《操作系统》课第六次实验报告

学院:	软件学院
姓名:	杨万里
学号:	2013774
邮箱:	2013774@mail.nankai.edu.cn
时间:	2022/10/23

0. 开篇感言

如果说上一次实验通过获取所有进程的信息让我们触摸到了操作系统的进程，这一次实验采用多进程进行目录拷贝则让我们实实在在地操纵着系统的进程，让进程为我们所用。

多进程编程非常有趣，当看到自己写的多进程程序迅速完成目录拷贝、程序运行时间大大缩短时，感受到了很大的成就感。

不过本次实验中自己犯了一个低级的错误导致多进程程序的运行时间一开始大于单进程程序，思索检查了大半天才发现问题所在，编程的时候还是要冷静啊！

1. 实验题目

编写一个 C/C++程序实现利用多进程拷贝一个文件夹

2. 实验目标

（1） 编写 C/C++程序

（2） 实现利用多进程拷贝一个文件夹及其所有子文件夹

（3） 采用最新的 Linux Kernel 作为测试文件夹

（4） 验证拷贝结果是否正确

（5） 比较多进程与单进程拷贝文件夹的效率

3. 原理方法及源代码

（1） 单进程拷贝目录的原理方法：在此前的实验中已经实现过单进程拷贝目录的功能，其大致原理方法是获取到指向源目录的指针，逐个遍历其下的文件/子目录，判断其类型，如果是文件夹则递归调用该拷贝函数、如果是普通文件则正常拷贝（实现一个拷贝普通文件的函数）、如果是链接文件则采用 link 函数进行拷贝。

源代码如下所示。

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <dirent.h>
5. #include <unistd.h>
6. #include <sys/types.h>
7. #include <sys/stat.h>
8. #include <fcntl.h>
9. #define  BUFSIZE 1024
10.
11.//拷贝普通文件
12.int copyfile(const char* src, const char* dest) {
13.    FILE* fp1=NULL, * fp2=NULL;
14.    fp1 = fopen(src, "r");
15.    if (fp1 == NULL) {
```

```
16.         return -1;
17.     }
18.     //fopen_s(&fp2,src,"w");
19.     fp2 = fopen(dest, "w");
20.     if (fp2 == NULL) {
21.         return -2;
22.     }
23.     char buffer[BUFSIZE];
24.     int readlen, writelen;
25.     while ((readlen = fread(buffer, 1, BUFSIZE, fp1))>0)
26.     {
27.         writelen = fwrite(buffer, 1, readlen, fp2);
28.         if (readlen != writelen) {
29.             return -3;
30.         }
31.     }
32.     fclose(fp1);
33.     fclose(fp2);
34.     return 0;
35.}
36.
37.//拷贝文件夹
38.int readFileList(char* sourcePath, char* targetPath){
39.    DIR* dir;
40.    struct dirent * ptr;
41.    char source[1000];
42.    char target[1000];
43.    // wrong path
44.    if((dir=opendir(sourcePath)) == NULL){
45.       perror("Open dir error");
46.       exit(1);
47.    }
48.    while((ptr=readdir(dir)) != NULL){
49.       if(strcmp(ptr->d_name,".")==0 || strcmp(ptr->d_name,"
   ..")==0)
50.           continue;
51.        //char arr set 0
52.        memset(target,'\0',sizeof(targetPath));
53.        memset(source,'\0',sizeof(source));
54.        strcpy(target,targetPath);
55.        strcpy(source,sourcePath);
56.        strcat(target,"/");
57.        strcat(source,"/");
58.        strcat(target,ptr->d_name);
```

