操作系统笔记

上官凝

2020 年 9 月 1 日

目录

第 1 章　导语

1.1　序幕

曾经有一位计算机学院的教授说过，计算机是文科。现在我信了。

建议多翻 PPT，以及 CS:APP。这门课的主题，正如 PPT 上所言（process lifecycle, process management, and other related issues are essential topics of this course），是围绕着进程来说的。注意每一个 PPT 的 summary。

OS 和 COD 除去绪论，我认为都是一个由简单到复杂，不断发现异常、引入新理论解决异常的过程。不过很多时候最终的往往是做了 tradeoff，比如内存不够的时候，可以用虚存。贴一张图[1]：

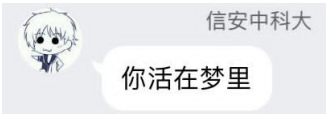

图 1.1: Virtual Memory

1.2　我愿意管他们叫定义

万一出个选择呢

1. The operating system contains the codes that are needed to work with the file system. The codes are called the kernel.
2. “The one program running at all times on the computer”is the kernel.
3. Kernel is the place where the system call implements.
4. C is the only language to interact with the OS directly.
5. Mechanisms determine ***how*** to do something; policies determine ***what*** will be done.
6. If a process fails, most operating systems write the error information to a log file to alert system operators or users that the problem occurred. The operating system can also take a core dump.

[1]看到虚存那一页 PPT 就理解这个梗了 233333

7. A process will switch its execution from user space to kernel space through invoking system call
8. Windows maintains a forest-like hierarchy. (ch3 p39)
9. A thread starts with one specific function. We name it the thread function. 其实就是说执行进程中的代码段，执行到某一句的时候，需要开线程了
10. Critical sections is the code segment that is accessing the shared object, and it should be kept as tight as possible.
11. A deadlock-free solution does not eliminate **starvation**
12. 数据存储里面的一个缩写：BSS (Block Started by Symbol) Containing uninitialized global and static variables.
13. A function can ask the CPU to read and to write anywhere in the stack, not just the "zone"belonging to the running function!
14. 在虚拟内存分页的时候，Unallocated zone does not occupy any pages.
15. Each process should have its own page table.
16. If a process does not have enough frames –number of frames required to support pages in active use, frequent page fault would occur, replacing a page that will be needed again right away. This is called thrashing（抖动）, spending more time paging than executing.
17. Opening a file only involves the pathname and the attributes of the file, instead of the file content! A file descriptor returned.
18. FAT 12/16/32: A block is named a cluster.
19. GDT: group descriptor table (in Ext FS, ch10 part2 page 6)
20. The number of files in the file system is fixed! Since…the inode bitmap has a fixed size.
21. A hard link is a directory entry pointing to an existing file. Conceptually speaking, this creates a file with two pathnames. Thus increases the link count of that file.
22. A symbolic link is a file. Unlike the hard link, a new inode is created for each symbolic link. It stores the pathname (shortcut). If the pathname is less than 60 characters, it is stored in the 12 direct block and the 3 indirect block pointers.
23. A big-endian system stores the most significant byte of a word at the smallest memory address and the least significant byte at the largest.

1.3 我觉得比较重要的 points

1. 操作系统结构（ch2 page 45 开始）
2. 杂记（ch2 page 65 开始）
3. `fork()`继承了什么（ch3 page 52, 53）
4. 线程的参数传递（ch4 page 39）
5. 多线程的进程，调用 `fork()` 和 `exec()`（ch4 page 55）
6. UNIX 管道：`fd[0]` 读端口 `fd[1]` 写端口
7. 进程间通信比较（ch5 page 44）
8. 进程同步涉及到问题汇总（ch5 part2 page 24）
9. 进程同步通信，race condition 及其解决（ch5 part2 page 31, 32）
10. Entry & Exit Section of Critical Section（ch5 part2 page 35, 36）
11. 几种 race condition 的解决方案问题汇总（注意只要有 busy waiting 就会有优先权的问题）（ch5 part2 page 55）
12. 信号量的命名：`down(),P,wait()` 以及 `up(),V,signal()`
13. Deadlock Requirements（ch5 part2 page 7, 8）
14. 对于死锁的判断：箭头指向圆圈表示这个实例已经分配给了圆圈里面的进程；箭头指向方框表示这个进程想要索取一个此资源的实例（ch5 part3 page 11）
15. 处理死锁：银行家 (FSM?) 和鸵鸟
16. 进程调度算法的优劣标准（ch6 page 30）注意有一个 throughput 指标：每时间段完成任务的进程数
17. 数据段的内容（ch7 part1 page 14∼24）
18. 关于 segmentation fault，可以看一下 Wikipedia，以及（ch7 part1 page 78 ∼ 85）
19. External Fragmentation & Internal Fragmentation（ch7 part1 page 77）
20. paging：虚拟内存的地址映射（ch7 part2 page 24）
21. 分页一页是 4kB，在 page table 里面，permission 和 Frame# 列，标有 NIL 的表示未分配，valid-invalid 列表示这个虚拟页是否在内存里面（还可以在外存的……）
22. TLB（caching 的思想）（ch7 part2 page 42）

