**主要结构体：**

typedef struct pcd //进程结构体

{

char processName[20]; //进程名称

int arriveTime; //进程到达时间

int startTime; //进程开始时间

int endTime; //进程结束时间

int runTime; //进程运行时间大小

int turnOverTime; //周转时间

float userweightTurnOverTime; //带权周转时间

char provessStatus[10]; //进程状态

int runTimeed; //进程已经运行的时间

int bz; //第一次运行标志位

} pcd;

**函数说明：**

|  |  |
| --- | --- |
| 函数 | 描述 |
| Void createProcess(PCB \*p, int n) | 创建进程 |
| Void sortOfArriveTime(PCB \*p, int n) | 按到达时间 冒泡排序 |
| Void runProcess(PCB \*p, int n) | 指定时间片进程运行函数 |
| Void printProcess(PCB \*p) | 打印进程，输出信息 |

**主要代码：**

pcd pcds[P\_NUM]; //进程数组

int currentTime = 0; //当前时间

int processIndex = 0; //进程的编号

int cpuTime = 2; //时间片的大小

int size = P\_NUM; //进程数组中的有效值

int main(int argc, char const \*argv[])

{

freopen("input.txt", "r", stdin);

// 以只读操作读文件，将输入有标准输入流重定向到input.txt

printf("进程名\t到达时间\t运行时间\t\n");

int index;

for (index = 0; index < P\_NUM; index++)

{

scanf("%s", pcds[index].processName);

scanf("%d", &pcds[index].arriveTime);

scanf("%d", &pcds[index].runTime);

pcds[index].bz = 0;

pcds[index].endTime = 0;

pcds[index].startTime = 0;

pcds[index].turnOverTime = 0;

pcds[index].userweightTurnOverTime = 0;

strcpy(pcds[index].provessStatus, "NONE");

printf("%s \t%d\t\t%d\n", pcds[index].processName, pcds[index].arriveTime,

pcds[index].runTime);

}

fclose(stdin); //关闭标准输入流重定向

printf("时间片大小：%d \n", cpuTime);

sortPcbs(); //按到达时间排序

startProcess(); //开始轮转调度算法

return 0;

}

void printfPcbsInfo()

{ //打印所有进程的所有信息

int index = 0;

printf("======================================");

printf("当前时间为：%d ", currentTime);

printf("======================================\n");

printf("进程名\t到达时间\t运行时间\t开始时间\t结束时间\t周转时间\t带权周转时间\t状态\n");

for (index = 0; index < P\_NUM; index++)

{

printf("%s\t%8d\t%8d\t%8d\t%8d\t%8d\t%8.2f\t%4s\n",

pcds[index].processName,

pcds[index].arriveTime,

pcds[index].runTime,

pcds[index].startTime,

pcds[index].endTime,

pcds[index].turnOverTime,

pcds[index].userweightTurnOverTime,

pcds[index].provessStatus);

}

}

void sortPcbs()

{ //按到达时间 冒泡排序

int index = 0, i, j;

int minindex = 0, minvalue = 0;

for (index = 0; index < P\_NUM; index++)

{

minvalue = pcds[index].arriveTime;

for (j = index; j < P\_NUM; j++)

{

if (pcds[j].arriveTime < minvalue)

{

pcd temp = pcds[index];

pcds[index] = pcds[j];

pcds[j] = temp;

}

}

}

}

void movePcbs(int pindex)

{ //改变就绪队列的顺序

int index = 0;

pcd temp = pcds[pindex];

for (index = pindex; index < size - 1; index++)

{

pcds[index] = pcds[index + 1];

}

pcds[size - 1] = temp;

}

void removeFromPcbs(int pindex)

{ //删除完成任务的进程

movePcbs(pindex);

strcpy(pcds[size - 1].provessStatus, "FINISH"); //更改完成进程的状态

if (size == 1)

{

printfPcbsInfo(); //最后一项的输出

}

processIndex++;

size--; //进程数组的逻辑大小减小一个

}

int proIsEnd(int pindex)

{ //判断一个进程是否完成

int x = -1;

if (pcds[pindex].runTimeed == pcds[pindex].runTime)

{

pcds[pindex].endTime = currentTime;

pcds[pindex].turnOverTime = currentTime - pcds[pindex].arriveTime; //计算周转时间

pcds[pindex].userweightTurnOverTime = (pcds[pindex].turnOverTime \* (1.0)) / pcds[pindex].runTime; //计算带权周转时间

removeFromPcbs(pindex); //删除完成全部服务的进程

return TRUE;

}

return x;

}

int runProcess(int pindex)

{ //指定时间片进程运行函数

int i = 0;

for (i = 0; i < cpuTime; i++)

{

if (pcds[pindex].bz == 0)

{

pcds[pindex].startTime = currentTime;

}

currentTime++; //当前单位时间计时+1

strcpy(pcds[pindex].provessStatus, "RUN"); //更改进程的状态

pcds[pindex].runTimeed++; //进程运行时间加一

pcds[pindex].bz = 1; //表示该进程不是第一次运行

printfPcbsInfo(); //保证每个单位时间输出一次结果

if (proIsEnd(pindex) == TRUE)

{ //判断一个进程 是否执行完毕

return 0;

}

}

movePcbs(pindex); //执行玩一个进程后需要对进程数组进行重新排序

strcpy(pcds[size - 1].provessStatus, "WAIT"); //对刚进行完的进程更改状态

return 0;

}

void startProcess()

{ //开始执行程序

int first = 0;

while (1)

{

if (currentTime >= pcds[first].arriveTime)

{

runProcess(first);

} //因为是按照队列的特征用数组实现的，所以每次执行的进程都是进程数组的第一个元素。

else

{

movePcbs(first);

strcpy(pcds[size - 1].provessStatus, "WAIT");

}

if (processIndex == P\_NUM) //判断进程数组是否全部执行完毕 。

break;

}

}

**运行结果：**

进程名 到达时间 运行时间

P1 0 5

P2 2 4

P3 4 1

P4 5 6

时间片大小：2

======================================当前时间为：1 ======================================

进程名 到达时间 运行时间 开始时间 结束时间 周转时间 带权周转时间 状态

P1 0 5 0 0 0 0.00 RUN

P2 2 4 0 0 0 0.00 NONE

P3 4 1 0 0 0 0.00 NONE

P4 5 6 0 0 0 0.00 NONE

======================================当前时间为：2 ======================================

进程名 到达时间 运行时间 开始时间 结束时间 周转时间 带权周转时间 状态

P1 0 5 0 0 0 0.00 RUN

P2 2 4 0 0 0 0.00 NONE

P3 4 1 0 0 0 0.00 NONE

P4 5 6 0 0 0 0.00 NONE

======================================当前时间为：3 ======================================

