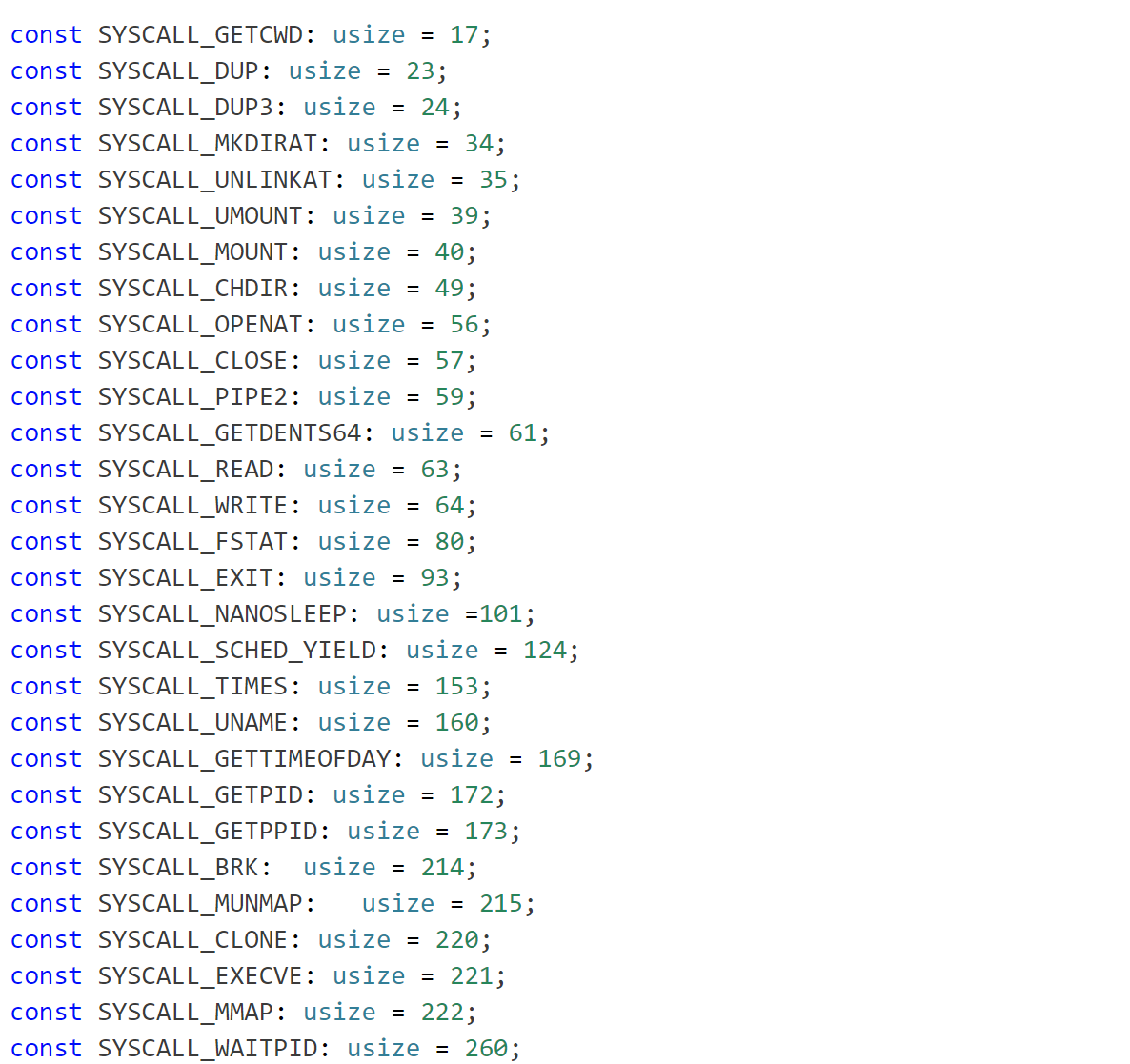
