操作系统导论要点总结

公众号： Jacen 的技术笔记

知乎： Jacenhu

欢迎关注我，共同学习和交流。

在公众号 Jacen 的技术笔记，回复操作系统，可获得本文 PDF。

Chapter0 前言

《Operating Systems: Three Easy Pieces》围绕 3 个主题展开： 虚拟 化（virtualization）、并发（concurrency）和持久性（persistence）。

online free ： http://pages.cs.wisc.edu/~remzi/OSTEP/

And now, the free online form of the book, in chapter-by-chapter form (now with chapter numbers!):

Intro	Virtualization		Concurrency	Persistence	Security
Preface	3 *Dialogue*	12 *Dialogue*	25 *Dialogue*	35 *Dialogue*	52 *Dialogue*
TOC	4 Processes	13 Address Spaces code	26 Concurrency and Threads code	36 I/O Devices	53 *Intro Security*
1 *Dialogue*	5 Process API code	14 Memory API	27 Thread API code	37 Hard Disk Drives	54 *Authentication*
2 Introduction code	6 Direct Execution	15 Address Translation	28 Locks code	38 Redundant Disk Arrays (RAID)	55 *Access Control*
	7 CPU Scheduling	16 Segmentation	29 Locked Data Structures	39 Files and Directories	56 *Cryptography*
	8 Multi-level Feedback	17 Free Space Management	30 Condition Variables code	40 File System Implementation	57 *Distributed*
	9 Lottery Scheduling code	18 Introduction to Paging	31 Semaphores code	41 Fast File System (FFS)	
	10 Multi-CPU Scheduling	19 Translation Lookaside Buffers	32 Concurrency Bugs	42 FSCK and Journaling	**Appendices**
	11 *Summary*	20 Advanced Page Tables	33 Event-based Concurrency	43 Log-structured File System (LFS)	*Dialogue*
		21 Swapping: Mechanisms	34 *Summary*	44 Flash-based SSDs	Virtual Machines
		22 Swapping: Policies		45 Data Integrity and Protection	*Dialogue*
		23 Complete VM Systems		46 *Summary*	Monitors
		24 *Summary*		47 *Dialogue*	*Dialogue*
				48 Distributed Systems	Lab Tutorial
				49 Network File System (NFS)	Systems Labs
				50 Andrew File System (AFS)	xv6 Labs
				51 *Summary*	

Chapter2 操作系统介绍

2.1 虚拟化 CPU

虚拟机 CPU：将单个 CPU 转换为看似无限数量的 CPU，从而让许多程序看似同时运行，这就是虚拟化 CPU（virtualizing the CPU)。

2.2 虚拟化内存

程序的每个指令都在内存中，因此每次读取指定都会访问内存。

虚拟化内存：每个进程访问自己的私有虚拟地址空间（virtual address space）(有时称为地址空间，address space)，操作系统以某种方式映射到机器的物理内存上。一个正在运行的程序中的内存引用不会影响其他进行的地址空间。对于正在运行的程序，它完全拥有自己的物理内存。

2.3 并发

操作系统本身和现代多线程 multi-threaded 程序都存在并发问题。

2.4 持久性

在系统内存中，数据容易丢失。如果断电或者系统崩溃，内存中的数据都会丢失，因此需要硬件和软件来持久的 persistently 存储数据。

硬件以输入/输出设备形式出现。

操作系统中管理磁盘的软件称为文件系统。大多数文件系统首先会延迟写操作，将其批量分组为较大的组。为了处理写入时的系统崩溃问题，文件系统还会包含复杂的写入协议，如日志和写时复制。

2.5 设计目标

目标：

（1）抽象，让系统方便和易于使用。

（2）高性能。

（3）提供保护。

（4）可靠性

（5）能源效率、安全性、移动性。

2.6 简单历史

（1）早期：一些库

（2）超越库：保护

（3）多道程序时代

（4）摩登时代

Chapter3 关于虚拟化的对话

假设一个计算机只有一个 CPU，虚拟化要做的就是将这个 CPU 虚拟成多个虚拟 CPU 并分给每一个进程使用，因此，每个应用都以为自己在独占 CPU，但实际上只有一个 CPU。这样操作系统就创造了美丽的假象——它虚拟化了 CPU。

