**操作系统课程设计报告**

**题目**

**院 、 系 计算机与控制工程学院**

**专 业 软件工程**

**班 级**

**姓 名**

**学 号**

**指导教师**

年 月 日**目录**

1 项目描述 1

2 系统结构分析 1

3 系统详细设计 1

4 系统主要算法 1

5 系统程序实现 1

6 程序实现结果 2

7 课程设计总结 2

附录：源程序 2

注：报告中，本行及【】内文字需要删除。

**1. 项目描述：**

该项目的目的是设计一个简单的操作系统内存管理模拟程序。程序模拟了一个具有页面置换和空闲块管理的内存系统。通过模拟进程的调度和页面错误的处理，可以观察和分析不同页面置换算法和空闲块管理方法的效果。

**2. 系统结构分析：**

该程序的结构较为简单，主要包括以下几个模块：

- 进程管理模块：负责创建进程、分配内存、回收内存等操作。

- 页面置换模块：负责处理页面错误，选择页面进行置换。

- 空闲块管理模块：负责管理空闲块链表，将置换出的页面添加到空闲块链表中。

- 就绪队列模块：用于存储就绪状态的进程，按照先来先服务的原则进行调度。

程序的整体仿真流程如下：

1) 创建进程，并分配内存给进程。

2) 将进程插入就绪队列。

3) 从就绪队列选择一个进程进行调度执行。

4) 执行过程中产生逻辑地址，进行地址重定位。

5) 模拟页面错误发生的情况，根据页面置换算法选择页面进行置换。

6) 处理页面错误后，将阻塞的进程放入就绪队列尾部。

7) 循环执行步骤3-6，直到就绪队列为空。

8) 回收进程占用的内存。

**3. 系统详细设计：**

数据结构和存储结构：

- PCB结构体：用于表示进程控制块，包含进程ID、页面表、页面错误次数和进程状态等信息。

- QueueNode结构体：用于表示就绪队列中的节点，包含进程指针和下一个节点指针。

- Queue结构体：用于表示就绪队列，包含队头指针和队尾指针。

- FreeBlock结构体：用于表示空闲块，包含起始页号、长度和下一个空闲块指针。