```
59.         strcat(source,ptr->d_name);
60.         if(ptr->d_type == DT_DIR)    //dir
61.         {
62.             int i=mkdir(target,S_IRWXU|S_IRGRP|S_IXGRP|S_IR
    OTH|S_IXOTH);
63.             readFileList(source,target);
64.         }
65.         else if(ptr->d_type == DT_LNK)    //link
66.         {
67.             link (source,target);
68.         }
69.         else if(ptr->d_type == DT_REG){// file
70.             copyfile(source,target);
71.         }
72.         else{
73.
74.         }
75.     }
76.     closedir(dir);
77.     return 1;
78.}
79.
80.int main(int argc, char *argv[])
81.{
82.     char* sourcePath = argv[1];
83.     char* targetPath = argv[2];
84.     printf(sourcePath);
85.     printf(" to ");
86.     printf(targetPath);
87.     printf("\n");
88.     readFileList(sourcePath,targetPath);
89.     return 0;
90.}
```

本次实验有两种多进程拷贝目录的方式，其主要区别是开辟进程数目的依据不同。

（2） 多进程拷贝目录的原理方法一：**进程数目固定的拷贝方法**。

对于父进程：首先简单地遍历一遍源目录，对于其中的普通文件和链接文件不需要开辟专门的子进程拷贝，因此可以直接在遍历过程中拷贝到目标目录下；对于其中的子目录，在本次遍历不进行拷贝，而是记录子目录的

名称，以及子目录的总数量。遍历之后，将所有要拷贝的子目录等分为 process_num 份（**固定进程数目，实验中取 process_num=4**）。开辟 proce_num 个子进程，每个进程分配一定数量的子目录进行拷贝工作。

父进程程序代码如下所示：

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
#include <fcntl.h>
#define  BUFSIZE 1024

// 拷贝文件
int copyfile(const char* src, const char* dest) {
    //这一部分代码和之前一样，就不重复展示了……
}

//拷贝文件夹
int readFileList(char* sourcePath, char* targetPath){
   //这一部分代码和之前一样，就不重复展示了……
}

int main(int argc, char *argv[])
{
    //接收源目录和目标目录
    char *sourcePath = argv[1];
    char *targetPath = argv[2];
    char source[1000];
    char target[1000];
    //打印信息
    int i;
    pid_t pid;
    int status;
    printf("Parent %d: begin\n", getpid());
    printf("arguments list of %s:\n", argv[0]);
    for(i = 0; i < argc; i++){
      printf("%3d %s\n", i, argv[i]);
```

```
37.     }
38.     char *files[100];
39.     int count = 0;
40.     DIR* dir;
41.     struct dirent * ptr;
42.     //简单遍历
43.     if((dir=opendir(sourcePath)) == NULL){
44.       perror("Open dir error");
45.       exit(1);
46.     }
47.     while((ptr=readdir(dir)) != NULL){
48.       if(strcmp(ptr->d_name,".")==0 || strcmp(ptr->d_name,"
   ..")==0)    ///current dir OR parrent dir
49.           continue;
50.       if(ptr->d_name[0] != '.'){
51.           if(ptr->d_type == DT_DIR)    //文件夹先记录名字
52.           {
53.              files[count] = ptr->d_name;
54.              count += 1;
55.           }
56.           else{ //普通文件和链接文件直接拷贝
57.              memset(target,'\0',sizeof(target));
58.              memset(source,'\0',sizeof(source));
59.              strcpy(target,targetPath);
60.              strcpy(source,sourcePath);
61.              strcat(target,"/");
62.              strcat(source,"/");
63.              strcat(target,ptr->d_name);
64.              strcat(source,ptr->d_name);
65.              if(ptr->d_type == DT_LNK)    //link
66.              {
67.                 link (source,target);
68.              }
69.              else if(ptr->d_type == DT_REG){// file
70.                 copyfile(source,target);
71.              }
72.              else{
73.
74.              }
75.           }
76.       }
77.     }
78.     closedir(dir);
79.     //分配子目录
```

```
80.     int process_num = 4;
81.     int length = count/process_num + 1;
82.     char *parms[process_num][length];
83.     int j;
84.     int k;
85.     for(k=0;k<process_num;k++){
86.       for(i=k*length,j=0;i<(k+1)*length;i++,j++){
87.          parms[k][j] = files[i];
88.       }
89.     }
90.     parms[3][length-1] = "none";
91.     //启动 4 个子进程执行拷贝
92.     for(i = 0;i < process_num;i++){
93.       pid = fork();
94.       if(pid < 0){
95.          fprintf(stderr, "Fork Failed\n");
96.          break;
97.       }
98.       else if(pid == 0){
99.            //子进程 copy
100.           execl("./copy", "./copy", sourcePath, parms[i][0
   ], parms[i][1], parms[i][2], parms[i][3], parms[i][4], parm
   s[i][5], targetPath, NULL);
101.       }
102.       else{
103.          printf("Parent %d: Create Child Process %d\n", g
   etpid(), pid);
104.       }
105.     }
106.     while(1){
107.       pid = wait(&status);
108.       if(pid == -1){
109.          break;
110.       }
111.       else{
112.          printf("Parent %d: Child %d exited with %d code\
   n", getpid (), pid,
113.  WEXITSTATUS (status));
114.       }
115.     }
116.     printf ("Parent %d: exited\n", getpid ());
117.     return 0;
118. }
```

对于子进程：接受的参数包括源目录、目标目录、一定数量的需要拷贝的子目录名称。子进程只需要顺序地将源目录当中地子目录逐个拷贝到目标目录当中即可。多个子进程并发执行。

