操作系统课程设计-Project1

周旭东 521021910829

2023.3.26

摘要

在操作系统课程设计 Project1 中，通过三个任务 Copy，Shell 和 Matrix，分别学习和实践了 fork 子进程，用 pipe 建立联系和传输数据；用 socket 搭建服务器主机并用 telnet 连接，用 execvp 执行命令，dup2 的 IO 重定向功能等；用 pthread 进行多线程操作。并加深了对 c 语言下的字符串处理，文件读取写入，调试程序，指针和动态内存管理的理解和掌握。

1 Task1: Copy File

使用系统调用 fork 两个进程来复制文件。这两个进程使用 pipe 系统调用进行通信。给定不同大小的 buffer 并分析性能。

1.1 实现概览

1.1.1 文件读写

使用函数 fopen(), fclose() 打开和关闭文件，使用参数 argv[] 获取所需文件名，在中部使用带 buffer 的 fread()，fwrite() 实现读写。

1.1.2 fork 进程以实现读写

在子进程中读取文件内容并写入管道，父进程中读取管道中内容，并写文件。管道的使用二者同步。

1.1.3 进程计时

使用 clock() 函数在读取文件打开前开始计时，到写入文件关闭后停止计时。

1.2 性能分析

使用程序对给定文件进行文件读写测试，为利于观察和得出规律，取定 BufferSize 为 2 的指数。即 1，2，... 512，1024 并以指数为 X 轴，使用 gnuplot 绘制折线图 (1)

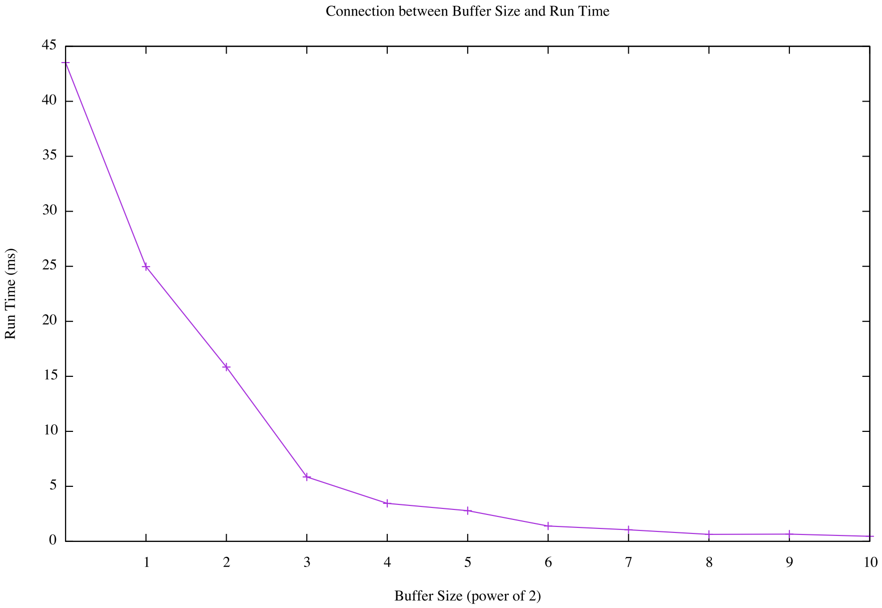


图 1: 不同 Buffer Size 的 Copy 性能分析

从图像中，我们可以发现 BufferSize 对时间影响较大。当 BufferSize < 8 时，翻倍的 BufferSize 带来近乎两倍（甚至大于）的性能提升，但是随着继续翻倍，提升的程度开始变小，以渐缓的趋势趋近于某一近 0 常数。这可能是由于当 BufferSize 足够大后，单次读写的操作逐渐占据影响性能的主要地位，使得 BufferSize 影响的占比降低，性能加速比变小。

