| 北京邮电大学<br>操作系统课程设计项目文档 | 产品名称 | 产品版本 | 密级 |
| --- | --- | --- | --- |
| | MiniOS | V1.0 | 公开 |
| | 提交日期： 2020 年 6 月 12 日 | | 共 28 页 |

MiniOS Ver 1.0

# 操作系统课程设计

## 项目中期进度汇报

学　　院：计算机学院

专　　业：计算机科学与技术

班　　级：2017211319 班

指 导 教 师：孟祥武老师

# 目录

# 1 项目概述

## 1.1 工作内容

- **明确项目解决方案与路线**

团队明确解决方案具体内容，并构建系统主要框架。组长安排组员进行分工协作，进行具体内容的详细说明与展开；各成员完成方案的编写，由组长进行汇总后在团队内发布，所有成员仔细阅读并提出意见、统一修改。

- **进行计划跟踪与监督**

按照作业内容与作业时间安排，结合组员时间安排具体的项目开放计划。组员严格执行计划安排，由组长定期跟踪、统一项目进度，实行小组内成员互相监督的合作机制。

- **依据项目预期目标进行系统开发**

详细内容见本方案的“预期目标”部分，该部分内容根据课程安排由团队共同制定。

- **按要求向老师反馈开发进度与问题**

团队应做到及时与老师沟通开发过程中遇到的关于项目问题，或者是团队在开放过程中遇到的技术难点，做到按要求与老师交流项目开发进度及完成情况。

- **系统产品及时进行受控管理**

对我们的系统产品进行受控管理，根据老师提供的设计要求，以及团队对于操作系统所应具备功能的调研，对我们的需求管理进行及时的更改和完善，确保系统的完整性，规避技术和质量的风险。

- **按要求接受阶段性评审检查与相关文档的提交**

团队在开发过程中应根据课程要求接收评审或检查（如日常审查、中期审查等），并且团队应注意在设计与编码阶段累积相关文档，在项目结束时做到产品、文档兼备。

- **项目验收与总结**

## 1.2 开发路线及预期目标

根据本次课程设计的相关要求，小组当前的整体工作与实现目标在于实现一个具有三大基本功能的操作系统模拟程序，该程序需要能够模拟现代操作系统所具备的三部分功能：

① 进程管理；
② 内存管理；
③ 文件管理。

该程序应能够支持在裸机上独立运行，或作为高层应用，能够支持多平台的运行（例如 Windows 操作系统环境、OpenEuler 操作系统环境）。除了程序外，小组还应当为本次完整的开发过程配备相应的说明性文档。

在第一次会议后，小组内部对本系统的开发路线以及设计目标进行了确定，即采用**模拟**的方式，将现代操作系统作为低层支撑架构，在其之上开发出精巧、高效的模拟操作系统程序，该程序的地位与其他计算机上的应用程序是一致的。模拟操作系统应具备清晰、精巧的用户交互能力，并将工作重心置于系统内部的具体实现算法（如进程调度算法、内存分配算法等）以及系统功能的完整性、多样性。

经过小组内部的协商，我们将采用 Python 3 作为模拟操作系统的开发语言。Python 语言具有简易、跨平台等优点，使得团队能够更加专注于算法与系统功能的设计。因此，以模拟系统为根本目标、将 Python 作为开发语言、注重算法与系统功能的设计是本项目团队的基本开发路线。

本项目的目标产出及成果的相关信息如下所示：

- **系统名称**
  MiniOS Ver 1.0
- **系统属性**
  跨平台、多功能的模拟操作系统程序
- **编程语言**
  Python 3
- **系统功能**

对于用户而言，本系统应具有良好的交互功能，包括对文件进行操作、模拟提交批处理任务（如打印任务、计算任务等）或运行普通程序的功能；对于系统开发人员而言，本系统应具备完整的进程管理、内存管理与文件管理功能，其中这三大功能中不仅包含对模拟硬件设备采用高效算法进行统一管理，而且能够支持对系统各功能组件的使用情况进行日志记录与分析，例如能够采用曲线图、网格图的形式将系统在此段时间内的运行效率（包括内存占用率、磁盘块使用情况、CPU 与打印机等资源的占用情况或使用率）进行记录。

此外，本系统要能够支持对外开放清晰、友好的交互界面，方便用户的操作与使用。

- **系统风格**

本系统应参考 Linux、Windows 等优秀的现代操作系统源代码或实现成果。经组内讨论，我们将把系统设计工作聚焦于内核功能集成开发，并对外提供简约、清晰的命令行交互界面(CLI, Command-Line Interface)，形成自己独特、精巧的风格。

- **项目文档**

本项目在开发过程中除程序外，还应根据客户需求与实际情况提供相关文档，具体文档内容需要根据要求进一步拟定。团队在全过程中应注意文档的累积，为项目的顺利进行和后期的维护工作打下良好的基础。

本项目的实施过程应当严格按照软件工程方法对系统程序进行设计与开发，团队的开发过程将主要分为前期准备、需求分析与系统设计、编码与测试、部署与维护这四个阶段，其中在第二个阶段团队应进行系统的概要设计与详细设计。

## 1.3 团队成员信息与总体分工情况

本项目的开发团队由北京邮电大学计算机学院的六名三年级本科生组成，所有成员的分工情况经成员个人选择以及组内所有成员的共同商讨与协调，在第一次小组会议后，经过前三周的系统设计与开发阶段组内对各成员的分工进行了调整。当前小组内各成员的基本信息与总体分工安排如下表所示：

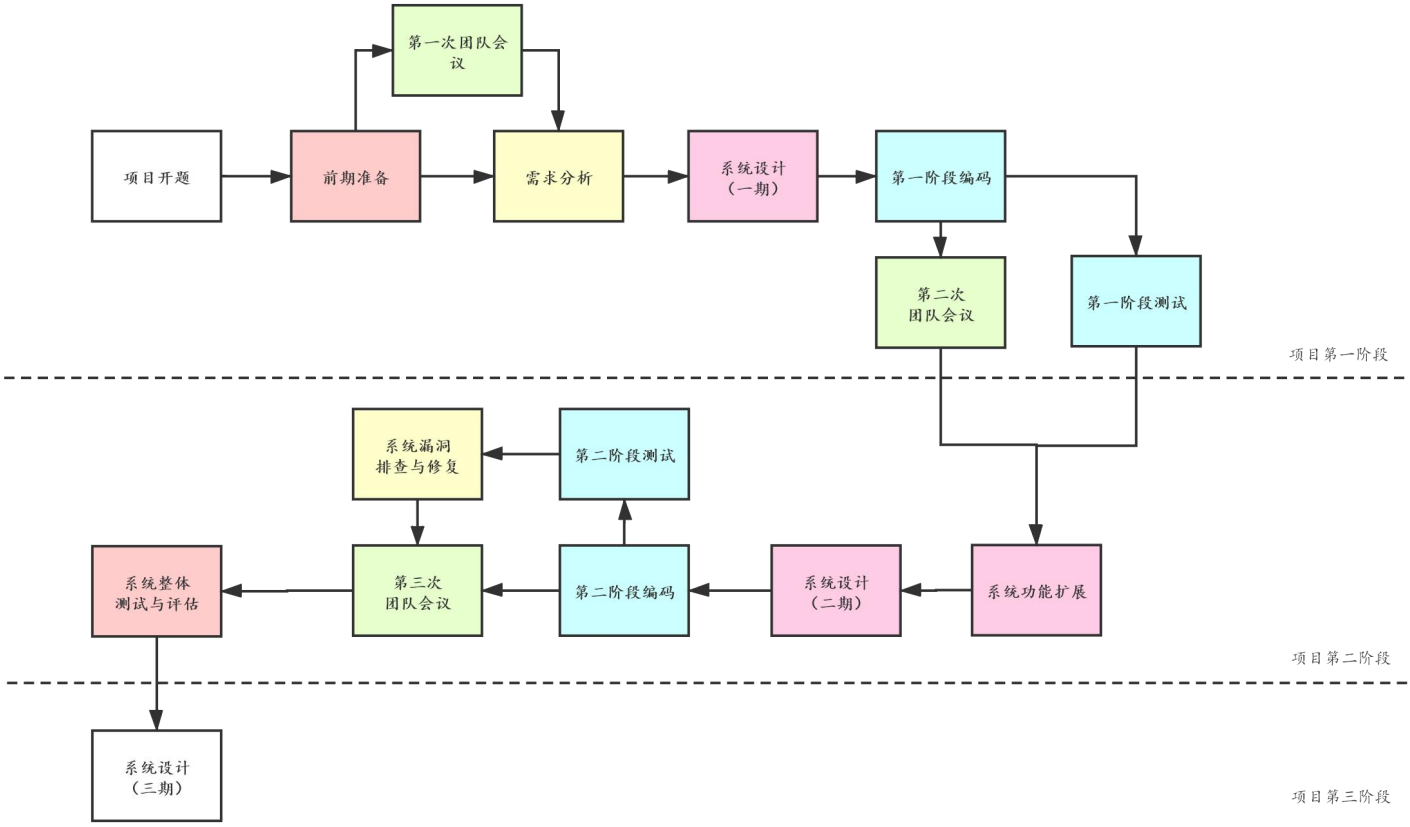
| 专业/班级 | 成员姓名 | 学号 | 成员分工 | 备注 |
|---|---|---|---|---|
| 计算机科学与技术/2017211319 | 陈斌 | 2017211661 | 系统设计、内核与 Shell | 小组组长 |
| | 吴桐子 | 2017211710 | 文件管理模块 | 小组成员 |
| | 许浩然 | 2017212193 | 文件管理模块 | |
| | 张炜晨 | 2017211835 | 进程管理模块 | |
| | 郑苹 | 2017211934 | 进程管理模块 | |
| | 周雯笛 | 2017211769 | 内存管理模块 | |

其中每位成员需要负责对应模块的代码书写、文档拟定等工作，并与负责其他部分的成员保持密切的联系与交流。小组在将各个模块独立调试完成后进行组装测试，最后小组需要将系统的各部分源代码、文档进行整合，由组长进行进一步的统一管理与审核，并根据最新进度开展小组会议，作出下一步的工作指导与计划安排。此外，系统开发全过程将在老师的指导与评审下进行。各小组成员的具体工作内容将在下文中进行详细描述。

# 2 项目当前进展

## 2.1 整体进度概览

本团队在课程项目开题后立即投入到设计与开发工作中，当目前为止，团队共完成了项目两个阶段的开发、三次组内会议，并在每次开发后均进行了系统的测试与问题的排查。本项目目前为止所经历的开发阶段及各阶段完成的具体内容可由下图进行归纳与示意：




