天文台智能观测运行操作系统

胡 义

中国科学院国家天文台

提纲

- 为什么需要天文台观测运行操作系统；
- 什么是观测系统；
- 观测系统的设计；
- 观测系统的实现；

- 程控自主观测成为（中小口径）望远镜的主流：
  - 提高观测效率；
  - 降低运行费用；
  - 提高设备安全；
  - 极端恶劣环境；

- 自动天文台运行控制软件：
  - RTS-2 (Kubanek et al. 2006, *BOOTES*)和其变种；
  - ASCOM (Windows-based，商业产品)；
  - AST3Suite (Hu et al. 2016, Ma et al. 2020, 高度定制)；
  - Auto-KLDIMM (Shang et al. 2018, Ma et al. 2020, 沿用部分ast3suite代码)；

为什么需要观测系统

* 通用性（用户程序不依赖于具体硬件）；
* 易用性：
  - 对用户，易于编写自动运行程序；
  - 对开发者，易于添加新型号设备和新的功能；
* 可用性和稳定性；
* 高自动化程度（由程控向无人值守运行的进化）；
* 智能化；
* 集群控制（多望远镜协同观测，e.g.，司天工程）；

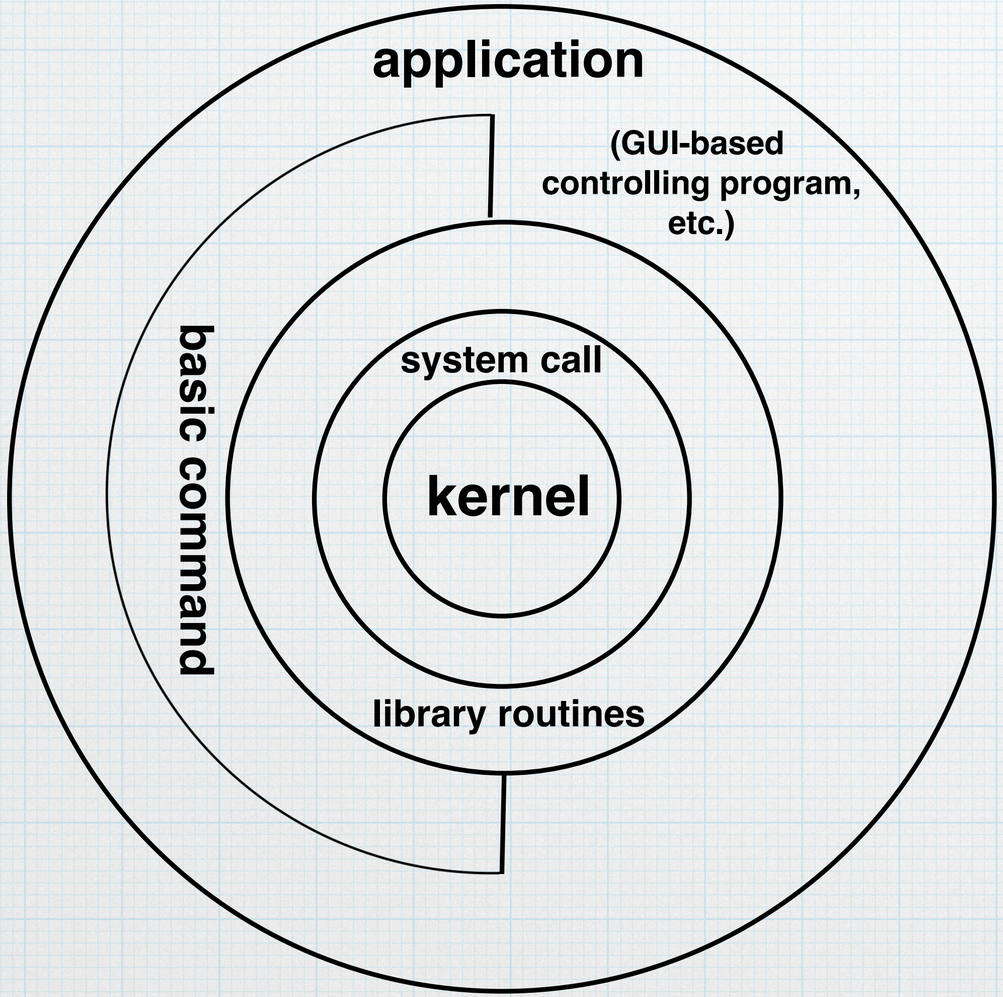
什么是观测系统

* 观测系统是天文台的操作系统，其功能包括：
  - ✓ 管理和协调天文台所有种类的硬件设备；
  - ✓ 代表用户程序操作硬件；
  - ✓ 提供设备设置和控制运行的统一接口；
  - ✓ 提供设备测试和运行的工具链；