这样的结果提示我们在优化程序的时候不能一味增加某一部分的性能，而应当以相互制约的角度进行分析和调试，以做出更具有“性价比”的选择

2 Task2: Shell

写一个服务器 Shell，处理带有管道的 linux 命令行。客户端用 Internet 套接字实现和主机的连接。并支持一个以上客户端的连接，工作，退出。

2.1 实现概览

2.1.1 多客户端支持

令服务器在一个无限循环 while(1) 中运行，每次用 fork 生产子进程，在父进程中接受新的客户端，在子进程中处理每个客户端的需求

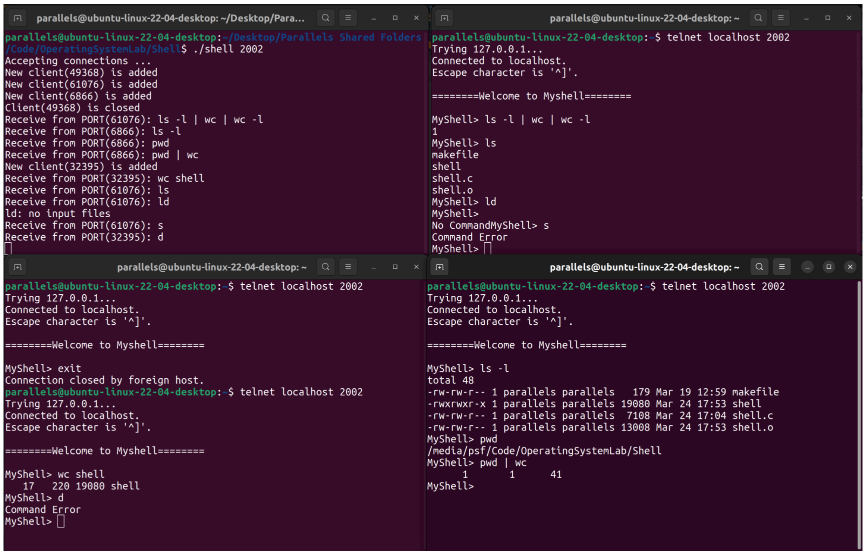
```
    do{
    clilen = sizeof(cli_addr);
    newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
    pid_t ForkPID;
        ForkPID = fork();
        switch (ForkPID) {
        case -1:
        //error
    case 0:
            //handle client
            default:
        close(newsockfd);
    }
}while(1);
```

2.1.2 pipe 指令支持

令执行函数在一个 n(n-1 个” | ” 将指令分为 n 个) 次循环中。每次 fork 出子进程，处理指令，其父进程进行对输入输出的 backup 备份。对第一个指令，将输出重定向到 pipe 中，对其余指令，重定向其输入输出。在最后一个指令的父进程中对客户端输出结果。

2.2 成果展示

该程序能够正确运行“ls -l”，包含多个管道的”ls -l | wc | wc -l”，接受“exit”后退出客户端，接受错误指令后给出反馈。并且每条指令被服务器接受后打印相应信息，如图（2）


parallels@ubuntu-linux-22-04-desktop: ~/Desktop/Para...
parallels@ubuntu-linux-22-04-desktop: ~
parallels@ubuntu-linux-22-04-desktop: ~
parallels@ubuntu-linux-22-04-desktop: ~

```
parallels@ubuntu-linux-22-04-desktop:~/Desktop/Parallels Shared Folders
/Code/OperatingSystemLab/Shell$ ./shell 2002
Accepting connections ...
New client(49368) is added
New client(61076) is added
New client(6866) is added
Client(49368) is closed
Receive from PORT(61076): ls -l | wc | wc -l
Receive from PORT(6866): ls -l
Receive from PORT(6866): pwd
Receive from PORT(6866): pwd | wc
New client(32395) is added
Receive from PORT(32395): wc shell
Receive from PORT(61076): ls
Receive from PORT(61076): ld
ld: no input files
Receive from PORT(61076): s
Receive from PORT(32395): d
```

```
parallels@ubuntu-linux-22-04-desktop:~$ telnet localhost 2002
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

========Welcome to Myshell========

MyShell> ls -l | wc | wc -l
1
MyShell> ls
makefile
shell
shell.c
shell.o
MyShell> ld
MyShell>
No CommandMyShell> s
Command Error
MyShell> 
```

```
parallels@ubuntu-linux-22-04-desktop:~$ telnet localhost 2002
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

========Welcome to Myshell========

MyShell> exit
Connection closed by foreign host.
parallels@ubuntu-linux-22-04-desktop:~$ telnet localhost 2002
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

========Welcome to Myshell========

MyShell> wc shell
   17   220 19080 shell
MyShell> d
Command Error
MyShell> 
```

```
parallels@ubuntu-linux-22-04-desktop:~$ telnet localhost 2002
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

========Welcome to Myshell========

MyShell> ls -l
total 48
-rw-rw-r-- 1 parallels parallels   179 Mar 19 12:59 makefile
-rwxrwxr-x 1 parallels parallels 19080 Mar 24 17:53 shell
-rw-rw-r-- 1 parallels parallels  7108 Mar 24 17:04 shell.c
-rw-rw-r-- 1 parallels parallels 13008 Mar 24 17:53 shell.o
MyShell> pwd
/media/psf/Code/OperatingSystemLab/Shell
MyShell> pwd | wc
      1       1      41
MyShell>
```

