操作系统课程设计报告

—概要设计

学　　院：计算机学院（国家示范性软件学院）

专　　业：计算机与科学技术

班　　级：2020211312

姓　　名：王天行

2023 年 3 月 1 日

1. 目录

1 总体设计

1.1 概述

1.1.1 功能描述

本系统旨在设计和实现一个具有操作系统基本功能的模拟程序，包括进程管理、内存管理、文件系统和设备管理等功能，同时提供用户友好的 UI 界面。

1.1.2 运行环境

操作系统：Windows 10 / Linux 硬件：支持 64 位 CPU、至少 2GB RAM

1.1.3 开发环境

编程语言：C/C++ 开发工具：Visual Studio / GCC

1.2 设计思想

1.2.1 软件设计构思

本系统采用模块化设计，将功能划分为进程管理、内存管理、文件系统和设备管理等模块，每个模块独立完成各自的功能，通过接口进行交互。

1.2.2 关键技术与算法

- 进程调度算法：如 FCFS, PRIORITY, RR 等；
- 存储管理：连续分区 / 页式 / 按需调页；
- 文件数据空间的组织方式：连续、链接、索引；
- 设备调度策略。

1.2.3 基本数据结构

①进程管理块 PCB：

```
class PCB
{
    int PID; //进程标识符
    int processState; //进程状态
    long IR; //指令寄存器，保存当前正在执行指令的地址
    long pSeg; //program segment，程序段地址
    long dSeg; //data segment，数据段地址
};
```

②就绪/阻塞队列：

```
class QUEUE
{
    int priority; //队列优先级
    int PID[MAX_Process]; //进程队列
};
```

③储存设备对象的基本信息：

```
struct device{
    string deviceName;//设备名称
    double tansmitRate;//传输速率（若有）
    int isBusy;//是否被占用
    int request[REQUESTNUM];//该设备的请求队列
};
```

④储存设备请求的信息：

```
class deviceRequest{
    int pid;//进程 ID
    string deviceName;//请求的设备名称
    string data;//数据内容（若有）
    double ioTime;//需使用该设备的时长（若有）
};
```

⑤进程存储位置：

```
class processStorage{
    int ID;
    int* dataSegment;
    int* codeSegment;
};
```

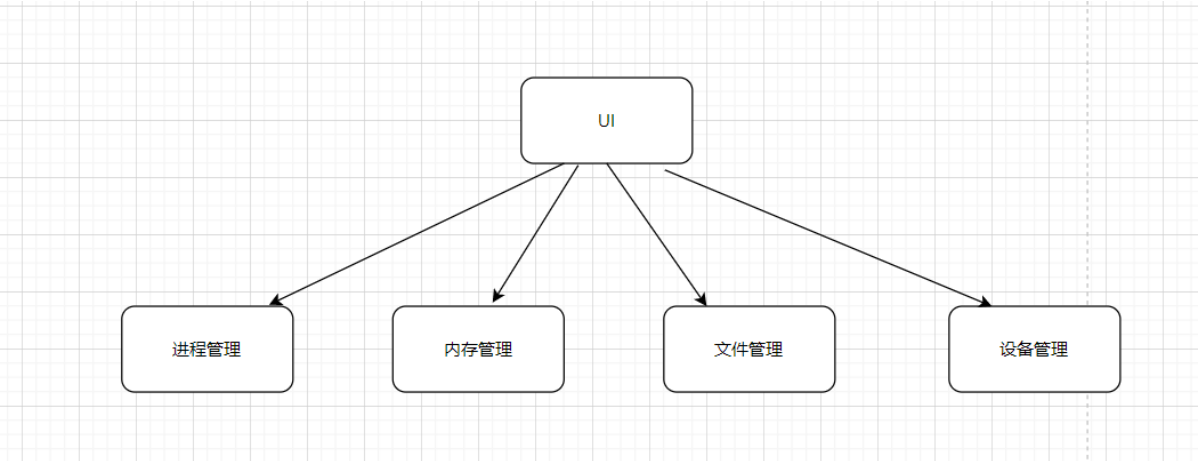
1.3 基本处理流程

系统启动时，初始化各模块，并创建用户界面。用户通过界面提交操作请求，系统根据请求调用相应模块处理，处理完成后将结果反馈给用户。

2 软件的体系结构和模块设计

2.1 软件的体系结构

2.1.1 软件体系结构框图



2.1.2 软件主要模块及其依赖关系说明

- 用户界面：与其他模块交互，接收用户请求并显示处理结果；
- 进程管理：负责进程的创建、调度、同步等功能，依赖文件存储源程序，依赖内存管理创建进程，依赖设备管理完成进程状态的转换；