子进程程序代码如下所示：

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>
#define  BUFSIZE 1024

// 拷贝文件
int copyfile(const char* src, const char* dest) {
   //和之前的函数一样
}

//拷贝文件夹
int readFileList(char* sourcePath, char* targetPath){
   //和之前的函数一样
}

int main(int argc, char *argv[])
{
    clock_t start,stop;
    start = clock();
    char source[1000];
    char target[1000];
    //接收参数
    char *files[6];
    //argv[1]传入原目录名称
    char *dir = argv[1];
    //文件
    for(int i=0;i<6;i++){
      files[i] = argv[i+2];
    }
```

```
    //目标文件夹
    char *targetPath = argv[8];
    for(int i=0;i<6;i++){
       if(files[i] == "none"){
         continue;
       }
       memset(target,'\0',sizeof(target));
       memset(source,'\0',sizeof(source));
       strcpy(target,targetPath);
       strcpy(source,dir);
       char *sourcePath = files[i];
       strcat(target, "/");
       strcat(source, "/");
       strcat(target, sourcePath);
       strcat(source, sourcePath);
       printf("%s to %s", source, target);
       printf("\n");
         //是文件夹
       int i=mkdir(target,S_IRWXU|S_IRGRP|S_IXGRP|S_IROTH|S_IXOTH);
       readFileList(source, target);
    }
    stop = clock();
    double duration = ((double)(stop-start))/CLOCKS_PER_SEC;
    printf("  use_time: ");
    printf("%f", duration);
    printf("\n");
    return 0;
}
```

(3) 多进程拷贝目录的原理方法二：**每个子目录对应一个进程**。

对于父进程：同样简单遍历一次源目录下的所有文件/文件夹，对于普通文件和链接文件进行拷贝，对于文件夹只需要记录其名称和总数 count。开辟 count 个子进程，每个子进程只拷贝源目录下的一个子目录。

父进程程序代码如下所示：

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

```
#include <unistd.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
#include <fcntl.h>
#define  BUFSIZE 1024

// 拷贝文件
int copyfile(const char* src, const char* dest) {
    //与之前的函数一样
}

//拷贝文件夹
int readFileList(char* sourcePath, char* targetPath){
   //与之前的函数一样
}

int main(int argc, char *argv[])
{
    //接收源目录和目标目录
    char *sourcePath = argv[1];
    char *targetPath = argv[2];
    char source[1000];
    char target[1000];
    //打印信息
    int i;
    pid_t pid;
    int status;
    printf("Parent %d: begin\n", getpid());
    printf("arguments list of %s:\n", argv[0]);
    for(i = 0; i < argc; i++){
      printf("%3d %s\n", i, argv[i]);
    }
    char *files[100];
    int count = 0;
    //遍历源目录
    DIR* dir;
    struct dirent * ptr;
    if((dir=opendir(sourcePath)) == NULL){
      perror("Open dir error");
      exit(1);
    }
    while((ptr=readdir(dir)) != NULL){
```

```
48.       if(strcmp(ptr->d_name,".")==0 || strcmp(ptr->d_name,"
   ..")==0)    ///current dir OR parrent dir
49.          continue;
50.       if(ptr->d_type == DT_DIR)    //文件夹先记录名字
51.       {
52.          files[count] = ptr->d_name;
53.          count += 1;
54.       }
55.       else{ //普通文件和链接文件直接拷贝
56.          memset(target,'\0',sizeof(target));
57.          memset(source,'\0',sizeof(source));
58.          strcpy(target,targetPath);
59.          strcpy(source,sourcePath);
60.          strcat(target,"/");
61.          strcat(source,"/");
62.          strcat(target,ptr->d_name);
63.          strcat(source,ptr->d_name);
64.          if(ptr->d_type == DT_LNK)    //link
65.          {
66.             link (source,target);
67.          }
68.          else if(ptr->d_type == DT_REG){// file
69.             copyfile(source,target);
70.          }
71.          else{
72.
73.          }
74.       }
75.    }
76.    //每个子目录对应一个进程
77.    for(i = 0;i < count;i++){
78.       pid = fork();
79.       if(pid < 0){
80.          fprintf(stderr, "Fork Failed\n");
81.          break;
82.       }
83.       else if(pid == 0){
84.          execl("./copy2", "./copy2", sourcePath, files[i],
   targetPath, NULL);
85.       }
86.       else{
87.          printf("Parent %d: Create Child Process %d\n", get
   pid(), pid);
88.       }
```