Chapter4 抽象：进程

进程就是运行中的程序（非正式定义），是操作系统的最基本抽象。

操作系统通过虚拟化（ virtualizing）CPU 来 提供这种假象。通过让一个进程只运行一个时间片，然后切换到其他进程，操作系统 提供了存在多个虚拟 CPU 的假象。这就是时分共享（ time sharing）CPU 技术，允许用户运行多个并发进程。

时分共享技术：资源由一个实体使用一小段时间，然后由另一个实体使用一段时间。

空分共享技术：资源在空间上被划分。

机制：低级方法或协议，功能。解决**如何**“how”问题

策略：操作系统内做出某种决定的算法。解决**哪个**“which”问题

4.1 抽象：进程

操作系统为正在运行的程序提供的抽象，就是所谓的进程。

进程的机器状态组成部分：内存；寄存器。

4.2 进程 API

- 创建
- 销毁
- 等待
- 其他控制
- 状态

4.3 进程创建：更多细节

操作系统运行程序做的第一件事是系统从磁盘读取字节，将代码和所有静态数据加载到内存中，加载到进程的地址空间中。

操作系统还需要为程序运行时栈 run-time stack 分配内存，也可能为程序的堆 heap 分配内存。

操作系统还将执行一些其他初始化任务，特别是输入/输出（IO）相关的任务。

最后一项任务：启动程序。

4.4 进程状态

3 种状态：

- 运行：进程正在处理器上运行，正在执行指令。
- 就绪：进程已准备好，但不在此时运行。
- 阻塞：进程执行了某种操作，直到其他事件时才会准备运行。

4.5 数据结构

有关键的数据结构来跟踪各种相关的信息。

如：

进程列表 process list

进程控制块 Process Control Block，PCB

Chapter5 插叙： 进程 API

5.1 fork()系统调用

系统调用 fork()用于创建新进程。

新创建的进程成为子进程，原来的进程称为父进程。

子进程不会从 main() 函数开始执行，而是直接从 fork()系统调用返回。

父进程的获得的返回值是子进程的 PID，而子进程获得的返回值是 0。

CPU 调度程序决定了某个时刻哪个进程被执行。

5.2 wait () 系统调用

wait 系统调用：父进程等待子进程执行完毕。

5.3 exec()系统调用

exec 系统调用让子进程执行与父进程不同的程序。

5.5 其他 API

kill() 向进程发送信号

ps 查看进程

top 展示系统中进程消耗 CPU 或其他资源的情况

Chapter6 机制：受限直接执行

控制权对操作系统非常重要，操作系统负责资源管理。

6.1 基本技巧：受限直接执行

Limited direct execution

6.2 问题 1：受限制的操作

不同的执行模式：

- 用户模式，应用程序不能完全访问硬件资源。
- 内核模式，操作系统可以访问机器的全部资源。

程序可通过执行特殊的陷阱（trap）指令，跳入内核并将特权级别提升到内核模式。进入内核后，系统就可以执行任何需要的特权操作，从而为调用进程执行所需的工作。完成后，操作系统调用一个特殊的从陷阱返回（return from trap）指令，该指令返回到发起调用的用户程序中，同时将特权级别降低，回到用户模式。

6.3 问题 2：在进程之间切换

协作方式：等待系统调用

在协作调度系统中，OS 通过等待系统调用，或某种非法操作发生，从而重新获得 CPU 的控制权。

非协作方式：操作系统进行控制

时钟中断（timerinterrupt）：时钟设备每隔几毫秒产生一次中断。产生中断时，当前正在运行的进程停止，操作系统中预先配置的中断处理程序（interrupthandler）会运行，这时操作系统会重新获得 CPU 的控制权。