- 内存管理：负责内存空间的分配与回收，依赖文件和进程分配内存，依赖设备管 理中的磁盘管理把文件读入内存；
- 文件系统：负责文件及目录的操作，依赖磁盘管理存储文件，依赖进程管理 创建和修改删除文件，依赖内存管理把文件调入内存；
- 设备管理：负责设备资源的申请、分配、使用、释放，依赖进程管理分配 IO 设备，依赖文件管理整理磁盘内容。

2.2 软件数据结构设计

2.2.1 全局数据结构说明

- PCB 表：用于存储系统中所有进程的 PCB；
- 设备表：用于记录系统中所有设备的信息；
- 文件控制块表：存储文件系统中所有文件的 FCB。

2.2.2 数据结构与系统单元的关系

说明各个数据结构与访问这些数据结构的各个模块级产品组件的分配之间的对应关系，可采用如下的矩阵图的形式：

	进程管理	文件系统	设备管理
PCB	√		
FCB		√	
设备表			√
设备请求队列			√

2.3 软件接口设计

2.3.1 外部接口

- 用户界面 API：用于与用户界面交互，接收用户操作并返回处理结果；
- 文件系统 API：用于操作文件和目录；
- 设备管理 API：用于申请、分配、使用、释放设备资源。

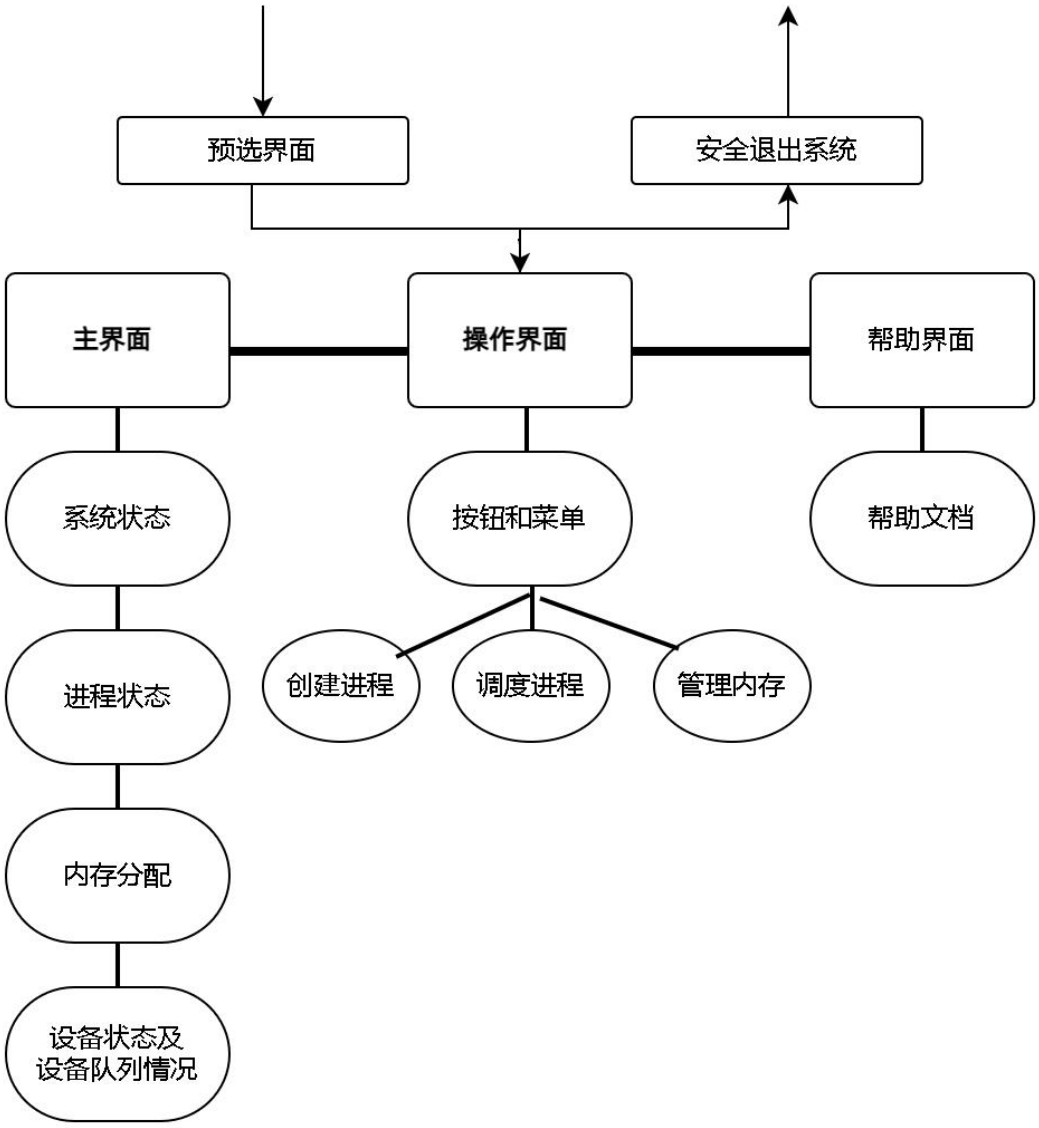
2.3.2 内部接口

- 进程管理内部接口：用于实现进程创建、调度等功能；
- 内存管理内部接口：用于内存分配和回收；
- 设备管理内部接口：用于设备申请、分配、使用、释放。

3 用户界面设计

3.1 界面的关系图

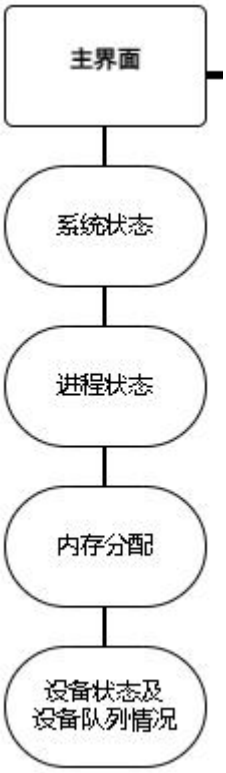
以图形界面展示多道程序并发执行的过程，界面包括：
- 主界面：显示系统状态、进程状态、内存分配情况等；
- 操作界面：提供用户进行操作的按钮和菜单；
-帮助界面：提供使用系统的帮助文档。