下面本文将围绕本项目当前结构，对系统中的各组成部分以及整体情况进行详细介绍。

## 2.2 系统基本结构与模块划分

本系统经团队成员共同商议，由组长提出、组员们共同认可，我们得到了目标系统程序的基本结构，并经过修正与完善，本系统的组成有了更加清晰的划分。

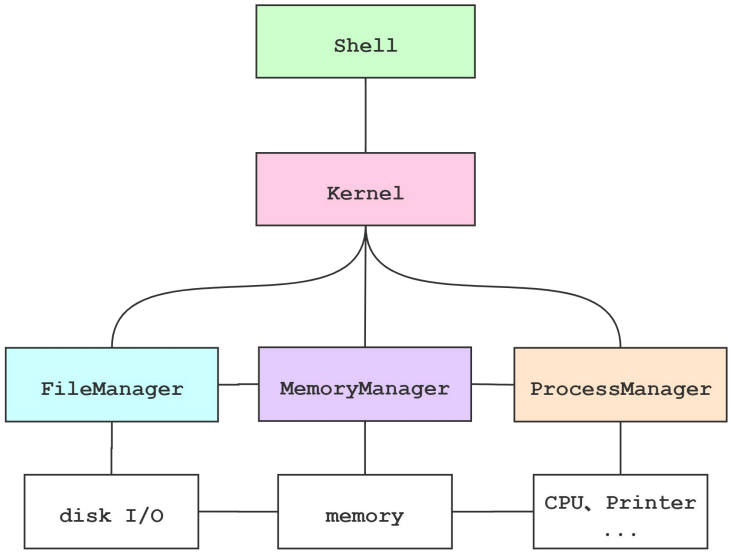
具体地，当前 MiniOS 可分为四个组成部分，它们分别对应四个不同的模块：Shell、Kernel、File Manager、Memory Manager、Process Manager。其中 Shell 是用户与 MiniOS 沟通的桥梁，其在我们设计的系统中既是一种命令语言的存在，又是第一个在本系统之上运行的默认应用程序。Shell 提供了一个界面，用户通过这个界面访问操作系统内核的服务；Kernel 是 MiniOS 的核心部分，其实际上又由本身以及剩余的三个模块构成，负责管理系统的进程、内存、设备、文件等，决定着系统的性能和稳定性。

进一步划分，Kernel 实际上能够调用本系统剩余的三大核心模块。其中 File Manager 负责对系统文件进行文件的逻辑组织和物理组织、文件树的结构和管理。所谓文件管理，就是操作系统中实现文件统一管理的一组软件，亦或是被管理的文件以及为实施文件管理所需要

的一些数据结构的总称。在我们设计的系统中，该模块还负责与模拟磁盘设备进行交互，实现文件在物理存储体上的读写，并负责对空闲磁盘块的调度；另外，其具有驱动功能，能够对模拟磁盘设备进行直接控制，接管了与外存交互的所有事务。

对于 Process Manager，由于在 MiniOS 中，进程是正在运行的程序实体，也包括这个运行的程序中占据的所有系统资源，比如说 CPU、打印机等。而这一模块能够实现对这些实体对系统核心资源使用的管理，如采用高效的调度算法（如优先级调度算法、高响应比调度算法等）以及通过维护 PCB（进程控制块）结构来实现对各进程实体关键信息的记录，以及使用资源的合理调度。

以上是对当前系统各模块的简单介绍，具体内容将在下面的小节中详细展开。这里本文给出当前系统的整体结构与模块划分示意图，如下所示：




其中连线表示存在调用或耦合关系。上图中的白色方框内所表示的为硬件资源，可以看到，三大内核功能模块均分别主要对其中一类资源进行管控。对以上文字内容进行总结，我们可以得到各模块的整体功能或职责，将其归纳为如下文所示的表格：

| 系统模块 | 属性、功能或职责 |
| --- | --- |
| Shell | 为用户提供与系统进行交互的命令行形式的界面接口；对用户命令进行初步处理（如分割、转换等）；提供诸如缓存命令等机制，使得用户的使用体验以及使用效率能够得到大幅提升 |
| Kernel | 是整个系统最重要、最核心的程序；调度其之下所管辖的三大系统功能模块，处理计算、打印等任务请求，实现对系统资源的充分利用以及高效、统一的管理 |
| File Manager | 实现系统与外存的信息交换；构建文件系统，包括文件逻辑结构与树状存储结构，将系统中的所有文件进行组织；负责驱动并管理磁盘设备，指挥其进行指定磁道、扇区的块的读写；通过使用高效的算法实现对资源的合理利用；提供访问文件系统的应用接口；实现对磁盘等外设资源的监控 |
| Memory Manager | 围绕内核发出的分配或释放请求，对内存资源进行管理；实现虚拟内存机制，其中包括逻辑地址到物理地址的转换、运用多种算法对虚拟页进行替换操作等；对虚拟地址空间、物理内存等资源进行跟踪与管控 |

| | |
|---|---|
| Process Manager | 针对进程实体，通过 PCB、队列等数据结构实现对这类实体的内部及实体间的调配，使得计算机的核心功能资源（如 CPU、打印机等）得到高效利用；对批处理任务进行解析与转换，使得各程序的任务能够有效地被分配到各功能部件上运行；提供多种批处理任务命令格式及系统调用（如 fork、access 等），使得程序的设计更加灵活，功能更丰富；对系统核心资源的使用情况进行跟踪监控 |

## 2.3 模块内部结构与功能说明

在这一小节中，本文将围绕系统各模块的具体结构以及详细功能进行展开，向读者阐述项目当前的最新细节情况，帮助用户对系统的各大功能以及整体运行模式有更深刻的认识。

### 2.3.1 Shell 与 Kernel

这两个模块与本系统的控制功能紧密相关。Shell 通过解析用户输入的命令，向内核 Kernel 传递信息，以特定的方式调用子程序，从而完成指定功能。到目前为止，MiniOS 向用户开放了如下的系统内置命令，各命令的名称、格式、功能分别如下所示：

| 系统内置命令 | 格式 | 功能 |
|---|---|---|
| man | man [command1] [command2] ... | 展示单条或多条命令的帮助信息，若未携带具体命令，则默认对所有命令进行展示 |
| ls | ls [-a\|-l\|-al][path] | 列出指定路径 path 的内容，-a 选项列出全部内容（包括隐藏文件或目录），-l 选项列出文件或目录的详细信息，使用-al 同时指明以上两个选项。若未提供 path 参数，则默认 path 为当前目录 |
| cd | cd [path] | 修改当前工作目录，若未提供 path 参数，则默认 path 为系统根目录 |
| rm | rm [-r\|-f\|-rf] path | 删除文件或目录。其中待删除的文件或目录的路径必须提供，-r 选项能够递归地删除目录，-f 选项对应于强制删除功能，使用-rf 同时指明以上两个选项 |
| mkdir | mkdir path | 创建目录 |
| dss | dss | 展示系统外存各磁盘块的占用状态 |
| dms | dms | 展示系统物理内存占用状态 |
| exec | exec path | 执行指定路径下的文件，该文件须为可执行文件 |
| ps | ps | 展示当前系统所有进程状态 |
| rs | rs | 展示当前系统所有资源的使用状态 |
| mon | mon [-o] | 开始监控系统资源使用情况，将监视结果以图片的方式实时输出；-o 选项停止监控 |
| td | td | 整理系统磁盘外部碎片，即“紧凑”操作 |
| kill | kill pid | 强制结束指定进程 |
| exit | exit | 退出 MiniOS |

此外，Shell 支持用户在单个命令行内键入多条命令，如：① man ls; man cd，② mkdir n; cd n 等。在系统获取到用户输入的命令后，首先该命令会经过 Shell 的初步处理，如进行分割操作后，将其传递至内核，由内核调用系统内置子程序，并将对应的参数进行转化后传

入该子程序，共同完成指定功能的实现。

上示的 ps、dms、rs 命令将动态实时刷新当前系统资源占用情况，该展示子程序可以通过组合键 Ctrl 和 C 进行退出。捕捉该热键产生的信号、创建监视进程、实现动态刷新等也是由 Shell 和 Kernel 共同控制完成的。

在当前的系统中，Shell 命令已支持正则表达式功能，其旨在同时处理多个名字匹配的文件，如执行、删除等；Shell 命令还支持使用分号“;”进行分隔，支持一次性读入多条命令，顺序处理。

除此之外，Shell 和 Kernel 对命令中存在的错误进行第一层级的把控。当用户键入命令后，首先这两个模块会对命令进行初步检查，若命令名称、基本格式正确无误，则将调用指定的子程序来完成功能；否则系统将报告错误信息。特别地，对于携带过多参数的命令，如 ls path1 path2 path3，则内核将取合法前缀进行处理，而对其他内容进行忽略，因此上条命令所完成的功能实际上是列出 path1 的内容。

Shell 是系统的基本应用程序，而 Kernel 是系统核心程序，在他们的互相配合下，MiniOS 能够接收并在内核空间中处理用户输入的命令。此外，Kernel 负责对三个系统核心功能模块进行初始化操作，并通过这三大模块，实现对整个系统所有资源的统一、高效的管控。

### 2.3.2 File Manager

本模块为 MiniOS 文件系统的主要承载模块，同时负责对外存设备的驱动和读写操作，是整个系统与外存之间实现交互与控制的首要接口。下面本文将从文件结构开始，对系统的文件组织模式以及与外存间的交互过程进行展示。

#### 2.3.2.1 文件结构

MiniOS 的文件以原生操作系统的文本文件进行模拟。文本内容为 json 文本格式，右图为一个文件的内容；name 表示文件名，与原生操作系统钟该文件的文件名保持一致。文件名可以由任何字符组成，当以“.”开头时，视该文件为隐藏文件；type 表示文件属性，是一个长度为 4 的字符串。从左到右第 1 位表示文件类型，‘c’表示普通文件；‘x’表示可执行文件，“d”表示目录文件；第 2~4 位为“r”、“w”、“x”时依次表示文件的文件具有读、写、执行属性，对应位为“-”时表示不具有该属性；size 表示文件所占磁盘空间大小，单位为 byte，其值实际上代表系统模拟大小，不严格要求与 content 的实际大小相匹配；content 为模拟文件的内容。

如下所示为一个普通模拟磁盘文件“f3”的表示方式，其大小为 1000 字节，用户对其具有读、写权限，不具有执行权限：

```
{
    "name": "f3",
    "type": "crw-",
    "size": "1000",
    "content": [
        null
    ]
}
```

#### 2.3.2.2 文件组织结构

文件管理系统以文件树字典的形式将文件组织起来。一个文件对应字典中的一个元素。元素的键对应文件名，元素的值根据文件是否是目录而有所不同：当文件是目录时，目录内的所有文件组成的字典作为该元素的值；当文件不是目录时，文件的 type 字符串作为该元素的值。

采用该文件组织结构有两个好处：一是便于访问文件，当以路径的形式对文件进行某种

访问时，可以很方便地从字典中判断路径是否合法、文件是否允许该形式的访问。二是易于保持保持文件管理系统与实际文件的一致性。当 File Manager 初始化时，以与 MiniOS 同一目录下的文件夹 MiniOS_files 作为根目录，递归地遍历该目录以初始化文件树字典。之后对文件进行增删改时，同时对字典进行修改即可。

#### 2.3.2.3 模块 API

File Manager 提供了如下 API 接口：ls，cd，mkdir，mkf，rm，chmod，get_file，get_file_demo ，dss，tidy_disk。