保存和恢复上下文

上下文切换：context switch。操作系统要做的就是为当前正在执行的进程保存一些寄存器的值，并为即将执行的进程恢复一些寄存器的值。

6.5 小结

- 受限直接执行
- 重新启动时有用的

- 时钟中断

Chapter7 进程调度：介绍

7.1 工作负载假设

7.2 调度指标

周转时间=任务完成时间−任务到达时间

性能指标和公平是矛盾的

7.3 先进先出（FIFO）

先进先出：First In First Out，FIFO

7.4 最短任务优先（SJF）

最短任务优先：Shortest Job First，SJF

非抢占式

7.5 最短完成时间优先（STCF）

最短完成时间优先：Shortest Time to completion First，STCF。

抢占式

当有新工作进入系统时，调度剩余时间最少的工作。

7.6 新度量指标：响应时间

响应时间：从任务到达系统到首次运行的时间。

响应时间=首次运行−到达时间。

7.7 轮转调度 Round-Robin，RR

在一个时间片内运行一个工作，然后切换到运行队列中的下一个任务，而不是运行一个任务直到结束。它反复执行，直到所有任务完成。

7.8 结合 IO

当交互式作业正在执行 I/O 时，其他 CPU 密集型作业将运行，从而更好地利用处理器。

7.10

- 运行最短的工作，优化周转时间。
- 交替运行所有工作，优化响应时间。

Chapter8 调度：多级反馈队列

我们对进程一无所知，应该如何构建调度程序来实现这些目标?

多级反馈队列是用历史经验预测未来。

8.1 MLFQ:基本规则

MLFQ 中有许多独立的队列（queue），每个队列有不同的优先级（prioritylevel）。任何时刻，一个工作只能存在于一个队列中。MLFQ 总是优先执行较高优先级的工作（即在较高级队列中的工作）。