3.2 界面说明

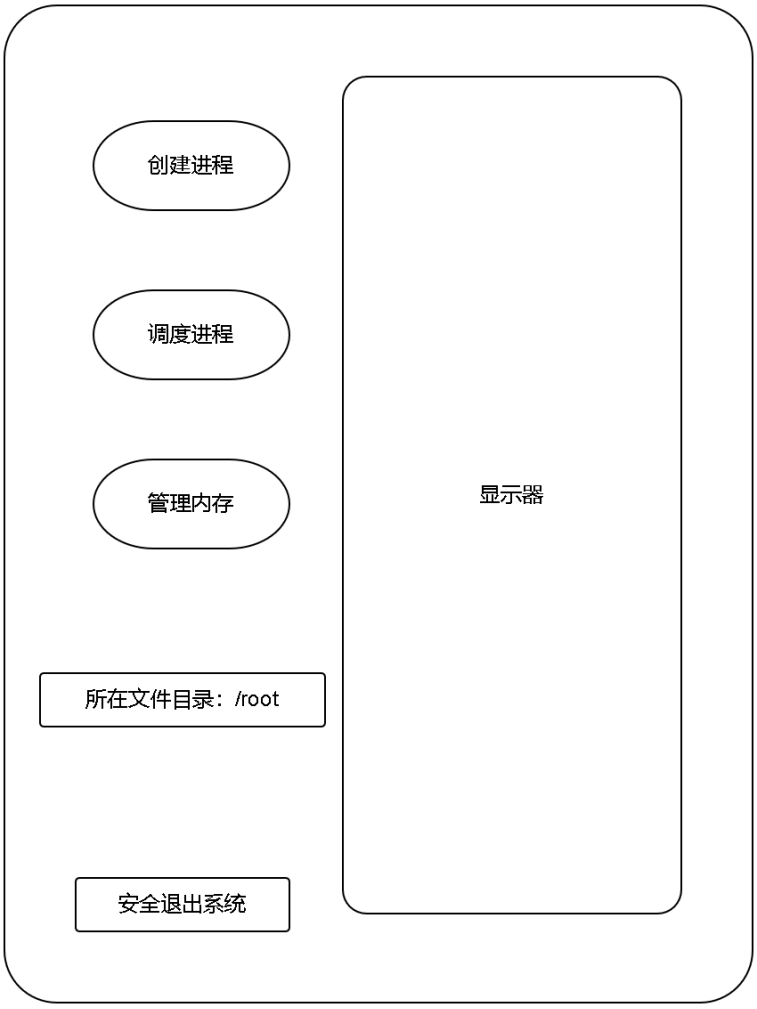
3.2.1 界面 1：主界面 1

- 显示系统状态、进程状态、内存分配情况；
- 显示设备状态及设备队列情况。



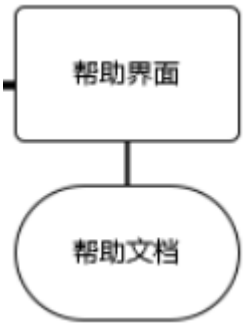
3.2.2 界面 2：操作界面

- 提供按钮和菜单，让用户可以进行创建进程、调度进程、管理内存等操作。



3.2.3 界面 3：帮助界面

- 提供帮助文档，详细介绍各个功能如何使用。

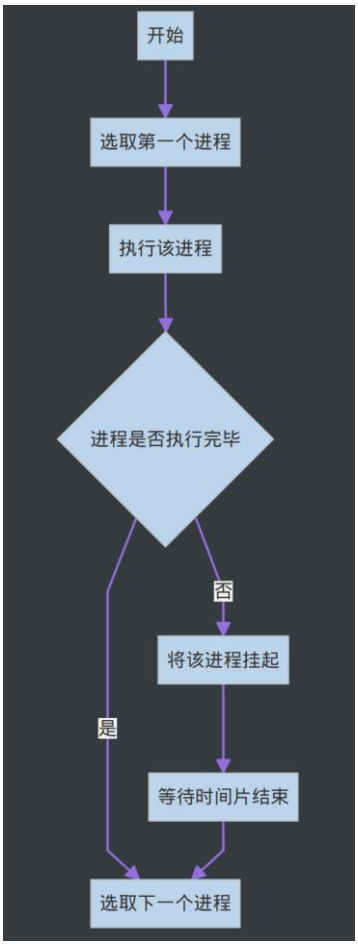


4 相关处理流程

本章逐个地给出各个层次中的每个程序单元的设计考虑。

4.1 进程管理设计说明

4.1.1 进程管理算法及流程说明



一.进程调度算法

我们设计的操作系统提供两种进程调度算法。

1) FCFS 算法

先请求 CPU 的进程先分配到 CPU。

该算法利用 FIFO 队列来实现：当一个进程进入到就绪队列，其 PCB 链接到队列的尾部。当 CPU 空闲时，CPU 分配给位于队列头的进程，接着该运行 进程从就绪队列中删除。

2) SJF 算法——支持抢占、非抢占两种模式

具有最短 CPU 区间的进程先分配到 CPU，若两进程具有相同的 CPU 区间长度，则按照 FCFS 原则来调度。

在非抢占模式下，只有当 CPU 空闲时才会使用调度算法从就绪队列中选 择进程来运行；而在

抢占模式下，CPU 空闲、有新的进程加入到就绪队列两种情况都会启用调度算法。

二. 进程同步的实现

在我们设计的操作系统中，多个进程的临界区问题发生在对同一文件的读写操作时。为了解决这一问题，对每一个文件都设置一个状态量 is_writing，当 is_writing == 1 时表示有进程正在对该文件进行“写”的操作，此时不允许其他进程再次访问 该文件。当“写”的操作结束后，将该状态量更改为 0，以此避免并发访问过程中可能的冲突。