图 2: 多用户 Shell 服务器实现

2.3 问题解决和思考

- strtok 函数在做嵌套使用时发生错误。在解析包含"|"命令时，我的思路是先用 strtok 对指令进行划分（两个 | 之间是一个指令），然后对空格分割，得到指令的二维字符串。但是实际实现时，strtok 似乎使用了一个静态变量进行分隔符的村粗，无法在确定分割符后临时变更。并且分割符只能为一个字符。针对这个问题，我复制了一份"mystrtok"，并用"ltrim"函数对字符串首保留的空格进行消除。

- 在用循环进行指令的操作时，发现 pipe 无法将前一个指令的输出传递至后面的指令。原因是每次循环后管道将被关闭，需要将管道备份传入到后方指令以实现管道的正确运行。

3 Task3: Matrix Multiplication using Pthread

3.1 实现概览

3.1.1 分块矩阵乘法

首先限定线程数为不大于矩阵大小的 2 的幂次。用创建 pthread 的 tid 确定分块矩阵的位置（分离第一个矩阵的行和第二个矩阵的列），对该部分（size/thread）的矩阵进行乘法操作，算法如下，复杂度为 $O(n^3)$

```
row_start = tid * portion_size;
row_end = (tid+1) * portion_size;

for (i = row_start; i < row_end; ++i) { // hold row index of 'matrix1'
    for (j = 0; j < size; ++j) { // hold column index of 'matrix2'
        sum = 0; // hold value of a cell
        for (k = 0; k < size; ++k) {
            sum += matrix1[i][k] * matrix2[k][j];
        }
        res_mat[i][j] = sum;
    }
}
```

3.1.2 多线程的创建和运行

将两部分分离，以方便进行计时和性能分析。

```
//创建线程
for (int i = 0; i < num_threads; ++i ) {
    int *tid;
    tid = (int *) malloc( sizeof(int) );
    *tid = i;
    pthread_create( &threads[i], NULL, matrix_multiply, (void *)tid );
}
//添加开始时钟
...
//执行多线程
for (int i = 0; i < num_threads; ++i ) {
    pthread_join( threads[i], NULL );
}
```

3.2 性能分析

线程数量为 1，2，4 ... 256，512。矩阵大小为 1，2，4 ... 256，512。同时矩阵大小大于等于线程数量，得出每个组合的运行时间如表（1）所示

Treads/MatrixSize	1	2	4	8	16	32	64	128	256	512
1	0.117	0.119	0.126	0.118	0.112	0.317	1.656	14.141	84.823	490.963
2	0	0.118	0.157	0.149	0.159	0.382	1.711	15.857	96.985	519.587
4	0	0	0.034	0.046	0.122	0.413	1.706	15.857	96.985	519.587
8	0	0	0	0.07	0.059	0.088	2.391	17.816	94.312	511.465
16	0	0	0	0	0.121	0.13	3.083	18.246	95.329	517.567
32	0	0	0	0	0	70.158	0.914	2.318	77.399	515.174
64	0	0	0	0	0	0	0.817	17.454	95.305	511.214
128	0	0	0	0	0	0	0	0.779	31.089	513.524
256	0	0	0	0	0	0	0	0	5.812	362.315
512	0	0	0	0	0	0	0	0	0	63.072

