项目名称：**SimpleKernel-编写实现文档**

项目详细方案：

目前的实现文档分布在开发日志、代码注释中，有的部分还有缺少，因此需要一份文档告诉使用者我们的代码的组织结构以及其背后的原理。

所编写文档主要包含两个模块：

1. 编写从计算机启动到多任务系统为止的实现文档

2. 从软件硬件两个角度讲

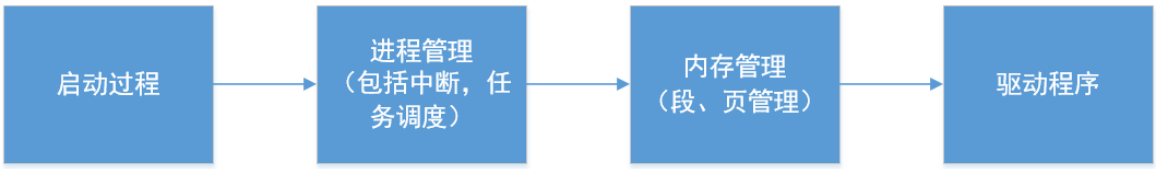
计划所撰写文档的组织流程：

1.宏观描述操作系统从boot启动到最后进入运行的流程等相关背景知识

2.操作系统和硬件交互的结构介绍

3.结合代码进行流程描述

4.分模块进行描述，包括操作系统任务管理，内存管理，中断设置，以及结构相关代码，基本上实现每个文件夹下都要有讲解。



进度：

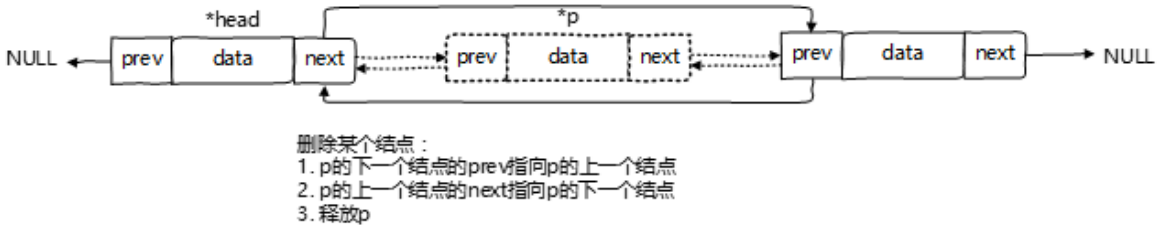
前期：2020.6-2020.7 编译代码，尝试用单步调试工具在感性上认识代码运行流程，并阅读源码。

中期：2020.6-2020.8 使用gitbook进行文档撰写，编写从计算机启动到多任务系统为止的实现文档，结合软硬件实现

后期：2020.8 -2020.9 完善文档，校正错别字以及不规范的表述，实现阅读文档就可以使得初学者能够理解操作系统运行流程，试图找几个初学者进行试读得到反馈并修改。

例子：

在操作系统中需要用到很多数据结构，其中最常用的就是链表，链表一般是内核中最常用最普通简单的数据结构，一般为包括单向链表和双向链表。链表是一种能够存放和操作可变数量元素的数据结构，其最大优势在于能够动态插入和删除，在进程管理中通过使用链表来管理进程。



这里可以看到在src\ds_alg的目录下有双向链表描述，_ListEntry为节点的描述node，其中包含三个成员，分别是data，prev和next。其中data用于存储数据内容，prev和next分别用于对节点进行操作。对于双向链表的操作方式可以参考其中代码实现。下面用进程的管理来描述下双线链表在操作系统中的应用。

举例：在进程中对进程的管理需要通过进程process control block(pcb)这个结构体来进行管理，参考文件src\include\task\task.h中task_pcb结构体，而这个**进程控制块是通过双向链表来进行操作**，可以打开这个文件看:

```
typedef
    struct task_pcb {
    // 任务状态
    volatile task_status_t      status;
    // 任务的 pid
    pid_t pid;
    // 任务名称
    char * name;
    // 当前任务运行时间
    uint32_t run_time;
    // 父进程指针
    struct task_pcb *       parent;
    // 任务的内存信息
    task_mem_t * mm;
    // 任务中断保存的寄存器信息
    pt_regs_t * pt_regs;
    // 任务切换上下文信息
    task_context_t *        context;
    // 任务的退出代码
    int32_t exit_code;
} task_pcb_t;
```

这个task_pcb结构体中有进程的状态，进程pid，进程名，进程运行时间，以及他的父进程指针，内存信息，中断保存的寄存器信息，以及任务切换上下文信息以及任务退出代码。而在进程控制块中有两个毕竟重要的结构体，task_context_t和task_mem_t。

task_context_t代表进程切换时候的上下文的寄存器，这部分寄存器的定义是根据x86架构中需要保存的寄存器地址而来的。

```
typedef
    struct task_context {
    uint32_t    eip;
    uint32_t    esp;
    uint32_t    ebp;
    uint32_t    ebx;
    uint32_t    ecx;
    uint32_t    edx;
    uint32_t    esi;
    uint32_t    edi;
} task_context_t;
```

task_mem_t为进程的内存地址结构，进程内存地址结构的可以参考后文中xx章内容，涉及到段页寻址等会着重讲解这个。

```
// 进程内存地址结构
typedef
    struct task_mem {
    // 进程页表
    pgd_t *     pgd_dir;
    // 栈顶
    ptr_t       stack_top;
    // 栈底
    ptr_t       stack_bottom;
    // 内存起点
    ptr_t       task_start;
    // 代码段起止
    ptr_t       code_start;
    ptr_t       code_end;
    // 数据段起止
    ptr_t       data_start;
    ptr_t       data_end;
    // 内存结束
    ptr_t       task_end;
} task_mem_t;
```

Reference：

[1] https://blog.csdn.net/bingfeilongxin/article/details/88654997

[2] https://www.cnblogs.com/fanweisheng/p/11138557.html

[3] 关于任务管理的资料见 intel 手册 3ACh7