- ls(dir_path='', mode='', method='print')：列出目标路径下的文件。dir_path，字符串类型，目标路径，支持相对或绝对路径，当目标路径不是目录时，仅列出该目标路径的文件信息。mode，字符串类型，ls 的功能模式：“-l”列出详细信息，“-a”列出隐藏文件。ls 的实现思路是：首先，分析目标路径参数，如果是相对路径，则转化为绝对路径。第 2 步，以绝对路径每一段的文件名为键访问文件树字典，找到目标路径对应字典元素的的值，即一个字典。第 3 步，分析该字典的每一个元素并按情况输出：元素的值是一个字典时，元素对应的文件是目录，以绿色输出；元素的键的字符串以 . 开头时，该文件是隐藏文件，不输出。cd，mkdir，mkf，rm，chmod 的实现思路与 ls 大致都相同，先处理路径，再扎到字典，再对字典进行操作。下文不再详细分析他们的实现思路。
- cd(dir_path='')：改变当前工作目录。dir_path，字符串类型，目标目录的路径，支持相对或绝对路径。当前工作目录以一个字符串变量记录在 File Manager 类中，cd 对其进行修改。
- mkdir(dir_path)：新建文件夹。dir_path 为字符串，即目标新建目录的路径，本系统支持相对或绝对路径。
- mkf(file_path, file_type='**crwx**', size='**0**', content=None)：创建文件。file_path，字符串类型，创建文件的路径，支持相对或绝对路径。file_type，size，content，字符串类型，指定文件的类型、大小、内容。
- rm(file_path, mode='')：删除文件。file_path，字符串类型，创建文件的路径，支持相对或绝对路径。mode，字符串类型，rm 的功能模式：为空时表示删可读的文件，‘-r’删空文件夹，‘-f’强制删文件，‘-rf’强制删文件夹。‘-rf’的实现思路：递归地对该目录进行 rm 操作，当文件是非空目录时，进入其中；当文件是空目录时，‘-r’删除；当文件是普通文件时，‘-f’删除。
- chmod(file_path, file_type)：修改文件的属性。file_path，字符串类型，创建文件的路径，支持相对或绝对路径。file_type，字符串类型，文件属性修改目标值。
- get_file(file_path, mode, seek_algo)：获得文件，成功时将绘图展示访存时磁头的移动曲线。 file_path，字符串类型，创建文件的路径，支持相对或绝对路径。mode，字符串类型，对文件操作的模式。seek_algo 为字符串类型，其含义为系统读取文件时，驱动磁道进行文件读取的算法，本系统当前支持 FCFS、SSTF、SCAN、C_SCAN、LOOK、C_LOOK 这六种经典算法。
- get_file_demo(seek_algo)：演示寻到算法优劣的展示函数，模拟访问一个位于多个磁盘块上的文件，成功时绘图展示磁头移动曲线。seek_algo，字符串类型，磁道寻道算法。
- dss()：打印外存块信息，将展示文件系统中的总容量、已分配空间、剩余空间（此处我们不考虑内部碎片部分的剩余空间，仅计算所有空闲文件块的总容量）。此外，还将展示每个外存块中的剩余空间和所属文件路径。dss 功能的效果示例将在后面内容中展示。
- tidy_disk()：磁盘碎片整理。经过磁盘碎片整理后，所有外部碎片将被清除，磁盘块内容将重新排列。通过这个方法，我们可以获取更多存放文件的空间。

#### 2.3.2.4 磁盘块分配算法

我们的系统支持 first-fit、best-fit、worst-fit 三种策略，通过对配置文件的设置，可以选

择不同的策略。而在文件系统初始化时，我们将直接调用 first-fit 将文件信息载入文件块中。

1. first-fit：即第一次找到足够的连续空间后，直接写入其中。在这里，我们通过将 bitmap 列表拼接转换为字符串的形式，通过 str.find()方法在此母串中寻找子串（即目标连续空闲块）。假设我们需要连续 5 块连续空闲块，则寻找的子串为“11111”，str.find()方法将返回找到的 index，如果没有找到，则返回-1。
2. best-fit：找到所有能够容纳文件的连续空间后，选择其中最小的写入。在这里，我们通过遍历的方法将所有空闲块的信息存于一个列表 free_blocks，然后对其进行排序，找到大小足够且最小的空间，返回该段空闲块的 index。
3. worst-fit：找到所有能够容纳文件的连续空间后，选择其中最大的写入。注意，三种方法都会引入外部碎片。在这里，我们通过遍历的方法将所有空闲块的信息存于一个列表 free_blocks，然后对其进行排序，找到大小足够且最大的空间，返回该段空闲块的 index。

如下图，在新建文件“123”之前，我们的空闲文件块列表为[0, 3, 6, 8:]。然后，我们采用 worst-fit 策略填入文件“123”，通过调用 API 中的 dss 方法对磁盘占用分布情况进行查看，可以发现文件被填充于大小足够的最大空间[8:]中。

```
total: 1228800 B,    allocated: 3072 B,  free: 1225728 B

block #0         0 / 512 Byte(s)   None
block #1       512 / 512 Byte(s)   \dir1\f3
block #2       174 / 512 Byte(s)   \dir1\f3
block #3         0 / 512 Byte(s)   None
block #4       233 / 512 Byte(s)   \f1
block #5       233 / 512 Byte(s)   \f3
block #6         0 / 512 Byte(s)   None
block #7       233 / 512 Byte(s)   \fork_test
block #8       300 / 512 Byte(s)   \123
```

#### 2.3.2.5 磁盘碎片整理算法

按照实际上的做法，应该是将文件块一一前移，以消除非空闲块间的空闲块碎片。但是，在我们的模拟系统上，其实不需要如此麻烦，只需要重新载入、分配一次文件系统的文件块信息即可（这个分配是紧密排列的）。于是，我们直接在 tidy_disk()函数中调用_init_blocks()函数进行磁盘碎片整理，在此就不详述了。磁盘整理的效果如下：

- **整理前：**

```
total: 1228800 B,    allocated: 3072 B,  free: 1225728 B

block #0         0 / 512 Byte(s)   None
block #1       512 / 512 Byte(s)   \dir1\f3
block #2       174 / 512 Byte(s)   \dir1\f3
block #3         0 / 512 Byte(s)   None
block #4       233 / 512 Byte(s)   \f1
block #5       233 / 512 Byte(s)   \f3
block #6         0 / 512 Byte(s)   None
block #7       233 / 512 Byte(s)   \fork_test
block #8       300 / 512 Byte(s)   \123
```

- **整理后：**

```
total: 1228800 B,    allocated: 3072 B,  free: 1225728 B

block #0       512 / 512 Byte(s)   \dir1\f3
block #1       174 / 512 Byte(s)   \dir1\f3
block #2       233 / 512 Byte(s)   \f1
block #3       233 / 512 Byte(s)   \f3
block #4       233 / 512 Byte(s)   \fork_test
block #5       300 / 512 Byte(s)   \123
block #6         0 / 512 Byte(s)   None
block #7         0 / 512 Byte(s)   None
block #8         0 / 512 Byte(s)   None
block #9         0 / 512 Byte(s)   None
```

#### 2.3.2.7 磁头寻道算法

在 MiniOS 中，我们从磁盘中读取文件时有六种算法可供系统人员配置：

1. FCFS，先来先服务策略，按照服务到来的顺序依次访问磁道。

2. SSTF，最短寻道时间优先策略，下一个寻道目标总是与当前磁头位置最近。

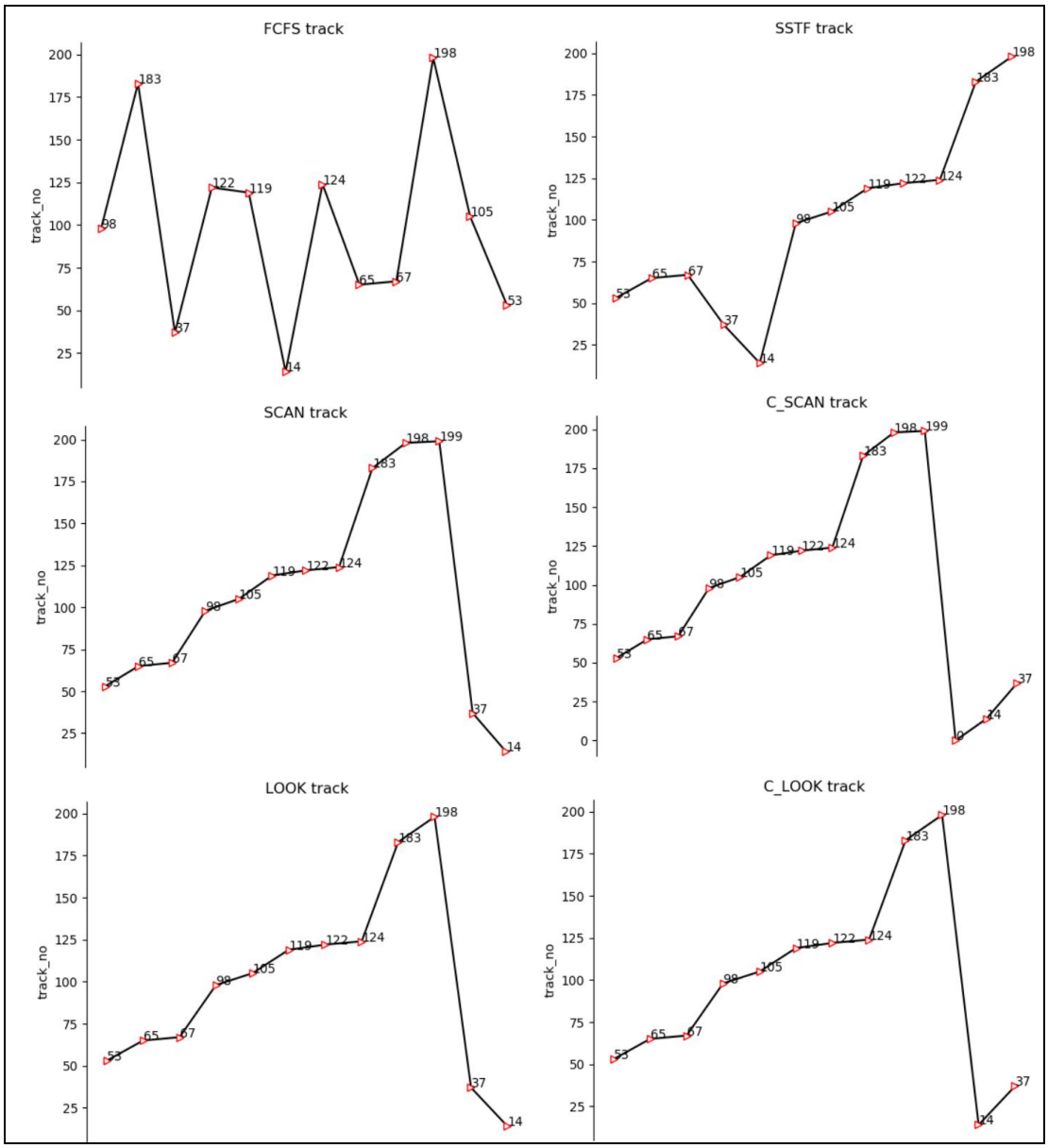
3. SCAN，扫描算法，向着磁盘的大端扫描，到边缘后再反向扫描，直到处理完服务。