什么是观测系统

- 观测系统不包含科学调度功能；
  - 留有调用和反馈接口；
- 观测系统不是硬件固件/驱动程序；
  - 与硬件驱动紧密配合实现运行功能；
  - 实现自我故障探测、报警和恢复功能；

类计算机操作系统的设计



* 内核：　　与硬件通讯；
* 系统调用：　请求内核服务；
* 库函数：　　系统调用包装；
* 基本命令：　命令行调试工具；
* 用户程序；

系统调用和库函数标准化

- 定义设备操作指令集（映射为库函数）；
- 每个指令，定义：
  - ✓指令输入、输出和功能；
  - ✓执行返回代码（同步）；
  - ✓异步取消行为；
  - ✓指令间的相互作用；

对比计算机操作系统文件操作

创建	• **Creating a fil** . Two steps are necessary to create a file. First, space in the file system must be found for the file. We discuss how to allocate space for the file in Chapter 14. Second, an entry for the new file must be made in a directory.	`creat()/open()`
打开	• **Opening a fil** . Rather than have all file operations specify a file name, causing the operating system to evaluate the name, check access permissions, and so on, all operations except create and delete require a file `open()` first. If successful, the open call returns a file handle that is used as an argument in the other calls.	`open()`
写入	• **Writing a fil** . To write a file, we make a system call specifying both the open file handle and the information to be written to the file. The system must keep a **write pointer** to the location in the file where the next write is to take place if it is sequential. The write pointer must be updated whenever a write occurs.	`write()`
读取	• **Reading a fil** . To read from a file, we use a system call that specifies the file handle and where (in memory) the next block of the file should be put. Again, the system needs to keep a **read pointer** to the location in the file where the next read is to take place, if sequential. Once the read has taken place, the read pointer is updated. Because a process is usually either reading from or writing to a file, the current operation location can be kept as a per-process **current-file-positio pointer**. Both the read and write operations use this same pointer, saving space and reducing system complexity.	`read()`
重定位	• **Repositioning within a file**. The current-file-position pointer of the open file is repositioned to a given value. Repositioning within a file need not involve any actual I/O. This file operation is also known as a file **seek**.	`lseek()`
删除	• **Deleting a fil** . To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase or mark as free the directory entry. Note that some systems allow **hard links**—multiple names (directory entries) for the same file. In this case the actual file contents is not deleted until the last link is deleted.	`delete()`
截断	• **Truncating a fil** . The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged—except for file length. The file can then be reset to length zero, and its file space can be released.	`trancate()`

from *Operating System Concepts*

* 操作系统只定义少数的一些系统调用；
* 用户程序通过系统调用实现统一的的文件操作；
* 操作系统完成用户层的抽象和硬件通讯；

对比计算机操作系统文件操作

创建
- **Creating a fil** . Two steps are necessary to create a file. First, space in the file system must be found for the file. We discuss how to allocate space for the file in Chapter 14. Second, an entry for the new file must be made in a directory.

creat()/open()

打开
- **Opening a fil** . Rather than have all file operations specify a file name, causing the operating system to evaluate the name, check access permissions, and so on, all operations except create and delete require a file `open()` first. If successful, the open call returns a file handle that is used as an argument in the other calls.

open()

写入
- **Writing a fil** . To write a file, we make a system call specifying both the open file handle and the information to be written to the file. The system must keep a **write pointer** to the location in the file where the next write is to take place if it is sequential. The write pointer must be updated whenever a write occurs.

write()

读取
- **Reading a fil** . To read from a file, we use a system call that specifies the file handle and where (in memory) the next block of the file should be put. Again, the system needs to keep a **read pointer** to the location in the file where the next read is to take place, if sequential. Once the read has taken place, the read pointer is updated. Because a process is usually either reading from or writing to a file, the current operation location can be kept as a per-process **current-file-positio pointer**. Both the read and write operations use this same pointer, saving space and reducing system complexity.

read()

重定位
- **Repositioning within a file**. The current-file-position pointer of the open file is repositioned to a given value. Repositioning within a file need not involve any actual I/O. This file operation is also known as a file **seek**.

lseek()

删除
- **Deleting a fil** . To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase or mark as free the directory entry. Note that some systems allow **hard links**—multiple names (directory entries) for the same file. In this case the actual file contents is not deleted until the last link is deleted.

delete()

截断
- **Truncating a fil** . The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged—except for file length. The file can then be reset to length zero, and its file space can be released.