23. 关于 thrashing：为什么 local replacement 可以部分解决问题（可以避免踢出去别的进程的 page）；关于工作集 working-set 方案，当 $\sum WSS_i > m$ 即所有进程的工作集加起来比可用的物理页框要多，就有可能会崩掉
24. metadata 元数据，见附件的 Wikipedia，后面文件系统也会用到
25. SSD 的分级 package > die/chip > plane > block > page
26. 特别注意 SSD 是不能 overwrite 的，只能 program-erase-reprogram，有擦写寿命
27. 在 RAID 那一块：MTTF（mean time to failure）
28. RMW：利用旧的检验块和新旧修改块确定新的检验块，即读取旧的检验块和修改块；RRW：读取未改动块的数据
29. open file table（ch9 part1 page 31）
30. FS 的 3.0 策略（ch9 part2 page 56 starts）注意 57 页的 block 的 definition
31. metadata journaling 两种模式（ch10 part2 page 70）
32. TLB 和大页表同时访问，所以计算所需时间的时候就只会加上其中的一个

第 2 章　Mass Storage

2.1 RAID

1. Purpose :
   (a) In the past, combine small and cheap disks as a ***cost-effective*** alternative to large and expensive disks
   (b) Nowadays
      i. Higher performance
      ii. Higher reliability via redundant data
      iii. Larger storage capacity
2. RAID-0: Block level stripping, no redundancy
3. RAID-1: Data mirroring
4. RAID-01: First stripping, then mirroring
5. RAID-10: On the contrary of above
6. RAID-4: Parity generation: Each parity block is the XOR value of the corresponding data disks $A_p = A_1 \otimes A_2 \otimes A_3$
   (a) RMW (read modify write) $A'_p = A_p \otimes A_1 \otimes A'_1$
   (b) RRW (read reconstruct write) $A'_p = A_3 \otimes A'_2 \otimes A'_1$
   (c) Problem: ***Imbalance*** – Disk bandwidth are not fully utilized
      i. Parity disk will not be accessed under normal mode
      ii. Parity disk may become the bottleneck
7. RAID-5: One parity per stripe
   (a) Key difference: Uniform parity distribution
8. RAID-6: 2 parities

第 3 章　File System

3.1　Programmer's perspective

1. What are stored inside a storage device?
   (a) File
   (b) Directories
   (c) Interfaces/Operations
2. Layout
   (a) what are stored inside the device
   (b) Where the stored things are
   (c) The set of FS operations defines how the OS should work with the FS layout. In other words, OS knows the FS layout and works with that layout.
3. There are ***two basic things*** that are stored inside a storage device, and are common to all existing file systems: File and Directories
4. How does a FS store data into the disk? −That is, the ***layout*** of file systems.

3.2　Why do we need files

1. File provides a long-term information storage.
2. File is also a shared object for processes to access concurrently.
3. A unique pathname lead to the file's attributes and its content, which are usually stored ***separately***

3.2.1　File permissions

1. ***First field:*** File/director
2. ***2nd /3rd /4th fields (3 bits each):*** controls read/write/execute
3. for the file owner/file's group/others (e.g., 111:7,110:6)

3.2.2 Opening a File

1. The process supplies a path name to the OS.
2. The OS looks for the ***file attributes*** of the target file in the disk.
3. The disk returns the file attributes.
4. The OS then associates the attributes to a number and the number is called the ***file descriptor***.
5. The OS returns the file descriptor to the process.
6. Opening a file only involves the ***pathname*** and the ***attributes of the file***, instead of the file content!

3.2.3 Read From Opened Files

1. The process supplies a file descriptor to the OS.
   (a) A file descriptor is just an ***array index*** for each process to locate its ***opened files***.
2. The OS reads the file attributes and uses the stored attributes to ***locate the required data***. (In the OPEN FILE TABLE !)
3. The disk returns the required data. – File data is stored in a fixed size ***cache in the kernel***.
4. The OS fills the ***buffer provided by the process*** with the data. Write data to the userspace buffer.