表 1: 矩阵、线程量级对时间的影响

根据表格可以画出 3 维散点图（3）

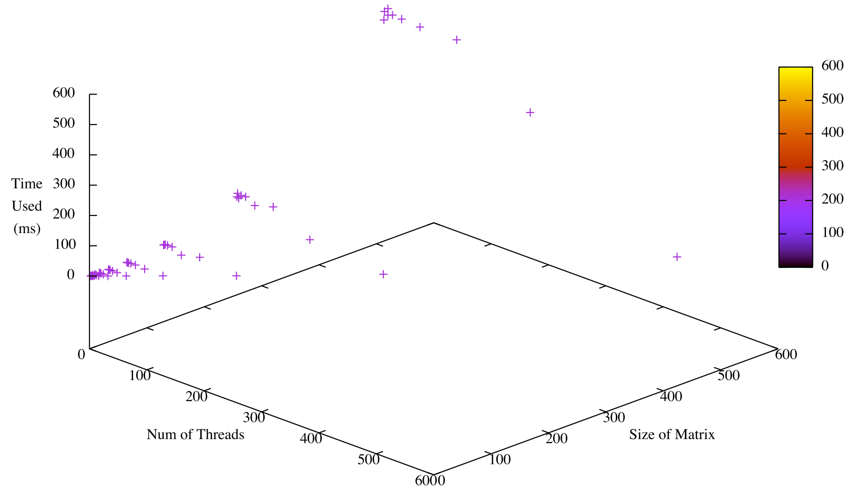


图 3: Treads 数量对不同量级矩阵乘法的影响-3D 散点

为了更直观地研究不同量级矩阵下同一 Treads 运行时间趋势，做出二维多组折线图（4）

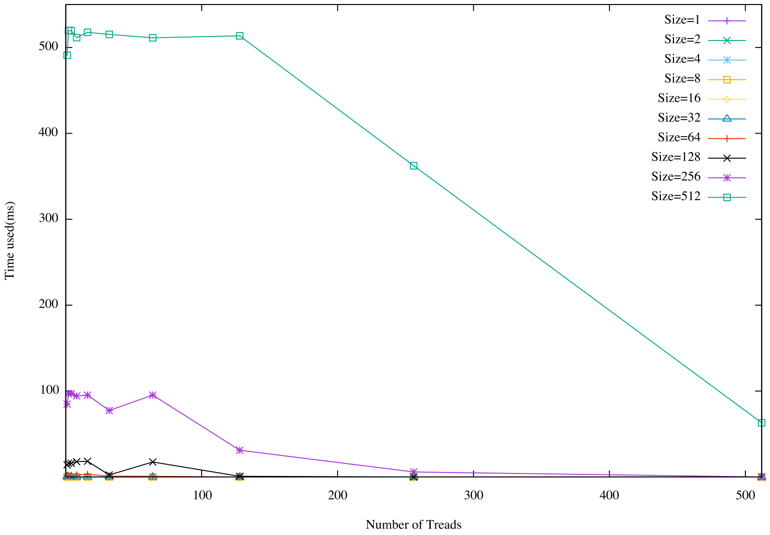


图 4: 不同量级矩阵下同一 Treads 运行时间趋势

发现该图由于比例原因，难以展示矩阵 Size = 1-128 的情况，且趋势并不明晰。我将所有的数据取 log2（Treads 个数和运行时间的数据）绘制出图（5）

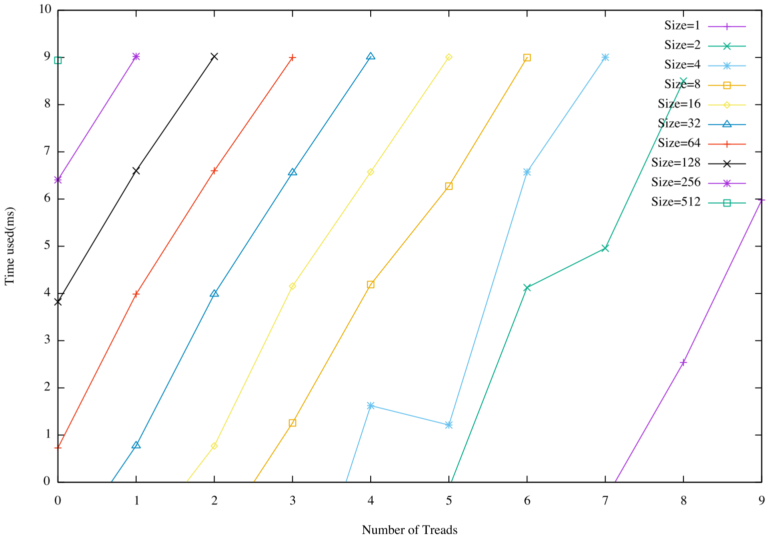


图 5: 不同量级矩阵下同一 Treads 运行时间趋势-取 log2

从图（5）呈现的结果来看，除去由于随机生成产生的数据抖动，可以近似认为 Treads 的个数和运行时间在同一大小的矩阵下进行乘法运算具有一定程度的线性关系。

3.3 问题解决和思考

在进行不同线程的的时间统计时，我先将时间的起始位置放在创建线程之前。发现多线程时间超过单线程（对于 sample-data.in 给出的矩阵，单线程在 17ms 左右，多线程却达到 22 ms，并且增加线程数量，时间变化不大）。然而将时间记录限定在线程创建之后的运行阶段，则能得到理想的结果。于是我认为创建新的线程耗时情况不容乐观，在数据量不大的情况下使用单线程是一种优良的选择。