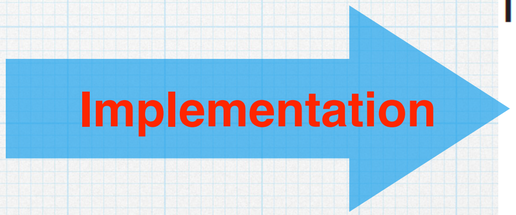
trancate()

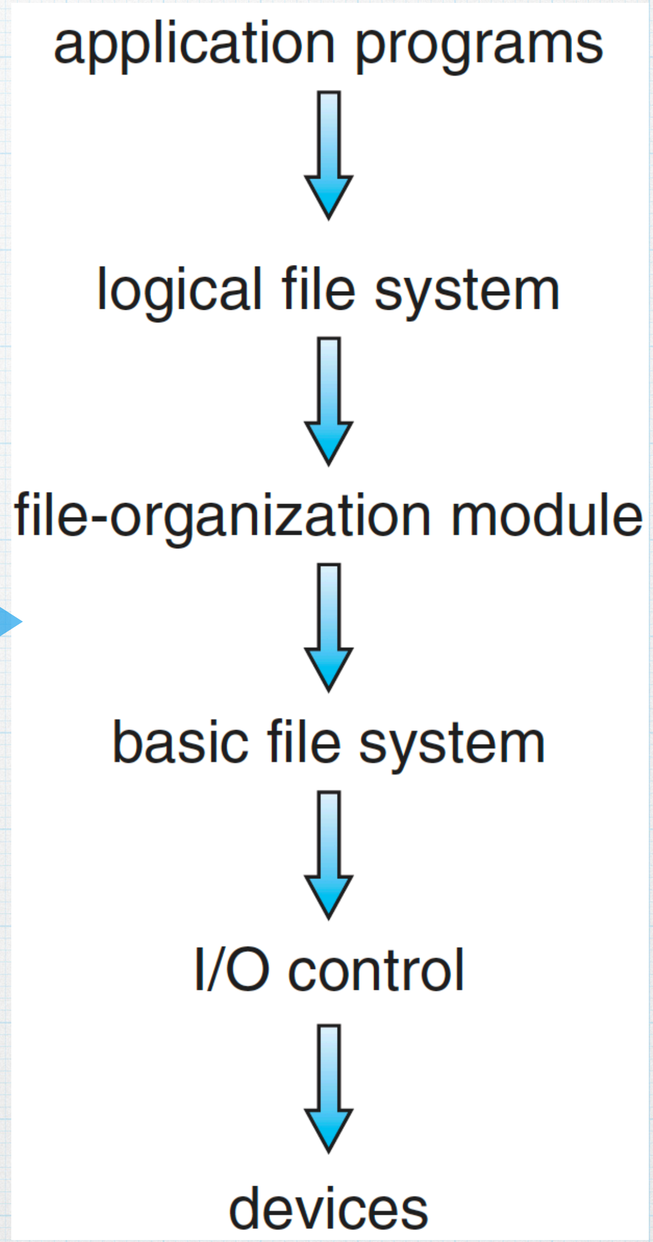
from *Operating System Concepts*

对比计算机操作系统文件操作

创建	• **Creating a fil** . Two steps are necessary to create a file. First, space in the file system must be found for the file. We discuss how to allocate space for the file in Chapter 14. Second, an entry for the new file must be made in a directory.	`creat()/open()`
打开	• **Opening a fil** . Rather than have all file operations specify a file name, causing the operating system to evaluate the name, check access permissions, and so on, all operations except create and delete require a file `open()` first. If successful, the open call returns a file handle that is used as an argument in the other calls.	`open()`
写入	• **Writing a fil** . To write a file, we make a system call specifying both the open file handle and the information to be written to the file. The system must keep a **write pointer** to the location in the file where the next write is to take place if it is sequential. The write pointer must be updated whenever a write occurs.	`write()`
读取	• **Reading a fil** . To read from a file, we use a system call that specifies the file handle and where (in memory) the next block of the file should be put. Again, the system needs to keep a **read pointer** to the location in the file where the next read is to take place, if sequential. Once the read has taken place, the read pointer is updated. Because a process is usually either reading from or writing to a file, the current operation location can be kept as a per-process **current-file-positio pointer**. Both the read and write operations use this same pointer, saving space and reducing system complexity.	`read()`
重定位	• **Repositioning within a file**. The current-file-position pointer of the open file is repositioned to a given value. Repositioning within a file need not involve any actual I/O. This file operation is also known as a file **seek**.	`lseek()`