MLFQ 根据观察到的行为调整它的优先级。

8.2 尝试 1：如何改变优先级

8.3 尝试 2：提升优先级

8.4 尝试 3：更好的计时方式

调度程序应该记录一个进程在某一层中消耗的总时间，而不是在调度时重新计时。

8.5 MLFQ 调优及其他问题

8.6 小结

规则 1：如果 A 的优先级>B 的优先级，运行 A（不运行 B）。

规则 2：如果 A 的优先级=B 的优先级，轮转运行 A 和 B。

规则 3：工作进入系统时，放在最高优先级（最上层队列）。

规则 4：一旦工作用完了其在某一层中的时间配额（无论中间主动放弃了多少次 CPU），就降低其优先级（移入低一级队列）。

规则 5：经过一段时间 S，就将系统中所有工作重新加入最高优先级队列。

Chapter9 调度：比例份额

比例份额算法基于一个简单的想法：调度程序的最终目标，是确保每个工作获得一定比例的 CPU 时间，而不是优化周转时间和响应时间。

9.1 基本概念：彩票数表示份额

彩票表示一个进程占有的 CPU 份额

ticket，利用随机性

9.2 彩票机制

- ticket currency 彩票货币
- ticket transfer 彩票转让
- ticket inflation 彩票通胀

9.3 实现

9.4 一个例子

9.5 如何分配彩票

9.6 为什么不是确定的

步长调度：确定性的公平分配算法。

彩票调度算法只能一段时间后，在概率上实现比例，而步长调度算法可以在每个调度周期后做到完全正确。

9.7 小结

彩票调度、步长调度。

未被广泛应用，只有在知道比例的场景中使用。

Chapter10 多处理器调度（高级）

多核处理器（ multicore） 将多个 CPU 核组装在一块芯片上。

10.1 背景：多处理器架构

单 CPU 系统中存在多级的硬件缓存。

局部性：时间局部性、空间局部性。

缓存一致性问题，解决方法：总线窥探。

10.2 同步问题

跨 CPU 访问共享数据或数据结构时，需要使用互斥原语（如锁）。

10.3 缓存亲和问题

尽可能让那个进程保持在同一个 CPU

10.4 单队列调度

单队列多处理器调度：Single Queue Multiprocessor Schduling, SQMS。

10.5 多队列调度

多队列多处理器调度：Multi-Queue Multiprocessor Schduling, SQMS。

迁移 migration 实现负责均衡。

工作窃取 work stealing

10.6 Linux 多处理器调度

10.7 小结

SQMS 容易构建、负载均衡较好，但在扩展性和缓存亲和度方面不足。

MQMS 有较好的扩展性和缓存亲和度，但负责均衡较困难。

Chapter11 关于 CPU 虚拟化的总结对话

- 陷阱和陷阱处理程序
- 时间中断以及操作系统和硬件在进程间切换时保存和恢复状态
- 调度

Chapter12 关于内存虚拟化的对话

- 基址/界限
- TLB
- 多级页表
- 现代虚拟内存管理程序
- 隔离和保护

Chapter13 抽象：地址空间

13.1 早期系统

13.2 多道程序和时分共享

进程切换的时候，仍然将进程信息放到内存中，操作系统可更有效的实现时分共享。

多个程序驻留在内存中，因此内存保护 protection 非常重要。

13.3 地址空间

- 物理内存抽象-->地址空间 address space
- 地址空间包括：代码段（指令）、栈 stack、堆 heap

13.4 目标

虚拟内存系统的目标：

- 1、透明 transparency

程序不应该感知到内存被虚拟化

- 2、效率 efficiency

时间和空间上的高效

- 3、保护 protection

每个进程在自己的独立环境中运行

13.5 小结

虚拟内存系统负责为程序提供地址空间的假象。

Chapter14 插叙：内存操作 API

14.1 内存类型

两种类型：

- 1、栈内存

申请和释放由编译器隐式管理

- 2、堆内存

申请和释放由程序显式管理

14.2 malloc()调用

```
void *malloc(size_t size);
double *d = (double *)malloc(sizeof(double));
```

size_t 单位为字节

sizeof 为编译时操作符

malloc(strlen(s)+1)

为字符串声明空间，使用上面的用法，因为需要为字符串结束符留出空间。

14.3 free 调用

```
int *x = malloc(10 * sizeof(int));


free(x);
```

调用 free 时，大小不用传入，这个必须由内存分配库本身追踪。

14.4 常见错误

- 1、忘记分配内存
- 2、分配内存不足
- 3、申请内存后，忘记初始化
- 4、忘记释放内存 memory leak
- 5、在用完前释放了内存 –>悬挂指针 dangling pointer
- 6、反复释放内存 double free
- 7、错误调用 free

工具：purity、valgrind

14.5 底层操作系统支持

malloc 和 free 是库调用，不是系统调用。

系统调用：brk、sbrk、mmap

14.6 其他调动

- calloc 分配并置零
- realloc

14.7 小结

本章介绍内存分配的 API

Chapter15 机制：地址转换

地址转换：硬件对每次内存访问进行处理，将指令中的虚拟地址转换为实际的物理地址。

15.1 假设

15.2 一个例子

反汇编工具：objdump(Linux)、otool（Max）

```
movl 0x0(%ebx), %eax;
addl $0x03, %eax;
movl %eax, 0x0(%ebx)
```

15.3 动态重定位

基址加界限机制（base and bound），也称为动态重定位（dynamic relocation）。

CPU 需要 2 个硬件寄存器：

基址寄存器、界限寄存器。

pyhsical address = virtual adress + base

界限寄存器提供了访问保护。

15.4 硬件支持：总结

两种 CPU 模式：

- 内核模式，kernel mode
- 用户模式，user mode

15.5 操作系统的问题

- 内存管理
- 基址/界限管理
- 异常处理

15.6 小结

地址转换确保访问在地址空间的界限内。

基址加界限的动态重定位。

内部碎片问题--》引入分段 segmentation。

Chapter16 分段

简单的通过基址寄存器和界限寄存器实现的虚拟内存存在浪费。

16.1 分段：泛化的基址/界限

MMU 引入多基址和界限寄存器对。

典型有 3 个逻辑不通的段：

代码、栈和堆。

将不同的段放到不同的物理内存区域。

16.2 我们引用哪个段

用开头几位标识段类型。

16.3 栈怎么办

栈的地址空间反向增长。--》栈的增长方向?