**功能流程：**

1) 创建进程：通过create\_process函数创建PCB结构体并初始化，返回进程指针。

2) 分配内存：通过allocate\_memory函数将物理块分配给进程，更新页面表。

3) 回收内存：通过deallocate\_memory函数将进程占用的物理块回收，更新页面表。

4) 插入就绪队列：通过insert\_ready\_queue函数将进程插入就绪队列尾部。

5) 选择进程：通过select\_process函数从就绪队列选择一个进程进行调度。

6) 处理页面错误：通过handle\_page\_fault函数处理页面错误，包括页面置换和空闲块管理。

7) 打印空闲块链表：通过print\_free\_blocks函数打印空闲块链表的起始页号和长度信息。

8) 打印就绪队列：通过print\_ready\_queue函数打印就绪队列中的进程ID。

9) 执行主函数中的循环，直到就绪队列为空。

**4. 系统主要算法：**

**-** 页面置换算法：FIFO（先进先出）算法。

当发生页面错误时，选择页面表中的第一个页面（队头页面）进行置换。

- 空闲块管理方法：空闲块链接法。

空闲块以链表的形式组织，每个空闲块包含起始页号和长度信息。

当页面被置换出去时，将置换出的页面添加到空闲块链表中。

**5. 系统程序实现：**

- 实现环境：C语言

- 部分核心代码：

- 创建进程函数create\_process：

创建PCB结构体，并初始化进程ID、页面表、页面错误次数和进程状态等信息。

- 分配内存函数allocate\_memory：

将物理块分配给进程，更新页面表。

- 回收内存函数deallocate\_memory：

回收进程占用的物理块，更新页面表。

- 插入就绪队列函数insert\_ready\_queue：

将进程插入就绪队列尾部。

- 选择进程函数select\_process：

从就绪队列选择一个进程进行调度。

- 处理页面错误函数handle\_page\_fault：

处理页面错误，包括页面置换和空闲块管理。

- 打印空闲块链表函数print\_free\_blocks：