三. 中断机制

每个进程管理块 PCB 都包含一个指令寄存器，记录该进程正在执行的指令。 在我们的操作系统中，发生中断的情况有以下几种：

1.时间片中断——时间片用尽但进程并未结束，指令寄存器的值修改为下一条指令，将该进程移入就绪队列重新等待调度，等到再次被调度时从指令寄存器显示的指令处开始执行。

2. I/O 中断——修正当前进程指令寄存器的值，将该进程由运行态移入阻塞态，等待 I/O 请求完成后将其移入就绪队列等到再次被调度时从指令寄存器显示的指令处开始执行。

3. 缺页中断——当前进程需要访问磁盘中的文件，修正当前进程指令寄存器的值，将该进程由运行态移入阻塞态，等待文件系统的返回信息，此后过程同上。

对于每一种中断情形都设置对应的中断处理程序，并制成中断向量表。

4.1.2 数据存储说明

进程的代码以文件形式存储，扩展名为.exe，里面包含一个进程要执行的指令。

4.2 内存管理设计说明

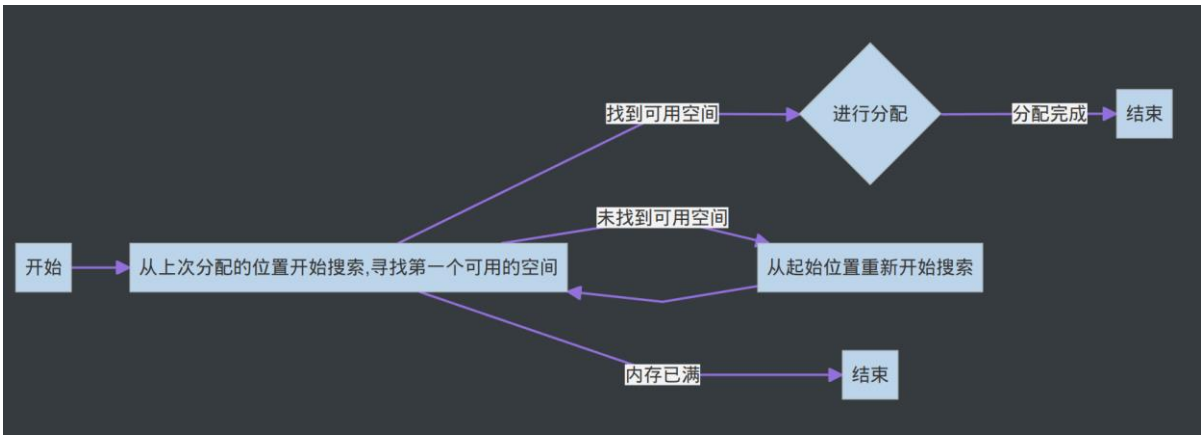
4.2.1 内存管理数据结构说明（可选）

内存分配表

4.2.2 内存管理算法及流程说明

内存分配和回收算法

内存分配:
循环首次适应算法



内存回收：
引用计数算法

4.2.3 数据存储说明（可选）

内存分配状态存储在内存分配表中

4.3 文件管理设计说明

4.3.1 文件管理数据结构说明（可选）

①模块内的全局变量

```
static int MAX_INDEX; //最多有多少直接索引
```

②

```
class FCB
{
    string name; //文件名，在每个目录下唯一
    char type; //目录(.f)、程序文件(.e)、用户文件(.t)
    pointer; //从根目录开始的绝对路径，指向所在的目录
    int size; //文件长度，每次写操作后都需要更新
    int auth; //权限，无限制（auth=0），不可删除（auth=1），只读（auth=2）
    vector<int> index; //索引，第 i 个元素表示文件第 i 块地址，最多 MAX_INDEX 块
    int singleIndirect; //一级间接索引，对大文件可用
```

}

③

```
class OftEntry
{
    FCB file; //文件控制块副本，包含文件信息
    int readFlag; //读文件标识符，计数信号量，标识正在读的进程数量
    int writeFlag; //写文件标识符，二元信号量，一次只允许一个进程写入
```

成员函数包括更改两个信号量的函数，以及返回 FCB 中的一些信息

```
}
```

④class OpenFileTable

```
class OpenFileTable
{
    vector<OftEntry> OFT; //打开文件表的主体，包括所有打开文件的 FCB 副本、读写情况
成员函数：新增/删除条目
}
```

