Project 1　　Report

姓名：吴本利　　　学号：522031910763

Contents

1　Copy

1.1　Description

Copy file from one to another using three different ways.

```
./Copy <InputFile> <OutputFile>
```

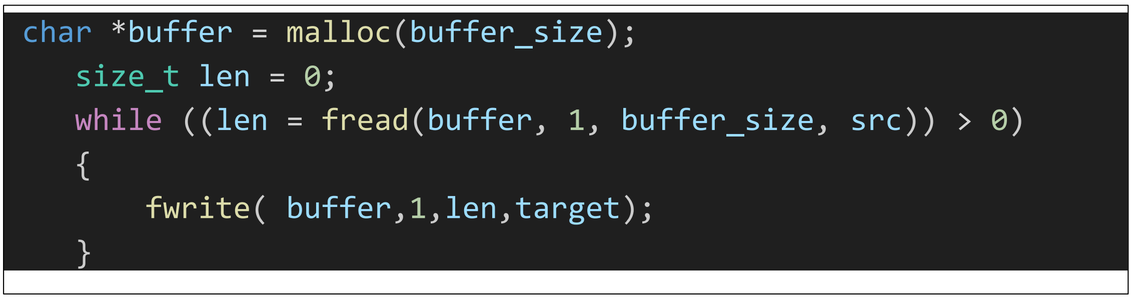
This command will copy the InputFile to OutputFile .The buffer size is set by the program.

1.2　implementation

1.2.1　MyCopy

It is a simple program to copy the file from inputFile to OutFile.

Just use a buffer to read and writer.



```
char *buffer = malloc(buffer_size);
   size_t len = 0;
   while ((len = fread(buffer, 1, buffer_size, src)) > 0)
   {
       fwrite( buffer,1,len,target);
   }
```

1.2.2 ForkCopy

It just fork a process to do the copy. And it will just call the “MyCopy” to do the work.

1. **Fork**

   It use the fork() to do the thing. And there mey be some wrong. So we should to handle the error.

```
pid_t pid;
    pid = fork();
    if (pid < 0)
    {
        printf("Error: Failed to fork.\n");
        exit(-1);
    }
```

2. **call MyCopy**

   Fork a process to do the thing And the main process should wait for it.

```
else if (pid == 0)
    {
        execl("./MyCopy", argv[0],argv[1],argv[2],NULL) ;
    }
    // 父进程
    else if (pid > 0)
    {
        wait(NULL);
        return 0;
    }
```

1.2.3 PipeCopy

1. **Create pipe**

```
int mypipe[2];
    if (pipe(mypipe))
    {
        fprintf(stderr, "Pipe failed.\n");
        return -1;
    }
```

3. **Fork a process to read from resource file and write to the pipe**

```
else if(pid==0){
        FILE *src=fopen(src_path,"r");
        if(src==NULL){
            printf("Error: Could not open file %s\n",src_path);
            exit(-1);
        }
        close(mypipe[0]); //要向 pipe 中写
        size_t len=0;
        while ((len = fread(rbuffer, 1, buffer_size, src)) > 0)
        {
            write(mypipe[1], rbuffer, len);//向 pipe 中写
        }
        close(mypipe[1]);
        fclose(src);
        free(rbuffer);
    }
```

4. **The main process to read from the pipe and write to the target file**

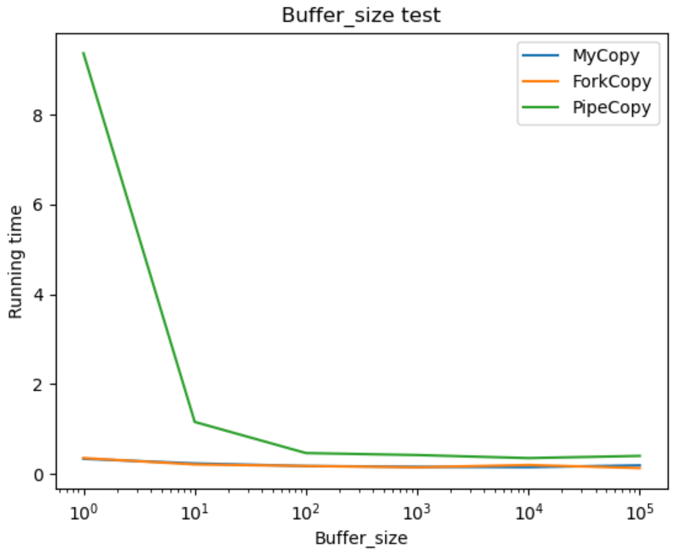
```
else if(pid>0){
        FILE *target = fopen(target_path, "w");
        if (target == NULL)
        {
            printf("Error: Could not open file %s\n", 
target_path);
            exit(-1);
        }
        close(mypipe[1]);
        size_t len = 0;
        while((len=read(mypipe[0],wbuffer,buffer_size))>0){
            fwrite(wbuffer, 1, len, target);
        }
        close(mypipe[0]);
        fclose(target);
        free(wbuffer);
}
```