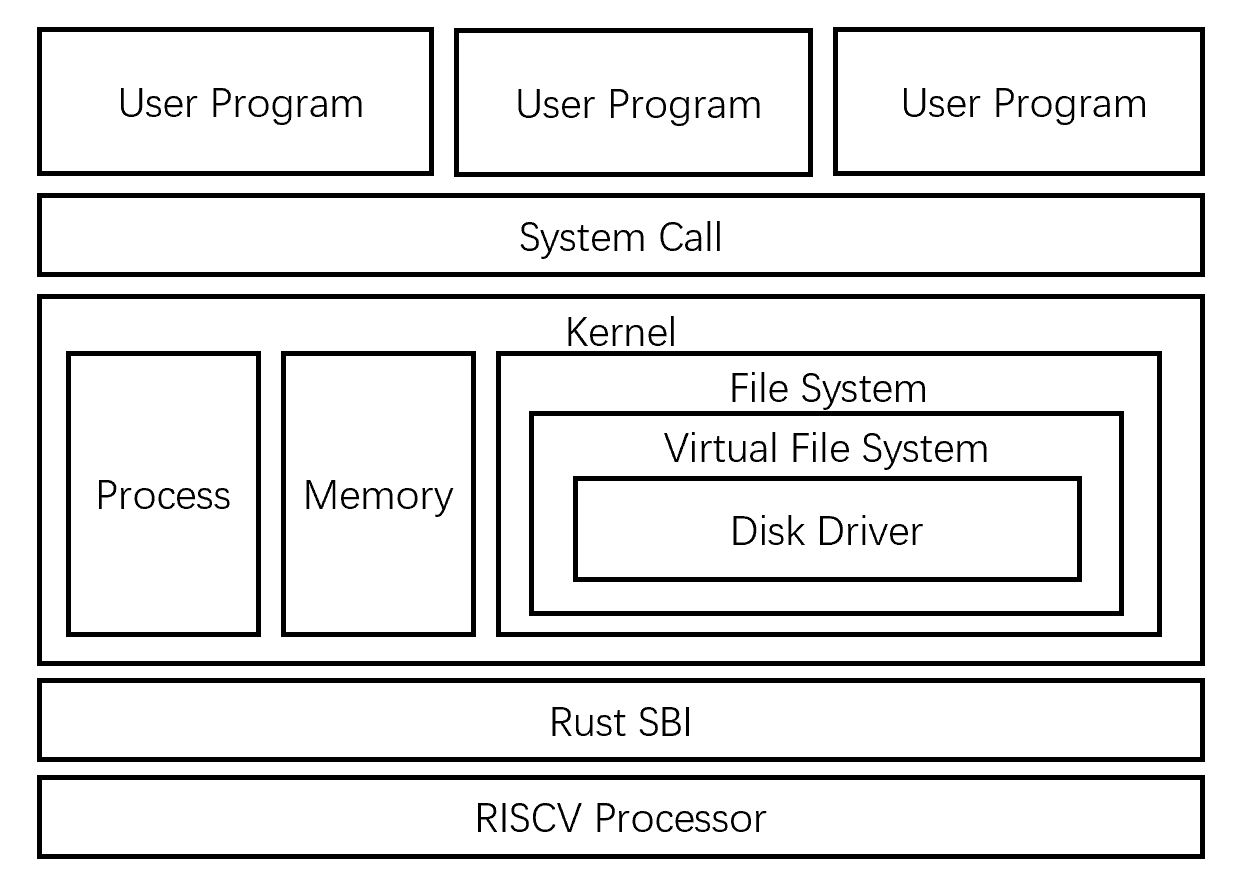