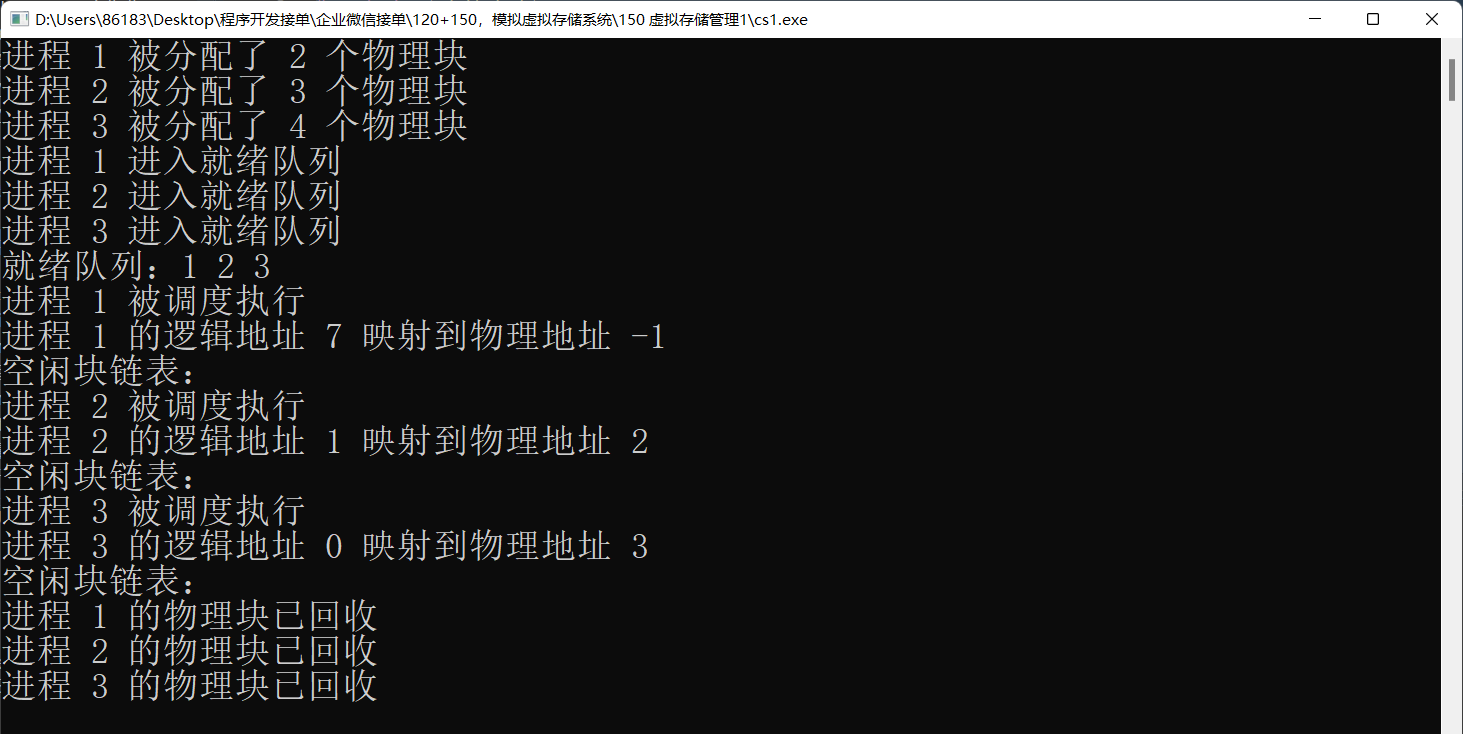
打印空闲块链表的起始页号和长度信息。

- 打印就绪队列函数print\_ready\_queue：

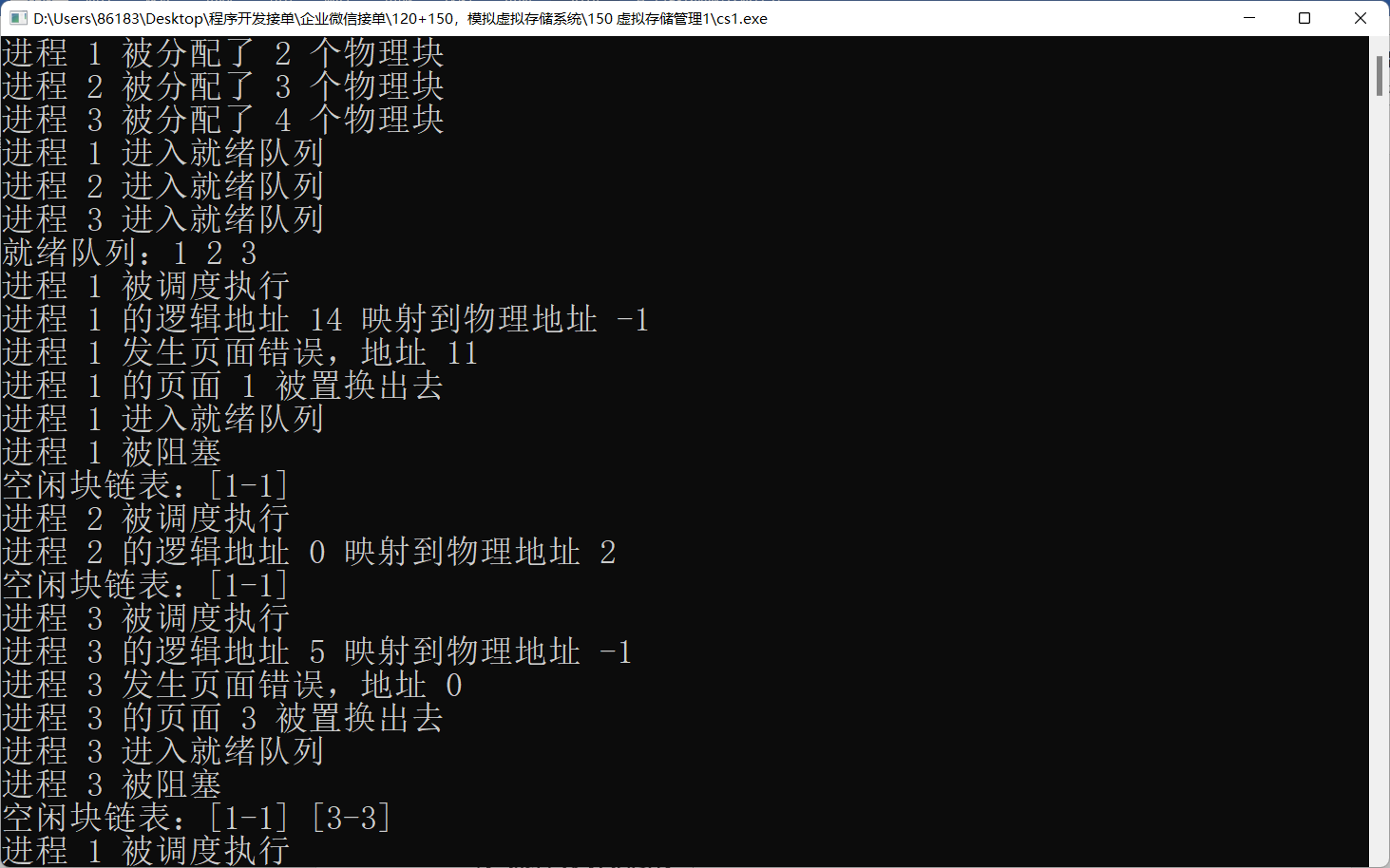
打印就绪队列中的进程ID。

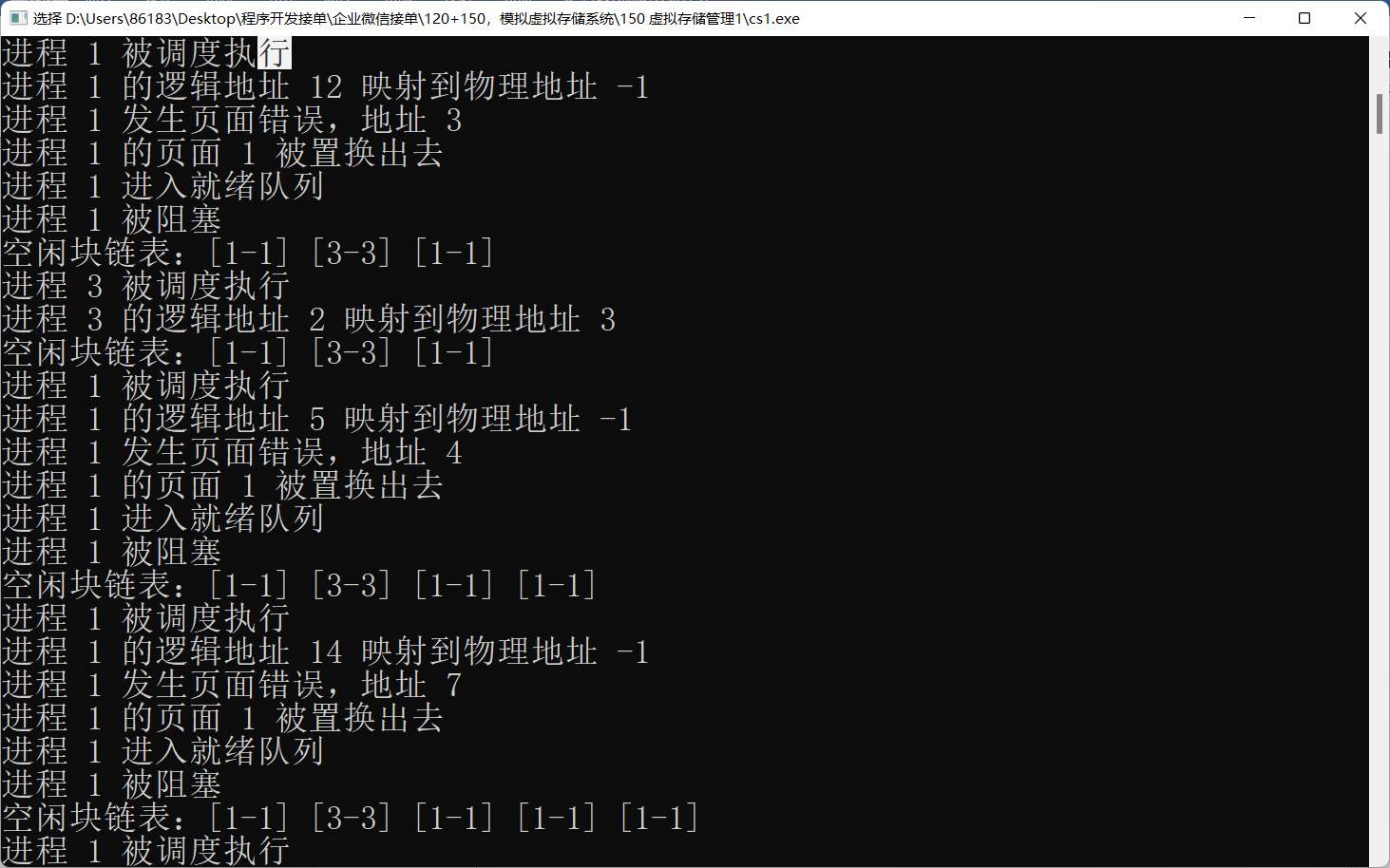
**6. 程序实现结果：**

该部分需要根据实际运行程序的结果进行实验设计、实验结果截图、实验结果截图说明、实验结果分析与讨论。



**第二次运行：**





**7. 课程设计总结：**

**建议：**

1. 在主函数中，你可以使用循环来模拟进程的执行，而不仅仅是执行就绪队列中的第一个进程。这样可以更好地模拟实际的进程调度。

2. 你可以考虑添加一些错误处理的机制，例如当分配内存失败时，输出错误信息并终止程序的执行。

3. 为了提高代码的可读性，你可以使用一些宏定义或者常量来代替一些魔法数字，例如 MAX\_PROCESS、MAX\_PAGES 和 PAGE\_SIZE。

**总结：**

1. 你的程序实现了进程管理、页面置换、空闲块管理和进程调度等功能，结构清晰，代码逻辑清楚。

2. 你使用了FIFO页面置换算法和空闲块链接法进行模拟，这些算法在实际的操作系统中也是常用的。

3. 你的程序可以通过调用 allocate\_memory 函数来为进程分配物理块，通过调用 deallocate\_memory 函数来释放进程占用的物理块。

4. 你使用了就绪队列来调度进程的执行，并处理页面错误，这样可以更好地模拟实际的进程调度过程。

5. 你的程序中还包括了打印空闲块链表和就绪队列的函数，这样可以方便地查看当前系统的状态。

# 附录：源程序

* 每个文件开头注释(示例)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 版权所有 (C)2019, AiBianChen

\*

\* 文件名称： SuffixTest.c

\* 文件标识：无

\* 内容摘要：该代码用于获取满足后缀要求的第一个文件