3.2.4 Read System Call

1. Check whether the end of the file is reached or not. [Comparing ***size*** and ***file seek***.]
   (a) File attributs: Name, Identifier (*Unique tag (a number which identifies the file within the FS)*), Type, Location, Size, Owner, Permission, Access, creation, modification time, etc.
   (b) Runtime Attributes: reading position count, etc
2. Reading data.
3. File data is stored in a fixed size cache in the kernel.
4. Write data to the userspace buffer.

3.2.5 Write System call

1. Write data to the kernel buffer.
2. According to the data length,
   (a) change in file size, if any (File attributes)
   (b) change in the file seek (Runtime attributes)

3. The call returns
4. The buffered data will be flushed to the disk from time to time.

3.3 Directories

1. It's a file
2. Whether it has file attributes is FS-dependent
3. It must have file content

3.3.1 Locate a File Using Pathname

Suppose that the process wants to open the file `/bin/ls`.

1. The process then supplies the OS the unique pathname `/bin/ls`.
2. The OS retrieves the directory file of the root directory `/`.
3. The disk returns the directory file.
4. The OS looks for the name "bin"in the directory file.
5. If found, then the OS retrieves the directory file of `/bin` using the information of the file attributes of `bin`.
6. The OS looks for the name `ls` in the directory file `bin`. If found, then the OS knows that the file `/bin/ls` is found, and it starts the previously-discussed procedures to open the file `/bin/ls`

3.3.2 Creation and Deletion

1. ***File creation == Update of the directory file***
   (a) `touch text.txt` will only create the directory entry, and there is no allocation for the file content.
2. Removing a file is just delete the information in Directory file.
   (a) Note that we are not ready to talk about de-allocation of the file content yet.

3.4 File System Layout

3.4.1 Trial 1.0: The Contiguous Allocation

Just like a book! Yet drawbacks:

1. External Fragmentation (We have enough space, but there is no holes that I can satisfy the request.) Therefore we need to move files to clean up enough space.
2. When a file need to grow, it may also do not have enough space.
3. Used suitable for read-only cases

第 4 章　一些常见的题目

整理整理作业，得到一个 cache，命中率还是可以很高哒

4.1　内存管理

4.1.1　Segmentation Fault

段错误简单来说是由非法访问内存引起的，有三种（其实差不多的）说法：

1. When you are accessing a piece of memory that is not allowed to be accessed, then the OS returns you an error called – segmentation fault.
2. (WikiPedia) In computing, a ***segmentation fault*** (often shortened to ***segfault***) or ***access violation*** is a fault, or failure condition, raised by hardware with memory protection, notifying an operating system (OS) the software has attempted to access a restricted area of memory (a memory access violation).
3. The memory in a process is separated into ***segments***. So, when you visit a segment in an illegal way, then...***segmentation fault***.

下面这张图标有 `YES` 的表示这种访问会引发段错误，标为 `NO` 的不会引发

简单分一下类，可以这么看访问（代码 + 常数段是只读段；已分配的栈、堆、BSS (Block Started by Symbol)、数据段都是 Allocated 部分；栈从高地址向低地址长，堆相反，这两个会在中间遇上?）：

	Segments to Visit		
	Read-only Segments	Allocated Segments	Unused or Unallocated Segments
Reading	No Problem	No Problem	Segmentation Fault
Writing	Segmentation Fault	No Problem	Segmentation Fault

Read	0xffffffff	Write
YES	Unusable	YES
NO	Allocated Stack	NO
YES	Unallocated Zone	YES
NO	Allocated Heap	NO
NO	BSS	NO
NO	Data	NO
NO	Code + Constant	YES
YES	Unusable	YES
	0x00000000	

图 4.1: 非法访问内存触发段错误的条件

4.2 文件系统

4.2.1 FAT 系统的读取流程

大致分为两步，即读取目录文件找到第一块的位置编号，和从 FAT 表链式遍历找到需要读取的块编号。FAT 表里面存储了磁盘的 content 部分，各级文件（包括目录文件）所有文件的信息。其实查找一个 cluster 就是在一个 array 里面找到相应的 hash 关系。有两种找法：一个文件刚开始的位置，是在 root directory 里面，文件中间的是用 FAT 表来查找下一块的位置。