```
    }
    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;
 }
```

对于子进程：每个子进程只接收源目录、目标目录和一个子目录，只需要拷贝单个目录即可。

子进程程序代码如下所示：

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
#include <fcntl.h>
#define  BUFSIZE 1024

// 拷贝文件
int copyfile(const char* src, const char* dest) {
    //和之前的函数一样
}

//拷贝文件夹
int readFileList(char* sourcePath, char* targetPath){
   //和之前的函数一样
}

int main(int argc, char *argv[])
{
```

```
    char source[1000];
    char target[1000];
    //argv[1]传入原目录名称
    char *dir = argv[1];
    //源文件
    char *sourcePath = argv[2];
    //目标文件夹
    char *targetPath = argv[3];
    memset(target,'\0',sizeof(target));
    memset(source,'\0',sizeof(source));
    strcpy(target,targetPath);
    strcpy(source,dir);
    strcat(target, "/");
    strcat(source, "/");
    strcat(target, sourcePath);
    strcat(source, sourcePath);
    printf("%s to %s", source, target);
    int i=mkdir(target,S_IRWXU|S_IRGRP|S_IXGRP|S_IROTH|S_IX
OTH);
    readFileList(source, target);
    return 0;
}
```

4. 具体步骤

（1） 单进程拷贝目录的时间测量与结果验证：

```
wanliyang2013774@wanliyang2013774-virtual-machine: ~
wanliyang2013774@wanliyang2013774-virtual-machine:~$ time ./single ./linux-5.19.10 ./linux-5.19.10-backSingle
./linux-5.19.10 to ./linux-5.19.10-backSingle

real    3m10.664s
user    0m5.586s
sys     1m25.763s
wanliyang2013774@wanliyang2013774-virtual-machine:~$ diff -r ./linux-5.19.10 ./linux-5.19.10-backSingle
wanliyang2013774@wanliyang2013774-virtual-machine:~$
```

单进程拷贝目录用时 3m10s，也就是 190s。

验证得出**拷贝结果正确**。

（2） 多进程拷贝目录方法一的时间测量与结果验证：

```
wanliyang2013774@wanliyang2013774-virtual-machine:~$ time ./main ./linux-5.19.10 ./linux-5.19.10-backMulti
Parent 3701: begin
arguments list of ./main:
  0 ./main
  1 ./linux-5.19.10
  2 ./linux-5.19.10-backMulti
Parent 3701: Create Child Process 3709
Parent 3701: Create Child Process 3710
Parent 3701: Create Child Process 3711
Parent 3701: Create Child Process 3712
./linux-5.19.10/init to ./linux-5.19.10-backMulti/init
./linux-5.19.10/lib to ./linux-5.19.10-backMulti/lib
./linux-5.19.10/io_uring to ./linux-5.19.10-backMulti/io_uring
./linux-5.19.10/kernel to ./linux-5.19.10-backMulti/kernel
./linux-5.19.10/scripts to ./linux-5.19.10-backMulti/scripts
./linux-5.19.10/Documentation to ./linux-5.19.10-backMulti/Documentation
./linux-5.19.10/block to ./linux-5.19.10-backMulti/block
./linux-5.19.10/crypto to ./linux-5.19.10-backMulti/crypto
./linux-5.19.10/virt to ./linux-5.19.10-backMulti/virt
./linux-5.19.10/tools to ./linux-5.19.10-backMulti/tools
./linux-5.19.10/samples to ./linux-5.19.10-backMulti/samples
```

……

```
Parent 3701: Child 3710 exited with 0 code
./linux-5.19.10/certs to ./linux-5.19.10-backMulti/certs
./linux-5.19.10/arch to ./linux-5.19.10-backMulti/arch
./linux-5.19.10/fs to ./linux-5.19.10-backMulti/fs
./linux-5.19.10/mm to ./linux-5.19.10-backMulti/mm
  use_time: 47.722851
Parent 3701: Child 3709 exited with 0 code
Parent 3701: exited

real    2m9.506s
user    0m4.658s
sys     1m19.045s
wanliyang2013774@wanliyang2013774-virtual-machine:~$
```

```
wanliyang2013774@wanliyang2013774-virtual-machine:~$ diff -r ./linux-5.19.10 ./linux-5.19.10-backMulti
wanliyang2013774@wanliyang2013774-virtual-machine:~$
```

多进程拷贝目录用时 2m9s，也就是 129s。

验证得出拷贝结果正确。

加速比分析：本方法总共启动四个子进程，加速比约为 1.5，效果不甚理想。但是得到这样不理想的加速比情有可原。首先由于采用多进程拷贝目录，并不像单进程只需要进行一次遍历即可，还需要有更多的**准备工作**（记录和分配子目录），同时调用进程的过程也更耗费时间。虽然子目录的数量是等分的，但是每个子目录所含的文件数目并不一致，这也导致了**负载不均衡**，任务量大的进程拖慢了整体的运行时间。综上，加速比不理想是正常的。