4 完整代码

4.1 Copy File

```
        #include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<stdlib.h>
#include<time.h>
int main(int argc, char* argv[]){
        //open file
        FILE *src;
        src = fopen(argv[1],"r");
        if(src == NULL){
                printf("Error!");
                fclose(src);
                exit(-1);
        }
        FILE *dest;
        dest = fopen(argv[2],"w+");
        if(dest == NULL){
                printf("Error!");
                fclose(src);
                exit(-1);
        }

        int length = atoi(argv[3]);
        char buffer[length];

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

        //start timer
        clock_t start, end;
        double elapsed;
        start = clock();

        //fork process
        pid_t ForkPID;
        ForkPID = fork();
        switch (ForkPID) {
        case -1:
                printf("Error: Failed to fork.\n"); break;
        // 0, this is the child process
        //read process
        case 0:
                close(mypipe[0]);
                do {
                        fread(buffer,length,1,src);
                        if( feof(src) ) {
                                break ;
                        }
                        write(mypipe[1],buffer,sizeof(buffer));
                } while(1);
                close(mypipe[1]);
                fclose(src);
                printf("Read file end.\n");
        break;
        // > 0, parent process and the PID is the child's PID
        //write process
        default:
                close(mypipe[1]);
                while (read(mypipe[0],buffer,sizeof(buffer)) > 0) {
                        fwrite(buffer,length,1, dest);
                }
                close(mypipe[0]);
                fclose(dest);
                printf("Write file end.\n");

                //end timer
                end = clock();
                elapsed = ((double) (end - start)) / CLOCKS_PER_SEC * 1000;
                printf("Time used: %f millisecond.\n", elapsed);
        }
        return 0;
}
```