**MoOS**

**系统设计**

该OS为实验性OS，仅为了小组成员熟悉OS架构以及RUST语言所写，作为一个基础框架支持后续的性能优化，内存部分参考了rCore-Tutorial实现。目前支持的系统调用如下，也即所有初赛要求的系统调用：





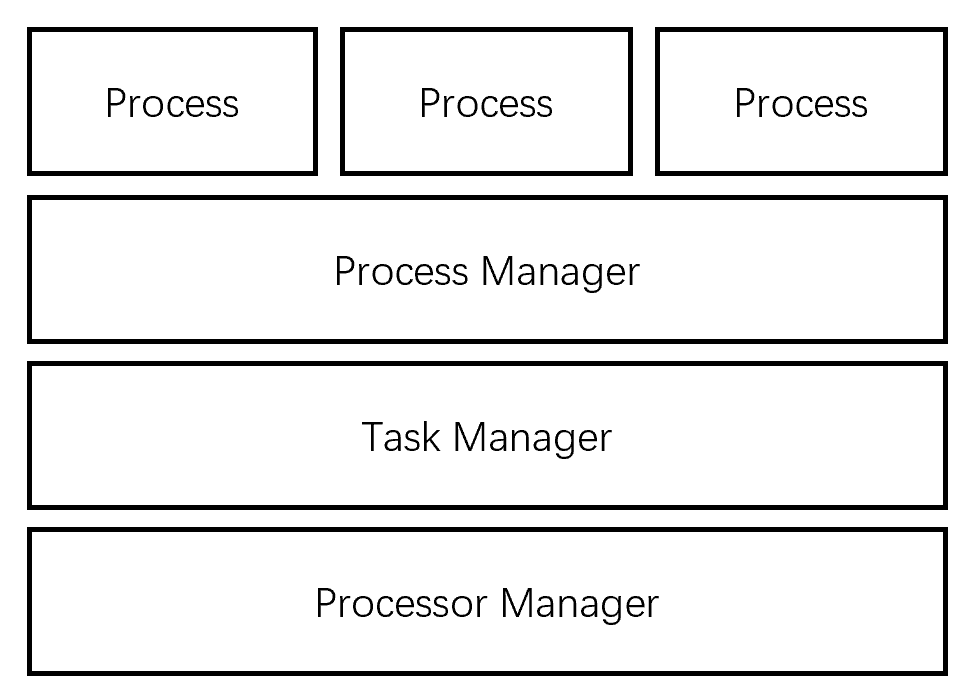
MoOS采用了分层模块设计的思想，将操作系统的功能划分为不同的层次和模块。MoOS的主要部分由两个层次构成：核心态和用户态。核心态是运行在Supervisor模式下的内核，它负责操作系统的核心功能，如进程管理、内存管理和文件系统。用户态是运行在User模式下的用户程序，它们通过系统调用与内核交互，实现各种应用功能。MoOS提供了一个基本的用户态程序，即shell，它可以接收用户的命令，并执行相应的操作。MoOS还提供了C语言标准库，使得用户可以使用C语言开发自己的用户态程序。

MoOS的核心态内核与硬件之间通过Rust SBI进行通信。Rust SBI是一个基于Rust语言开发的软件二进制接口（SBI），它实现了RISC-V SBI规范，并提供了一些扩展功能。Rust SBI可以屏蔽硬件的差异，为上层的操作系统提供统一和安全的接口。MoOS的核心态内核与用户态程序之间通过系统调用进行通信。MoOS的系统调用遵循Unix规范，提供了一系列标准的接口，如文件操作、进程控制、信号处理等。

MoOS的核心态内核主要由三个模块组成：进程管理、内存管理和文件系统。进程管理模块负责创建、销毁、调度、中断、唤醒等进程相关的功能；内存管理模块负责虚拟内存和物理内存的管理；文件系统包含一个虚拟文件系统，能够通过各级抽象统一调用接口，自管理缓存实现内存与外设的高速准确交互。