进程名 到达时间 运行时间 开始时间 结束时间 周转时间 带权周转时间 状态

P2 2 4 2 0 0 0.00 RUN

P3 4 1 0 0 0 0.00 NONE

P4 5 6 0 0 0 0.00 NONE

P1 0 5 0 0 0 0.00 WAIT

======================================当前时间为：4 ======================================

进程名 到达时间 运行时间 开始时间 结束时间 周转时间 带权周转时间 状态

P2 2 4 2 0 0 0.00 RUN

P3 4 1 0 0 0 0.00 NONE

P4 5 6 0 0 0 0.00 NONE

P1 0 5 0 0 0 0.00 WAIT

======================================当前时间为：5 ======================================

进程名 到达时间 运行时间 开始时间 结束时间 周转时间 带权周转时间 状态

P3 4 1 4 0 0 0.00 RUN

P4 5 6 0 0 0 0.00 NONE

P1 0 5 0 0 0 0.00 WAIT

P2 2 4 2 0 0 0.00 WAIT

======================================当前时间为：6 ======================================

进程名 到达时间 运行时间 开始时间 结束时间 周转时间 带权周转时间 状态

P4 5 6 5 0 0 0.00 RUN

P1 0 5 0 0 0 0.00 WAIT

P2 2 4 2 0 0 0.00 WAIT

P3 4 1 4 5 1 1.00 FINISH

======================================当前时间为：7 ======================================

进程名 到达时间 运行时间 开始时间 结束时间 周转时间 带权周转时间 状态

P4 5 6 5 0 0 0.00 RUN

P1 0 5 0 0 0 0.00 WAIT

P2 2 4 2 0 0 0.00 WAIT

P3 4 1 4 5 1 1.00 FINISH

======================================当前时间为：8 ======================================

进程名 到达时间 运行时间 开始时间 结束时间 周转时间 带权周转时间 状态

P1 0 5 0 0 0 0.00 RUN

P2 2 4 2 0 0 0.00 WAIT

P4 5 6 5 0 0 0.00 WAIT

P3 4 1 4 5 1 1.00 FINISH

======================================当前时间为：9 ======================================

进程名 到达时间 运行时间 开始时间 结束时间 周转时间 带权周转时间 状态

P1 0 5 0 0 0 0.00 RUN

P2 2 4 2 0 0 0.00 WAIT

P4 5 6 5 0 0 0.00 WAIT

P3 4 1 4 5 1 1.00 FINISH

======================================当前时间为：10 ======================================

进程名 到达时间 运行时间 开始时间 结束时间 周转时间 带权周转时间 状态

P2 2 4 2 0 0 0.00 RUN

P4 5 6 5 0 0 0.00 WAIT

P1 0 5 0 0 0 0.00 WAIT

P3 4 1 4 5 1 1.00 FINISH

======================================当前时间为：11 ======================================

进程名 到达时间 运行时间 开始时间 结束时间 周转时间 带权周转时间 状态

P2 2 4 2 0 0 0.00 RUN

P4 5 6 5 0 0 0.00 WAIT

P1 0 5 0 0 0 0.00 WAIT

P3 4 1 4 5 1 1.00 FINISH

======================================当前时间为：12 ======================================

进程名 到达时间 运行时间 开始时间 结束时间 周转时间 带权周转时间 状态

P4 5 6 5 0 0 0.00 RUN

P1 0 5 0 0 0 0.00 WAIT

P2 2 4 2 11 9 2.25 FINISH

P3 4 1 4 5 1 1.00 FINISH

======================================当前时间为：13 ======================================

进程名 到达时间 运行时间 开始时间 结束时间 周转时间 带权周转时间 状态

P4 5 6 5 0 0 0.00 RUN

P1 0 5 0 0 0 0.00 WAIT

P2 2 4 2 11 9 2.25 FINISH

P3 4 1 4 5 1 1.00 FINISH

======================================当前时间为：14 ======================================

进程名 到达时间 运行时间 开始时间 结束时间 周转时间 带权周转时间 状态

P1 0 5 0 0 0 0.00 RUN

P4 5 6 5 0 0 0.00 WAIT

P2 2 4 2 11 9 2.25 FINISH

P3 4 1 4 5 1 1.00 FINISH

======================================当前时间为：15 ======================================

进程名 到达时间 运行时间 开始时间 结束时间 周转时间 带权周转时间 状态

P4 5 6 5 0 0 0.00 RUN

P1 0 5 0 14 14 2.80 FINISH

P2 2 4 2 11 9 2.25 FINISH

P3 4 1 4 5 1 1.00 FINISH

======================================当前时间为：16 ======================================

进程名 到达时间 运行时间 开始时间 结束时间 周转时间 带权周转时间 状态

P4 5 6 5 0 0 0.00 RUN

P1 0 5 0 14 14 2.80 FINISH

P2 2 4 2 11 9 2.25 FINISH

P3 4 1 4 5 1 1.00 FINISH

======================================当前时间为：16 ======================================

进程名 到达时间 运行时间 开始时间 结束时间 周转时间 带权周转时间 状态

P4 5 6 5 16 11 1.83 FINISH

P1 0 5 0 14 14 2.80 FINISH

P2 2 4 2 11 9 2.25 FINISH

P3 4 1 4 5 1 1.00 FINISH

**主要结构体：**

typedef struct memoryNode {

string name;

int address;

int length;

memoryNode \*next;

}memoryNode, \*MemoryNode;

MemoryNode freeMemoryList;

MemoryNode usedMemoryList;

int choise1;

MemoryNode create(MemoryNode &p, int address = 0, int length = 0, string name = "") {

p->next = new(memoryNode);

MemoryNode q = p->next;

q->name = name;

q->address = address;

q->length = length;

q->next = NULL;

return q;

}

**主要实现函数：**

//删除内存节点

void deleteMemoryNode(MemoryNode &p, MemoryNode &q) {

p->next = q->next;

delete(q);

}

//插入内存节点

void insertMemoryNode(MemoryNode &p) {

MemoryNode before = usedMemoryList;

MemoryNode now = usedMemoryList->next;

while (now != NULL) {

before = now;

now = now->next;

}

before->next = p;

p->next = now;

}

//初始化内存，选择分配算法

void initMemoryList() {

freeMemoryList = new(memoryNode);

freeMemoryList->address = freeMemoryList->length = 0;

freeMemoryList->next = NULL;

usedMemoryList = new(memoryNode);

usedMemoryList->address = usedMemoryList->length = 0;

usedMemoryList->next = NULL;

int x;

cin >> x;

create(freeMemoryList, 0, x);

printf("1：firstFit\n");

printf("2：bestFist\n");

scanf("%d", &choise1);

while (choise1 != 1 && choise1 != 2) {

printf("非法的选择项请重新选择：");

scanf("%d", &choise1);

