进程管理系统计划文档

一、需求分析

1. **进程管理核心功能**：
   - 进程创建、阻塞、唤醒、挂起、激活和终止
   - 指令的按时间顺序执行与上下文切换
   - 内存和设备资源的分配与释放
   - 进程状态的实时更新与监控
2. **输入/输出规范**：
   - 输入：进程的指令文件（包含初始化参数和指令序列）
   - 输出：进程执行结果及状态变化日志

二、模块设计

1. 数据结构

- PCB结构

  ：

```
Cpp1struct PCB {
2    int PID;          // 进程唯一标识符
3    int arriveTime;   // 进程到达时间
4    int needTime;     // 总需时间
5    int remainTime;   // 剩余时间
6    int size;         // 内存需求
7    int dataSize;     // 数据段大小
8    v_address address; // 内存地址
9    State state;      // 当前状态（READY/BLOCK/SUSPEND/END）
10    int finalTime;   // 完成时间
11    stack<cmd> cmdStack; // 指令栈（按先进后出顺序执行）
12    myFile* myFile;  // 指令文件指针
13};
```

- 指令结构

  ：

```
Cpp1struct cmd {
2    int time;         // 执行时间
3    int num;          // 指令类型（CREATE/DELETE/APPLY/RELEASE/BLOCK/WAKE）
4    int num2;         // 设备ID或其他参数
5    string path;      // 文件路径（仅CREATE/DELETE需要）
6};
```

---

2. 核心函数说明

进程创建（createProc）

- **功能**：从指令文件中读取进程初始化信息并创建PCB
- 逻辑：
    1. 打开指令文件
    2. 解析前三个参数（`needTime`, `size`, `dataSize`）
    3. 验证参数合法性（时间>0，内存足够，指令格式正确）
    4. 分配内存资源
    5. 将PCB加入**进程表**和**就绪队列**

指令解析（getNum & getCmd）

- 改进点：
    - 使用`fgets`替代`strcpy`（避免缓冲区溢出）
    - 使用栈结构存储指令（先进后出）
    - 添加路径和参数的合法性验证

状态管理

- 阻塞（blockProc）：
    - 从**就绪队列**移除
    - 状态设为BLOCK
- 唤醒（wakeProc）：
    - 状态设为READY
    - 重新加入**就绪队列**
- 挂起（suspendProc）：
    - 状态设为SUSPEND
    - 释放内存和设备资源
- 激活（activeProc）：
    - 状态设为READY
    - 重新分配资源并加入**就绪队列**

调度器（scheduler）

- 实现FCFS调度算法：
    1. 每个时间片检查到达时间匹配的进程
    2. 选择优先级最高的进程执行（此处简化为先进先出）
    3. 执行指令并更新剩余时间
    4. 根据状态变化决定是否放回就绪队列或阻塞

三、功能实现要点

1. 进程指令执行规则

- 指令按`time`字段的**绝对时间**排序执行：

```
Cpp1// 需要定义比较函数（按时间顺序）
2struct Compare {
3    bool operator()(const PCB& a, const PCB& b) {
4        return a.cmdStack.top().time > b.cmdStack.top().time;
5    }
6};
```

（注：当前实现简化为按栈顶指令时间执行，需根据实际需求调整）

- **中断处理**：
  - 在指令执行过程中调用`handle_interrupt()`模拟设备中断
  - 中断触发时进程转为BLOCK状态

2. 内存与设备管理接口

- 内存分配：
  - `alloc(v_address*, int size, int pid)`：分配连续内存空间
  - `free(v_address address, int pid)`：释放已分配内存
- 设备控制：
  - `acquire(int pid, int deviceID)`：申请设备
  - `release(int pid, int deviceID)`：释放设备

四、测试计划

1. 单元测试

- 测试用例：

情景	操作	预期结果
正常创建进程	输入合法指令文件	创建成功，PCB加入进程表和就绪队列
内存不足创建	输入`size < dataSize`	输出错误提示，创建失败
非法指令类型	输入`num=5`	输出错误提示，指令读取失败
阻塞未就绪进程	对BLOCK状态进程调用blockProc	输出错误提示"状态错误"
挂起非阻塞/就绪进程	对RUNNING状态进程调用suspendProc	输出错误提示"状态错误"

2. 集成测试

- 多进程协作测试：
  - 进程A通过`BLOCKCMD`阻塞进程B
  - 进程B通过`WAKE`指令被唤醒后继续执行
- 资源竞争测试：
  - 进程同时申请同一设备，验证`acquire`是否正确阻塞进程