4. C-SCAN，循环扫描，向着磁盘的大端扫描，到边缘后返回磁盘的小端，再重新向着磁盘的大端扫描，直到处理完服务为止。

5. LOOK，不触及端点的扫描算法，向着服务队列中最大的磁道号正向扫描，到达后再向着服务队列中最小的磁道号负向扫描。

6. C-LOOK，不触及端点的循环扫描算法，向着服务队列中最大的磁道号正向扫描，到达后磁头返回服务队列中最小的磁道号，再正向扫描，直到处理完服务。

设当前系统收到如下所示的寻道请求序列：98，183，37，122，119，14，124，65，67，198，105，53。初始磁头位置设置为 53，分别以以上六种寻道算法进行磁盘读取，系统可以对磁头进行跟踪记录，获得的磁头寻道轨迹如下所示：



### 2.3.3 Memory Manager

本模块实现对系统内存资源的统一调配与管理，围绕 Kernel 的请求，通过虚拟内存机制，对可执行文件在逻辑地址空间内的进行有效的装载与释放。

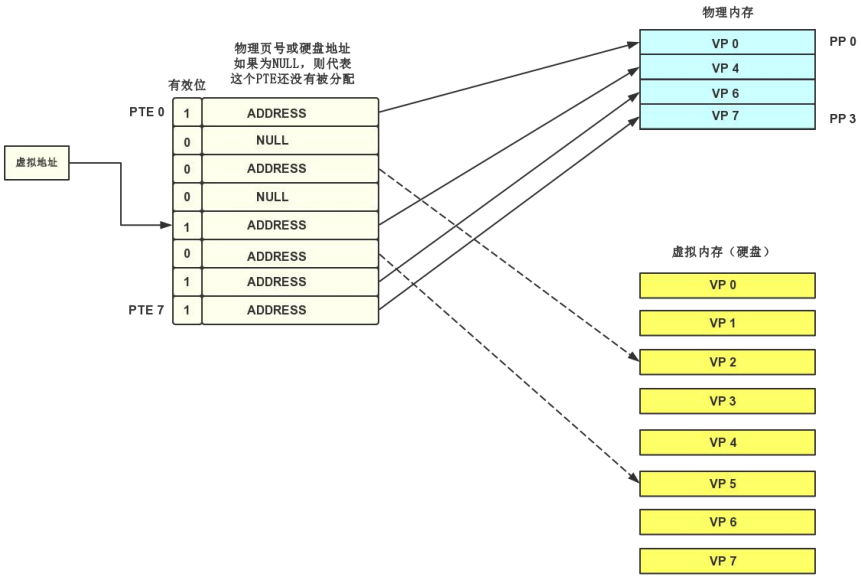
#### 2.3.3.1 主要任务

**a)** 我们采用两类算法实现了 Kernel 对内存的分配及释放，即页式分配与连续分配，采用连续分配时，出于对内存管理性能的考虑，实现了 Best-fit 算法，当 kernel 调用该算法申请一个存储区时，系统选中一个满足要求的最小空闲区分配给提出申请的进程；

**b)** 实现了虚拟内存机制并采用 Demand paging 算法，在为进程进行存储的分配与释放时都是对虚拟内存的分配与释放，只有当进程的某一页被访问时，才会将这一页调入物理内存中，物理内存中记录被调入的虚拟页号，初始值全为-1；

**c)** 在内存中为每个进程记录页表，页表以类的形式实现：其中以字典的格式记录这一页表的页面状态，该进程被分配到的虚拟内存 page 作为 Key，其对应的物理内存 frame 和页面是有效位作为 value，针对页表的操作（插入新页/删除页/修改有效位/查询页）则作为页表这个类的方法；

**d)** 支持 Kernel 通过 access 接口访问某个进程的进程内偏移地址，实现了虚实地址的转换与 LRU 的页面调度算法。以相对偏移地址除以页面大小的商作为页表偏移量来查找虚页号，以余数作为页内的地址偏移，通过查找页表的结果及页内偏移判断该访问地址是否合法，若合法则判断该页面是否在物理内存中，若在则页命中，否则记录页不命中，并调用页面替换算法执行换页操作，换页操作结束后再次执行地址访问操作，具体的示意图如下所示：




#### 2.3.3.2 核心算法

**e)** 虚拟内存的分配与回收算法：

**i.** 页式分配：采用页式分配时可以指定页面数量及页面大小，并采用 ndarray 结构来记录每一页的分配情况（占用空间/占用进程 ID/分配 ID），页式分配算法在进行分配与回收时都是以页面为单位；

**ii.** 连续分配的 Best-fit 算法：使用 list 结构分别记录内存的分配情况（基址/分配大小/占用进程 ID/分配 ID）及空闲区的情况（基址/空闲区域大小），在每一次分配时，遍历空闲记录表，挑选符合条件的空闲区进行分配，在每一次回收时，需要考虑是否有与释放区相邻的空闲区，若有，则需要将该释放区合并到相邻的空闲区中，并修改该区的大小和首址，否则，将释放区加入空闲区的记录表中。

**f)** 页面替换的 LRU 算法：

依据进程页表判断当前所访问的虚拟页是否已经被调入到物理内存中：

- 若已在物理内存中，则将该页放在调度队列尾部（表示最近访问）；
- 若未在物理内存中，则判断物理内存是否空闲，若空闲，则将访问页面调入物理内存中，并将该页放在调度队列尾部；若不空闲，则将调度队列头部的页面替换出去，将该页面所在页表的有效位置为-1（无效），并将该页面从调度队列中删除，后将访问页面调入物理内存，并将访问页面放在调度队列尾部。

假设系统在某时刻按照 7，0，1，2，0，3，0，4 的顺序访问虚拟页面，使用 LRU 调度的算法示意图如下所示：


### 2.3.3.3 监视功能

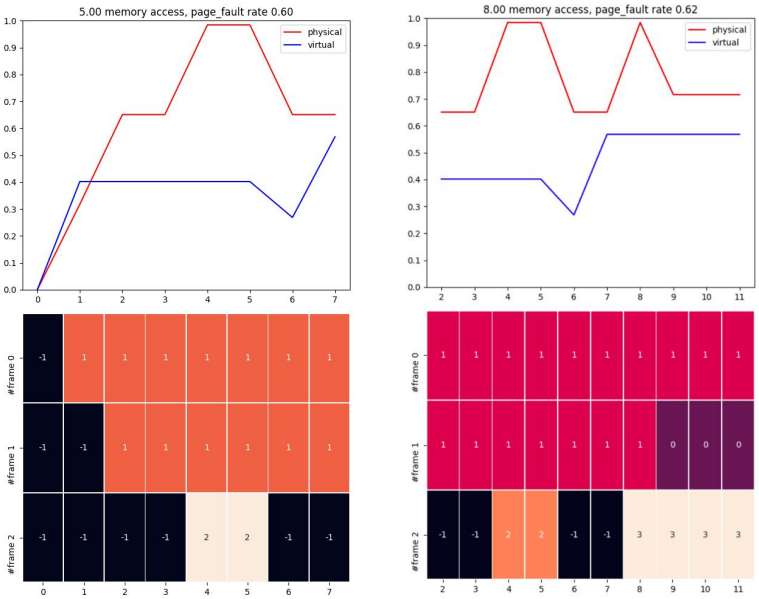
实现了具有内存监视功能的函数，以折线图的形式显示最近 n 个时刻虚拟内存的占用率及物理内存的占用率，显示出 access 的次数及页错误率，并以热力图的形式显示出最近 n 个时刻物理内存是被哪些进程占用的。

具体值的计算方式如下：

- 虚拟/物理内存的占用率=（已分配的虚拟/物理内存空间）/（总虚拟/物理内存空间）；
- 内存访问次数在每次调用 access 函数时加一，页错误次数在每次调入页面进入物理内存时加一；
- 页错误率=页错误次数/内存访问次数。

实现方式：

采用 list 结构存储每个时刻的物理/虚拟内存大小，以及物理内存的状态。



上图为系统在两个时间段中的内存状态曲线图，下方的网格图中标明了每个时刻占用各物理帧(frame)的进程 ID。在最上方的表头注明了到当前为止页面的访问总数及页错误率。

### 2.3.4 Process Manager

本模块负责对批处理任务进行解析，对内核创建的用户进程实体进行统一的调配与管理，实现系统核心资源与外设的充分利用，并通过时间片、优先级等机制，使得用户能够以更加灵活的方式与系统进行任务交互。

#### 2.3.4.1 主要任务

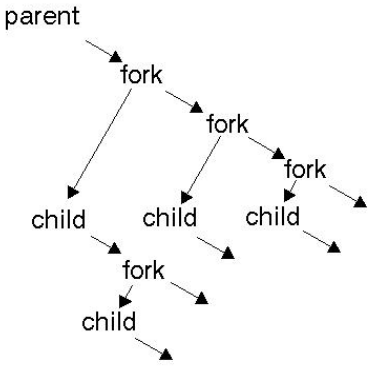
首先，我们需要对用户提交的批处理任务进行解析，我们预先设定了几种可能出现的批处理指令如下表所示：

| 批处理指令 | 格式 | 说明 |
|---|---|---|
| cpu | cpu [time] | 本指令用于模拟程序请求计算任务，需要调度 CPU 部件来执行，执行时间为 time 个单位，不需要连续执行 |
| printer | printer [time] | 本指令用于模拟程序请求打印任务，需要调度 Printer 部件来执行，执行时间为 time 个单位，需要连续执行 |
| fork | | 本指令用于在当前进程的运行基础上创建进程，需要调度 CPU 部件来执行，执行时间为 1 个单位，为原子性操作 |
| access | access [address] | 本指令用于模拟进程访问其所占用逻辑空间中的任一地址（读或写），执行时间为 1 个单位，为原子性操作 |

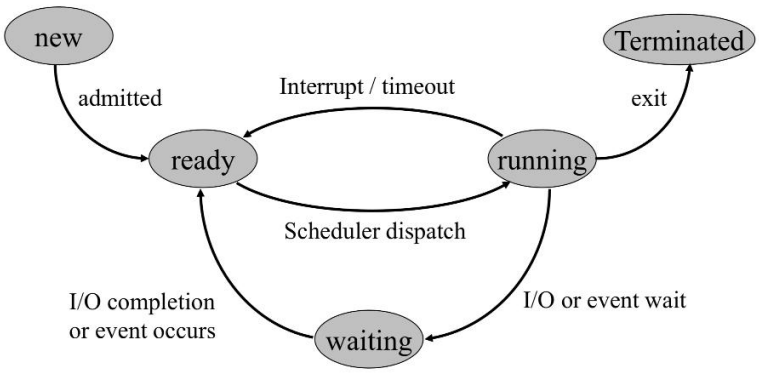
之后，我们创建进程来执行解析完的批处理任务，使用 PCB 作为进程唯一存在的标志。PCB 中包含了与过程相关的信息，包括：

进程号(pid) --记录该进程的 ID 号
进程名(name) --记录该进程的名字
创建时间(create_time) --记录该 PCB 创建的时间，即程序开始运行的时间
进程状态(status) --例如准备(ready)，运行(running)，等待(waiting)，终止(terminated)等
优先级(priority) --记录该进程所属优先级
执行队列(command_queue) --记录该进程仍需要执行的批处理指令

目前 MiniOS 支持两种进程创建的方法，一种是由操作系统进行创建，初始化 PCB；另一种是由进程创建(fork)，复制父进程 PCB，设置父进程号，分配子进程号，加入等待队列。创建子进程的树状展开过程可由下图进行示意：