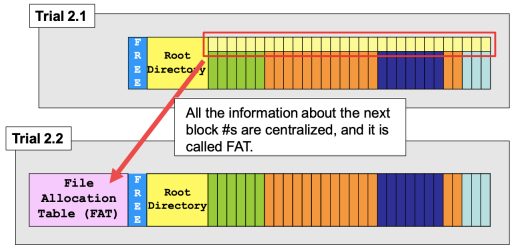

图 4.2: FAT Structure

图 4.3: FAT 表包含目录和多级文件

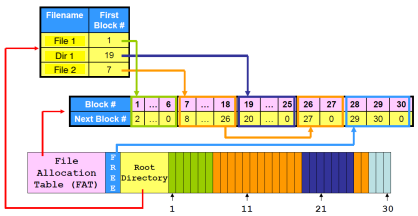

图 4.4: FAT 表包含目录和多级文件

另外一个需要注意的是 FAT 表必须 cache 进 kernel，至少一部分要被 cache 进去，否则依然

解决不了 random access 效率很低的问题

4.2.2 Ext 系统的读取流程

Inode 的构建 回看一下 FAT 系统，效率上的亮点在于 cache 到 kernel 的 FAT，但磁盘容量的增加会导致需要缓存的表变得很大。于是在 Inode 实现方案里面就不再缓存了（但是还是需要保存一个 Open File Table 的）。Ext 一上来就先将 FAT 按照文件切开，每一个文件有一个自己的小 FAT，构成一个结构体数组。这是为了解决 FAT 结构里面缓存不够灵活，导致性能和空间必须有很明显的 Tradeoff 的问题。为什么要用一个整齐的结构体数组呢，因为只有每一个 Node 的大小是一样的，才可以在从目录文件中查找得知目标文件的 Inode 编号的时候，计算出偏移量得到这个文件对应的 Inode 的具体位置。如果是变长 Node，那么就像 Trial 1.0 一样，会面临外部碎片化的问题。而至于为什么这里就可以使用一个统一的结构来强制规范化对齐，而 1.0 就不可以，是因为 Inode 里面存的是元数据，比较小，这些内部碎片化就可以接受了。

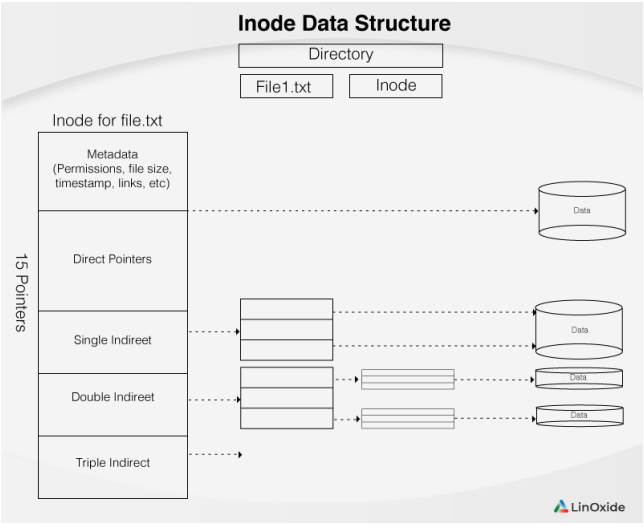

图 4.5: Index Node Structure

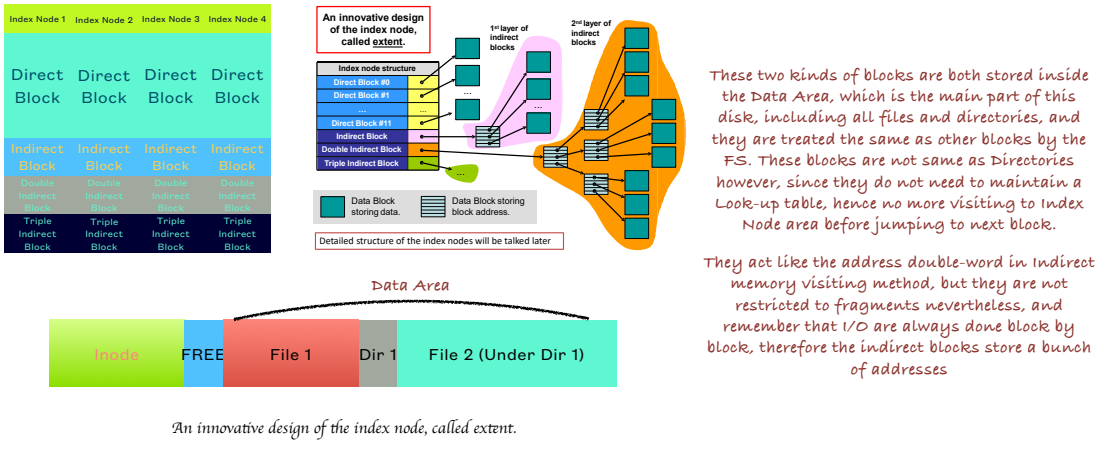

图 4.6: 基于 Inode 的 FS 索引结构