4.2 Shell

```
        #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/wait.h>
#include <ctype.h>

//define valuable
int sockfd, newsockfd, portno;
socklen_t clilen;
char buffer[256];
char *line = NULL;
char buf[1024];
struct sockaddr_in serv_addr, cli_addr;
int n;
int* com_num;

//get valid string in buffer
char* getValidString(const char* buffer) {
    int len = strlen(buffer);
    int start = 0, end = len - 1;
    //find the first position with char
    while (start < len && isspace(buffer[start])) {
        start++;
    }
    //find the last position with char
    while (end >= start && isspace(buffer[end])) {
        end--;
    }
    //copy valid string
    int strLen = end - start + 1;
    if (strLen <= 0) {
        return NULL;
    }
    char* validStr = (char*)malloc(sizeof(char) * (strLen + 1));
    if (validStr == NULL) {
        return NULL;
    }
    memcpy(validStr, buffer + start, strLen);
    validStr[strLen] = '\0';
    return validStr;
}

void error(const char *msg)
{
    perror(msg);
    exit(1);
}

//set a new strtok to prevent nested strtok error
char * myStrtok;
char * mystrtok( char * s,const char * ct)
{

    char *sbegin, *send;
    sbegin  = s ? s : myStrtok;//
    if (!sbegin) {
    return NULL;
    }
    sbegin += strspn(sbegin,ct);
    if (*sbegin == '\0'){
    myStrtok = NULL;
    return (NULL);
    }
    send = strpbrk(sbegin, ct);
    if(send && *send != '\0')
    *send++ = '\0';
    myStrtok = send;
    return (sbegin);
}

int parseLine(char *line, char **command_array) {
    char *p;
    int count = 0;
```

```c
    p = strtok(line, " ");
    while (p && strcmp(p,"|")!= 0){
        command_array[count] = p;
        count++;
        p = strtok(NULL," ");
    }
    return count;
}

char* substr(const char *src, int m, int n)
{
    int len = n - m;
    char *dest = (char*)malloc(sizeof(char) * (len + 1));
    for (int i = m; i < n && (*(src + i) != '\0'); i++)
    {
        *dest = *(src + i);
        dest++;
    }
    *dest = '\0';
    return dest - len;
}

//count number of commands devided by "|"
int countPipes(char *line){
    if (line == "") return 0;
    int count = 1;
    for(int i = 0; line[i] != '\0'; i++){
        if(line[i] == '|') count ++;
    }
    return count;
}
//remove " " at the begin of strings
char *ltrim(char *str)
{
    if (str == NULL || *str == '\0')
    {
        return str;
    }
    int len = 0;
    char *p = str;
    while (*p != '\0' && isspace(*p))
    {
        ++p; ++len;
    }
    memmove(str, p, strlen(str) - len + 1);
    return str;
}
//cammand function
void command(int num, char ***cmd){
        int fd[2];
        pid_t pid;
        int backup = 0;

        for(int i = 0; i < num; i++) {
                if(pipe(fd) < 0)error("Pipe Error");
                if ((pid = fork()) == -1) {
                        perror("fork");
                        exit(1);
                }
                else if (pid == 0) {
                        dup2(backup, STDIN_FILENO);
                        if (*cmd != NULL) {
                close(STDOUT_FILENO);
```

```
                                dup2(fd[1], STDOUT_FILENO);
                        }
                        close(fd[0]);
                        if(execvp((*cmd)[0], *cmd) == -1){
                write(newsockfd, "Command Error\n", 15);
                _exit(1);
            };
                        exit(1);
                }
                else {
            if(i == num - 1){
                wait(NULL);
                close(fd[1]);
                backup = fd[0];
                while(read(fd[0], buf, sizeof(buf)) > 0){}
                n = write(newsockfd, buf, sizeof(buf));
                if (n < 0) error("Error writing to socket");
                close(fd[0]);
                bzero(buf,1024);
                cmd++;
            }
                        else{
                wait(NULL);
                close(fd[1]);
                backup = fd[0];
                cmd++;
            }
                }
        }
}

int main(int argc, char *argv[])
{
    if (argc < 2) {
        fprintf(stderr,"Error, no port provided\n");
        exit(1);
    }
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
    error("Error opening socket");
    printf("Accepting connections ...\n");

    bzero((char *) &serv_addr, sizeof(serv_addr));
    portno = atoi(argv[1]);
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portno);

    if (bind(sockfd, (struct sockaddr *) &serv_addr,sizeof(serv_addr)) < 0)
        error("Error on binding");
    listen(sockfd,5);

    //serve the client in a loop
    do{
        clilen = sizeof(cli_addr);
        newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
        if (newsockfd < 0)
            error("Error on accept");
        printf("New client(%d) is added\n", cli_addr.sin_port);

        pid_t ForkPID;
            ForkPID = fork();
            switch (ForkPID) {
```

```
        case -1:
            printf("Error: Failed to fork.\n");
            break;

        //child process deal with clients
        case 0:
            close(sockfd);
            write(newsockfd,"\n========Welcome to Myshell========\n\n",38);
            do{
                //clean the buffer every loop
                bzero(buffer,256);
                bzero(buf,1024);

                n = write(newsockfd,"MyShell> ",10);
                if (n < 0) error("Error writing to socket");
                n = read(newsockfd, buffer, 255);
                if (n < 0) error("Error reading from socket");

                line = getValidString(buffer);
                //no command
                if(line == NULL){
                    write(newsockfd, "No Command\n", 10);
                    continue;
                }
                //when meet "exit", quit client
                if(strcmp(line, "exit") == 0){
                    printf("Client(%d) is closed\n", cli_addr.sin_port);
                    close(newsockfd);
                    exit(1);
                    return 1;
                }

                printf("Receive from PORT(%d): %s", cli_addr.sin_port, buffer);

                int num_of_pipes = countPipes(line);
                char *cmd;
                char *cmd_mod;
                char ***cmd_array = malloc((num_of_pipes+1)*sizeof(char**));
                for(int i = 0; i < num_of_pipes; i++){
                    cmd_array[i] = malloc(3*sizeof(char*));
                }
                //seperate commands
                char *t = mystrtok(line, "|");
                for(int i = 0; i < num_of_pipes; i++){
                    cmd = t;
                    cmd_mod = ltrim(cmd);
                    parseLine(cmd_mod, cmd_array[i]);
                    t = mystrtok(NULL,"|");
                }
                command(num_of_pipes, cmd_array);

                //free the space
                for(int i = 0; i < num_of_pipes; i++){
                    free(cmd_array[i]);
                }
                free(cmd_array);

            }while(1);
            break;

        //parent process accept new client
        default:
            close(newsockfd);
        }
        free(line);
     }while(1);

     close(sockfd);
     return 0;
}
```