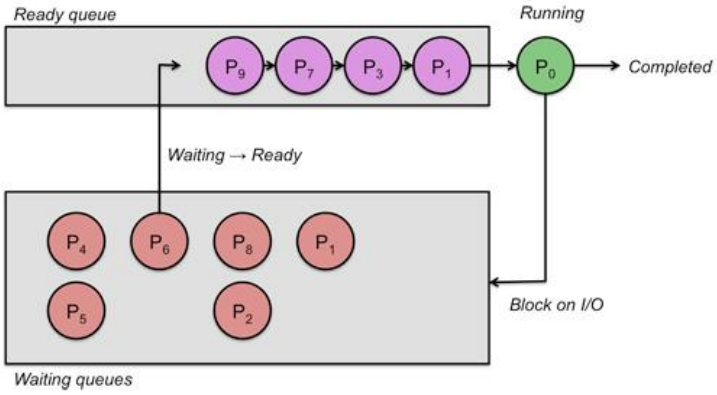


在整个进程的生命周期中，往往包括五个状态，如下图所示。在建立进程后，进程就从新建(new)状态转换到准备(ready)状态，等待其需要的资源。如果 CPU 调度到该进程，则进入运行(running)状态。在运行状态，如果发生中断或者时钟超时，则会重新回到准备(ready)状态。在运行状态，如果发生 I/O 或者其他事件的等待，则会从运行(running)状态中退出，进入等待(waiting)状态。在等待状态(waiting)等待其 I/O 完成，或者事件发生后，则从等待(waiting)状态中退出，重新进入准备(ready)状态等待 CPU 调度。在程序运行的过程中，如果接收到退出指令，或者程序运行完毕，则转入终止(terminated)状态。




#### 2.3.4.2 进程调度算法

在第一阶段优先级与时间片的调度算法的实现上，我们对等待队列根据优先级做了分层的设定，先考虑高优先级的进程，等到高优先级进程的等待队列为空时再考虑低优先级进程。在此基础上增加时间片调度，每个进程在占用一个时间片后会释放 CPU 资源，同时重新进入自己对应优先级队列的尾部排队。本算法可由下图进行简单示意：




#### 2.3.4.3 系统关键资源的监视与跟踪

我们实现了三个监视功能，包括对进程的监视 ps、对资源的监视 rs 和绘制资源占用图 mon。ps 功能会将目前系统中仍在运行的程序的 PID 号，以及对应的程序名字，当前所处的运行状态还有创建该进程的时间展现出来。

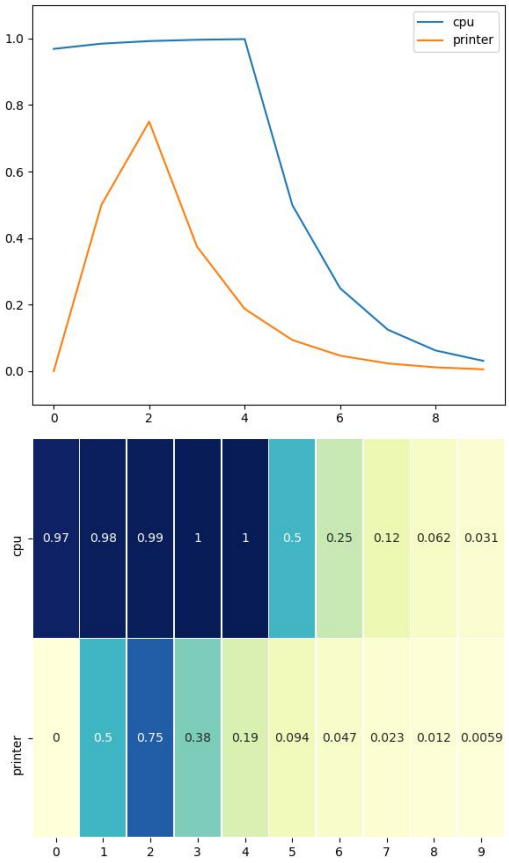
rs 功能会将目前系统中资源的使用情况打印出来，目前只包括 printer 资源。具体包括目前空闲的资源数目和使用的资源数目，资源正在被哪个进程使用，开始使用的时间，已经使用的时间以及预计释放该资源的时间

mon 能够用于绘制出近段时间系统资源占用率随时间的曲线图以及热力图。如下图所示，上半部分是曲线图，蓝线表示 CPU 各时刻的占用情况，黄线表示 Printer 各时刻的占用情况；下半部分是资源使用热力图进行表示，颜色越深，则代表其值越大。

MiniOS 在时刻 $t$，其资源占用率的计算公式为：

$$utilization_t = \alpha \times utilization_{t-1} + (1-\alpha) \times usage_t$$

其中 $usage_t$ 取值为 0 或 1，表示该资源在时刻 $t$ 是否正在使用；$\alpha$在系统运行过程中始终为常数，这里在 MiniOS 中默认配置的参数为 0.5。




# 3 团队成员阶段性工作内容总结

至项目中期，本团队已经顺利地完成了两个阶段的系统设计与开发，并已做好第三阶段开发的充足准备。在 2.1 小节，本文给出了项目的进度概览示意图。我们从开题至今，共进行了三次会议；在前期准备阶段，团队完成了初步调研，确立了项目的技术路线；在第一阶段，我们完成了系统的基本结构、命令格式的设计，以及 MiniOS 框架的编码实现，该阶段的产出为具备了基本功能的系统程序，确定了用户以命令行的方式与系统进行高效交互。

在第二阶段，我们引入了更具通用性的算法、更灵活的功能，以及更高效的资源管理模式。这其中包括设计并实现了完整的虚拟内存机制、多级反馈队列调度算法、磁盘寻道算法等操作系统核心算法，并增添了图形化资源实时监控与跟踪功能，使得不同的算法对系统资源的使用效率能够进行对比分析，帮助系统人员对 MiniOS 的运行模式有更为清晰的认识，使得系统灵活性与健壮性得到大幅提升。

对于开题至今，即前两个阶段的详细开发过程，本小节将围绕各团队成员的工作详细展开，对团队成员的阶段性工作内容进行完整的归纳与总结。

- **陈斌同学（小组组长，负责部分：系统结构，Kernel、Shell 模块，参与其他模块开发）**

  在本项目的开发过程中，我对项目工作进行合理的组织与分配，及时召开团队会议，对

项目进度进行实时监管，确保各阶段的推进在我们的严格控制之内。同时我负责对整个系统的结构进行设计。实现上，我对各个软件类的职责和功能进行了明确的定义，并对所有接口的输入输出都作了明确的规定，明确的定义、清晰的接口也是团队成员之间能够进行高效协作的重要保障。

在第一个阶段中，我的工作可以分为三个部分：提出路线、系统结构设计、编程开发。我组织了团队会议，召集小组成员们共同商讨，与小组成员们在前期对不同实现方式进行调研后提出当前的开发路线，即以模拟的方式进行操作系统功能实现，并以 Python 3 作为编程语言，专注于系统性能、算法与功能的设计上。

在提出开发路线的基础上，我与成员们协商探讨，确立了当前系统的基本结构与模块划分。其中为了使得不同功能的实现能够被清晰地划分并独立设计与测试，本系统尽可能地将不同功能划分到不同的模块中：将用户命令的接收、解析与处理工作划分至 Shell 模块，对于底层外设的使用并实现文件、内存、进程管理的功能分别划分到三大管理模块 FileManager、MemoryManager、ProcessManager 中，并由 Kernel 模块对整个操作系统的其他模块进行管理。其中三大管理模块之间不存在耦合关系，它们都是可以独立设计与测试的。

在完成了前两部分的工作后，我提出采用面向对象的设计方法对上示结构进行设计，并编写了代码框架，规定各变量的含义、各功能模块接口规范，并将任务分配予对应的小组成员，其中我主要负责 Shell 与 Kernel 模块的开发。以此将团队内部的 6 个成员进一步划分为 4 个不同的分组开始并行开发，在各模块第一阶段开发完成后进行测试，与成员进行问题反馈与修复，并将模块进行了初步整合，形成一个基本功能相对完整的版本。

在第二个阶段，我完整地回顾了操作系统理论课程中学习到的知识，参考了《实用操作系统概念》、《操作系统概念》这两本书籍以及其他经典文献，提出了向 MiniOS 引入虚拟内存机制、资源使用情况的跟踪与监控机制的构想，并和组员们共同针对当前系统，引入了页面替换算法、进程的优先级调度算法、磁盘空闲块的分配与管理算法以及寻道算法等，并全程参与开发实现，**具体的实现内容将由小组成员们进行更加细致的阐述**。此阶段中，我根据三大核心功能模块的设计与实现情况对 Kernel 进行调整。具体地，引入了 ps、dms、dss 等系统资源使用情况展示功能命令，并添加了对内存利用率、各虚拟页占用情况、CPU 等核心资源的**实时使用率的监控功能**。此外，我向 Shell 中引入了**单行多命令解析机制以及正则表达式功能**，在经过不断优化和改进后，我们的系统在保证高效性与健壮性的同时，也使得用户的使用效率和体验得到了进一步的提升。

关于编码与文档部署方面，我在两个阶段的工作中均进行了主体框架的设计，并严格规定代码和文档各部分的逻辑、内容与格式，在团队成员们完成了各自部分工作的记录与文档书写后，我对各部分内容进行审核并及时安排修订，并对全部工作进行统一格式化整理。在此前团队进行了两次会议后，我分别以此模式组织全员撰写了会议纪要以及中期项目汇报文档，在编程开发的同时，我们也时刻保持文档兼备，最终顺利完成了项目前两阶段的收尾。

- **吴桐子同学（负责部分：文件管理模块）**

第一阶段我的工作是完成了文件管理 FileManager 中对于外存文件块的管理。为了尽快完成最小版本开发，我们采用了连续分配的策略。我们为外存块 block 单独声明为一个类 Block。具体的外存块管理操作包括：初始化、新建文件时对 block 的写入、删除文件时对 block 的清除、展示各 block 的信息。其中，又涉及到对于 block 的重要操作，具体说明如下：

- 寻找空闲的连续 n 个 block。在本次作业中，我们采用了 first-fit 策略，即第一次找到足够的连续空间后，直接写入其中。这个方法将会引入外部碎片；
- 将文件填充进 block。指定文件大小和文件路径后，此操作将文件存放于连续的若干块中，并将更新 block 目录（其中每小项的格式为文件名：起始 block 号，使用的 block 数量）。这个操作在初始化 block 信息时会被使用，即读取目录下的所有

文件，并放入的 block 中。此外，在新建文件时也会被使用。文件新建时会指定文件所占的空间，当外存空间中无法找到足够的空闲连续块时，将会报告空间不足的错误；

- 删除文件后清除 block。指定文件路径后，系统将从 block 目录中检索该文件所占有的空间，并将相应的外存空间释放。

正如上述文字所说，我们在外存块管理中会引入外部碎片，所以我们将在第二阶段工作中引入整理磁盘碎片的功能。此外，在判断 block 是否空闲的操作中，我们采用的策略是对每个块的剩余空间进行判断，这个操作较为不经济，而实际上可以采用 bit map 的方式。