}

printf("\n");

}

// 首次适应算法

void firstFit(MemoryNode &p) {

MemoryNode before = freeMemoryList;

MemoryNode now = freeMemoryList->next;

bool flag = false;

while (now != NULL) {

if (now->length >= p->length) {

flag = true;

now->length = now->length - p->length;

p->address = now->address;

now->address = now->address + p->length;

if (now->length == 0) {

deleteMemoryNode(before, now);

}

insertMemoryNode(p);

break;

}

before = now;

now = now->next;

}

if (flag) {

printf("内存分配成功！该进程起始地址为：%d\n", p->address);

} else {

printf("内存不足，分配失败！\n");

}

}

// 最佳适应算法

void bestFist(MemoryNode &p) {

MemoryNode before = freeMemoryList;

MemoryNode now = freeMemoryList->next;

MemoryNode bestBefore = NULL;

MemoryNode bestNow = NULL;

bool flag = false;

while (now != NULL) {

if (now->length >= p->length) {

if (flag == false) {

flag = true;

bestBefore = before;

bestNow = now;

} else {

if (bestNow->length >= now->length) {

bestBefore = before;

bestNow = now;

}

}

}

before = now;

now = now->next;

}

if (flag) {

bestNow->length = bestNow->length - p->length;

p->address = bestNow->address;

bestNow->address = bestNow->address + p->length;

if (bestNow->length == 0){

deleteMemoryNode(bestBefore, bestNow);

}

insertMemoryNode(p);

printf("分配成功，进程起始地址：%d\n", p->address);

} else {

printf("内存不足，分配失败！\n");

}

}

// 分配内存

void allocateMemory(int choise) {

MemoryNode p = new(memoryNode);

p->next = NULL;

printf("进程名：");

cin >> p->name;

printf("进程容量：");

scanf("%d", &p->length);

if (choise == 1) {

firstFit(p);

} else {

bestFist(p);

}

}

**运行结果：**

0：退出程序 1：显示主内存使用情况 2：请求内存 3：释放内存

2

进程名：1

进程容量：130

分配成功，进程起始地址：0

0：退出程序 1：显示主内存使用情况 2：请求内存 3：释放内存

2

进程名：2

进程容量：60

分配成功，进程起始地址：130

0：退出程序 1：显示主内存使用情况 2：请求内存 3：释放内存

2

进程名：3

进程容量：100

640

1：firstFit

2：bestFist

1

0：退出程序 1：显示主内存使用情况 2：请求内存 3：释放内存

2

进程名：1

进程容量：130

内存分配成功！该进程起始地址为：0

0：退出程序 1：显示主内存使用情况 2：请求内存 3：释放内存

2

进程名：2

进程容量：60

内存分配成功！该进程起始地址为：130

0：退出程序 1：显示主内存使用情况 2：请求内存 3：释放内存

2

进程名：3

进程容量：100

内存分配成功！该进程起始地址为：190

0：退出程序 1：显示主内存使用情况 2：请求内存 3：释放内存

3

请输入进程名：2

回收内存成功！

0：退出程序 1：显示主内存使用情况 2：请求内存 3：释放内存

2

进程名：4

进程容量：200

内存分配成功！该进程起始地址为：290

0：退出程序 1：显示主内存使用情况 2：请求内存 3：释放内存

3

请输入进程名：3

回收内存成功！

0：退出程序 1：显示主内存使用情况 2：请求内存 3：释放内存

3

请输入进程名：1

回收内存成功！

0：退出程序 1：显示主内存使用情况 2：请求内存 3：释放内存

2

进程名：5

进程容量：140

内存分配成功！该进程起始地址为：0

0：退出程序 1：显示主内存使用情况 2：请求内存 3：释放内存

2

进程名：6

进程容量：60

内存分配成功！该进程起始地址为：140

0：退出程序 1：显示主内存使用情况 2：请求内存 3：释放内存

2

进程名：7

进程容量：50

内存分配成功！该进程起始地址为：200

0：退出程序 1：显示主内存使用情况 2：请求内存 3：释放内存

3

请输入进程名：6

回收内存成功！

0：退出程序 1：显示主内存使用情况 2：请求内存 3：释放内存

1

未分配的内存：

起始位置：140 大小：60 KB

起始位置：250 大小：40 KB

起始位置：490 大小：150 KB

已分配的内存：

进程名：4 起始位置：290 容量：200KB

进程名：5 起始位置：0 容量：140KB

进程名：7 起始位置：200 容量：50KB

**主要结构体：**

typedef struct

{

// 块号

int blockNumber;

// 状态位（states）

int P;

} PageTableItem, \*PageTableItemArray;

typedef struct PageTable

{

PageTableItemArray \*pageTableItemArray;

int length;

} PageTable;

typedef struct quickPageTableNode

{

int pageNumber;

int blockNumber;

quickPageTableNode \*next;

} quickPageTableNode, \*QuickPageTableNode;

typedef struct QuickPageTable

{

int length;

QuickPageTableNode quickPageTableNode;

} QuickPageTable;

typedef struct stackNode

{

stackNode \*before;

int pageNumber;

stackNode \*next;

} stackNode, \*StackNode;

typedef struct StackList

{

int length;

StackNode stackNode;

} StackList;

**主要实现函数：**

// FIFO算法（进先出算法）更新快表

void updateQuickPageTableNode(QuickPageTable &quickPageTable, int pageNumber, int blockNumber)

{

QuickPageTableNode newQuickPageTableNode = new (quickPageTableNode);

QuickPageTableNode quickPageTableNode = quickPageTable.quickPageTableNode;

newQuickPageTableNode->pageNumber = pageNumber;

newQuickPageTableNode->blockNumber = blockNumber;

if (quickPageTableNode)

{

if (quickPageTable.length > 0)

{

quickPageTable.length--;

newQuickPageTableNode->next = quickPageTable.quickPageTableNode;

}

else

{

printf("快表已满！\n");

QuickPageTableNode tempQuickPageTableNode = quickPageTableNode;

quickPageTableNode = quickPageTableNode->next;

while (quickPageTableNode->next != NULL)

{

tempQuickPageTableNode = quickPageTableNode;

quickPageTableNode = quickPageTableNode->next;

}

tempQuickPageTableNode->next = NULL;

delete (quickPageTableNode);

newQuickPageTableNode->next = quickPageTable.quickPageTableNode;

}

}

else

{

quickPageTable.length--;

newQuickPageTableNode->next = NULL;

}

quickPageTable.quickPageTableNode = newQuickPageTableNode;

}

// 向栈中加入结点页

void addStackNode(PageTable &pageTable, StackList &stackList, int pageNumber)

{

// OS命令CPU从外存读缺页，启动I/O硬件

if (stackList.length > 0)