4.3 Single Thread Matrix

```
        #include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <pthread.h>

int **matrix1;
int **matrix2;
int **res_mat;

struct mat{
int row1;//row of mat1
int column1;//column of mat1
int row2;//row of mat2
int column2;//column of mat2
};

//run in pthread
void *matrix_multiply(void *args){
   struct mat *a = args;
   for (int i = 0; i < a->row1; i++){
                for (int j = 0; j < a->column2; j++){
                        res_mat[i][j] = 0;
                        for (int k = 0; k < a->column1; k++)
                                res_mat[i][j] += matrix1[i][k] *matrix2[k][j];
                }
        }
   pthread_exit(NULL);
}

int main(int argc, char *argv[]) {
   //set and start timer
        clock_t start, end;
        double elapsed;
        start = clock();

   //open read file1
   FILE *src1;
        src1 = fopen("data.in","r");
        if(src1 == NULL){
                printf("Error!");
                fclose(src1);
                exit(-1);
        }
   FILE *src2;
        src2 = fopen("data.in","r");
        if(src2 == NULL){
                printf("Error!");
                fclose(src2);
                exit(-1);
        }

   struct mat *data = (struct mat *) malloc(sizeof(struct mat));
   //read file and parse the matrix
   fscanf(src1, "%d", &data->row1);
   data->column1 = data->row1;
   matrix1 = malloc(data->row1 * sizeof(int*));
   for(int i = 0; i < data->row1; ++i){
      matrix1[i] = malloc(data->column1 * sizeof(int));
      for(int j = 0; j < data->column1; ++j)
         fscanf(src1, "%d", &matrix1[i][j]);
   }
   fclose(src1);
   //second matrix
   fscanf(src2, "%d", &data->row2);
   data->column2 = data->row2;
   matrix2 = malloc(data->row2 * sizeof(int*));
   for(int i = 0; i < data->row2; ++i){
      matrix2[i] = malloc(data->column2 * sizeof(int));
      for(int j = 0; j < data->column2; ++j)
         fscanf(src2, "%d", &matrix2[i][j]);
   }
   fclose(src2);

   res_mat = malloc(data->row1 * sizeof(int*));
   for(int i = 0; i < data->row1; ++i){
      res_mat[i] = malloc(data->row1 * sizeof(int));
   }

   pthread_t tid;
   pthread_attr_t attr;
   pthread_attr_init(&attr);

   int rc = pthread_create(&tid,&attr,matrix_multiply,data);
   if (rc) {
      printf("ERROR; return code from pthread_create(tid) is %d\n", rc);
      exit(-1);
   }
   pthread_join(tid, NULL);

   // //Print out the resulting matrix
   // for(int i = 0; i < data->row1; i++) {
   //    for(int j = 0; j < data->column2; j++) {
   //       printf("%d ", res_mat[i][j]);
   //    }
   //    printf("\n");
   // }

   FILE *dest;
        dest = fopen("data.out","w+");
        if(dest == NULL){
                printf("Error!");
                fclose(dest);
                exit(-1);
        }
   for(int i = 0; i < data->row1; i++) {
      for(int j = 0; j < data->column2; j++) {
         fprintf(dest, "%d ", res_mat[i][j]);
      }
      fprintf(dest, "\n");
   }
   fclose(dest);

   //end timer
```

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

    return 0;
}
```

4.4 Multiple Threads Matrix

```
        #include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <pthread.h>

int **matrix1;
int **matrix2;
int **res_mat;

int size, num_threads;

struct mat{
int row1;//row of mat1
int column1;//column of mat1
int row2;//row of mat2
int column2;//column of mat2
};

//run in pthread
void *matrix_multiply(void *args)
{
    int i, j, k, tid, portion_size, row_start, row_end;
    double sum;

    tid = *(int *)(args); // get the thread ID assigned sequentially.
    portion_size = size / num_threads;
    row_start = tid * portion_size;
    row_end = (tid+1) * portion_size;

    for (i = row_start; i < row_end; ++i) { // hold row index of 'matrix1'
    for (j = 0; j < size; ++j) { // hold column index of 'matrix2'
        sum = 0; // hold value of a cell
        for (k = 0; k < size; ++k) {
                sum += matrix1[i][k] * matrix2[k][j];
        }
        res_mat[i][j] = sum;
    }
  }
}

int main(int argc, char *argv[]) {
    if(argc == 1){
        printf("more argv needed\n command: ./multi <number of threads> <(size of matrix)>");
        return -1;
    }
    //version 1
    else if(argc == 2){
        pthread_t * threads;
        num_threads = atoi(argv[1]);

        //open read file1
```

```c
        FILE *src1;
        src1 = fopen("data.in","r");
        if(src1 == NULL){
            printf("Error!");
            fclose(src1);
            exit(-1);
        }
        FILE *src2;
        src2 = fopen("data.in","r");
        if(src2 == NULL){
            printf("Error!");
            fclose(src2);
            exit(-1);
        }
        struct mat *data = (struct mat *) malloc(sizeof(struct mat));
        //read file and parse the matrix
        fscanf(src1, "%d", &data->row1);
        data->column1 = data->row1;
        size = data->column1;
        matrix1 = malloc(data->row1 * sizeof(int*));
        for(int i = 0; i < data->row1; ++i){
            matrix1[i] = malloc(data->column1 * sizeof(int));
            for(int j = 0; j < data->column1; ++j)
                fscanf(src1, "%d", &matrix1[i][j]);
        }
        fclose(src1);
        //second matrix
        fscanf(src2, "%d", &data->row2);
        data->column2 = data->row2;
        matrix2 = malloc(data->row2 * sizeof(int*));
        for(int i = 0; i < data->row2; ++i){
            matrix2[i] = malloc(data->column2 * sizeof(int));
            for(int j = 0; j < data->column2; ++j)
                fscanf(src2, "%d", &matrix2[i][j]);
        }
        fclose(src2);

        if ( size % num_threads != 0 ) {
            fprintf( stderr, "size %d must be a multiple of num of threads %d\n",
                size, num_threads );
            return -1;
        }

        res_mat = malloc(data->row1 * sizeof(int*));
        for(int i = 0; i < data->row1; ++i){
            res_mat[i] = malloc(data->row1 * sizeof(int));
        }