第二阶段我的工作针对第一阶段中的遗留问题展开。在这个阶段，我实现了三种磁盘块的分配方式、磁盘碎片的整理和 bitmap 空闲块表示方式。详述如下：

- 建立 bitmap，即记录每个 block 空闲与否的一个 ndarray（numpy 数组）。通过 bitmap，我们可以轻松寻找连续的空闲块，从而加快运算速度；
- 寻找空闲的连续 n 个 block。在本次作业中，我们共加入了 first-fit、best-fit、worst-fit 三种策略，通过对配置文件的设置，可以选择不同的策略。first-fit：即第一次找到足够的连续空间后，直接写入其中。best-fit：找到所有能够容纳文件的连续空间后，选择其中最小的写入。worst-fit：找到所有能够容纳文件的连续空间后，选择其中最大的写入。注意，三种方法都会引入外部碎片；
- 整理磁盘碎片：调用 tidy_disk()函数后，系统将会对磁盘内容进行整理，从而消除所有外部碎片。

在第二阶段工作结束后，我在外存块管理中的工作已经基本算是面面俱到。

- **许浩然同学（负责部分：文件管理模块）**

第一阶段我负责与吴桐子同学一起完成文件管理模块的功能分析，并迭代地、由易到难地实现这些功能。在其中，我主要负责各个文件操作命令的实现。我们和小组组长进行探讨，并明确了文件管理模块中针对文件存储体系的模拟方式：

模拟系统的文件存储结构在程序之外以对应文件夹的形式存在，在模拟系统程序启动后，该系统程序会依据外部的文件存储结构，通过复刻该结构形成本系统内部的文件体系；而对于文件内容，系统也将会把其读取到模拟磁盘中。这一模拟方式较为简易，其核心在于如何对文件树进行高效、合理的组织。

我第一阶段开发工作的第一部分在于构建文件树字典。字典中的一个键值对可以代表一个目录, 以目录名为键, 以目录子字典为值。以此方式遍历这个字典结构，就可以获取到所有文件的信息。

在第二部分，我们参考了 Linux 系统中的多个文件操作命令：ls(显示目录下的内容)，cd(切换当前工作目录)，mkdir(创建目录)，mkf(创建普通文件)，rm(删除文件)，chmod(更改文件属性)，pwd(显示当前工作目录)等并进行相应的功能实现。

在第三部分，我们完善了文件操作命令的参数。从仅支持文件名，到支持相对路径，再到支持绝对路径；从无选项，到支持部分命令行选项功能，如 rm 命令中的“-f”(强制删除)，“-r”(递归地删除文件夹)等具有多种功能的命令选项。

在完成了以上三个部分的系统开发后，我们经过大量测试，发现系统中仍存在一些问题，如 chmod 未设置格式错误检测、绝对路径不允许以多个“\”结尾等等。因此我们下一步的工作在于致力解决这些问题，并尝试将系统对存储体系的模拟方式从“完全模拟”方式转化为“部分模拟”方式，即通过采用更加底层的方法对系统的文件存储体系进行二次设计与开发实现。

第二阶段我的主要工作实现磁盘类。实现各寻道算法的模拟磁盘读写，绘图展示轨迹、对比算法效率功能。工作的核心在于六种磁盘寻道算法：FCFS，SSTF，SCAN，C_SCAN，

LOOK，C_LOOK。参考 OS 理论课知识首先实现 FCFS，它根据服务队列依次寻道。剩下五种算法只需对服务队列按算法进行修改，再调用 FCFS，以修改过的服务队列进行寻道即可。此方法能模拟每一次寻道，将寻道信息记录下来，便可绘制轨迹图；统计每次寻道所用的时间和读写量，便可绘制速率图。

由于我们的操作系统文件较少且不复杂，访存服务队列总是短而规律的，六种寻道算法在应用于这些服务队列上并不能体现出差距。故人为设计一个长而复杂的内置服务队列，用于专门模拟寻道算法，绘制轨迹并对比速率。

除磁盘之外，我对第一阶段文件管理模块遗留下的问题进行修补。如 ls 指令可支持文件，修复 cd 和 ls 存在的 bug 等。

- **张炜晨同学（负责部分：进程管理模块）**

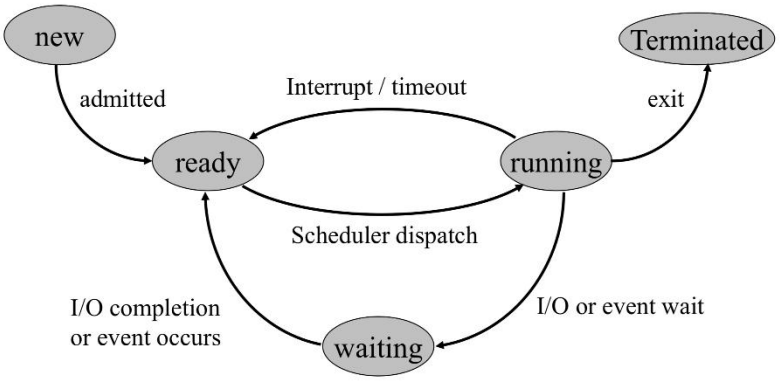
本阶段我主要负责进程管理模块的基础功能实现并与最后其他模块的对接工作，我将我的工作内容总结如下：

1．通过 exec 命令执行可执行文件，创建进程，创建后的进程在其整个生命周期中可以以 PCB 形式存在。每个进程用一个 PCB 表示，进程的属性在 PCB 中保存；

2．使用 ps 命令查看进程状态，rs 命令查看目前硬件资源使用状态，和预计被释放的时间（目前硬件资源仅支持打印机 Printer）；

3．使用 kill 命令杀死当前运行的进程，杀死进程后会释放其占用的 CPU，硬件 IO 等资源；

4．进程根据其执行情况在不同队列(就绪队列,阻塞队列,运行队列)间迁移。进程从 new 后在 ready、waiting、running 之间状态转换，直到最后进程执行完毕或者被 kill 后进入 terminated 状态，同时也模拟了时钟中断，硬件 I/O 中断等功能：




5．实现使用单处理器进程调度功能，目前的 CPU 的调度算法只设置了一个，即优先级调度与时间片相结合的调度算法。考虑到硬件资源的特殊性，对于硬件资源我们采用非抢占的先到先服务的调度算法。

当前关于进程管理模块依然存在的问题有：

1．CPU 调度算法仍是采用静态优先级和时间片相结合的策略，未使用动态优先级；

2．在进程进入 terminated 状态后，该进程 PCB 仍会保留，不会被删除；

3．未对内存资源进行竞争使用；

4．操作时间以一秒（时间片）为单位，若该时间片还未结束，该进程就结束 CPU 使用的话，并不会立即调度，而是会等到时间片结束后统一执行。

- **郑莘同学（负责部分：进程管理模块）**

本阶段我负责了四个方面的工作：关于实现裸机运行方法的调研、内核代码研究、进程管理模块程序设计与代码审阅、系统程序的跨平台测试。

我先研究学习了操作系统的启动过程。首先，计算机启动 BIOS，再加载位于硬盘引导

扇区的引导程序(bootloader)至内存，并跳转开始运行引导程序。引导程序初始化 CPU 运行状态，再调用引导主程序将操作系统内核加载进内存，并将控制权交给它。最后，操作系统设置页表，初始化内存管理、进程管理、中断控制、文件管理，并启动第一个用户进程，操作系统启动完毕。我还成功在 QEMU 硬件模拟器以及 VMWare 虚拟机上启动运行了 xv6 操作系统。此外，我制作了 haribote OS 的 U 盘启动盘，学习研究了 BIOS U 盘启动的操作步骤。裸机启动的实现方法较为复杂，且实现难度大，这些前期调研工作的结果为后续团队开发路线的确定提供了较大的参考。

然后，我查阅了 linux 0.96、ucore、xv6 等小型操作系统的内核源码，重点研究了他们的进程管理模块，认真学习了代码风格，了解到进程管理模块需要设计的部分分别为：进程控制块(PCB)、初始化进程(init)、创建子进程(fork)、进程退出(exit)、进程等待(wait)、进程调度(scheduler)、进程睡眠(sleep)等，对进程管理模块构建了总体框架。

之后，我和张炜晨同学共同变成实现了进程管理模块，并对代码进行审阅，初步实现了创建进程、进程调度、创建子进程、资源使用情况图绘制的功能。其中，创建子进程的具体步骤为：复制父进程 PCB，设置父进程号，分配子进程号，加入等待队列。下一步，我们还计划添加进程间通信的功能设计，例如管道、虚拟网络环路等。

最后，我对本系统进行了跨平台运行测试，测试结果表明本次设计的系统程序在当前主流的操作系统 Windows 10 与 Ubuntu 20.04 LTS 下都能够正常运行，再次验证了系统程序的跨平台能力。

- **周雯笛同学（负责部分：内存管理模块）**

在第一阶段，我主要负责内存管理模块，包括设计实现内存的分配与回收模式，作为其他模块的调用部分。内存分配大体分为连续型分配与不连续型分配，于是在本阶段我针对两大类型实现了两种具体内存分配方式：固定页面大小的页式分配及应用 best-fit 算法的连续分配，使用者可以在初始化时选择采用的分配模式，也针对这两种内存分配算法实现了两种内存回收方式，可以实现内存的正确分配及回收。除此之外，实现了针对不同内存存储方式的跟踪程序，可以在调用时格式化的打印出当前内存所处的状态。

当前关于内存管理模块依然存在的问题有：

① 针对不同的内存分配方式，我设计了不同的内存存储结构，但我认为在后期应当尝试设计出统一的存储结构，可以对多种算法进行兼容；

② 当前还未实现从配置文件中读取对内存分配的设置功能；

③ 还未尝试通过多线程的形式，通过调用跟踪程序来跟踪打印内存状态的功能。

在第二阶段，我主要实现了虚拟内存机制，实现了虚实地址的转换，LRU 的页面调度策略，以及可以跟踪绘制内存状态的函数。具体说明如下：

1. 我基于页式存储模式实现了虚拟内存机制，并利用页表来实现虚实地址的转化，开始时由于内部碎片的存在，导致地址的转换过程比较复杂且有错误，在和组长讨论过后决定将内部碎片页划归入该进程的占用空间，但却不算在有效寻址的范围内，以此正确的实现了地址的转换过程；

2. 对每个进程的页表进行管理：进程的页面在申请内存时被创建，并在释放内存后被回收，且只有在被访问时才调入物理内存。这样的设计思路是基于上学期操作系统所学的 Demand paging 策略；

3. 通过查看上学期操作系统的课件，我将页表设计为 dict 结构，以虚拟页号为键，对应的物理页号及有效位为值，以此实现了可变长的页表，并且可以通过该进程的进程内偏移地址来计算偏移量，访问相应的页面；

4. 为了避免过多的参数传递，因此将绘图函数也放在主类(MemoryManager)中，并且通过存储近 n 个时刻的值，实现了类似于坐标轴滚动的效果。

本阶段解决了上一阶段关于内存的跟踪及打印问题，并且经过和组长讨论，我们的系统将不会把连续存储与页式存储统一为相同的内存结构。除这些外，经过分析与总结，当前内存管理模块的问题以及未来的工作有：

① 页面调度算法可以进行进一步扩充，实现 FIFO、OPT 算法等；
② 页错误时的中断机制没有得到模拟；
③ 可以进一步补充调度物理内存中页面时，对于修改后页面的写回过程；
④ 尝试编写多级页表结构，考虑向系统中加入 cache 机制。