**模块设计**

**1．进程管理**



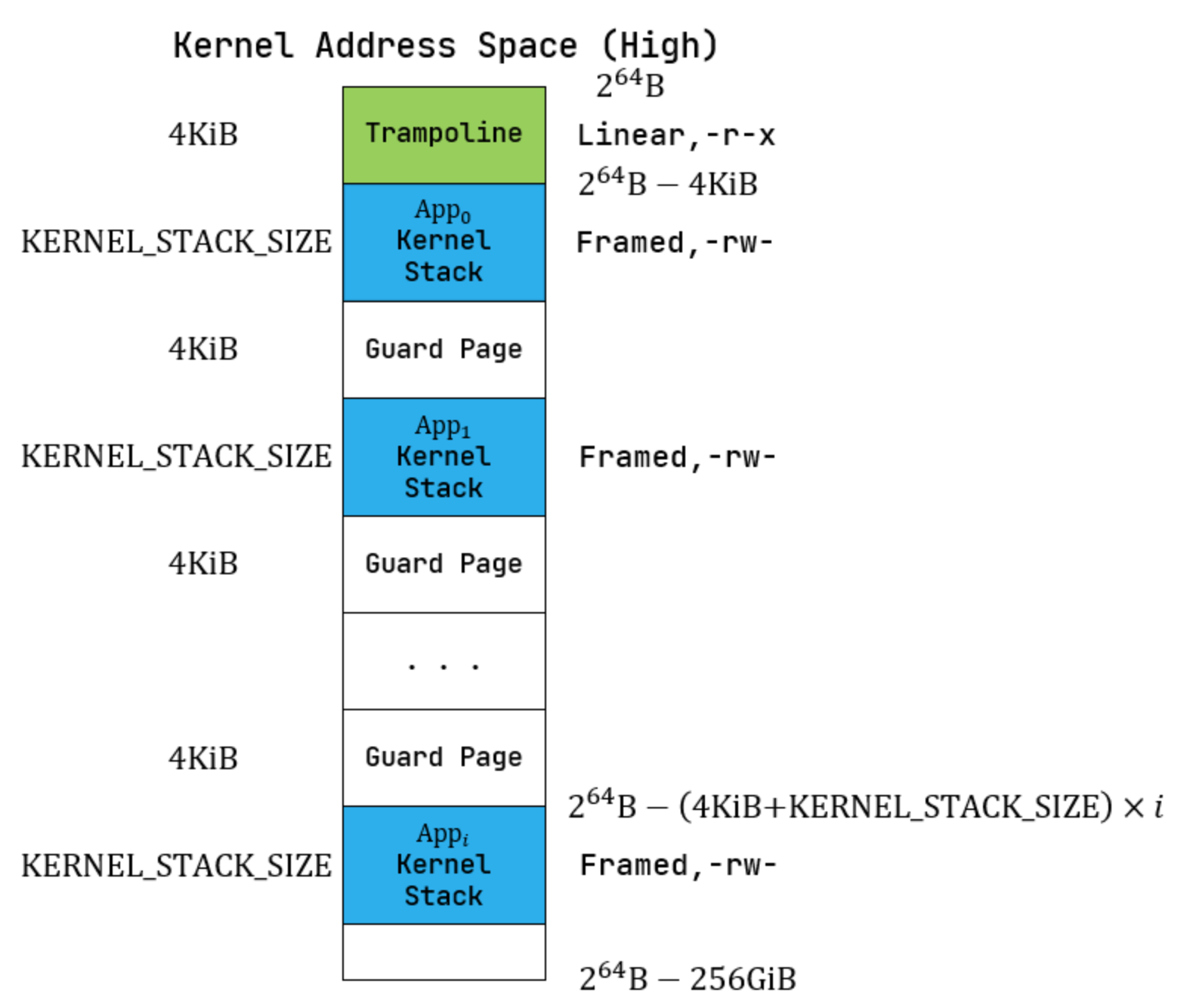
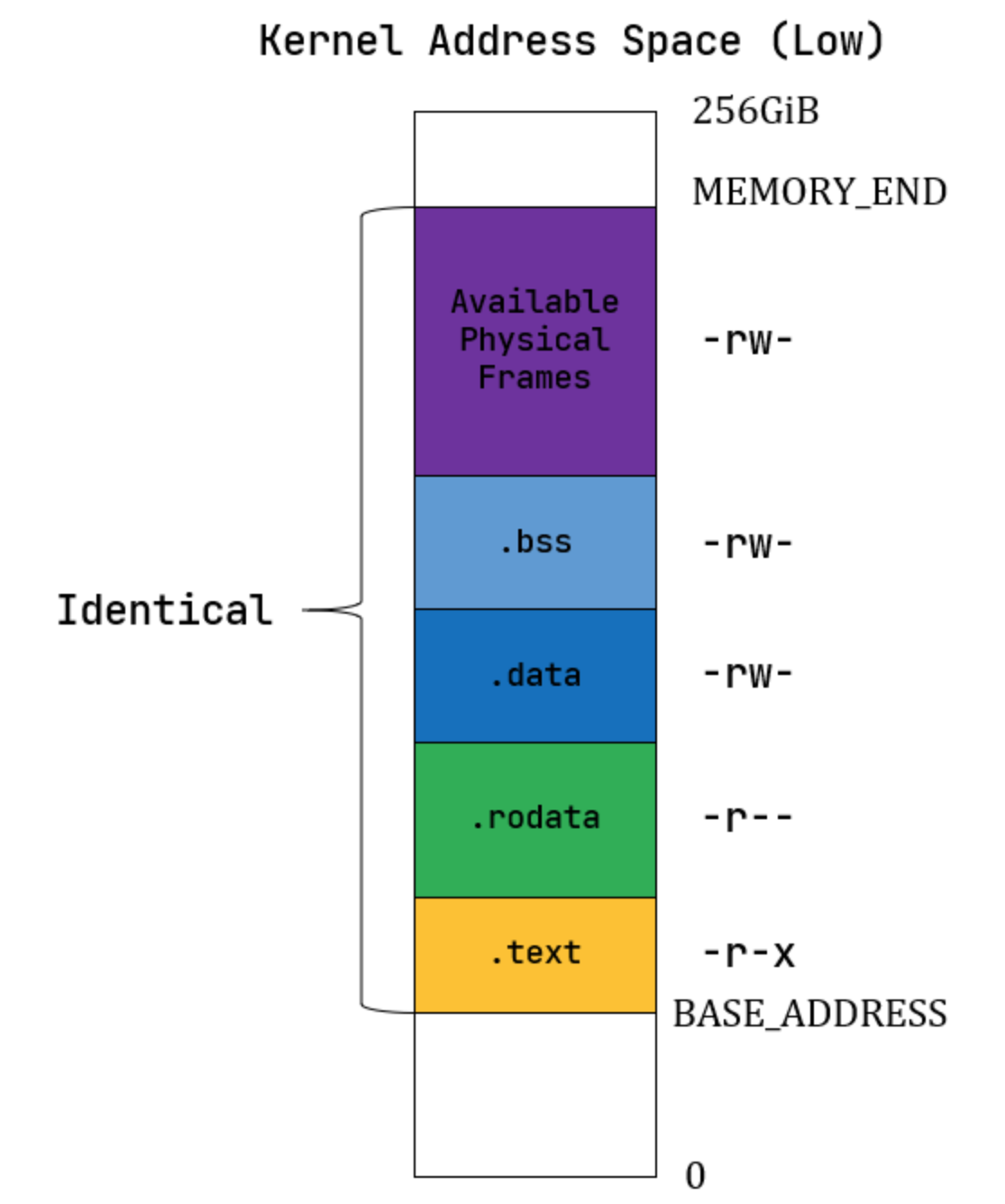
进程管理是操作系统的一个重要功能，它负责对系统中的所有进程进行有效的管理和控制。为了实现这一功能，MoOS将进程管理分为三个层次，分别是处理器管理、进程管理和进程控制。每个层次都对应了不同的抽象对象和功能模块。