⑤目录 逻辑上：树形结构，根节点为根目录，每个节点的孩子标识目录下的文件，叶子结点 标识可执行文件或文本文件
存储上：哈希表

```
class directoryTree
{
    unordered_map<string, void *> directoryTree
    每个目录下的内容用哈希表存储，key 是文件名，内容是一个无类型指针：
    若文件为目录，则指针类型为 unordered_map*，指向另一个哈希表；
    若文件为.t 或.e，则指针类型为 FCB*，指向文件控制块
}
```

⑥空闲空间表 vector<int> freeSpaceList
用位向量 bit vector 实现，位数等于磁盘块数，每位标识一个磁盘块,每位只能为 0 或 1，0 表示块不可用，1 表示块为空

⑦磁盘 暂定为 Vector 的三层数组 Disk[i][j][k]：标识第 i 个柱面第 j 道第 k 分区。
暂定每个磁道扇区数量相同，且一个文件块对应一个扇区

4.3.2 文件管理算法及流程说明

一、 存储逻辑结构

选用索引分配，为简化操作，将索引块并入 FCB。

每个索引块的最后一条为一级间接块。考虑到本 OS 规模，应该不需要更多层次的间接块。

与真实操作系统不同的是，真实 OS 需要内存先读入索引再根据索引读文件，但因为我们限制了文件操作的复杂度，所以文件系统会遍历索引段将文件块全部载入内存。

二、 存储物理结构

磁盘寻道算法：SCAN,优先考虑磁头的当前移动方向，并且考虑当前磁道与下一磁道之间的距离。例如，当磁头正在自里向外移动时，扫描算法所选择的下一个访问对象应该是，即在当前磁道之外，且距离最近，直到再无更外的磁道需要访问才将磁头换向，自外向里移动。移动原则同前一致。

4.3.3 数据存储说明（可选）

一方面，磁盘具有非易失的特点，要求每一次重启操作系统，磁盘上的数据不应丢失或损坏；另一方面，频繁的文件操作效率低、性能差。因此我们决定在每一次关闭操作系统时，将文件树和所有文件以文件的形式写入程序所在文件夹；重新启动时，作为初始化，将文件树和存储的文件载入内存，后续的磁盘读写操作在程序内存中实现。

需要存储的文件：目录结构、对应每个文件的 txt 文档。

4.4 4.4 设备管理设计说明

4.4.1 设备管理数据结构说明

系统设备表：

```
struct SDT {

    int nd;        // 设备数量
```

```
    char *fpdata     // 设备配置文件
    DCB *pd;         // 设备控制块表指针
};
```

设备控制块：

```
struct DCB {
    char *name;       // 设备名称
    int id;           // 设备标识符
    int type;         // 设备类型
    int status;       // 设备状态
    int pid;          // 获得设备进程的进程号
};
```