删除	• **Deleting a fil** . To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase or mark as free the directory entry. Note that some systems allow **hard links**—multiple names (directory entries) for the same file. In this case the actual file contents is not deleted until the last link is deleted.	`delete()`
截断	• **Truncating a fil** . The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged—except for file length. The file can then be reset to length zero, and its file space can be released.	`trancate()`

from *Operating System Concepts*





望远镜的指令集

望远镜指令集 [编辑]

status指令

status指令功能是获取望远镜的当前运行参数。运行参数的数目取决于具体硬件，但是必须包含当前望远镜指向的RA和DEC，以及望远镜所处的状态。status指令是必须的。status指令是快指令。

raw指令

raw指令的功能是向望远镜发送原始命令。raw仅供硬件研发人员调试使用，不会更新望远镜状态，会导致系统记录望远镜状态与实际硬件状态不一致。raw指令与望远镜指令集中的其他指令混用的行为是undefined behavior。raw指令是可选的。raw指令可以是快指令也可以是慢指令。

slew指令

slew指令功能为将望远镜指向观测目标。slew执行过程中，望远镜处于SLEWING状态；slew成功执行完毕，无论望远镜之前处于何种运动状态，之后望远镜都处于TRACKING状态，这时可以开始曝光。slew在执行过程中可以被后续新的slew、move、stop、park、go_home指令打断，当前slew指令执行提前结束。slew 指令是必须的。slew指令是慢指令。

move指令

move指令的功能是将望远镜的单个轴以设定的速度向设定的方向移动设定的时间。move指令执行过程中，望远镜处于MOVING状态；move成功执行完毕，无论望远镜之前处于何种运动状态，之后，之后望远镜都处于TRACKING状态。move在执行过程中可以被后续新的move、slew、stop、park、go_home指令打断，当前move指令执行提前结束。move指令是可选的。move指令是慢指令。

power_on指令

power_on指令的功能是开启望远镜所有设备的电源。power_on指令执行成功后，望远镜处于UNINITIALIZED状态。但是，如果电源已经打开，再次执行power_on指令不会产生任何效果。power_on指令是可选的。power_on指令是快指令

power_off指令

power_off指令的功能是关闭望远镜所有设备的电源。如果电源已经关闭，再次执行power_off指令不会产生任何效果。power_off指令执行成功后望远镜处于POWER_OFF状态。power_off指令是可选的。power_off指令是快指令。

init指令

望远镜在通电后处于UNINITIALIZED的状态，这时一般需要对其进行初始化。初始化的操作取决于具体的望远镜硬件，可能的操作有，向硬件提供当前时间、台站地理坐标、望远镜当前姿态等等信息。init指令的功能是完成望远镜的初始化。init指令执行成功后，望远镜处于PARKED的状态。如果望远镜已经成功初始化，之后执行init指令不会产生任何效果。init指令是必须的。init指令是快指令。

park指令

park指令的功能是将望远镜停靠在当前位置。park成功执行完毕，无论望远镜之前处于何种运动状态，之后望远镜都处于PARKED状态。park指令是必须的。park指令是快指令。

park_off指令

park_off指令的功能是将望远镜双轴运动设置为跟踪模式。park成功执行完毕，如果望远镜之前处于PARKED状态，之后望远镜处于TRACKING状态；如果望远镜之前处于MOVING、SLEWING或TRACKING状态，执行park指令不产生任何效果。park_off指令是必须的。park_off指令是快指令。

stop指令

stop指令功能是强行将望远镜双轴运动设置为跟踪模式。stop成功执行完毕，如果望远镜之前处于MOVING和SLEWING状态，立刻停止move和slew，之后望远镜处于TRACKING状态；如果望远镜之前处于PARK状态，之后望远镜处于TARCKING状态；如果望远镜之前处于TRACKING状态，执行stop指令不产生任何效果。stop指令是必须的。stop指令是快指令。