# 4 系统运行说明及测试结果

## 4.1 启动系统与查看手册

使用 Python 3 平台运行操作系统内核程序：kernel.py，可以看到，MiniOS 已经启动，接下来我们可以使用键盘输入命令，使用“man”命令，我们可以查看当前系统所支持的所有命令手册，如下所示：

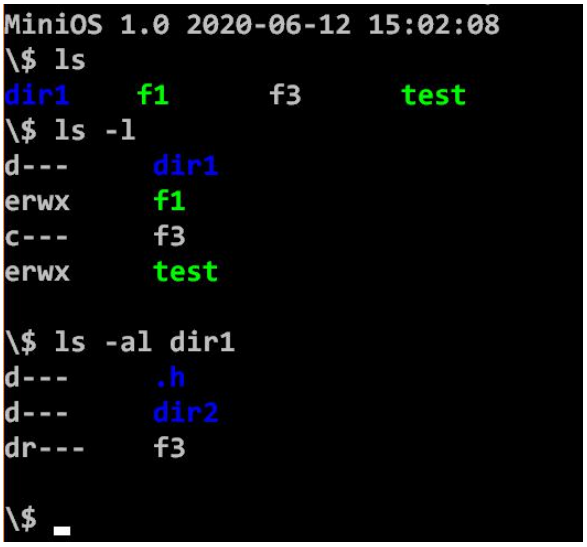
```
MiniOS 1.0 2020-06-12 14:55:37
\$ man
man - manual page, format: man [command1] [command2] ...
ls - list directory contents, format: ls [-a|-l|-al][path]
cd - change current working directory, format: cd [path]
rm - remove file or directory recursively, format: rm [-r|-f|-rf] path
mkdir - make directory, format: mkdir path
dss - display storage status, format: dss
dms - display memory status, format: dms
exec - execute file, format: exec path, e.g. exec test
chmod - change mode of file, format: chmod path new_mode, e.g. chmod test erwx
ps - display process status, format: ps
rs - display resource status, format: rs
mon - start monitoring system resources, format: mon [-o], use -o to stop
td - tidy and defragment your disk, format: td
kill - kill process, format: kill pid
exit - exit MiniOS
\$
```

“man”命令为 MiniOS 内核的内置命令，我们可以在其后跟随若干个需要查看的命令名称，这样可以单独对这些命令进行查询，如下所示：

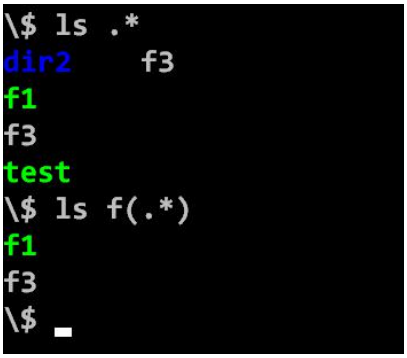
```
\$ man ls rm
ls - list directory contents, format: ls [-a|-l|-al][path]
rm - remove file or directory recursively, format: rm [-r|-f|-rf] path
\$ _
```

## 4.2 文件浏览、属性修改与目录切换

使用“ls”命令，我们可以列出当前目录下的所有文件名称。我们也可以在其后添加待查看目录或文件，其能够进行筛选与列出，值得一提的是，当前系统已支持正则表达式功能，因此我们可以使用正则表达式来进行文件名的筛选与查看。使用“-a”选项可查看所有文件，使用“-l”选项可查看文件属性，使用“-al”选项可以同时选中以上两个选项。如下所示：

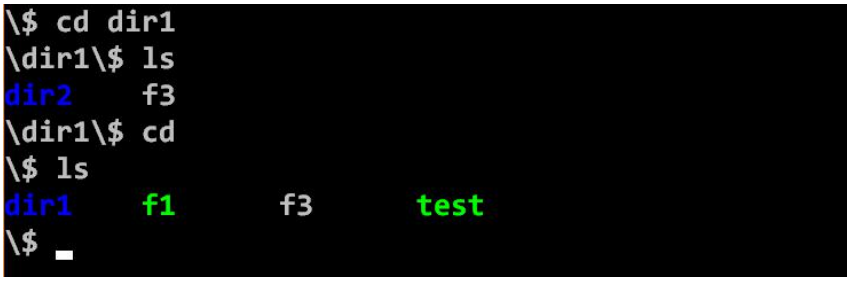



本系统还支持使用正则表达式，该表达式针对文件名，能够被 Shell 所解析，并与当前目录下的文件名匹配。如下所示：




使用“chmod”命令，我们可以对某文件的属性进行修改。如下所示：


使用“cd”命令，我们可以将当前工作目录进行切换。若其后不携带参数，则默认切换至根目录下，如下所示：



## 4.3 目录创建与文件删除

使用“mkdir”命令，我们可以在指定路径创建空白目录，如下所示：

```
\$ mkdir test
mkdir: cannot create directory 'test': File exists
\$ mkdir test_dir
mkdir success
\$ ls
dir1    f1      f3      test    test_dir
```

使用“rm”命令，我们可以删除文件，选项“-r”可删除空白目录，“-f”可用于强制删除文件，“-rf”可以递归地删除目录，如下所示：

```
\$ rm -r test_dir
\$ ls
dir1    f1      f3      test
\$ mkdir dir2
mkdir success
\$ ls
dir1    f1      f3      test    dir2
\$ cd dir2
\dir2\$ mkdir dir3
mkdir success
\dir2\$ cd
\$ rm dir2
rm: cannot remove 'dir2': Is a dir. Try to use -r option
\$ rm -rf dir2
\$ _
```

## 4.4 执行与停止批处理任务

在 MiniOS 中，我们设计了一套模拟文件格式，其中文件的“content”部分为可执行程序的代码段，本次用于测试的可执行文件 test 的内容如下所示：

```
{
    "name": "test",
    "type": "erwx",
    "size": "2000",
    "priority": 2,
    "content": [
        "fork",
        "cpu 1",
        "access 10",
        "access 100",
        "access 100",
        "access 100",
        "fork",
        "printer 1",
```

```
        "cpu 1",
        "printer 1",
        "access 1999",
        "cpu 1",
        "access 535",
        "access 535",
        "printer 1",
        "cpu 1",
        "access 535",
        "printer 1",
        "cpu 1",
        "printer 1",
        "access 535",
        "access 535",
        "cpu 1",
        "fork",
        "printer 1",
        "access 1999",
        "access 1999",
        "access 1999",
        "cpu 1",
        "access 1999"
    ]
}
```

其中由该程序创建的进程在运行过程中将使用系统调用 fork 创建出更多的子进程。使用“exec”命令，我们可以运行 test 程序，运行效果如下所示：

```
\$ ls
dir1    f1      f3      test
\$ exec test
disk access success: time used:  21.4 ms
[pid 0] process created successfully
   [pid 1] process forked successfully by [pid 0]
[pid 2] process forked successfully by [pid 0]
[pid 3] process forked successfully by [pid 1]
[pid 4] process forked successfully by [pid 0]
[pid 5] process forked successfully by [pid 1]
[pid 6] process forked successfully by [pid 2]
[pid 7] process forked successfully by [pid 3]
[pid #0] finish!
[pid #1] finish!
[pid #4] finish!
[pid #2] finish!
[pid #5] finish!
[pid #3] finish!
[pid #6] finish!
[pid #7] finish!
```

在程序运行过程中，我们可以使用“kill”命令并携带对应的进程 pid 将指定进程强制结束。如下所示：

```
MiniOS 1.0 2020-06-12 15:11:49
\$ ls
dir1    f1      f3      test
\$ exec test
disk access success: time used:  21.4 ms
[pid 0] process created successfully
   [pid 1] process forked successfully by [pid 0]
kill 0 1
\$
```

## 4.5 查看系统进程与资源状态

在 MiniOS 中，我们提供了如下系统内置功能命令：dss、dms、ps、rs，分别用于查看磁盘占用情况，物理内存占用情况，进程状态以及系统功能资源（包括 CPU、打印机）使用状态。查看系统相关资源或进程状态的运行示例如下所示：

```
\$ dss
total: 1228800 B,        allocated: 6144 B,      free: 1222656 B

block #0       512 / 512 Byte(s)   \dir1\dir2\dir3\dir6\f4
block #1       174 / 512 Byte(s)   \dir1\dir2\dir3\dir6\f4
block #2       233 / 512 Byte(s)   \f1
block #3         0 / 512 Byte(s)   None
block #4       512 / 512 Byte(s)   \dir1\dir2\dir3\f5
block #5       174 / 512 Byte(s)   \dir1\dir2\dir3\f5
block #6         0 / 512 Byte(s)   None
block #7       512 / 512 Byte(s)   \dir1\f3
block #8       174 / 512 Byte(s)   \dir1\f3
block #9         0 / 512 Byte(s)   None
block #10      233 / 512 Byte(s)   \f3
block #11      512 / 512 Byte(s)   \test
block #12      512 / 512 Byte(s)   \test
block #13      512 / 512 Byte(s)   \test
block #14      464 / 512 Byte(s)   \test
```

```
\$ dms
total: 16384B allocated: 0B free: 16384B
\$ ls -l
d---     dir1
erwx     f1
crw-     f3
erwx     test

\$ exec f1
disk access success: time used:  9.3 ms
[pid 0] process created successfully
\$ dms
total: 16384B allocated: 233B free: 16151B
block #0  233 /1024 Byte(s)  pid =0    aid =0
   [pid 1] process forked successfully by [pid 0]
[pid 2] process forked successfully by [pid 0]
[pid 3] process forked successfully by [pid 1]
dms
total: 16384B allocated: 932B free: 15452B
block #0  233 /1024 Byte(s)  pid =0    aid =0
block #1  233 /1024 Byte(s)  pid =1    aid =1
block #2  233 /1024 Byte(s)  pid =2    aid =2
block #3  233 /1024 Byte(s)  pid =3    aid =3
\$
```

```
\$ ps
[pid #    0] name: f1         status: ready              create_time: 2020-06-12 15:19:28
[pid #    1] name: f1         status: ready              create_time: 2020-06-12 15:19:32
[pid #    2] name: f1         status: ready              create_time: 2020-06-12 15:19:35
[pid #    3] name: f1         status: ready              create_time: 2020-06-12 15:19:37
\$ _
```

```
[pid #    0] name: f1        status: ready              create_time: 2020-06-12 15:19:28
[pid #    1] name: f1        status: ready              create_time: 2020-06-12 15:19:32
[pid #    2] name: f1        status: ready              create_time: 2020-06-12 15:19:35
[pid #    3] name: f1        status: ready              create_time: 2020-06-12 15:19:37
[pid #    4] name: test      status: ready              create_time: 2020-06-12 15:22:46
[pid #    5] name: test      status: running            create_time: 2020-06-12 15:22:46
[pid #    6] name: test      status: running            create_time: 2020-06-12 15:22:57
[pid #    7] name: test      status: running            create_time: 2020-06-12 15:22:59
\$ rs
1 Printer is using,the recent free time is 2020-06-12 15:24:05
[Printer #0] pid: #5    starting_time: 2020-06-12 15:23:05   used time: 0    expect_free_time: 2020-06-12 15:24:05
\$ _
```