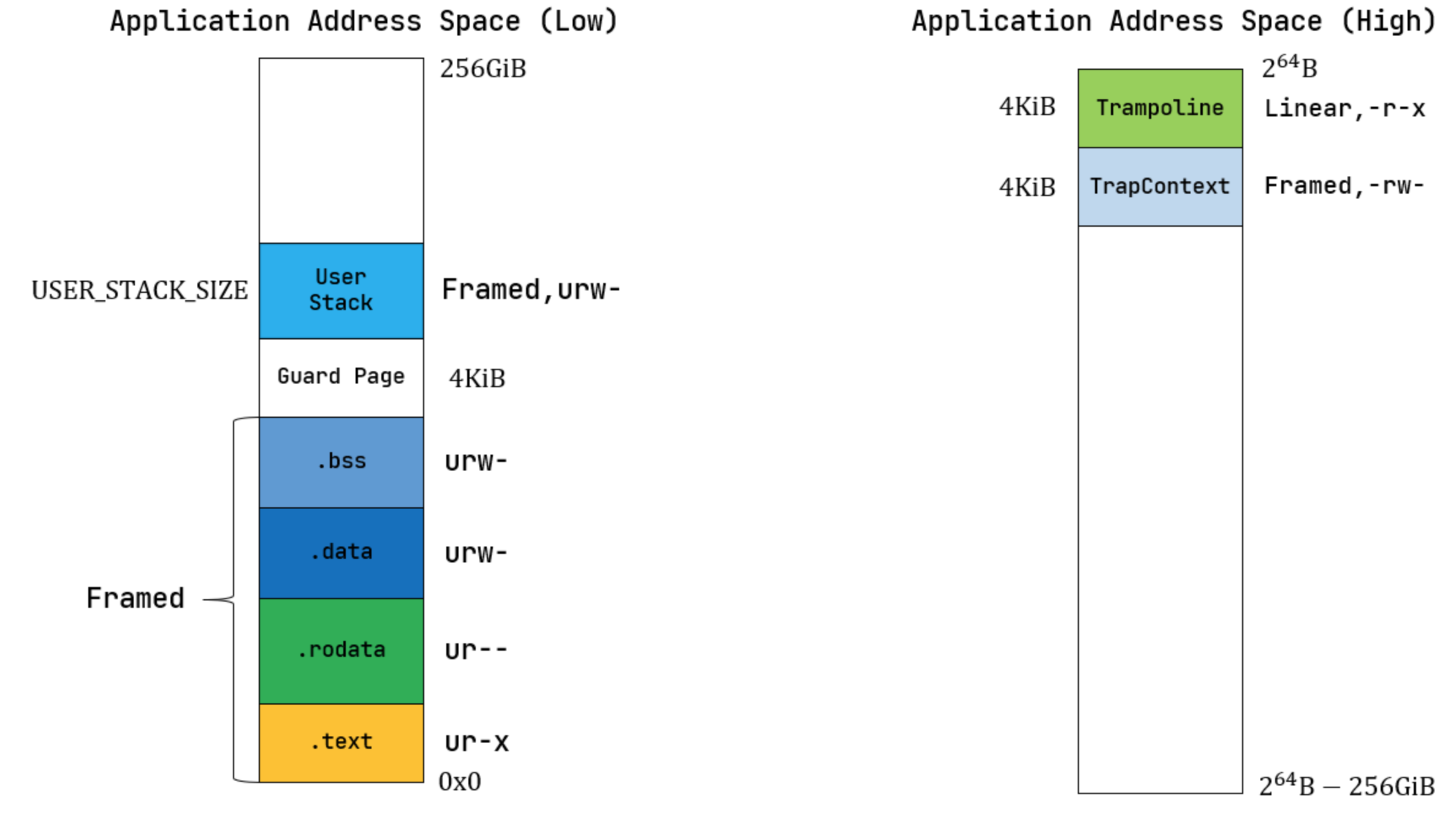
处理器管理是最高层次的进程管理，它主要负责处理机或CPU的分配和调度。在这一层次中，操作系统把处理机抽象为一个或多个逻辑处理器，每个逻辑处理器可以运行一个进程。操作系统维护了一个就绪队列，用来存放等待运行的进程。同时，操作系统还提供了一个调度器，用来根据一定的算法选择合适的进程送入处理机运行。目前实现的调度器为轮转调度器，即按照PID（进程标识符）的顺序就近原则进行调度，被调度器选中的进程将被加载运行。为了保证调度过程的正确性和一致性，操作系统在进行调度时持有全局锁，也即只允许一个核心占有调度器。