go_home指令

go_home指令的功能是将望远镜指向到预先定义的的位置后停靠在该位置。预先定义的位置应当是望远镜望远镜断电前停靠的安全位置，如果有圆顶，在这个位置下关闭也能安全关闭圆顶。go_home执行过程中，望远镜处于SLEWING状态；go_home成功执行完毕，无论望远镜之前处于何种运动状态，之后望远镜都处于PARKED状态。go_home在执行过程中可以被后续新的go_home、slew、move、stop、park指令打断，当前go_home指令执行提前结束。go_home指令是可选的。go_home指令是慢指令。

http://wiki.kunlunstation.org

- **init**
- **slew**
- **move**
- **stop**
- **park**
- **park off**
- **status**
- **…**

系统调用和库函数标准化

- 定义设备操作指令集（映射为库函数）；

- 每个指令，定义：

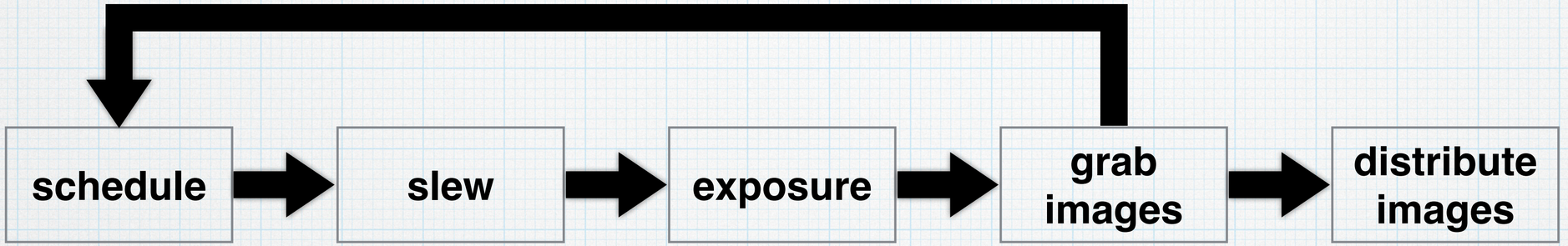
  ✓指令输入、输出和确切功能；

  ✓执行返回代码（同步）；

  ✓取消行为（异步）；

  ✓指令间的相互作用；

系统调用和库函数标准化



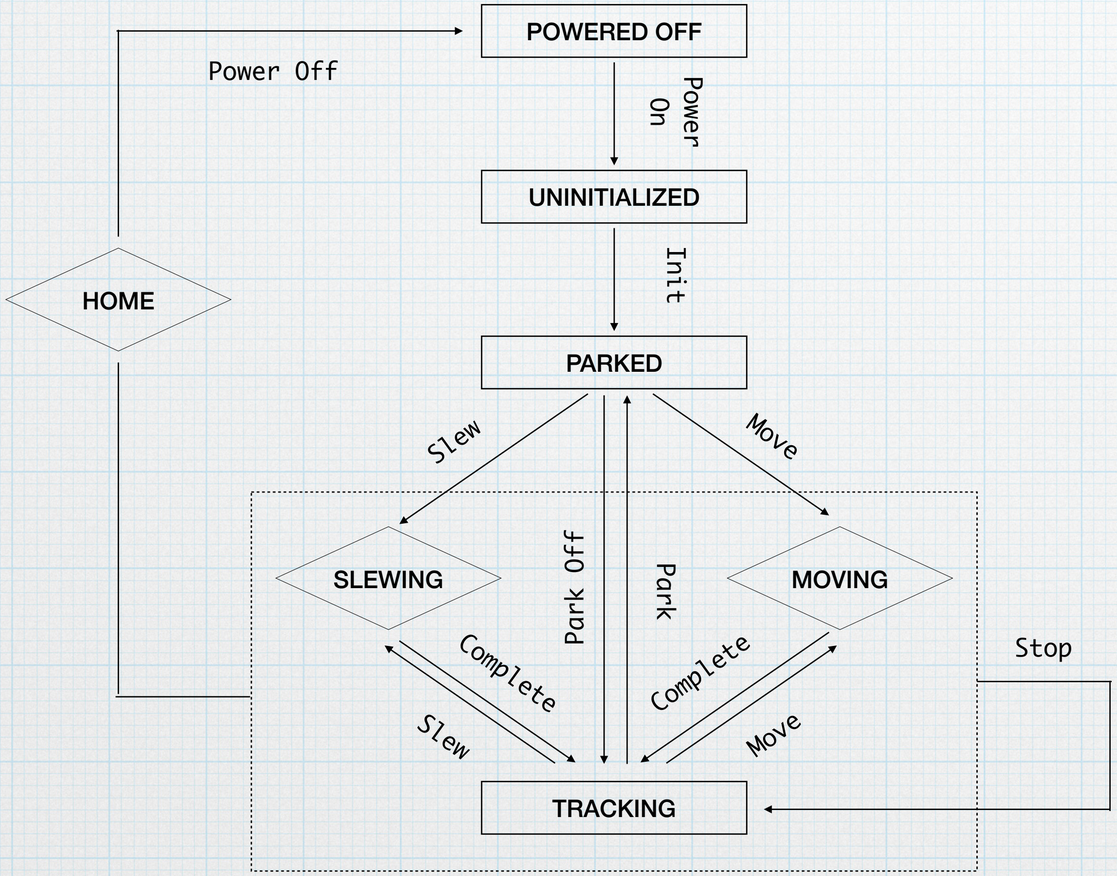
自动观测流程

- slew指令执行完成后意味着什么?
- slew执行执行失败后怎么办?
- 在slew过程中如果因为天气原因需要立刻停止望远镜，望远镜的行为是什么?

系统调用和库函数标准化

- 定义设备操作指令集（映射为库函数）；
- 每个指令，定义：
  - ✓指令输入、输出和确切功能；
  - ✓执行返回代码（同步）；
  - ✓取消行为（异步）；
  - ✓指令间的相互作用；