\* 其它说明：无

\* 当前版本： V1.0

\* 作 者：艾边晨

\* 完成日期： 20191106

\*

\* 修改记录1：

\* 修改日期： 20191106

\* 版本号： V1.0

\* 修改人： AiBianChen

\* 修改内容：创建

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

源代码：

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define MAX\_PROCESS 10

#define MAX\_PAGES 20

#define PAGE\_SIZE 4096

typedef struct PCB {

int pid; // 进程ID

int page\_table[MAX\_PAGES]; // 页面表

int page\_fault; // 页面错误次数

int status; // 进程状态

} PCB;

typedef struct QueueNode {

PCB\* process; // 进程指针

struct QueueNode\* next; // 下一个节点指针

} QueueNode;

typedef struct Queue {

QueueNode\* front; // 队头指针

QueueNode\* rear; // 队尾指针

} Queue;

typedef struct FreeBlock {

int start; // 空闲块起始页号

int length; // 空闲块长度

struct FreeBlock\* next; // 下一个空闲块指针

} FreeBlock;

PCB\* create\_process(int pid) {

PCB\* process = (PCB\*)malloc(sizeof(PCB));

process->pid = pid;

process->page\_fault = 0;

process->status = 0; // 就绪状态

// 初始化页面表

for (int i = 0; i < MAX\_PAGES; i++) {

process->page\_table[i] = -1; // -1表示未分配页面

}

return process;

}

void allocate\_memory(PCB\* process, int num\_pages) {

// 分配物理块给进程

int allocated = 0;

for (int i = 0; i < MAX\_PAGES && allocated < num\_pages; i++) {

if (process->page\_table[i] == -1) {

process->page\_table[i] = process->pid;

allocated++;

}

}

printf("进程 %d 被分配了 %d 个物理块\n", process->pid, allocated);

}

void deallocate\_memory(PCB\* process) {

// 释放进程占用的物理块

for (int i = 0; i < MAX\_PAGES; i++) {

if (process->page\_table[i] == process->pid) {

process->page\_table[i] = -1;

}

}

printf("进程 %d 的物理块已回收\n", process->pid);

}

void insert\_ready\_queue(Queue\* ready\_queue, PCB\* process) {

// 将进程插入就绪队列尾部

QueueNode\* node = (QueueNode\*)malloc(sizeof(QueueNode));

node->process = process;

node->next = NULL;

if (ready\_queue->rear == NULL) {

ready\_queue->front = node;

ready\_queue->rear = node;