{

StackNode newStackNode = new (stackNode);

StackNode stackNode = stackList.stackNode;

if (stackNode)

{

newStackNode->pageNumber = pageNumber;

newStackNode->before = stackNode->before;

newStackNode->next = stackNode;

stackNode->before = newStackNode;

stackList.stackNode = newStackNode;

}

else

{

newStackNode->pageNumber = pageNumber;

newStackNode->before = newStackNode;

newStackNode->next = NULL;

stackList.stackNode = newStackNode;

}

stackList.length--;

// NEW 2

printf("PageSum 1: %d \n", PageSum);

// 记录内存栈的长度：MEMORYSIZE - stackList.length

printf("stack.length : %d \n", MEMORYSIZE - stackList.length);

// 当内存栈要大于页表长度时，触发 “踢出最先进入页表的页，将块号分配给新进来的页”

if (PageSum+1 >= (MEMORYSIZE - stackList.length))

{

/\* code \*/

pageTable.pageTableItemArray[pageNumber]->P = 1;

pageTable.pageTableItemArray[pageNumber]->blockNumber = rand() % (10) + 1;

}

else

{

//“踢出最先进入页表的页，将块号分配给新进来的页”

printf("BeforeP : %d \n", BeforeP);

pageTable.pageTableItemArray[pageNumber]->P = 1;

pageTable.pageTableItemArray[pageNumber]->blockNumber = pageTable.pageTableItemArray[BeforeP]->blockNumber;

pageTable.pageTableItemArray[BeforeP]->P = 0;

pageTable.pageTableItemArray[BeforeP]->blockNumber = -1;

}

}

else

{

printf("内存已满！\n");

}

}

// 栈中已有结点页更新使用程度

void updateStackNode(PageTable &pageTable, StackList &stackList, int pageNumber)

{

StackNode stackNode = stackList.stackNode;

while (stackNode != NULL)

{

if (stackNode->pageNumber == pageNumber && stackList.stackNode != stackNode)

{

stackNode->before->next = stackNode->next;

if (stackNode->next != NULL)

{

stackNode->next->before = stackNode->before;

}

stackNode->before = stackList.stackNode->before;

stackNode->next = stackList.stackNode;

stackList.stackNode->before = stackNode;

stackList.stackNode = stackNode;

return;

}

stackNode = stackNode->next;

}

}

// 栈中已有结点页与新结点页置换

void replaceStackNode(PageTable &pageTable, StackList &stackList, int pageNumber)

{

StackNode tailStackNode = stackList.stackNode->before;

tailStackNode->before->next = NULL;

// 换出淘汰页，释放内存

delete (tailStackNode);

stackList.length++;

// printf("Now PageNumber: %d ", pageTable.pageTableItemArray[pageNumber]->P);

pageTable.pageTableItemArray[pageNumber]->P = 0;

// 换入当前页

addStackNode(pageTable, stackList, pageNumber);

}

// 最近最久未使用算法（LRU）

void LRU(PageTable &pageTable, QuickPageTable &quickPageTable, StackList &stackList, int pageNumber)

{

// 判断该页是否在内存中

if (pageTable.pageTableItemArray[pageNumber]->P == 1)

{

// 该页在内存中

printf("当前页已在内存中！更新栈表。n");

updateStackNode(pageTable, stackList, pageNumber);

}

else if (pageTable.pageTableItemArray[pageNumber]->P == 0)

{

// 该页不在内存中

printf("当前页不在在内存中！保留CPU现场，从外存中找到缺页。\n");

if (stackList.length > 0)

{

// 内存未满

printf("内存未满！直接为当前页分配内存。\n");

addStackNode(pageTable, stackList, pageNumber);

}

else

{

// 内存已满

printf("内存已满！置换页\n");

replaceStackNode(pageTable, stackList, pageNumber);

}

}

else

{

printf("系统错误！\n");

