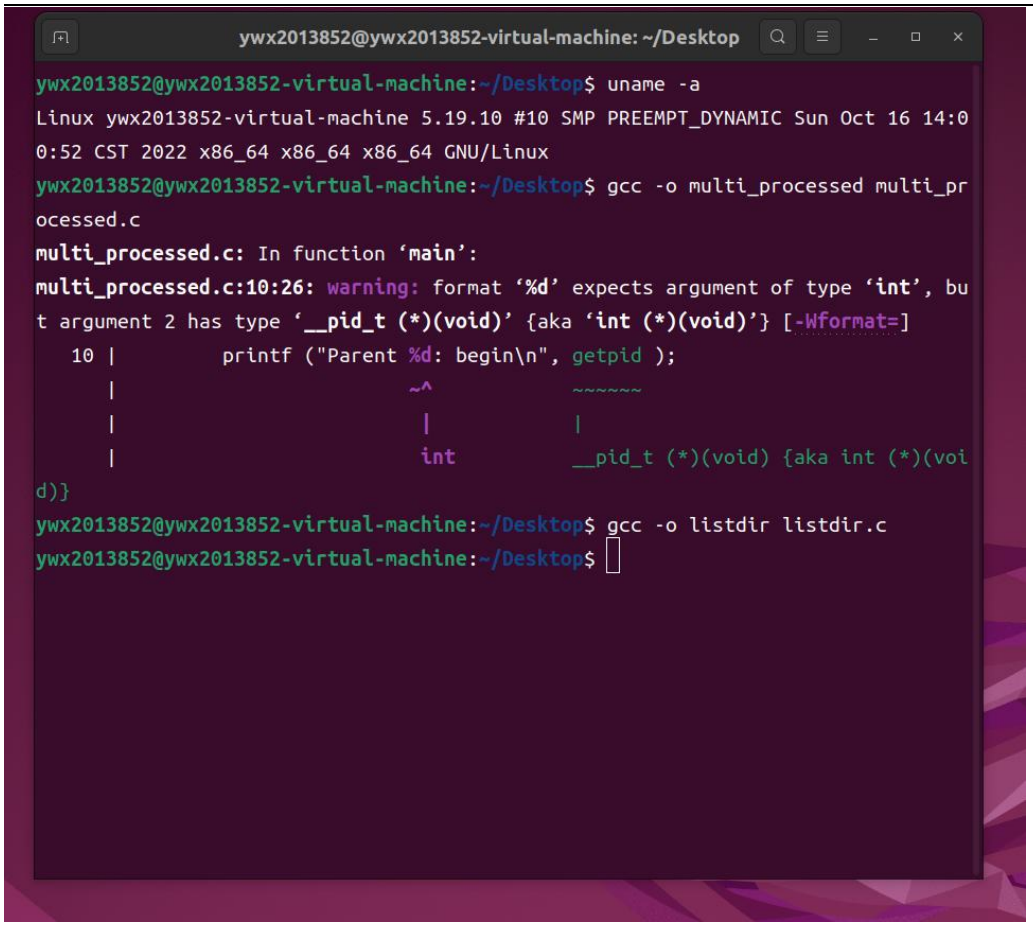
3、exec 函数族：在使用 fork 创建多进程的时候，子进程不仅只能够执行子进程代码块里面的程序，还可以让子进程执行任意想执行的代码，这通过 exec 函数族实现。本次实验使用了 execlp 函数，通过传入参数：拷贝目录程序的绝对地址，拷贝目录程序的参数（即源地址和目的地址）从而实现多个进程运行一个程序。

4. 具体步骤

1、编写拷贝目录和多进程并发的 C 语言程序（源码见附录）。

2、编译 C 文件。

```
gcc -o multi_processed multi_processed.c
gcc -o listdir listdir.c
```