（3） 多进程拷贝目录方法二的时间测量与结果验证：

```
wanliyang2013774@wanliyang2013774-virtual-machine:~$ time ./main2 ./linux-5.19.10 ./linux-5.19.10-backMulti2
Parent 4890: begin
arguments list of ./main2:
  0 ./main2
  1 ./linux-5.19.10
  2 ./linux-5.19.10-backMulti2
Parent 4890: Create Child Process 4904
Parent 4890: Create Child Process 4905
Parent 4890: Create Child Process 4906
Parent 4890: Create Child Process 4907
Parent 4890: Create Child Process 4908
Parent 4890: Create Child Process 4909
Parent 4890: Create Child Process 4910
Parent 4890: Create Child Process 4911
Parent 4890: Create Child Process 4912
Parent 4890: Create Child Process 4913
Parent 4890: Create Child Process 4914
Parent 4890: Create Child Process 4915
Parent 4890: Create Child Process 4916
Parent 4890: Create Child Process 4917
Parent 4890: Create Child Process 4918
Parent 4890: Create Child Process 4919
Parent 4890: Create Child Process 4920
Parent 4890: Create Child Process 4921
Parent 4890: Create Child Process 4922
Parent 4890: Create Child Process 4923
Parent 4890: Create Child Process 4924
Parent 4890: Create Child Process 4925
Parent 4890: Create Child Process 4926
Parent 4890: Create Child Process 4927
./linux-5.19.10/.tmp_6112 to ./linux-5.19.10-backMulti2/.tmp_6112Parent 4890: Child 4909 exited with 0 code
./linux-5.19.10/LICENSES to ./linux-5.19.10-backMulti2/LICENSESParent 4890: Child 4922 exited with 0 code
./linux-5.19.10/usr to ./linux-5.19.10-backMulti2/usrParent 4890: Child 4920 exited with 0 code
./linux-5.19.10/virt to ./linux-5.19.10-backMulti2/virtParent 4890: Child 4912 exited with 0 code
./linux-5.19.10/io_uring to ./linux-5.19.10-backMulti2/io_uringParent 4890: Child 4923 exited with 0 code
```

……

```
./linux-5.19.10/Documentation to ./linux-5.19.10-backMulti2/DocumentationParent 4890: Child 4918 exited with 0 co
./linux-5.19.10/arch to ./linux-5.19.10-backMulti2/archParent 4890: Child 4907 exited with 0 code
./linux-5.19.10/drivers to ./linux-5.19.10-backMulti2/driversParent 4890: Child 4905 exited with 0 code
Parent 4890: exited

real    2m14.955s
user    0m7.282s
sys     2m16.573s
wanliyang2013774@wanliyang2013774-virtual-machine:~$ diff -r ./linux-5.19.10 ./linux-5.19.10-backMulti2
wanliyang2013774@wanliyang2013774-virtual-machine:~$
```

可以发现**该方法用时 135s**，加速比只有 1.4，启用的进程数更多，效果反而更不好。主要是因为**虚拟机的 CPU 核数较少**，支持同步进行的进程数有限，进程开辟多了，同样需要轮流使用 CPU，因此加速比并不理想。

验证结果表明该方法同样拷贝成功。

5. 总结心得

（1） 如今我们使用的电脑基本都采用多核的 CPU，因此对于多进程、多线程的支持能力比较强大。在做比较复杂的工作时尝试采用多进程和多线程的方式，使得总任务量分摊到不同进程上，多进程并发执行，能很大程度上提

升程序的性能。

（2） 本次实验的目标：拷贝目录及所有子目录是非常适合并行优化的，把不同的子目录分配给不同的子进程，非常符合并行编程的思想。

（3） 但是多进程编程对于程序员提出了更多的要求，因为需要启用和调度不同的进程，因此在程序编码上会比单进程更为复杂。

（4） 多进程的加速比并不会很理想地接近进程数目。理想状态下，开启 n 个进程分担总任务量，希望达到 n 倍的加速比。但是这并不现实，因为多进程编程需要做更多的准备与进程间通讯工作以及负载并不一定均衡等因素，加速比会受到较大的影响。

（5） 通过本次实验对于 UNIX 系统的进程创建过程（fork、execl）有了更为深刻的理解。

6. 参考资料

实验指导书《Lab06CopyDirWithMultiProcesses_README.pdf》