} else {

ready\_queue->rear->next = node;

ready\_queue->rear = node;

}

printf("进程 %d 进入就绪队列\n", process->pid);

}

PCB\* select\_process(Queue\* ready\_queue) {

// 从就绪队列选择一个进程调度

if (ready\_queue->front == NULL) {

return NULL;

}

QueueNode\* node = ready\_queue->front;

ready\_queue->front = ready\_queue->front->next;

if (ready\_queue->front == NULL) {

ready\_queue->rear = NULL;

}

PCB\* process = node->process;

free(node);

printf("进程 %d 被调度执行\n", process->pid);

return process;

}

void handle\_page\_fault(PCB\* process, Queue\* ready\_queue, FreeBlock\*\* free\_blocks) {

// 处理页面错误

int page\_fault\_addr = rand() % MAX\_PAGES; // 随机选择一个页面错误的地址

process->page\_fault++;

process->status = 2; // 阻塞状态

printf("进程 %d 发生页面错误，地址 %d\n", process->pid, page\_fault\_addr);

// 页面置换算法：FIFO，选择队头页面进行置换

int victim\_page = process->page\_table[0];

process->page\_table[0] = process->pid;

printf("进程 %d 的页面 %d 被置换出去\n", process->pid, victim\_page);

// 空闲块管理算法：空闲块链接法，将置换出的页面添加到空闲块链表

FreeBlock\* new\_block = (FreeBlock\*)malloc(sizeof(FreeBlock));

new\_block->start = victim\_page;

new\_block->length = 1;

new\_block->next = NULL;

if (\*free\_blocks == NULL) {

\*free\_blocks = new\_block;

} else {

FreeBlock\* tail = \*free\_blocks;

while (tail->next != NULL) {

tail = tail->next;

}

tail->next = new\_block;

}

// 将阻塞的进程放入就绪队列尾部

insert\_ready\_queue(ready\_queue, process);

printf("进程 %d 被阻塞\n", process->pid);

}

void print\_free\_blocks(FreeBlock\* free\_blocks) {

// 打印空闲块链表

printf("空闲块链表：");

FreeBlock\* block = free\_blocks;

while (block != NULL) {

printf("[%d-%d] ", block->start, block->start + block->length - 1);

block = block->next;

}

printf("\n");

}

void print\_ready\_queue(Queue\* ready\_queue) {

// 打印就绪队列

printf("就绪队列：");

QueueNode\* node = ready\_queue->front;

while (node != NULL) {

printf("%d ", node->process->pid);

node = node->next;

}

printf("\n");

}

int main() {

srand(time(NULL));

Queue ready\_queue;

ready\_queue.front = NULL;

ready\_queue.rear = NULL;

FreeBlock\* free\_blocks = NULL;

// 创建进程

PCB\* process1 = create\_process(1);

PCB\* process2 = create\_process(2);

PCB\* process3 = create\_process(3);

// 分配内存给进程

allocate\_memory(process1, 2);

allocate\_memory(process2, 3);

allocate\_memory(process3, 4);

// 将进程插入就绪队列

insert\_ready\_queue(&ready\_queue, process1);

insert\_ready\_queue(&ready\_queue, process2);

insert\_ready\_queue(&ready\_queue, process3);

print\_ready\_queue(&ready\_queue);

while (ready\_queue.front != NULL) {

PCB\* current\_process = select\_process(&ready\_queue);

// 执行中产生逻辑地址并进行地址重定位

int logical\_address = rand() % MAX\_PAGES;

int physical\_address = current\_process->page\_table[logical\_address];

printf("进程 %d 的逻辑地址 %d 映射到物理地址 %d\n", current\_process->pid, logical\_address, physical\_address);

// 模拟页面错误发生

if (rand() % 10 < 3) { // 产生页面错误的概率为30%

handle\_page\_fault(current\_process, &ready\_queue, &free\_blocks);