}

}

**运行结果：**

Page：0 BlockNumber：-1 State：0

Page：1 BlockNumber：-1 State：0

Page：2 BlockNumber：-1 State：0

Page：3 BlockNumber：-1 State：0

Page：4 BlockNumber：-1 State：0

Page：5 BlockNumber：-1 State：0

Page：6 BlockNumber：-1 State：0

Page：7 BlockNumber：-1 State：0

快表：

null

Stack：

null

now BeforeP : 0

pagetable.lenth: 0请输入逻辑地址：99

不在快表中！

当前页不在在内存中！保留CPU现场，从外存中找到缺页。

内存未满！直接为当前页分配内存。

PageSum 1: 0

stack.length : 1

Page：0 BlockNumber：8 State：1

Page：1 BlockNumber：-1 State：0

Page：2 BlockNumber：-1 State：0

Page：3 BlockNumber：-1 State：0

Page：4 BlockNumber：-1 State：0

Page：5 BlockNumber：-1 State：0

Page：6 BlockNumber：-1 State：0

Page：7 BlockNumber：-1 State：0

快表：

page：0 BlockNumber：8

Stack：

====

0

====

now BeforeP : 0

pagetable.lenth: 1修改访问位和修改位。

当前页的物理地址是：8291

请输入逻辑地址：199

在快表中！

Stack：

====

0

====

修改访问位和修改位。

当前页的物理地址是：8391

请输入逻辑地址：1999

不在快表中！

当前页不在在内存中！保留CPU现场，从外存中找到缺页。

内存未满！直接为当前页分配内存。

PageSum 1: 1

stack.length : 2

Page：0 BlockNumber：8 State：1

Page：1 BlockNumber：10 State：1

Page：2 BlockNumber：-1 State：0

Page：3 BlockNumber：-1 State：0

Page：4 BlockNumber：-1 State：0

Page：5 BlockNumber：-1 State：0

Page：6 BlockNumber：-1 State：0

Page：7 BlockNumber：-1 State：0

快表：

page：1 BlockNumber：10

page：0 BlockNumber：8

Stack：

====

1

====

0

====

now BeforeP : 0

pagetable.lenth: 2修改访问位和修改位。

当前页的物理地址是：11215

请输入逻辑地址：2999

不在快表中！

当前页不在在内存中！保留CPU现场，从外存中找到缺页。

内存未满！直接为当前页分配内存。

PageSum 1: 2

stack.length : 3

Page：0 BlockNumber：8 State：1

Page：1 BlockNumber：10 State：1

Page：2 BlockNumber：4 State：1

Page：3 BlockNumber：-1 State：0

Page：4 BlockNumber：-1 State：0

Page：5 BlockNumber：-1 State：0

Page：6 BlockNumber：-1 State：0

Page：7 BlockNumber：-1 State：0

快表：

page：2 BlockNumber：4

page：1 BlockNumber：10

page：0 BlockNumber：8

Stack：

====

2

====

1

====

0

====

now BeforeP : 0

pagetable.lenth: 3修改访问位和修改位。

当前页的物理地址是：5047

请输入逻辑地址：3999

不在快表中！

当前页不在在内存中！保留CPU现场，从外存中找到缺页。

内存未满！直接为当前页分配内存。

PageSum 1: 3

stack.length : 4

Page：0 BlockNumber：8 State：1

Page：1 BlockNumber：10 State：1

Page：2 BlockNumber：4 State：1

Page：3 BlockNumber：9 State：1

Page：4 BlockNumber：-1 State：0

Page：5 BlockNumber：-1 State：0

Page：6 BlockNumber：-1 State：0

Page：7 BlockNumber：-1 State：0

快表：

page：3 BlockNumber：9

page：2 BlockNumber：4

page：1 BlockNumber：10

page：0 BlockNumber：8

Stack：

====

3

====

2

====

1

====

0

====

now BeforeP : 0

pagetable.lenth: 4修改访问位和修改位。

当前页的物理地址是：10143

请输入逻辑地址：4999

不在快表中！

当前页不在在内存中！保留CPU现场，从外存中找到缺页。

内存未满！直接为当前页分配内存。

PageSum 1: 4

stack.length : 5

快表已满！

Page：0 BlockNumber：8 State：1

Page：1 BlockNumber：10 State：1

Page：2 BlockNumber：4 State：1

Page：3 BlockNumber：9 State：1

Page：4 BlockNumber：1 State：1

Page：5 BlockNumber：-1 State：0

Page：6 BlockNumber：-1 State：0

Page：7 BlockNumber：-1 State：0

快表：

page：4 BlockNumber：1

page：3 BlockNumber：9

page：2 BlockNumber：4

page：1 BlockNumber：10

Stack：

====

4

====

3

====

2

====

1

====

0

====

now BeforeP : 0

pagetable.lenth: 5修改访问位和修改位。

当前页的物理地址是：1927

**主要结构体：**

typedef struct fileContent

{// 实际文件内容

string Content;

} \* FileContent;

typedef struct file

{

// 文件名

string fileName;

// 文件类型

string fileType;

// 文件大小

int fileSize;

// 文件内容指针

FileContent fileContent;

// 文件保护码（index = 0：读权限，index = 1：读权限，index = 2：执行权限；权限位含义：0为禁止，1为允许）

char fileProtectCode[3];

// 创建时间

time\_t createTime;

// 修改时间

time\_t updateTime;

// 访问时间

time\_t accessTime;

} \* File;

// 二级目录：用户文件目录（每个用户的独立文件目录）

typedef struct uFD

{

// 当前文件

File file;

// 上一个文件指针

uFD \*before;

// 下一个文件指针

uFD \*next;

} \* UFD;

typedef struct user

{

// 用户名

string username;

// 密码

string password;

// 用户目录指针

UFD ufd;

} \* User;

// 一级目录：主文件目录（所有用户和对应用户的相关信息）

typedef struct mFD

{

// 当前用户

User user;

// 上一个用户指针

mFD \*before;

// 下一个用户指针

mFD \*next;

} \* MFD;

// 打开文件目录（运行文件目录）

typedef struct aFD

{

// 当前文件

File file;

// 上一个文件指针

aFD \*before;

// 下一个文件指针

aFD \*next;

} \* AFD;

**主要实现函数：**

// 创建二级目录（用户文件目录：每个用户的独立文件目录）

UFD createUFD()

{

UFD ufd = new (uFD);

initUFD(ufd);

return ufd;

}

// 创建打开文件目录（运行文件目录）

AFD createAFD()

{

AFD afd = new (aFD);

initAFD(afd);

return afd;

}

// 判断用户是否存在（已被创建过）

bool isExistForUser(MFD &mfd, string useranme)

{

MFD tempMFD = mfd->next;

while (tempMFD != NULL)

{

if (strcmp(tempMFD->user->username.c\_str(), useranme.c\_str()) == 0)

{

return true;

}

tempMFD = tempMFD->next;

}

return false;

}

// 创建用户

void createUser(MFD &mfd)

{

cout << "正在创建新用户......" << endl;

string username;

cout << "请输入用户名：";

getline(cin, username);

cout << "校验用户存性中......" << endl;

if (isExistForUser(mfd, username))

{

cout << username + "用户已存在，创建失败！" << endl;

return;

}

cout << "用户名“" + username + "”可用！" << endl;

string password;

cout << "请输入密码：";

getline(cin, password);

User newUser = new (user);

newUser->username = username;

newUser->password = password;

newUser->ufd = createUFD();

MFD newMFDNode = new (mFD);

newMFDNode->user = newUser; //把新用户存入到用户中

MFD tempMFD = mfd;

while (tempMFD->next != NULL)

{

tempMFD = tempMFD->next;

}

tempMFD->next = newMFDNode;

newMFDNode->before = tempMFD;

newMFDNode->next = NULL;

mfd->before = newMFDNode;

cout << "创建" + username + "用户成功！" << endl;

}

// 删除用户

void deleteUser(MFD &mfd)

{

cout << "正在删除用户......" << endl;

if (mfd->next == NULL)

{

cout << "一级目录（主文件目录：所有用户和对应用户的相关信息）为空！无用户！此操作无效！" << endl;

return;

}

else

{

string username;

string password;

cout << "请输入用户名：";

getline(cin, username);

cout << "检索用户中......" << endl;

MFD tempMFD = mfd->next;

do

{

if (strcmp(tempMFD->user->username.c\_str(), username.c\_str()) == 0)

{

cout << "正在递归删除" + tempMFD->user->username + "用户的相关信息和所有文件......" << endl;

UFD tempUFD = tempMFD->user->ufd;

while (tempUFD->next != NULL)

{

tempUFD = tempUFD->next;

delete (tempUFD->before);

tempUFD->before = NULL;

}

delete (tempUFD);

tempUFD = NULL;

tempMFD->before->next = tempMFD->next;

if (tempMFD->next != NULL)

{

tempMFD->next->before = tempMFD->before;

}

else

{

mfd->before = tempMFD->before;

}

delete (tempMFD);

tempMFD = NULL;

cout << "删除" + username + "用户成功！" << endl;

return;

}

tempMFD = tempMFD->next;

} while (tempMFD != NULL);

cout << "没有匹配到相应用户！" << endl;

return;

}

}

// 显示一级目录（主文件目录：所有用户和对应用户的相关信息）

void showMFD(MFD &mfd)

{

cout << endl;

cout << "显示一级目录（主文件目录：所有用户和对应用户的相关信息）:" << endl;

if (mfd->next == NULL)

{

cout << "用户列表为空！无用户！" << endl;

return;

}

else

{

cout << "用户列表：" << endl;