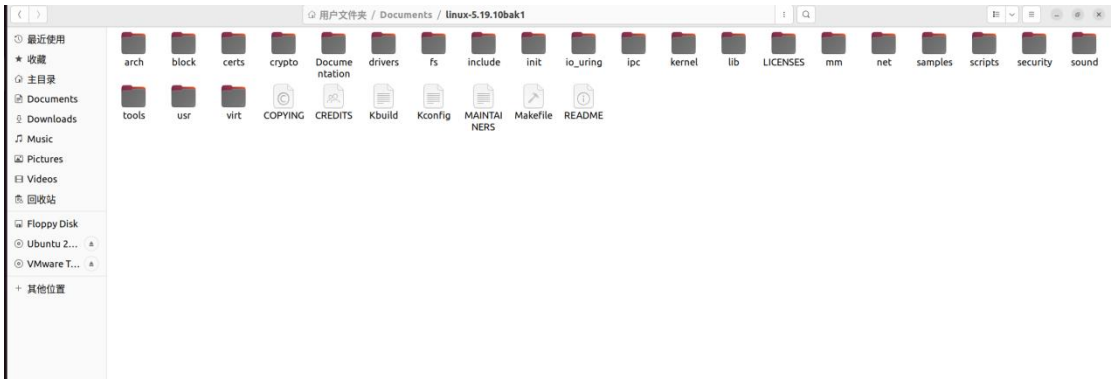
3、运行 C 文件。

./multi_processed /home/ywx2013852/Documents/linux-5.19.10bak /home/ywx2013852/Documents/linux-5.19.10bak1（第一个参数是源文件的地址，第二个参数是目的地址）

```
ywx2013852@ywx2013852-virtual-machine:~/Desktop$ gcc -o multi_processed multi_pr
ocessed.c
multi_processed.c: In function ‘main’:
multi_processed.c:10:26: warning: format ‘%d’ expects argument of type ‘int’, bu
t argument 2 has type ‘__pid_t (*)(void)’ {aka ‘int (*)(void)’} [-Wformat=]
   10 |         printf ("Parent %d: begin\n", getpid );
      |                         ~^            ~~~~~~
      |                          |            |
      |                          int          __pid_t (*)(void) {aka int (*)(voi
d)}
ywx2013852@ywx2013852-virtual-machine:~/Desktop$ gcc -o listdir listdir.c
ywx2013852@ywx2013852-virtual-machine:~/Desktop$ ./multi_processed /home/ywx2013
852/Documents/linux-5.19.10bak /home/ywx2013852/Documents/linux-5.19.10bak1
Parent -57753424: begin
arguments list of ./multi_processed:
  0 ./multi_processed
  1 /home/ywx2013852/Documents/linux-5.19.10bak
  2 /home/ywx2013852/Documents/linux-5.19.10bak1
Parent 7145: Create Child Process 7146
Parent 7145: Create Child Process 7147
Parent 7145: Child 7147 exited with 0 code
Parent 7145: Child 7146 exited with 0 code
Parent 7145: exited
ywx2013852@ywx2013852-virtual-machine:~/Desktop$
```

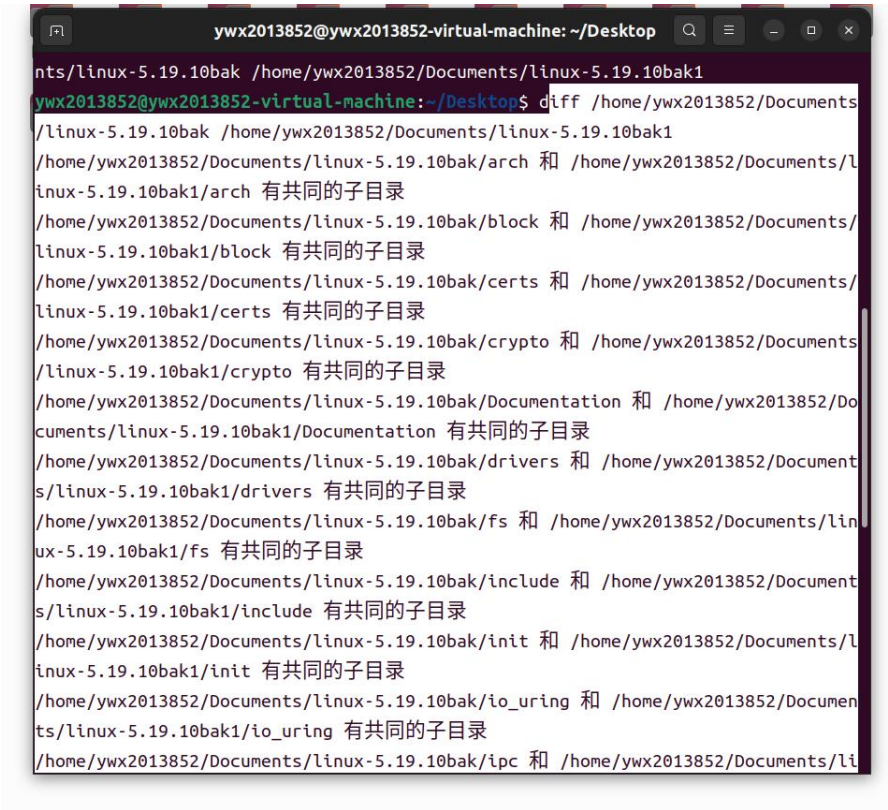
4、检验拷贝正确性。

拷贝情况：



diff 比较：

diff /home/ywx2013852/Documents/linux-5.19.10bak /home/ywx2013852/Documents/linux-5.19.10bak1