16.4 支持共享

保护位，protection bit

16.5 细粒度与粗粒度的分段

段表、segment table

16.6 操作系统支持

- 1、上下文切换时候，操作系统需要对每个段寄存器中的内容进行保存和回复。
- 2、管理物理内存的空闲空间。

  外部碎片的解决方案：一种方法是紧凑物理内存，compact；另外一种方法是空闲列表管理算法（如 best fit、worst fit、first fit 等）。

16.7 小结

分段可更好的支持稀疏地址空间。

Chapter17 空闲空间管理

17.1 假设

在堆上管理空闲空间的数据结构通常称为空闲列表（free list）。

内部碎片。

17.2 底层机制

分割与合并

追踪已分配空间的大小

为了对于给定的指针确定要释放空间的大小，大多数分配程序会在 header 中保存额外信息。

```
typedef struct header_t
{
        int size;
        int magix;
}header_t;
```

实际释放的是头块大小加上分配给用户的空间的大小。

嵌入空闲列表

碎片化解决方案：遍历列表，合并相邻块。

让堆增长

耗尽时，向操作系统申请更大的空间。

17.3 基本策略

- 最优匹配 ：best fit

  遍历并返回候选中的最小块。

  缺点：性能代价

- 最差匹配：worst fit

  找到最大的空闲块，分割使用。

- 首次匹配： first fit

  找到第一个足够大的块

- 下次匹配：next fit

  多维护一个指针，指向上一次查找结束的位置

17.4 其他方式

- 分离空闲列表

  Slab allocator

- 伙伴

17.5 小结

Chapter18 分页：介绍

分页：将一个进程的地址空间分割成固定大小的单元，每个单元称为一页。

页帧：将物理内存看成是定长槽块的阵列。

18.1 一个简单例子

页表：page table，记录地址空间的每个虚拟页在物理内存中的位置，每个进程都有一个这样的数据结构。主要作用是尾地址空间的每个虚拟页面保存地址转换。

两个组件：虚拟页面号 （virtual page number, VPN）和页内的偏移量（offset）

物理帧号：PFN，physical page number

18.2 页表存在哪里

- 存在内存中
- 交换到磁盘上

18.3 列表中究竟有什么

线性列表： linear page table，通过 VPN 查找页表项 PTE，找到 PFN。

18.4 分页：也很慢

```
VPN = (VirtualAddress & VPN_MASK) >> SHIFT;
PTEAddr = PageTableBseRegister + (VPN * sizeof(PTE)) ;
offset = VirtualAddress & OFFSET_MASK;
PhysAddr = (PFN << SHIFT) | offset;
```