## 4.6 消除磁盘碎片

使用“td”命令，我们可以将系统磁盘进行整理，消除其中存在的外部碎片。执行前后的系统磁盘占用状态如下所示：

```
\$ dss
total: 1228800 B,        allocated: 6144 B,      free: 1222656 B

block #0       512 / 512 Byte(s)   \dir1\dir2\dir3\dir6\f4
block #1       174 / 512 Byte(s)   \dir1\dir2\dir3\dir6\f4
block #2       233 / 512 Byte(s)   \f1
block #3         0 / 512 Byte(s)   None
block #4       512 / 512 Byte(s)   \dir1\dir2\dir3\f5
block #5       174 / 512 Byte(s)   \dir1\dir2\dir3\f5
block #6         0 / 512 Byte(s)   None
block #7       512 / 512 Byte(s)   \dir1\f3
block #8       174 / 512 Byte(s)   \dir1\f3
block #9         0 / 512 Byte(s)   None
block #10      233 / 512 Byte(s)   \f3
block #11      512 / 512 Byte(s)   \test
block #12      512 / 512 Byte(s)   \test
block #13      512 / 512 Byte(s)   \test
block #14      464 / 512 Byte(s)   \test
```

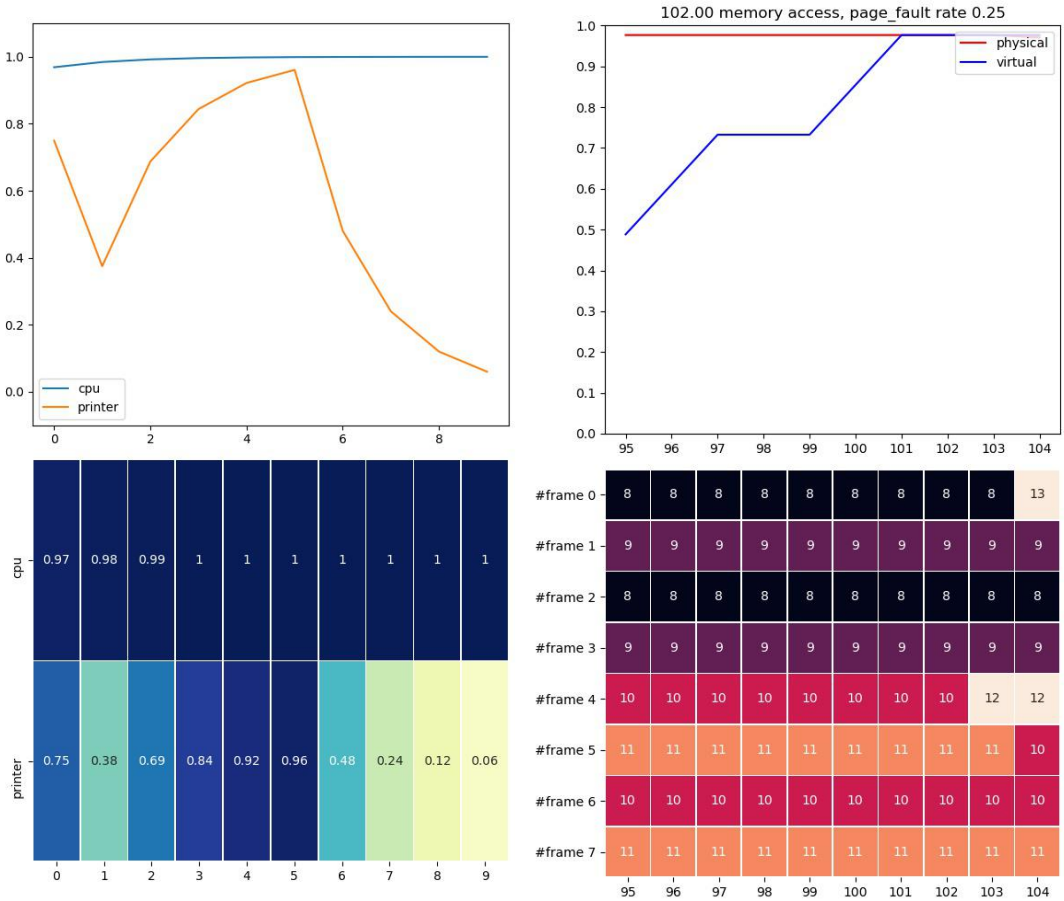
```
\$ td
\$ dss
total: 1228800 B,        allocated: 6144 B,      free: 1222656 B

block #0       512 / 512 Byte(s)   \dir1\dir2\dir3\dir6\f4
block #1       174 / 512 Byte(s)   \dir1\dir2\dir3\dir6\f4
block #2       512 / 512 Byte(s)   \dir1\dir2\dir3\f5
block #3       174 / 512 Byte(s)   \dir1\dir2\dir3\f5
block #4       512 / 512 Byte(s)   \dir1\f3
block #5       174 / 512 Byte(s)   \dir1\f3
block #6       233 / 512 Byte(s)   \f1
block #7       233 / 512 Byte(s)   \f3
block #8       512 / 512 Byte(s)   \test
block #9       512 / 512 Byte(s)   \test
block #10      512 / 512 Byte(s)   \test
block #11      464 / 512 Byte(s)   \test
block #12        0 / 512 Byte(s)   None
```

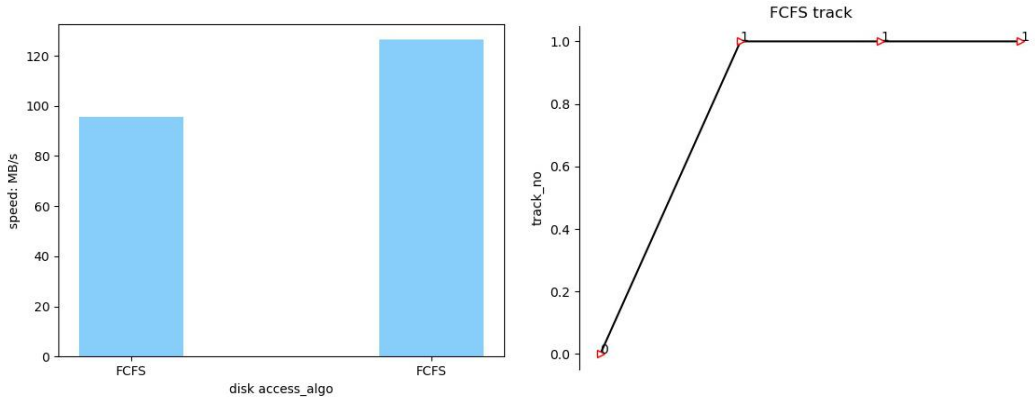
## 4.8 使用实时监视功能

使用“mon”命令，我们可以开启或关闭（通过“-o”选项）系统对三部分资源进行实时跟踪的功能。系统进行实时监控与跟踪的资源分别为：CPU 与打印机使用率、虚拟与物理内存空间使用情况与缺页率、磁盘读写情况（包括磁头移动轨迹等）。使用“mon”命令后，我们可以运行测试程序，可以看到，系统能够周期性地为我们生成当前系统的资源使用情况快照，本文择取其中的部分结果，如下所示：

```
MiniOS 1.0 2020-06-12 15:43:28
\$ mon
\$ exec test
disk access success: time used:  21.4 ms
[pid 0] process created successfully
   [pid 1] process forked successfully by [pid 0]
[pid 2] process forked successfully by [pid 0]
[pid 3] process forked successfully by [pid 1]
```




当前系统中共有资源 CPU、Printer，它们的数量均为 1，虚拟内存大小为 16 页，物理帧数为 8，其中物理帧的大小与虚拟页的大小是一致的，均为 1024Byte。进程调度策略采用优先级、抢占式调度，调度时间片为 1 秒。当前系统所采用的默认磁盘寻道算法为 FCFS，页面替换策略采用 LRU 算法。更为具体的配置信息，可以查看本系统下的 config.py 文件以及系统当前完整源代码。




上图所示为最近所进行的磁盘读写情况以及磁道运动轨迹。可以看到我们自系统启动以来，一共读取了两次同一个文件 test，由于系统磁头初始位置为 0，因此在系统两次读取过程中，第一次的平均读写速度是稍低一些的，大约为 95MB/s，而第二次读取的平均磁盘访问速度约为 125MB/s。

当我们需要关闭监视功能时，使用“mon -o”命令即可，如下所示：

```
[pid #11] finish!
[pid #14] finish!
[pid #15] finish!
mon -o
\$ _
```

## 4.9 安全退出

使用“exit”命令，我们就可以安全结束 MiniOS 系统程序，如下所示：

```
\$ exit
```

对于更详细的功能，我们本次随本报告亦提交了系统可执行脚本文件（即 Python 源代码），如需要运行测试可直接在与本报告同一目录下的 MiniOS 路径中通过终端执行“python kernel.py”即可。由于 MiniOS 仍处于测试阶段，因此其难免存在因各模块组合或成员未考虑周到等原因而遗留的系统漏洞，我们也将在接下来的开发阶段中不断进行修复与完善。

# 5 后续开发计划

经团队第二次会议中成员们的共同商讨，我们对团队在第一阶段开发完成的系统程序进行了归纳总结，并针对当前各模块中存在的问题以及当前的系统状态进行了探究，提出并部署了接下来在第二阶段中本团队的开发计划，其可大致总结如下：

① 目前本系统仍然存在一些技术上的问题，我们在接下来将会把各部分功能模块当前已经发现的问题首先予以全部解决；

② 在此版本的系统程序之上考虑新增以下功能或系统特性：

1. 对于文件管理模块，实际上该模块的基本模拟功能已趋于完善，并且我们对于磁盘读取信息的过程已经引入了多种经典算法。在接下来的开发阶段中，我们将对文件存储系统、磁盘空闲块以及分配引入更加高效的管理策略，参考现代操作系统，模拟更加真实的文件存取过程；

2. 在当前基础之上通过参考相关文献，尝试引入共享内存机制，亦或是类 Unix 系统中所提供的管道、通信环路等进程间通信机制；

3. 参考当前主流操作系统，向本系统程序中引入更多、更丰富的命令集合；

4. 将当前系统所支持的批处理程序的文件格式、所对应的指令集合进行进一步的修正与完善，例如引入文件最近修改时间属性、引入循环语句等等；

5. 引入多处理器（多 CPU）、多台外设（如多台打印机，或引入扫描仪等新设备）等，实现更为复杂、更真实的资源及配套的调度方法；

6. 参考现代操作系统性能评价方法，对本系统的性能进行综合考察，并通过设计基准程序、对比不同的算法，得出对当前系统的改良方案，进一步提升系统的性能；

6. 尝试将本系统的扩展能力进一步提升，使其能够被更多的平台所支持，并研究系统的编译方法，使得本系统能够发布被多平台所支持的不同版本。

③ 团队在第三阶段的开发过程应高效且有序，具体细节将在此次总结工作完成之后立即再次召开团队会议进行讨论与明确，与前一阶段类似，我们将在第三阶段的开发过程中对系统各功能模块及新增接口的输入输出作出统一规定，达成组内共识，并在此阶段的开发工作完成后对系统程序进行完整的测试与性能评估，将结果用于指导下一阶段的项目开发。