进程管理是中间层次的进程管理，它主要负责对系统中存在的所有进程进行统一的管理和维护。在这一层次中，操作系统把每个进程抽象为一个PCB（进程控制块），PCB是用来描述和记录进程各种状态信息的数据结构。操作系统维护了一个PCB集合或表格，用来存放所有进程的PCB。在这一层次并未集成大量功能，其职责是保证多核心下对于PCB集合或表格的互斥访问。为了实现这一点，操作系统使用了锁或信号量等同步机制。

进程控制是最低层次的进程管理，它主要负责对单个进程进行具体的控制和操作。在这一层次中，操作系统把每个PCB作为一个控制器，用来控制该PCB所代表的进程。每个控制器里集成了该PCB所需的全部资源和功能模块，如内存分配、文件打开、信号处理等。由于这些资源和功能模块与内存系统、文件系统等其他子系统强耦合，因此这一层次也是操作系统与其他子系统交互最频繁的地方。

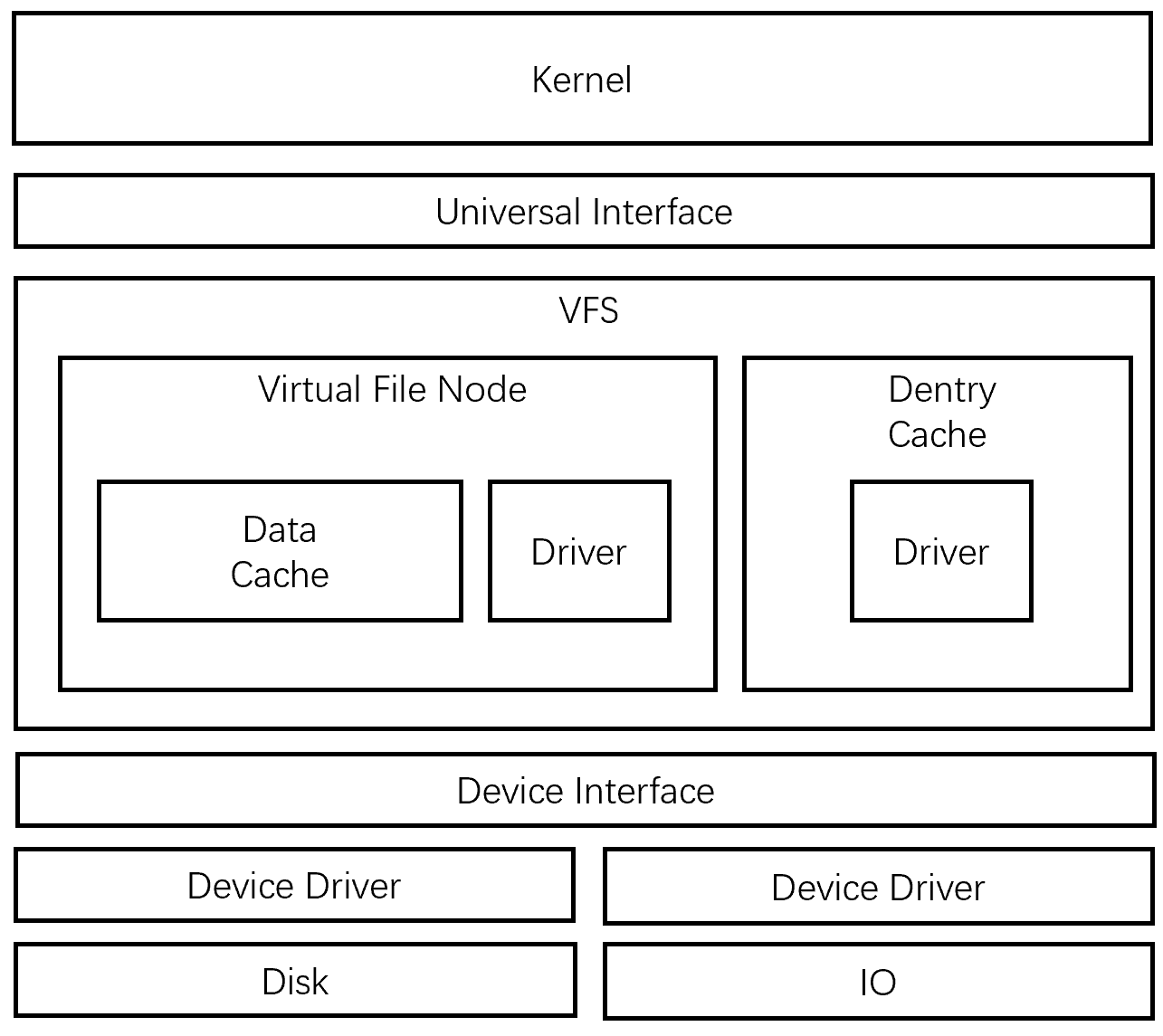
**2. 内存管理**



内存布局与rCore-tutorial 一致，采用分离的内核态页表和用户态页表，内核栈放置于内核空间中，每一进程内核占内核空间的一个绝对位置，由PID决定。对于内核态与用户态的数据交换采用复制移动的方式。为了保证Trap的连贯性，在用户空间顶部映射了Trampoline和Trap Frame。用户堆和页表项与内核堆一起由一个分配栈统一分配。

**3. 文件系统**



文件系统是一个与硬件强耦合且非常复杂的部分。它主要负责两方面的工作：一是维护操作系统目录文件系统的结构和内容，二是实现操作系统与外设之间的数据交互。由于内存与外设之间存在着巨大的速度差异，为了保证高效地读写数据，这一部分需要设计大量分层缓存机制。同时，由于外设设备种类繁多且不同类型之间有很大差异，为了简化内核对外设访问的逻辑和代码量，这一部分还需要向内核提供一个统一且抽象化的接口。因此，我们将文件系统分为三个层次：接口层、文件节点层和数据缓存层。