内存引用开销很大！

18.5 内存追踪

18.6 小结

分页较灵活，支持稀疏地址空间。但是存在较慢和内存访问的问题。

Chapter19 分页：快速地址转换（TLB）

硬件支持：地址转换旁路缓冲存储器。内存访问时，硬件检查 TLB，若命中则完成转换。

19.1 TLB 的基本算法

见图 19.1 TLB 控制流算法

https://pages.cs.wisc.edu/~remzi/OSTEP/vm-tlbs.pdf

19.2 示例：访问数组

典型页的大小一般为 4KB。

TLB 的成功依赖空间和时间局部性。

19.3 谁来处理 TLB 未命中

硬件方式

软件方式

19.4 TLB 的内容

VPN | PFN | 其他位

19.5 上下文切换时对 TLB 的处理

增加地址空间标识符 （Address Space Identifier，ASID）来区分不同进程的地址映射。

操作系统切换上下文时，需将某个特权寄存器设置为当前进程的 ASID。

19.6 TLB 替换策略

- 一种是 LRU， 替换最近最少使用的。
- 一种是随机策略，随机选择一项换出去。

19.7 实际系统的 TLB 表项

19.8 小结

- TLB
- 超过 TLB 覆盖范围 TLB coverage
- 支持更大的页

Chapter20 分页：较小的表

20.1 简单的解决方案：更大的项

问题：大内存页导致每页的浪费，内部碎片。

20.2 混合方法：分页和分段

```
SN = (VirtualAddress & SEG_MASK) >> SN_SHIFT;
VPN = (VirtualAddress & VPN_MASK) >> VPN_SHIFT;
AddressOfPTE = Base[SN] + (VPN * sizeof(PTE));
```

20.3 多级页表

多级页表：让线性页表的一部分消失，并用页目录来记录页表的哪些页被分配，支持稀疏的地址空间。

页目录项：Page Directory Entries，PDE

PDE = 有效位 valid bit + 页帧号 page frame number

多级页表存在成本，未命中 TLB 时，需要从内存加载两次。

20.4 反向页表

inverted page table，保留一个页表，每项代表系统的每个物理页，并且表明哪个进程正在使用该页，哪个进程的哪个虚拟页映射到该物理页。

20.5 将页表交换的磁盘

将一些页表放入内核虚拟内存 kernel virtual memory，在系统内存压力大时，将页表中的一部分交换到磁盘。

20.6 小结

时间与空间的折中！

Chapter21 超越物理内存：机制

21.1 交换空间

交换空间：swap space，在硬盘上开辟一部分空间用于物理页的移入和移出。

21.2 存在位

存在位：present bit

设置为 1，表示该页存在于物理内存中。

设置为 0，表示不在内存中，而在硬盘上。

21.3 页错误

页错误：page fault，访问不在物理内存中的页。实际上，应该称为页未命中（page miss）。

处理页错误的时候，操作系统需要将页交换到内存中，操作系统可以用 PTE 中的某些位来存储硬盘地址。

21.4 内存满了怎么办

页交换策略：page-replacement policy，选择哪些页被交换出或被替换的过程。

21.5 页错误处理流程

21.6 交换何时真正发生

大多数操作系统会设置高低水位线，High Watermark 和 Low Watermark。

操作系统还可通过执行多个交换过程，进行优化。

21.7 小结

- 存在位，present bit
- 页错误，page-fault
- 不在内存中的页从硬盘取回

Chapter22 超越物理内存：策略

22.1 缓存管理

目标：cache miss 最少，cache hit 最多。

22.2 最优替换策略

替换内存中最远将来才会被访问到的页。

22.3 简单策略：FIFO

先入先出 FIFO 无法确定页的重要性。

22.4 另一简单策略：随机

内存满的时候随机选择一个页进行替换。

22.5 利用历史数据：LRU

通过历史的访问情况作为参考。

LRU：Least-Recently-Used。

22.6 工作负载示例

22.7 实现基于历史信息的算法

22.8 近似 LRU

使用位：use bit，也称为引用位，referenced bit。

时钟指针算法：clock hand。周期性清除使用位，通过区分使用位是 1 和 0 来判定该替换哪个页。

22.9 考虑脏页

修改位：modified bit，又名脏位，dirty bit。

22.10 其他虚拟内存策略

页选择策略：操作系统决定何时将页载入内存。如预取。

页写入磁盘，聚集写入。

22.11 抖动

抖动：内存被超额使用时，系统不断的进行换页、

22.12 小结

页替换策略。

Chapter23 VAX/VMS 虚拟内存系统

23.1 背景

23.2 内存管理硬件

23.3 一个真实的地址空间

23.4 页替换

23.5 其他漂亮的虚拟内存技巧

按需置零、写入时复制。

23.6 小结

Chapter24 内存虚拟化总结对话

- 1、程序中观察到的所有地址都是虚拟地址。
- 2、TLB，为系统提供地址转换的小硬件缓存
- 3、页表
- 4、多级表
- 5、交换到磁盘