**课 程 设 计 报 告**

设计题目：进程调度与内存分配管理模拟

班 级： 计算机科学与技术二班

组长学号： E11514080

组长姓名： 曹泽宇

指导教师： 周爱武

设计时间： 2018年7月

设计分工

组长学号及姓名：E11514080 曹泽宇

分工： 成绩：

组员1学号及姓名：E11514024 江婉榕

分工： 成绩：

组员2学号及姓名：E11514084 戚林莉

分工： 成绩：

组员3学号及姓名：Z11514106 张子淳

分工： 成绩：

组员4学号及姓名：P31514052 倪靖泽

分工： 成绩：

# 摘 要

进程调度与内存分配是操作系统中核心的软件机构。进程调度的任务主要有三个：保存处理机的现场信息、按某种算法选取进程、把处理器分配给进程。为了让进程装入内存，必须为它分配一定大小的内存空间。存储器分配方式可分为连续分配存储管理方式和离散分配存储管理方式。本次课程设计我们小组选择的是模拟Windows操作系统下的进程调度与内存分配，其模拟过程大致可以分为两个部分：第一部分是通过选取抢占式或者非抢占式优先级调度算法来实现进程调度功能；第二部分是为了让进程装入内存，选取连续分配存储管理方式中的最佳适应算法或者离散分配存储管理方式中的分页存储管理算法来对进程进行管理。为了更好地实现抢占式进程调度算法，本实验选用动态优先级类型，规定在就绪队列中的进程随其等待时间的增长，使其优先级相应提高。本模拟系统主要针对进程的管理和操作功能有：添加进程、删除进程、挂起进程、解挂进程等。

**关键词**：操作系统，进程调度，内存管理，抢占式优先级调度算法，非抢占式优先级调度算法，分页式存储管理，最佳适应算法

**目 录**

摘 要 3

1 概述 5

2 课程设计任务及要求 6

2.1 设计任务 6

2.2 设计要求 6

3 算法及数据结构 7

3.1 算法的总体思想 7

3.2 优先级进程调度模拟模块 8

3.2.1 功能 8

3.2.2 数据结构 9

3.2.3 算法 9

3.3 内存分配存储管理模拟模块 15

3.3.1 功能 15

3.3.2 数据结构 16

3.3.3 算法 17

4 程序设计与实现 20

4.1 程序流程图 20

4.2 程序说明 23

4.3实验结果 24

5 结论 27

6 参考文献 27

7 收获、体会和建议 28

# 1 概述

我们组进程调度和内存分配管理模拟系统的设计主要是利用软件工程思想将设计任务进行模块划分，主要分为两个模块，优先级进程调度模拟和内存分配存储管理模拟。

1. 优先级进程调度模拟模块

（1）非抢占式优先级进程调度模拟

系统一旦把处理机分配给就绪队列中优先级最高的进程后，该进程便一直执行下去直至完成，或者因该进程发生某事件而放弃处理机时，系统方可将处理机重新分配给另一优先级最高的进程。

（2）抢占式优先级进程调度模拟

系统把处理机分配给优先级最高的进程，使之执行。但在其执行期间，只要出现了另一个其优先级更高的进程，调度程序就立即停止当前进程的执行，将处理机分配给新到的优先级最高的进程。并且随着进程的推进或者等待，进程的优先级会随着等待时间的增加而提高，以便获得更好的调度性能。这种抢占式的优先级调度算法，能更好地满足紧迫作业的要求，常用于要求比较严格的实时系统中，以及对性能要求高的批处理和分时系统中。

1. 动态优先级

动态优先级是指在创建进程之初，先赋予其一个优先级，然后其值随着进程的推进或等待时间的增加而改变，以便获得更好的调度性能。若所有的就绪进程具有各不相同的优先级初值，那么对于优先级初值低的进程在等待了足够的时间后，也可以获得处理机。当采用抢占式调度方式时，若再规定当前进程的优先级随着运行时间的推移而下降，则可防止一个长作业长期地垄断处理机。

1. 内存分配存储管理模拟模块
2. 最佳适应算法

它从全部空闲区中找出能满足作业要求的、且大小最小的空闲分区，这种方法能使碎片尽量小。为适应此算法，空闲分区表（空闲区链）中的空闲分区要按从小到大进行排序，自表头开始查找到第一个满足要求的自由分区分配。该算法保留大的空闲区，但造成许多小的空闲区。