MFD tempMFD = mfd->next;

do

{

cout << "\t" + tempMFD->user->username << endl;

tempMFD = tempMFD->next;

} while (tempMFD != NULL);

}

}

// 登录

void loginIn(MFD &mfd, User &currentUser)

{

cout << "正在登录......" << endl;

if (mfd->next == NULL)

{

cout << "一级目录（主文件目录：所有用户和对应用户的相关信息）为空！无用户！此操作无效！" << endl;

return;

}

else

{

string username;

cout << "请输入用户名：";

getline(cin, username);

cout << "检索用户中......" << endl;

MFD tempMFD = mfd->next;

do

{

if (strcmp(tempMFD->user->username.c\_str(), username.c\_str()) == 0)

{

cout << "用户名“" + username + "”有效！" << endl;

string password;

cout << "请输入密码：";

getline(cin, password);

cout << "校验密码中......" << endl;

if (strcmp(tempMFD->user->password.c\_str(), password.c\_str()) != 0)

{

cout << "密码错误！登录失败！" << endl;

return;

}

currentUser = tempMFD->user;

cout << "登录" + currentUser->username + "用户成功！" << endl;

cout << endl;

return;

}

tempMFD = tempMFD->next;

} while (tempMFD != NULL);

cout << "没有匹配到相应用户！" << endl;

return;

}

}

// 登出/注销登录

void loginOut(AFD &afd, User &currentUser)

{

cout << "正在登出/注销登录......" << endl;

if (currentUser == NULL)

{

cout << "无用户登录！此操作无效！" << endl;

return;

}

else

{

cout << "为确保账户安全、文件信息的一致性，正在关闭已打开的文件......" << endl;

AFD tempAFD = afd->next;

if (tempAFD != NULL)

{

while (tempAFD->next != NULL)

{

tempAFD = tempAFD->next;

delete (tempAFD->before);

tempAFD->before = NULL;

}

delete (tempAFD);

tempAFD = NULL;

}

afd->before = afd->next = NULL;

cout << "已打开的文件已全部关闭！" << endl;

currentUser = NULL;

cout << "登出/注销登录成功！";

}

}

// 获取完整文件名（文件名和文件类型）

string getFileNameWithType(File &file)

{

string fileNameWithType = file->fileName + "." + file->fileType;

return fileNameWithType;

}

// 判断完整文件名（文件名和文件类型）是否存在（已被创建过）

bool isExistForFile(UFD &ufd, string fileNameWithType)

{

UFD tempUFD = ufd->next;

while (tempUFD != NULL)

{

if (strcmp(getFileNameWithType(tempUFD->file).c\_str(), fileNameWithType.c\_str()) == 0)

{

return true;

}

tempUFD = tempUFD->next;

}

return false;

}

// 创建文件

void createFile(UFD &ufd)

{

cout << endl;

cout << "正在创建新文件......" << endl;

File newFile = new (file);

cout << "请输入文件名（不包含文件类型/文件扩展名）：";

getline(cin, newFile->fileName);

cout << "请输入文件类型：";

getline(cin, newFile->fileType);

cout << "校验文件存在性中......" << endl;

if (isExistForFile(ufd, getFileNameWithType(newFile)))

{

cout << newFile->fileName + "." + newFile->fileType + "文件已存在，创建失败！" << endl;

return;

}

newFile->fileSize = 0;

newFile->fileContent = new (fileContent);

newFile->fileContent->Content = "";

// 设定文件类型如果为exe则文件保护码中可执行位设定为1

if (strcmp(newFile->fileType.c\_str(), "exe") == 0)

{

strcpy(newFile->fileProtectCode, "111");

}

else

{

strcpy(newFile->fileProtectCode, "110");

}

time(&newFile->createTime);

newFile->updateTime = newFile->createTime;

newFile->accessTime = newFile->createTime;

UFD newUFDNode = createUFD();

newUFDNode->file = newFile;

UFD tempUFD = ufd;

while (tempUFD->next != NULL)

{

tempUFD = tempUFD->next;

}

tempUFD->next = newUFDNode;

newUFDNode->before = tempUFD;

ufd->before = newUFDNode;

cout << "创建" + newFile->fileName + "." + newFile->fileType + "文件成功！" << endl;

}

// 显示二级目录（用户文件目录：每个用户的独立文件目录）

void showUFD(UFD &ufd)

{

cout << endl;

cout << "显示二级目录（用户文件目录：每个用户的独立文件目录）：" << endl;

if (ufd->next == NULL)

{

cout << "文件列表为空！无文件！" << endl;

return;

}

else

{

cout << "文件列表：" << endl;

UFD tempUFD = ufd->next;

do

{

printf("\t%s.%s\n", tempUFD->file->fileName.c\_str(), tempUFD->file->fileType.c\_str());

tempUFD = tempUFD->next;

} while (tempUFD != NULL);

}

}

// 详细显示二级目录（用户文件目录：每个用户的独立文件目录）

void showUFDDetail(UFD &ufd)

{

cout << endl;

cout << "详细显示二级目录（用户文件目录：每个用户的独立文件目录）:" << endl;

if (ufd->next == NULL)

{

cout << "文件列表为空！无文件！" << endl;

return;

}

else

{

UFD tempUFD = ufd->next;

do

{

cout << getFileNameWithType(tempUFD->file) + "文件详情：" << endl;

cout << "\t文件名：" + tempUFD->file->fileName << endl;

cout << "\t文件类型：" + tempUFD->file->fileType << endl;

printf("\t文件大小：%d\n", tempUFD->file->fileSize);

printf("\t文件保护码：%s\n", tempUFD->file->fileProtectCode);

struct tm \*lt;

lt = localtime(&tempUFD->file->createTime);

printf("\t创建时间：%d/%d/%d %d:%d:%d\n", lt->tm\_year + 1900, lt->tm\_mon + 1, lt->tm\_mday, lt->tm\_hour, lt->tm\_min, lt->tm\_sec);

lt = localtime(&tempUFD->file->updateTime);

printf("\t修改时间：%d/%d/%d %d:%d:%d\n", lt->tm\_year + 1900, lt->tm\_mon + 1, lt->tm\_mday, lt->tm\_hour, lt->tm\_min, lt->tm\_sec);

lt = localtime(&tempUFD->file->accessTime);

printf("\t访问时间：%d/%d/%d %d:%d:%d\n", lt->tm\_year + 1900, lt->tm\_mon + 1, lt->tm\_mday, lt->tm\_hour, lt->tm\_min, lt->tm\_sec);

tempUFD = tempUFD->next;

} while (tempUFD != NULL);

}

}

// 显示打开文件目录（运行文件目录）

void showAFD(AFD &afd)

{

cout << endl;

cout << "显示打开文件目录（运行文件目录）：" << endl;

if (afd->next == NULL)