5. **Time**

```
    clock_t start, end;
    double elapsed;
    start = clock();
     end = clock();
        elapsed = ((double)(end - start)) /CLOCKS_PER_SEC * 1000;
        printf("Time used: %f millisecond\n", elapsed);
```

1.3 Test and analysis

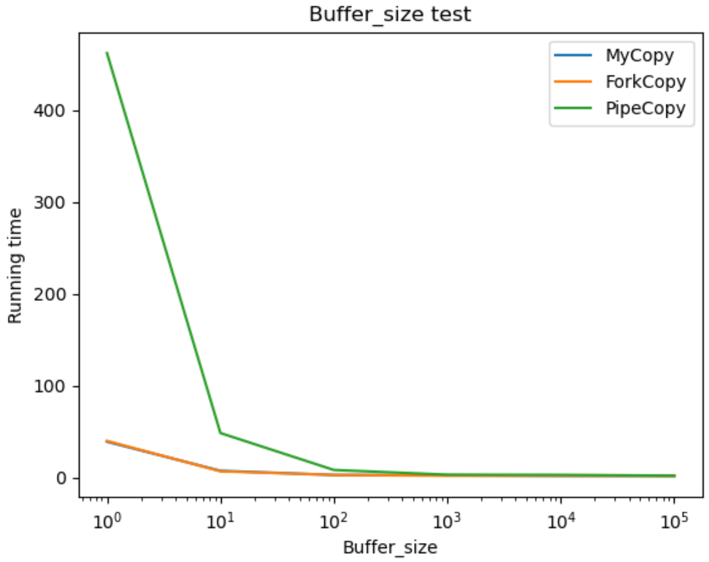
1. 10 KB test file(type: .text)



We can see the Running time is reduced quickly in the beginning for the PipeCopy. But when Buffer_size > 100, the rate is slow.

For MyCopy and ForkCopy the rate is always slow.

2. 612 KB test file(type: .jpg)



We can see the trend is just like the figure above.

2 shell

2.1 Description

A shell-like server, which can be connected by many clients, handling the commands with arguments and the commands connected by pipes.

```
./shell <Port>
```

2.2 Implementatio

2.2.1 Multi-clients

For every client, we can fork a process to do service just for the client. And we can use “while” to connect with different clients constantly.

```
 while (1)
    {
        socklen_t client_addr_len = sizeof(client_addr);
        client_sock = accept(serv_sock, (struct sockaddr
*)&client_addr, &client_addr_len);
        pid_t pid;
        pid = fork();
        else if (pid == 0)
        {……}
}
```

2.2.2 Parse commands

```
int parseLine(char *line, char *command_array[])
{
    char *p;
    int count = 0;
    p = strtok(line, " ");
    while (p)
    {
        command_array[count] = p;
        count++;
        p = strtok(NULL, " ");
    }
    //标记字符串数组结束
    command_array[count] = NULL;
    return count;
}
```

2.2.3 Execute commands

cd

```
else if (strcmp(command_array[0], cmd_cd) == 0)
                {
                    int res = chdir(command_array[1]);
                    if (res != 0)
                    {
                       strcpy(temp,"Error: can't change
directory\n");
                       write(client_sock, temp, strlen(temp));
                       continue;
```

```
          }
        }
```

Exit

```
else if (strcmp(command_array[0], cmd_exit) == 0)
          break;
```

No pipe command

```
dup2(client_sock, STDOUT_FILENO);
      if (execvp(command_array[0], command_array) == -1)
      {
        printf("Error: running the command ");
        for (int i = 0; i < n; i++)
          printf("%s ", command_array[i]);
        printf("error\n");
      }
      exit(-1);
```