系统调用和库函数标准化



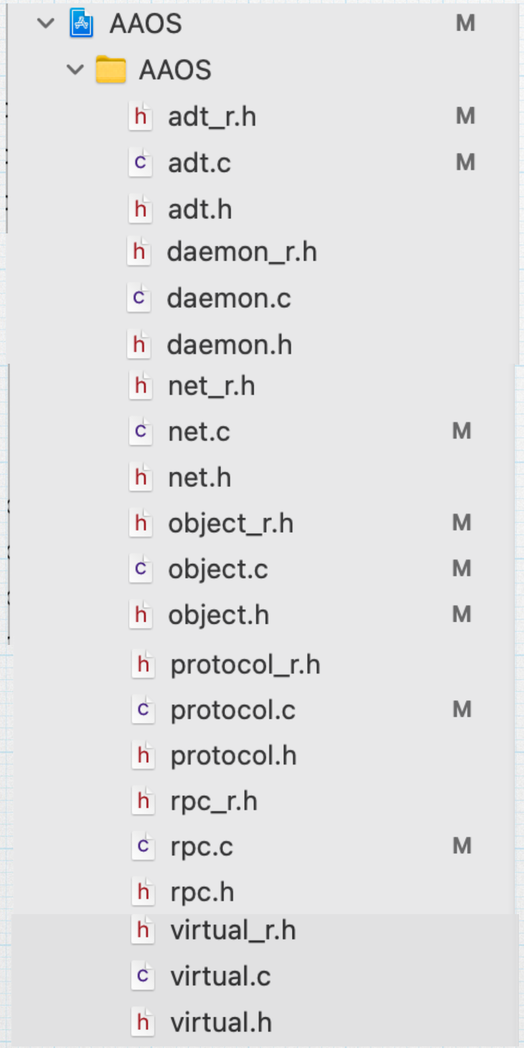
指令间的相互作用和状态迁移图

* 望远镜只处于有限数目的状态；
* 望远镜指令执行会改变望远镜的状态，且行为唯一；
* 定义指令间相互作用保证望远镜的状态一致性；

观测系统的实现

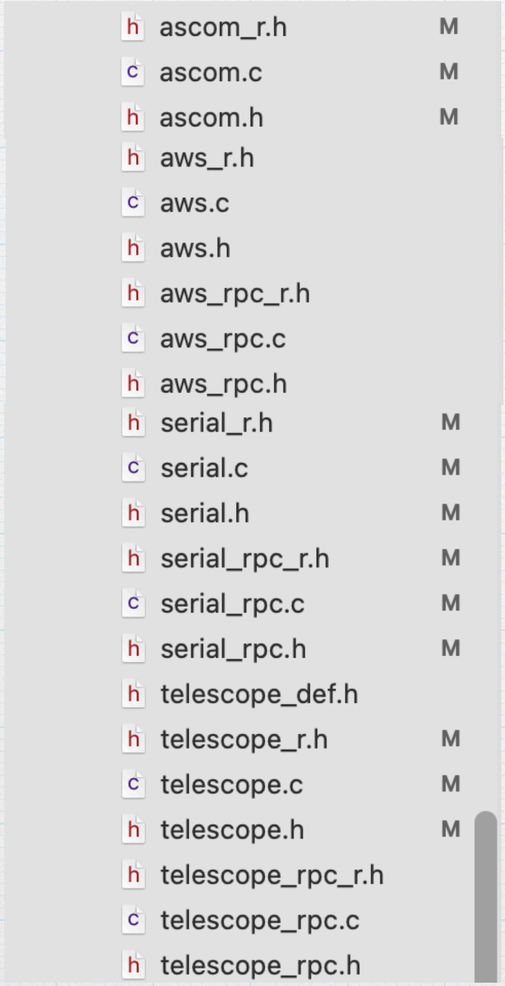
* 基于消息传递的微内核；
  - 稳定性；
  - 可容错；
* 服务器/客户端（C/S）构架；
  - 适合于远程和集群控制；
* 系统级的编程语言（C/C++, Rust?Go?）；
* 面向对象的编程风格（内核实现）；

观测系统的实现: AAOS (developing)



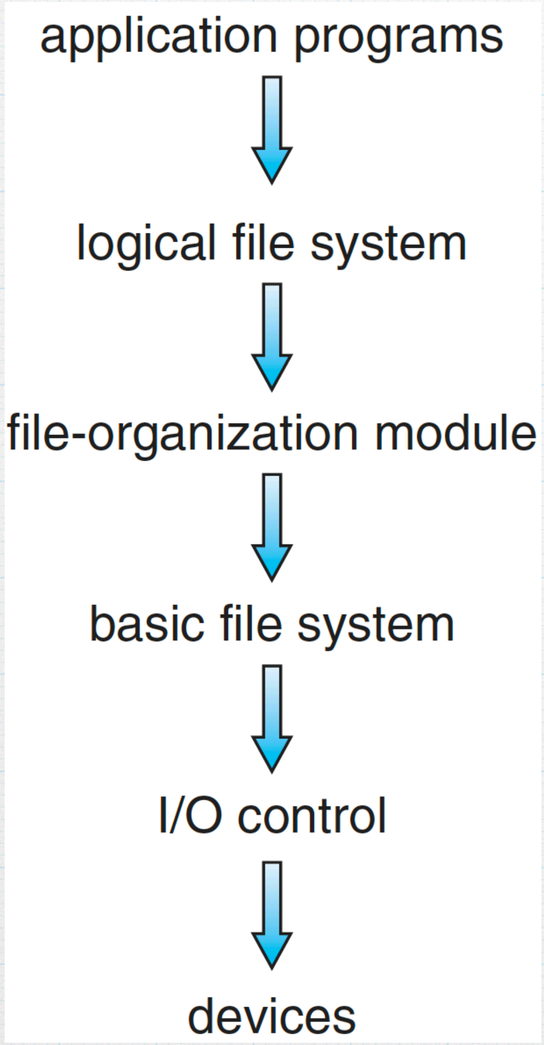
- 基础库:
  - 面向对象库;
  - 线程安全数据结构;
  - 基础网络库和远程调用库;

观测系统的实现: AAOS (developing)



- 内核部分功能：
  - 串口设备；
  - 自动气象站子系统；
  - 望远镜设备（虚拟望远镜，AstroPhysics 赤道仪，ASCOM Alpaca）；

AAOS实现slew功能




```
telescope_slew(void *telescope, double ra, double dec)
```

library function

translate to generic instruction

“driver”

```
rpc_call(void *rpc)
```

system call

```
rpc_process(void *rpc)
```

real device driver

```
“:Sr xxx#:Sd xxx#:MS#”
```

```
serial_raw(void *rpc)
```

library function

总结

* 自动天文台观测运行系统是操作系统级别的软件，类操作系统设计；
* 系统标准化是观测系统研发中的最为关键要素；
* 消息传递和基于网络的C/S构架体系；

谢 谢！ ！