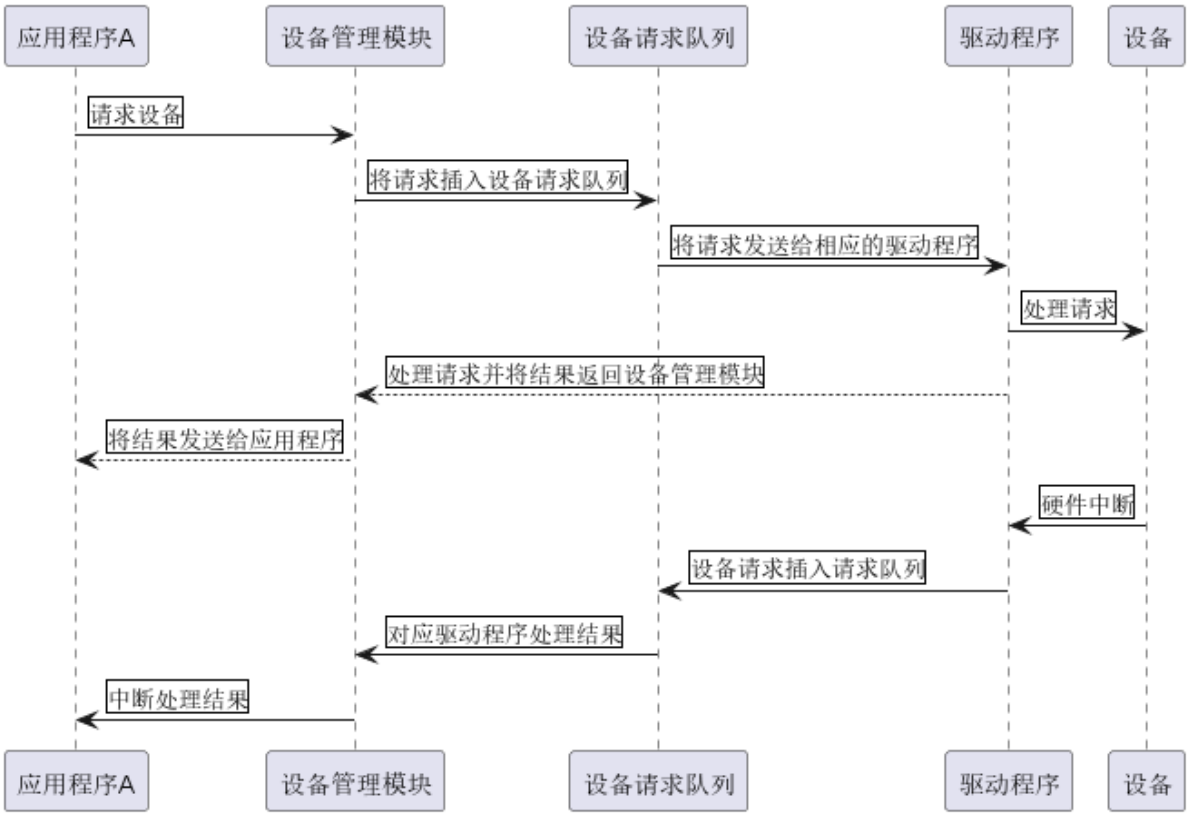
设备请求：

```
struct DeviceRequest {
    int pid;       // 请求进程号
    int id;        // 请求设备号
    char *data   // 请求内容
};
```

队列/循环队列	用于构造设备请求队列
哈希表	存放设备控制块指针，快速查找对应设备信息

4.4.2 设备管理算法及流程说明

设备管理模块流程：如图



流程说明：

设备管理模块的主要功能是管理计算机系统中的各种设备和响应并处理硬件中断，并为应用程序提供简单的接口来访问设备。

当应用程序需要访问设备时，它会向设备管理模块发送请求。
设备管理模块会根据设备请求的类型和优先级，将请求插入到设备请求队列中，
然后将请求发送给相应的驱动程序。
在驱动程序完成设备请求后，它会将结果返回给设备管理模块。
设备管理模块将结果发送给应用程序，然后将设备请求从设备请求队列中删除。

当设备产生硬件中断。对应驱动程序（中断处理程序）会对中断进行处理，并将对应设备请求插入请求设备队列，并按照调度顺序将结果返回给设备管理模块。设备管理模块将处理该结果并将设备管理模块处理结果发送给应用程序，然后将设备请求从设备请求队列中删除。

优先级调度策略：

通过设置不同设备的处理优先级，优化设备处理顺序，增强系统可用性。

对于优先级相同的设备请求，采用 FCFS 策略。

4.4.3 设备管理数据存储说明

设备信息及配置在文件中持久化保存

5 总结

本设计实现了一个具有操作系统基本功能的模拟程序，采用模块化设计，各模块独立完成各自的功能，通过接口进行交互。用户界面友好，方便用户操作。系统主要包括进程管理、内存管理、文件系统和设备管理等功能。通过本设计，我们可以更好地理解操作系统的基本原理和实现方法，为进一步研究操作系统提供了一个实践的基础。

2. 参考文献

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts (10th ed.). John Wiley & Sons.
2. Tanenbaum, A. S., & Bos, H. (2014). Modern Operating Systems (4th ed.). Pearson.
3. Stallings, W. (2018). Operating Systems: Internals and Design Principles (9th ed.). Pearson.
4. Elmasri, R., & Navathe, S. B. (2016). Fundamentals of Database Systems (7th ed.). Pearson.