Command with pipes

We can excute the command before '|' , then do the command after '|'. We can use recursion (递归) to do the command as before.

```
//执行'|'前的指令
    pid_t pid1;
    pid1 = fork();
    if(pid1 == 0)
    {
      close(my_pipes[0]);
      dup2(my_pipes[1], STDOUT_FILENO);
      close(my_pipes[1]);
      char *command_arrayA[100];
      for (int i = 0; i < pipe_id; i++)
        command_arrayA[i] = command_array[i];
      command_arrayA[pipe_id] = NULL;
      if (execvp(command_arrayA[0], command_arrayA) == -1)
      {
        printf("Error: running the command ");
        for (int i = 0; i < n; i++)
          printf("%s ", command_array[i]);
        printf("error\n");
      }
      exit(-1);
    }
    else
    {
      // 等子进程 1 运行完后
      wait(NULL);
      pid_t pid2;
      pid2 = fork();
      if (pid2 < 0)
      {
        strcpy(temp, "Error: Failed to execute the 
commmand.\n");
        write(client_sock, temp, strlen(temp));
      }
      else if (pid2 == 0)
      {
        close(my_pipes[1]);
        close(0);
        dup2(my_pipes[0], STDIN_FILENO);
        close(my_pipes[0]);
        char *command_arrayB[100];
        for (int i = pipe_id + 1; i < n; i++)
          command_arrayB[i-pipe_id-1] = 
command_array[i];
        command_arrayB[n-pipe_id-1] = NULL;
        execute_command(command_arrayB, n - pipe_id - 1, 
client_sock);
        exit(-1);
      }
      else{
        close(my_pipes[0]);
        close(my_pipes[1]);
        wait(NULL);
        return;
      }
```

3.Sort

3.1 Description

Do mergesort by single thread and multi threads.

```
./MergesortSingle

./MergesortMulti <max_thread_num>
```

3.2 Implementation

3.2.1 MergesortSingle

Just use Merge algorithm.

```
void MergeSort(int *array, int left, int right);
void Merge(int *array, int left, int mid, int right);
```

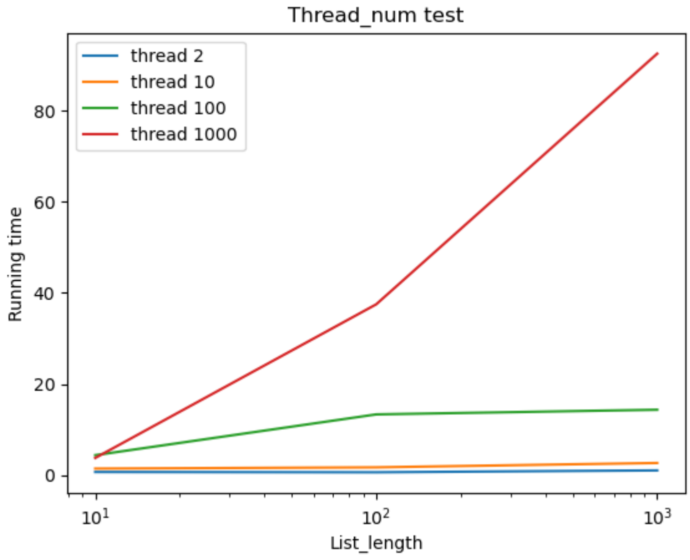
3.2.2 Multi-thread Mergesort

Just create thread in the MergeSort function.

```
        pthread_t t1, t2;
         used_thread += 2;
         int rc1 = pthread_create(&t1, NULL, MergeSort, 
&my_arguments[0]);
         int rc2 = pthread_create(&t2, NULL, MergeSort, 
&my_arguments[1]);
         int rc3 = pthread_join(t1, NULL);
         if (rc3)
         {…}
         used_thread--;
         int rc4 = pthread_join(t2, NULL);
         if (rc4)
         {…}
         used_thread--;
         Merge(array, left, mid, right);
```

3.3 Test and analysis

Because there are some wrong in the input .So I just test 3 different length.


Thread_num test
thread 2
thread 10
thread 100
thread 1000
Running time
10^1
10^2
10^3
List_length