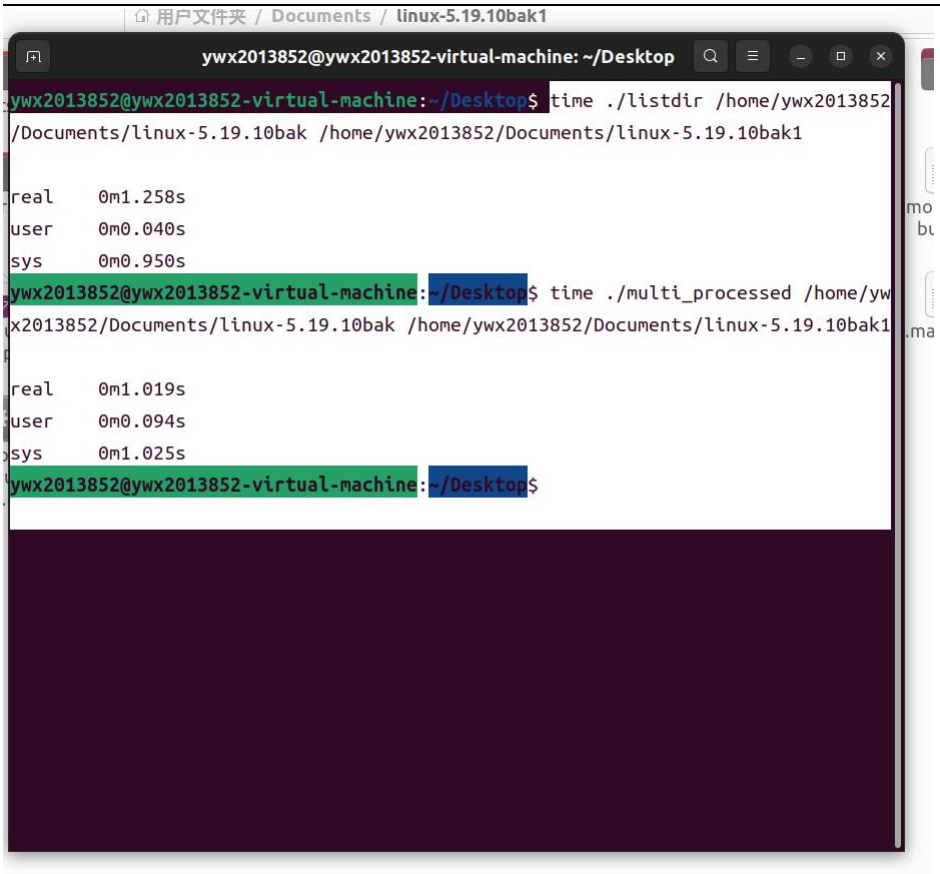
5、比较单进程拷贝和多进程拷贝的效率(以程序运行时间为衡量标准):

单进程拷贝:

time ./listdir /home/ywx2013852/Documents/linux-5.19.10bak /home/ywx2013852/Documents/linux-5.19.10bak1

多进程拷贝:

time ./multi_processed /home/ywx2013852/Documents/linux-5.19.10bak /home/ywx2013852/Documents/linux-5.19.10bak1



5. 总结心得

起初，在比较效率的时候，我发现，多进程拷贝程序所用的总时间和单进程拷贝程序所用时间相差无几，甚至略逊一筹，但是，当我把系统的输出语句删除后，这一现象就不会出现，因此我猜测是因为输出的时间造成了影响，通过思考与实践，进而我发现造成这一现象的原因其实还有本身拷贝程序所用的时间就算不是太多，因此才让程序输出的时间对实验结果造成了一定的影响。

6. 参考资料

源码：

listdir.c：

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int readFileList(char *basePath,char *targetPath)
{
    DIR *dir;
    struct dirent *ptr;
    char base[1000];
    char target[1000];

    if ((dir=opendir(basePath)) == NULL)
    {
        perror("Open dir error...");
        exit(1);
    }

    while ((ptr=readdir(dir)) != NULL)
    {
        if(strcmp(ptr->d_name,".")==0 || strcmp(ptr->d_name,"..")==0)    ///current dir
OR parrent dir
            continue;
        memset(target,'\0',sizeof(targetPath));
        memset(base,'\0',sizeof(base));
        strcpy(target,targetPath);
        strcpy(base,basePath);
        strcat(target,"/");
        strcat(base,"/");
        strcat(target,ptr->d_name);
        strcat(base,ptr->d_name);
        if(ptr->d_type == 4)    ///dir
        {
            int i=mkdir(target,S_IRWXU|S_IRGRP|S_IXGRP|S_IROTH|S_IXOTH);
            readFileList(base,target);
        }
        else {// file
            link (base,target);
        }
    }
    closedir(dir);
    return 1;
```

```
}

int main(int argc,char *argv[])
{
    char* basePath=argv[1];
    char* targetPath=argv[2];
    readFileList(basePath,targetPath);
    return 0;
}
```

multi_processed.c：

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
int main (int argc, char *argv[])
{
    int i;
    pid_t pid;
    int status;
    //printf ("Parent %d: begin\n", getpid )
    for (i = 1; i < 5; i++)
    {
        pid = fork ();
        if (pid < 0)
        {
            fprintf (stderr, "Fork Failed\n");
            break;
        }
        if (pid == 0)
        {
            execlp ("/home/ywx2013852/Desktop/listdir", "listdir", argv[1], argv[2], NULL);
        }
        /*
        else
        {
            printf ("Parent %d: Create Child Process %d\n", getpid (), pid);
        }
        */
    }
    while (1)
    {
        pid = wait (&status);
        if (pid == -1)
```

```
            {
                break;
            }
            else
            {
                //printf ("Parent %d: Child %d exited with %d code\n", getpid (), pid,
                WEXITSTATUS (status);
            }
        }
        //printf ("Parent %d: exited\n", getpid ());
        return 0;
}
```