        ////////////////////////////////////////////////////////////
        //ptread
        threads = (pthread_t *) malloc( num_threads * sizeof(pthread_t) );

        for (int i = 0; i < num_threads; ++i ) {
            int *tid;
            tid = (int *) malloc( sizeof(int) );
            *tid = i;
            pthread_create( &threads[i], NULL, matrix_multiply, (void *)tid );
        }

        //set and start timer
        clock_t start, end;
        double elapsed;
        start = clock();
        for (int i = 0; i < num_threads; ++i ) {
            pthread_join( threads[i], NULL );
        }
        //end timer
        end = clock();
        elapsed = ((double) (end - start)) / CLOCKS_PER_SEC * 1000;

        FILE *dest;
            dest = fopen("data.out","w+");
            if(dest == NULL){
                printf("Error!");
                fclose(dest);
                exit(-1);
            }
        for(int i = 0; i < data->row1; i++) {
            for(int j = 0; j < data->column2; j++) {
                fprintf(dest, "%d ", res_mat[i][j]);
            }
            fprintf(dest, "\n");
        }
        fclose(dest);
        printf("Time used: %f millisecond.\n", elapsed);

        return 0;
    }

    //version 2
    else if(argc == 3){
        pthread_t * threads;
        num_threads = atoi(argv[1]);
        size = atoi(argv[2]);
        if ( size % num_threads != 0 ) {
        fprintf( stderr, "size %d must be a multiple of num of threads %d\n",
            size, num_threads );
        return -1;
        }

        //create random matrix
        matrix1 = malloc(size * sizeof(int*));
        for(int i = 0; i < size; ++i){
            matrix1[i] = malloc(size * sizeof(int));
            for(int j = 0; j < size; ++j)
                matrix1[i][j] = rand()%100;
        }
        matrix2 = malloc(size * sizeof(int*));
        for(int i = 0; i < size; ++i){
            matrix2[i] = malloc(size * sizeof(int));
            for(int j = 0; j < size; ++j)
                matrix2[i][j] = rand()%100;
        }
        res_mat = malloc(size * sizeof(int*));
        for(int i = 0; i < size; ++i){
            res_mat[i] = malloc(size * sizeof(int));
        }


        ////////////////////////////////////////////////////////////
        //ptread
        threads = (pthread_t *) malloc( num_threads * sizeof(pthread_t) );

        for (int i = 0; i < num_threads; ++i ) {
            int *tid;
            tid = (int *) malloc( sizeof(int) );
            *tid = i;
            pthread_create( &threads[i], NULL, matrix_multiply, (void *)tid );
        }

        //set and start timer
        clock_t start, end;
        double elapsed;
        start = clock();
        for (int i = 0; i < num_threads; ++i ) {
            pthread_join( threads[i], NULL );
        }
        //end timer
        end = clock();
        elapsed = ((double) (end - start)) / CLOCKS_PER_SEC * 1000;

        FILE *dest;
            dest = fopen("random_data.out","w+");
            if(dest == NULL){
                printf("Error!");
                fclose(dest);
                exit(-1);
            }
        fprintf(dest, "%d\n", size);
        fprintf(dest, "Matrix A:\n");
        for(int i = 0; i < size; i++) {
            for(int j = 0; j < size; j++) {
                fprintf(dest, "%d %d %d\n", i, j, matrix1[i][j]);
            }
        }
        fprintf(dest, "Matrix B:\n");
        for(int i = 0; i < size; i++) {
            for(int j = 0; j < size; j++) {
                fprintf(dest, "%d %d %d\n", i, j, matrix2[i][j]);
            }
        }
        fprintf(dest, "Matrix AB:\n");
        for(int i = 0; i < size; i++) {
            for(int j = 0; j < size; j++) {
                fprintf(dest, "%d %d %d\n", i, j, res_mat[i][j]);
            }
        }
        fclose(dest);
        printf("Time used: %f millisecond.\n", elapsed);

        return 1;
    }
    else{
        printf("argv overflow \n command: ./multi <number of threads> <size of matrix>");
        return -1;
    }
}
```