{

cout << "已打开文件列表为空！无文件！" << endl;

return;

}

else

{

cout << "已打开文件列表：" << endl;

AFD tempAFD = afd->next;

do

{

printf("\t%s.%s\n", tempAFD->file->fileName.c\_str(), tempAFD->file->fileType.c\_str());

tempAFD = tempAFD->next;

} while (tempAFD != NULL);

}

}

// 详细打开文件目录（运行文件目录）

void showAFDDetail(AFD &afd)

{

cout << endl;

cout << "详细显示打开文件目录（运行文件目录）：" << endl;

if (afd->next == NULL)

{

cout << "已打开文件列表为空！无文件！" << endl;

return;

}

else

{

AFD tempAFD = afd->next;

do

{

cout << getFileNameWithType(tempAFD->file) + "已打开文件详情：" << endl;

cout << "\t文件名：" + tempAFD->file->fileName << endl;

cout << "\t文件类型：" + tempAFD->file->fileType << endl;

printf("\t文件大小：%d\n", tempAFD->file->fileSize);

printf("\t文件保护码：%s\n", tempAFD->file->fileProtectCode);

struct tm \*lt;

lt = localtime(&tempAFD->file->createTime);

printf("\t创建时间：%d/%d/%d %d:%d:%d\n", lt->tm\_year + 1900, lt->tm\_mon + 1, lt->tm\_mday, lt->tm\_hour, lt->tm\_min, lt->tm\_sec);

lt = localtime(&tempAFD->file->updateTime);

printf("\t修改时间：%d/%d/%d %d:%d:%d\n", lt->tm\_year + 1900, lt->tm\_mon + 1, lt->tm\_mday, lt->tm\_hour, lt->tm\_min, lt->tm\_sec);

lt = localtime(&tempAFD->file->accessTime);

printf("\t访问时间：%d/%d/%d %d:%d:%d\n", lt->tm\_year + 1900, lt->tm\_mon + 1, lt->tm\_mday, lt->tm\_hour, lt->tm\_min, lt->tm\_sec);

tempAFD = tempAFD->next;

} while (tempAFD != NULL);

}

}

// 检查读权限

bool checkReadablePermission(File &file)

{

if (file->fileProtectCode[0] == '1')

{

return true;

}

else

{

return false;

}

}

// 检查写权限

bool checkWritablePermission(File &file)

{

if (file->fileProtectCode[1] == '1')

{

return true;

}

else

{

return false;

}

}

// 检查执行权限

bool checkExecutablePermission(File &file)

{

if (file->fileProtectCode[2] == '1')

{

return true;

}

else

{

return false;

}

}

// 判断文件是否打开

File isOpen(AFD &afd, string fileNameWithType)

{

AFD tempAFD = afd->next;

while (tempAFD != NULL)

{

if (strcmp(getFileNameWithType(tempAFD->file).c\_str(), fileNameWithType.c\_str()) == 0)

{

return tempAFD->file;

}

tempAFD = tempAFD->next;

}

return NULL;

}

// 删除文件

void deleteFile(UFD &ufd, AFD &afd)

{

cout << endl;

cout << "删除文件中......" << endl;

if (ufd->next == NULL)

{

cout << "二级目录（用户文件目录：每个用户的独立文件目录）为空！无文件！此操作无效！" << endl;

return;

}

else

{

string fileNameWithType;

cout << "请输入要删除的文件（格式：文件名.文件类型）：";

getline(cin, fileNameWithType);

cout << "检索文件中......" << endl;

if (isOpen(afd, fileNameWithType))

{

cout << "此文件已打开，不可删除！" << endl;

return;

}

UFD tempUFD = ufd->next;

do

{

if (strcmp(getFileNameWithType(tempUFD->file).c\_str(), fileNameWithType.c\_str()) == 0)

{

cout << "已匹配到对应文件，正在删除......" << endl;

if (!checkWritablePermission(tempUFD->file))

{

cout << "权限不足，该文件不可删除！" << endl;

return;

}

tempUFD->before->next = tempUFD->next;

if (tempUFD->next != NULL)

{

tempUFD->next->before = tempUFD->before;

}

else

{

ufd->before = tempUFD->before;

}

delete (tempUFD);

tempUFD = NULL;

cout << "删除" + fileNameWithType + "文件成功！" << endl;

return;

}

tempUFD = tempUFD->next;

} while (tempUFD != NULL);

cout << "没有匹配到相应文件！" << endl;

return;

}

}

// 打开文件（加载到内存中）

void openFile(UFD &ufd, AFD &afd)

{

cout << endl;

cout << "打开文件中（加载到内存中）......" << endl;

if (ufd->next == NULL)

{

cout << "二级目录（用户文件目录：每个用户的独立文件目录）为空！无文件！此操作无效！" << endl;

return;

}

else

{

string fileNameWithType;

cout << "请输入要打开/执行的文件（格式：文件名.文件类型）：";

getline(cin, fileNameWithType);

cout << "检索文件中......" << endl;

if (isOpen(afd, fileNameWithType))

{

cout << "此文件已打开（已加载至内存中），无需再次打开！" << endl;

return;

}

else

{

UFD tempUFD = ufd->next;

do

{

if (strcmp(getFileNameWithType(tempUFD->file).c\_str(), fileNameWithType.c\_str()) == 0)

{

cout << "已匹配到对应文件，正在打开......" << endl;

if (!checkReadablePermission(tempUFD->file) && checkWritablePermission(tempUFD->file) && !checkExecutablePermission(tempUFD->file))

{

cout << "权限不足，该文件不可打开（加载到内存中）！" << endl;

return;

}

AFD newAFDNode = createAFD();

newAFDNode->file = tempUFD->file;

AFD tempAFD = afd;

while (tempAFD->next != NULL)

{

tempAFD = tempAFD->next;

}

tempAFD->next = newAFDNode;

newAFDNode->before = tempAFD;

afd->before = newAFDNode;

cout << "打开（加载到内存中）" + fileNameWithType + "文件成功！" << endl;

return;

}

tempUFD = tempUFD->next;

} while (tempUFD != NULL);

cout << "没有匹配到相应文件！" << endl;

return;

}

}

}

// 关闭文件

void closeFile(UFD &ufd, AFD &afd)

{

cout << endl;

cout << "关闭文件中......" << endl;

if (ufd->next == NULL)

{

cout << "二级目录（用户文件目录：每个用户的独立文件目录）为空！无文件！此操作无效！" << endl;

return;

}

else

{

string fileNameWithType;

cout << "请输入要关闭的文件（格式：文件名.文件类型）：";

getline(cin, fileNameWithType);

cout << "检索文件中......" << endl;

AFD tempAFD = afd->next;

while (tempAFD != NULL)