1. 分页存储管理方式

将用户程序的地址空间分为若干个固定大小的区域，称为“页”或“页面”。相应地，也将内存空间分为若干个物理块，页和块的大小相同。这样可将用户程序的任一页放入任一物理块中，实现离散分配。

# 2 课程设计任务及要求

**2.1 设计任务**

本系统采用优先级调度方式（非抢占式和抢占式），在内存管理方式上选取连续分配方式中的最佳适应算法以及离散分配方式中的分页存储管理方式，从而实现进程的存储以及调度。详细设计分为以下两个方面：

一．优先级进程调度

①设计进程控制块PCB表结构，适用于优先级调度算法（非抢占式和抢占式）。

②建立2-3个进程就绪队列，分别设置不同的时间片和优先级。

③编制进程调度算法，实现该算法的模拟。

二．内存管理方式

（1）最佳适应算法分配方式

①内存初始化。

②接受用户输入的进程消息，并为之分配内存。

③回收用户指定的进程所占用的空间。

④编制最佳适应算法，实现该算法的模拟。

（2）分页存储管理方式

①内存初始化。

②接受用户输入的进程消息，并为之分配内存。

③将一个进程直接分散地装入到许多不相邻接的内存物理块中。

④设计页面索引表，实现从页号到物理块号的地址映射。

⑤回收用户指定的进程所占用的空间。

⑥编制分页存储管理算法，实现该算法的模拟。

**2.2 设计要求**

1、在深入理解操作系统基本原理的基础上，对于选定的题目，以小组为单位，先确定设计方案；

2、设计系统的数据结构、算法和程序结构，设计每个模块的处理流程。要求设计合理，界面友好；

3、编程序实现选题的系统功能，要求实现可视化的运行界面，界面应清楚地反映出系统的运行结果；

4、确定测试方案，选择测试用例，对系统进行测试；

5、运行系统并要通过验收，讲解运行结果，说明系统的特色和创新之处，并回答指导教师的提问；

6、提交系统软件综合训练设计报告

# 3 算法及数据结构

**3.1 算法的总体思想**

非抢占式优先级调度算法流程如下：

****

抢占式优先级调度算法流程如下：

****

最佳适应算法流程如下：

****

分页存储管理算法流程如下：

**3.2 优先级进程调度模拟模块**

**3.2.1功能**

该模块的主要任务是实现优先级进程调度。

（1）可随时增加进程；

（2）规定道数，设置后备队列和挂起状态。若内存中进程少于规定道数，可自 动从后备队列调度一作业进入。被挂起进程入挂起队列，设置解挂功能用于将指定挂起进程解挂入就绪队列;

（3）每次调度后，显示各进程状态；

（4）采用优先级调度算法，实现抢占式和非抢占式两种效果。

**3.2.2数据结构**

该模块的实现使用的是面向对象的编程方法，其数据结构主要是一些类的实现，这些类封装了对进程调度的各种操作。在进程调度上，采取链表结构。

**一．数据结构总体实现**

class CpuWidget : public QWidget//CPU部件类

class InfoDialog : public QDialog//进程信息输入类

class MainWindow : public QMainWindow//系统界面类

class PCB//PCB类

class PCBManager : public QThread//PCB管理类

class QestionDialog : public QDialog//添加进程类

class SProcess : public QPushButton

**二．数据结构具体实现**

**（1）CPU部件类**

class CpuWidget : public QWidget

{

Q\_OBJECT

public:

explicit CpuWidget(QWidget \*parent = nullptr);

void refreshData();

protected:

void initData();

void setUsingState(SProcess\* &s);

void setNotUsingState(SProcess\* &s);

public:

std::vector<SProcess\*> readyLine; // 就绪队列

std::vector<SProcess\*> reserveLine; // 后备队列

std::vector<SProcess\*> hangLine; // 挂起队列

std::vector<SProcess\*> finishLine; // 完成队列

SProcess \*runningProcess;

signals:

public slots:

};

**（2）进程信息输入类**

class InfoDialog : public QDialog

{

Q\_OBJECT

public:

explicit InfoDialog(QWidget \*parent = nullptr);

InfoDialog(QWidget \*parent, int cpu\_time, int priority, int start\_time, int needtime, int over\_time);

signals:

public slots:

};

**（3）系统界面类**

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = 0);

~*MainWindow*();

private:

QMenu \*menu[3];

QAction \*act\_part1[4];

QAction \*act\_part2[2];

QAction \*act\_part3[3];

QMenuBar \*menu\_bar;

// 处理器区域

CpuWidget \*cpu\_widget;

// 主存区域

RAMWidget \*ram\_widget;

int count = 1;

QLineEdit \*deleteLineEdit;

QDialog \*deleteDialog;

QDialog \*hangDialog;

QLineEdit \*hangNum;

// QLineEdit \*hangTime;

QDialog \*solveHangDialog;

QLineEdit \*solveHangEdit;

public slots:

void showAddDialog(QAction\* act);

void deleteConfirm();

void hangConfirm();

void solveHangConfirm();

void mFreshRAMUI();

};

**（4）PCB类**

class PCB

{

public:

PCB(int priority, int needtime, int PID, int PCBsize);

int PID; // 进程名

int needtime; // 要求运行时间

int cputime; // 已经运行时间

int priority; // 优先权(越小越好)

int startTime; // 进入就绪队列时间

int overTime; // 运行结束时间

int state; // 状态：1就绪 2后备 3挂起 4完成 5运行

int hangTime;

int PCBsize;

int startPosition;

int r;

int g;

int b;

};

**（5）PCB管理类**

class PCBManager : public QThread

{

Q\_OBJECT

private:

PCBManager(QObject \*parent = nullptr);

public:

static PCBManager \*getInstance();

void addPCB(PCB\* pcb);

void deletePCB();

void hangPCB();

void freshUI();

void clearAll();

void *run*();

void clearList();

public:

std::vector<PCB\*> readyLine; // 就绪队列

std::vector<PCB\*> reserveLine; // 后备队列

std::vector<PCB\*> hangLine; // 挂起队列

std::vector<PCB\*> finishLine; // 完成队列

int runningState; // 为0是抢占式，为1是非抢占式

CpuWidget \*cpuWidget;

RAMWidget \*ramWidget;

PCB \*runningPCB; // 正在执行的PCB

QTimer \*timer; // 计时器

int timeCount; // 总时间

int deleteNum; // 删除进程编号

int hangNum; // 挂起教程编号

int solveHangNum; // 解挂进程编号

int stop = 0; // 系统是否暂停

int usedSize; // 已使用主存大小

std::vector<ProcessMemoryInfo\*> originInfoList; // 根据就绪队列和运行中的进程获取内存分配的原始信息

std::vector<ProcessMemoryInfo\*> positionInfoList; // 计算空闲分区的信息

signals:

void freshRAM();

void deleteOriginList();

public slots:

void update();

};

**（6）添加进程类**

class QestionDialog : public QDialog

{

Q\_OBJECT

public:

explicit QestionDialog(QWidget \*parent = nullptr);

QLineEdit \*start\_time\_line;

QLineEdit \*priority\_line;

QLineEdit \*needtime\_line;

QLineEdit \*need\_space\_line;

signals:

public slots:

void confirmPress();

};

**（7）进程类**

class SProcess : public QPushButton

{

Q\_OBJECT

public:

explicit SProcess(QWidget \*parent = nullptr);

SProcess(QWidget \*parent, int start\_time, int priority, int cpu\_time, int needtime);

int num;

int cpu\_time;

int priority;

int start\_time;

int needtime;

int overtime;··

int r;

int g;

int b;

protected:

void *mousePressEvent*(QMouseEvent \*event);

signals:

public slots:

};

**（8）在结构体中封装类**

Struct node{

SProcess sprocess；

node \* next；

}

### 3.2.3 算法

**3.3内存分配存储管理模拟模块**

**3.3.1功能**

该模块的主要任务是实现内存的分配存储管理。

（1）采用最佳适应算法或者分页存储管理方法分配主存空间；

（2）进程完成后，回收主存，并与相邻空闲分区合并；

（3）实现最佳适应算法以及分页存储管理方法。

**3.3.2数据结构**

**一．数据结构总体实现**

该模块的实现使用的是面向对象的编程方法，其数据结构也主要是一些类的实现，这些类封装了对内存分配回收的各种操作。

注：与上一模块相同的数据结构，不再做多余叙述

class PCBManager : public QThread//PCB管理类

class ProcessMemoryInfo//进程信息存储类

class RAMWidget : public QWidget//存储器部件类

**二．数据结构具体实现**

**（1）PCB管理类**

class PCBManager : public QThread

{

Q\_OBJECT

private:

PCBManager(QObject \*parent = nullptr);

public:

static PCBManager \*getInstance();

void addPCB(PCB\* pcb);

void deletePCB();

void hangPCB();

void freshUI();

void clearAll();

void *run*();

void clearList();

int getStartPostion(PCB \*pcb);

void getList();

public:

std::vector<PCB\*> readyLine; // 就绪队列

std::vector<PCB\*> reserveLine; // 后备队列

std::vector<PCB\*> hangLine; // 挂起队列

std::vector<PCB\*> finishLine; // 完成队列

CpuWidget \*cpuWidget;

RAMWidget \*ramWidget;

PCB \*runningPCB; // 正在执行的PCB

QTimer \*timer; // 计时器

int timeCount; // 总时间

int deleteNum; // 删除进程编号

int hangNum; // 挂起教程编号

int solveHangNum; // 解挂进程编号

int stop = 0; // 系统是否暂停

int usedSize; // 已使用主存大小

std::vector<ProcessMemoryInfo\*> originInfoList; // 根据就绪队列和运行中的进程获取内存分配的原始信息

std::vector<ProcessMemoryInfo\*> positionInfoList; // 计算空闲分区的信息

signals:

void freshRAM();

void deleteOriginList();

public slots:

void update();

};

**（2）进程信息存储类**

class ProcessMemoryInfo

{

public:

// ProcessMemoryInfo();

ProcessMemoryInfo(int start, int length, int end, int PID);

ProcessMemoryInfo(ProcessMemoryInfo \*p);

int start;

int length;

int end;

int PID;

};

**（3）存储器部件类**

class RAMWidget : public QWidget

{

Q\_OBJECT

public:

explicit RAMWidget(QWidget \*parent = nullptr);

public:

QFrame \*frame;

std::vector<QLabel\*> labelList;

void refreshRAMUI();

signals:

public slots:

};

**3.3.3算法**

# 4 程序设计与实现

**4.1 程序流程图**

（1）刷新UI操作函数流程图

****

（2）运行进程操作函数流程图

****

（3）获取空闲位置操作函数流程图

****

（4）获取内存信息操作函数流程图

****

（5）删除PCB操作函数流程图

****

（6）挂起PCB操作函数流程图

****

**4.2 程序说明**

我们小组设计的进程调度和内存分配管理系统是对进程调度以及内存分配回收的模拟。该模拟在进程调度上实现了非抢占式优先级进程调度以及抢占式优先级进程调度，在内存分配管理上实现了最佳适应算法以及分页存储管理方法。具体实现功能为随时增加进程、规定道数，设置后备队列和挂起状态，每次调度后，显示各进程状态，采用最佳适应算法和分页存储管理分配主存空间，进程完成后，回收主存，并与相邻空闲分区合并等。该模拟的实现基于QT的开发环境，使用QT进行windows桌面应用开发。为了实现以上功能，我们采用了链表以及类的数据结构。

**4.3 实验结果**

# 5 结论

这次的课程设计任务是在我们小组的共同努力下完成的，就课设要求而言，我们的成果可以说是相当成功。本系统不仅实现了老师要求的各项基本功能:采用抢占式动态优先级调度算法实现进程调度以及分页式存储管理算法进行进程内存分配。另外该系统还有不少的扩展功能，如：非抢占式动态优先级调度算法实现进行调度以及使用连续分配管理的最佳适应算法进行进程内存分配。并在上述基本功能和扩展功能基础上，实现添加进程、删除进程、挂起进程、解挂进程的操作。所以说我们组设计的进程调度与内存分配存储管理模拟系统已经基本实现Windows操作系统下的大部分进程操作功能。尽管如此，我们组的设计还是有许多待改进的地方，如：可以对调度算法进行优化，实现多级反馈队列调度算法；对内存分配的优化，比如模拟虚拟存储器进行存储；界面可以做得更加人性化……

# 6 参考文献

1.徐虹等编著.操作系统实验指导——基于Linux内核.北京: 清华大学出版社.

2.陈向群等编著. Windows内核实验教程. 北京: 机械工业出版社.

3.周苏等编著. 操作系统原理实验. 北京: 科学出版社.

4.张尧学编著. 计算机操作系统教程习题解答与实验指导. 北京: 清华大学出版社

# 7 收获、体会和建议