接口层包含向上与内核通信的接口层，与负责负责与硬件和外设进行直接通信的硬件驱动层。它包括物理设备驱动和设备翻译两个子层。物理设备驱动是针对不同类型的物理设备（如SD卡、串口等）编写的程序，它可以直接控制物理设备进行读写操作。设备翻译是针对不同类型的磁盘布局（如FAT32、EXT4等）编写的程序，它可以将虚拟文件节点中表示数据位置和大小的信息转换为物理设备上具体扇区和字节位置。这两个子层共同构成了驱动层，它们可以屏蔽不同硬件和外设之间的差异，向上层提供一个统一且简单化的读写接口。

文件节点层，它主要负责管理管理文件节点。它将所有可能被内核访问到的字节流（如普通文件、管道、mmap节点等）统一抽象为虚拟文件节点，并为每个虚拟文件节点提供了open\_at、read\_at、write\_at、close等基本操作接口。同时，它还维护了一个完整的文件目录树结构，并为其提供了路径查找、创建、删除等操作接口。这些接口都是通过调用下层驱动层来实现具体功能的。虚拟文件系统层可以向内核提供一个统一且抽象化的访问接口，使得内核无需关心具体物理设备和磁盘布局。

每个虚拟文件节点都设置了一个独立的数据缓存区域，并根据不同类型节点采用不同策略来管理缓存内容。例如，在普通文件节点中，在内存有限的情况下采用一些置换算法来淘汰最近最少使用过的缓存块；在管道节点中，采用FIFO算法来按顺序传输缓存块；在mmap节点中，缓存写回时结合mmap标志位来阻止非法的写入等。数据缓存层可以利用大容量内存和局部性原理来提高数据读写速度，并减少对下层驱动层的调用次数。

在本次比赛中，由于我们有足够大的内存空间可供使用，并且所有读写需求都可以在程序开始和结束时集中处理（即挂载和卸载时），所以我们没有实现缓存换出机制，并且在挂载时就将所有需要访问到的数据全部加载进了缓存区域。这样做可以极大地提高程序运行效率，并简化缓存管理逻辑。

**测试与交互**

我们在用户态实现了一个shell程序，可以用户简单的人机交互、调用用户态程序；一些简单的压力测试程序，以及测试用例自动化运行脚本程序，用于测例的自动运行。

在shell模式下输入runtest 命令或者在自动模式下直接运行会挂载磁盘并依次运行初赛测例文件。



初赛样例能够全部通过