{

if (strcmp(getFileNameWithType(tempAFD->file).c\_str(), fileNameWithType.c\_str()) == 0)

{

tempAFD->before->next = tempAFD->next;

if (tempAFD->next != NULL)

{

tempAFD->next->before = tempAFD->before;

}

else

{

afd->before = tempAFD->before;

}

delete (tempAFD);

tempAFD = NULL;

cout << "关闭" + fileNameWithType + "文件成功！" << endl;

return;

}

tempAFD = tempAFD->next;

}

cout << "此文件未打开，无需关闭！" << endl;

return;

}

}

// 读文件

void readFile(UFD &ufd, AFD &afd)

{

cout << endl;

cout << "读文件中......" << endl;

if (ufd->next == NULL)

{

cout << "二级目录（用户文件目录：每个用户的独立文件目录）为空！无文件！此操作无效！" << endl;

return;

}

else

{

string fileNameWithType;

cout << "请输入要读取的文件（格式：文件名.文件类型）：";

getline(cin, fileNameWithType);

cout << "正在检索文件中......" << endl;

File file = isOpen(afd, fileNameWithType);

if (!file)

{

cout << "没有匹配到相应文件（此文件还未打开，请先打开该文件再读！）" << endl;

return;

}

cout << "已匹配到对应文件，正在尝试读......" << endl;

if (!checkReadablePermission(file))

{

cout << "权限不足，该文件不可读！" << endl;

return;

}

// 修改文件访问时间

time(&file->accessTime);

cout << "读" + fileNameWithType + "文件成功！" << endl;

if (file->fileSize == 0)

{

cout << "文件内容：(无文件内容)" + file->fileContent->Content << endl;

}

else

{

cout << "文件内容：" + file->fileContent->Content << endl;

}

return;

}

}

// 写文件

void writeFile(UFD &ufd, AFD &afd)

{

cout << endl;

cout << "写文件中......" << endl;

if (ufd->next == NULL)

{

cout << "二级目录（用户文件目录：每个用户的独立文件目录）为空！无文件！此操作无效！" << endl;

return;

}

else

{

string fileNameWithType;

cout << "请输入要写入的文件（格式：文件名.文件类型）：";

getline(cin, fileNameWithType);

cout << "正在检索文件中......" << endl;

File file = isOpen(afd, fileNameWithType);

if (!file)

{

cout << "没有匹配到相应文件（此文件还未打开，请先打开该文件再写！）" << endl;

return;

}

cout << "已匹配到对应文件" << endl;

if (!checkWritablePermission(file))

{

cout << "权限不足，该文件不可写！" << endl;

return;

}

cout << "请输入要写入的文件内容（按回车确定内容即写入）：";

string fileContent;

getline(cin, fileContent);

cout << "保存中，正在尝试写入......" << endl;

file->fileSize = fileContent.length();

file->fileContent->Content = fileContent;

time(&file->updateTime);

// 修改文件更新时间

time(&file->updateTime);

cout << "写入" + fileNameWithType + "文件成功！" << endl;

return;

}

}

// 运行

void run(MFD &mfd, AFD &afd, User &currentUser)

{

string choice1;

string choice2;

while (true)

{

cout << "请选择操作：1、创建新用户 2、删除用户 3、显示所有用户 4、登录 5、结束程序" << endl;

cout << "直接输入操作选项前面对应数字即可：";

getline(cin, choice1);

while (strcmp(choice1.c\_str(), "1") != 0 && strcmp(choice1.c\_str(), "2") != 0 && strcmp(choice1.c\_str(), "3") != 0 && strcmp(choice1.c\_str(), "4") != 0 && strcmp(choice1.c\_str(), "5") != 0)

{

cout << "非法输入！请重新输入！" << endl;

cout << "直接输入操作选项前面对应数字即可：";

getline(cin, choice1);

}

if (strcmp(choice1.c\_str(), "1") == 0)

{

createUser(mfd);

}

else if (strcmp(choice1.c\_str(), "2") == 0)

{

deleteUser(mfd);

}

else if (strcmp(choice1.c\_str(), "3") == 0)

{

showMFD(mfd);

}

else if (strcmp(choice1.c\_str(), "4") == 0)

{

loginIn(mfd, currentUser);

while (currentUser)

{

cout << "请选择操作：1、创建新文件 2、删除文件 3、打开文件 4、关闭文件 5、读文件 6、写文件 7、显示所有文件 8、详细显示所有文件信息 9、显示已打开的文件 10、详细显示已打开的文件的信息 11、登出/注销登录" << endl;

cout << "直接输入操作选项前面对应数字即可：";

getline(cin, choice2);

while (strcmp(choice2.c\_str(), "1") != 0 && strcmp(choice2.c\_str(), "2") != 0 && strcmp(choice2.c\_str(), "3") != 0 && strcmp(choice2.c\_str(), "4") != 0 && strcmp(choice2.c\_str(), "5") != 0 && strcmp(choice2.c\_str(), "6") != 0 && strcmp(choice2.c\_str(), "7") != 0 && strcmp(choice2.c\_str(), "8") != 0 && strcmp(choice2.c\_str(), "9") != 0 && strcmp(choice2.c\_str(), "10") != 0 && strcmp(choice2.c\_str(), "11") != 0)

{

cout << "非法输入！请重新输入！" << endl;

cout << "直接输入操作选项前面对应数字即可：";

getline(cin, choice2);

}

if (strcmp(choice2.c\_str(), "1") == 0)

{

createFile(currentUser->ufd);

}

else if (strcmp(choice2.c\_str(), "2") == 0)

{

deleteFile(currentUser->ufd, afd);

}

else if (strcmp(choice2.c\_str(), "3") == 0)

{

openFile(currentUser->ufd, afd);

}

else if (strcmp(choice2.c\_str(), "4") == 0)

{

closeFile(currentUser->ufd, afd);

}

else if (strcmp(choice2.c\_str(), "5") == 0)

{

readFile(currentUser->ufd, afd);

}

else if (strcmp(choice2.c\_str(), "6") == 0)

{

writeFile(currentUser->ufd, afd);

}

else if (strcmp(choice2.c\_str(), "7") == 0)

{

showUFD(currentUser->ufd);

}

else if (strcmp(choice2.c\_str(), "8") == 0)

{

showUFDDetail(currentUser->ufd);

}

else if (strcmp(choice2.c\_str(), "9") == 0)

{

showAFD(afd);

}

else if (strcmp(choice2.c\_str(), "10") == 0)

{

showAFDDetail(afd);

}

else if (strcmp(choice2.c\_str(), "11") == 0)

{

loginOut(afd, currentUser);

}

cout << endl;

}

}

else if (strcmp(choice1.c\_str(), "5") == 0)

{

break;

}

cout << endl;

}

}

**运行结果：**