}

// 打印空闲块链表

print\_free\_blocks(free\_blocks);

}

// 打印被阻塞的进程

QueueNode\* node = ready\_queue.front;

while (node != NULL) {

printf("进程 %d 被阻塞\n", node->process->pid);

node = node->next;

}

// 回收内存

deallocate\_memory(process1);

deallocate\_memory(process2);

deallocate\_memory(process3);

return 0;

}

**操作系统课程设计成绩评价表**

注：请在验收答辩前，对除“答辩”以外的其他项目完成自评。

| **课程目标** | **评价依据** | **评价指标及标准** | **满分** | **学生自评** | **教师确认** |
| --- | --- | --- | --- | --- | --- |
| 课程目标1：针对相对复杂的实际的应用问题，能综合运用操作系统、数据结构、程序设计等多种知识，在对问题进行分析、建模基础上，选择建构合适的数据结构，设计较优的调度与管理算法，并通过编程与调试，完成仿真系统的构建。 | 源程序和课程设计报告 | 源程序：程序排版规范，一句一行，缩格排放；程序中的注释适当，每一个函数前，通过注释给出了关于功能、接口的说明。 | 5 |  |  |
| 结构：程序模块功能划分合理，接口清晰、高效。 | 5 |  |  |
| 功能：完整、正确实现系统管理功能，功能完善。 | 5 |  |  |
| 界面：采用命令行或菜单操作或图形用户界面，操作流程清晰便利。 | 5 |  |  |
| 创新：实现系统有什么功能特色 | 5 |  |  |
| 报告：按模板要求写作，（手写）字迹清楚，书写工整，或（打印）版面在边距、行距等方面处理恰当，整体可读性好。 | 5 |  |  |
| 验收答辩：操作熟练，程序运行正确，能正确回答相关问题。对程序出现的问题能做出相应处理和改进。 | 10 | / |  |
| 课程目标2：理解设计技术参数的要求及实现方式，能识别系统中对处理时间、时间复杂度和存储空间的要求，将之用于系统模型、存储结构和管理算法的设计中，并能分析算法的性能和效率。 | 课程设计报告“3 项目的主要算法”和“6 项目的实现结果”部分 | 用伪代码或流程图等形式规范、清晰地描述了主要算法 | 5 |  |  |
| 正确分析和讨论主要算法的性能和效率。 | 5 |  |  |
| 答辩：正确回答相关问题。连续两个问题回答错误此项不得分。 | 10 | / |  |
| 课程目标3：选择恰当的操作系统平台和程序开发环境，用于对进程管理、存储管理和文件管理中复杂的管理问题进行模拟仿真和开发 | 源程序与项目验收 | 系统平台：是否采用Linux平台，选用Windows平台只能得3分。 | 5 |  |  |
| 程序组织：程序采用多文件组织，采用单文件方案此项不得分。 | 5 |  |  |
| 答辩：正确回答相关问题。连续两个问题回答错误此项不得分。 | 10 | / |  |
| 课程目标4：能理解诚信守则是基本的道德底线，在课程设计环节中态度认真，不抄袭代码，独立或小组合作完成，自觉遵守纪律和规范。 | 源程序与项目验收 | 代码是否涉嫌抄袭，重复率超过50%此项不得分。 | 5 |  |  |
| 答辩：正确回答相关问题。连续两个问题回答错误此项不得分。 | 5 |  |  |
| 课程目标5：能认识到课程设计是一个小型软件项目，可以预测设计开发期间所遇到的困难和风险，能协调自己（或团队）开发时间与成本关系，能对课程设计的成果进行评价和分析。 | 课程设计报告“1 项目描述”和“2系统结构分析”部分 | 能否体现出对于整个设计开发流程各个环节的思考和预测，以及开发程序期间是否与其他同学团队合作。 | 10 |  |  |
| 合 计 | | | 100 |  |  |