OS 秦嘉俊 3210106182

OS: resource abstractor / allocator

OS Structures

User OS interface { CLI: shell / GUI (desktop)

system call 一般通过API访问 (e.g. win32 / POSIX / Java API)

table: 下标是系统调用号, 值为对应函数指针.

类型: process control; file management; device manage; information maintenance; communications; protection

Linker: 可重定位obj(s) ⇒ 1 binary executable file

Loader: 加载…开始执行; ELF: .text / .rodata / .data / .bss

（.data: 初始化 static const）

动态链接: 有 .interp段, entry point 指 loader（静态指 _start）

microkernel 微内核: 易扩展, 易移植, 更可靠, 更安全; 但性能差 e.g. MacOS

把fs, device… 放user space

↕

monolithic 宏内核

Process

Process: a unit of resource allocation / protection - dynamic (running)

code + data section + PC + reg + stack / heap

process control block (PCB) (context), 一个进程有且仅有一个.

进程状态, PC, CPU regs, 调度信息, mm, 统计信息, IO. task_struct

fork(): {父: 返回子进程pid; 子: 返回 0} Question

execv(): 加载新的, 扔掉之前, 一般不返回

wait(), waitpid(): 等child终止, 并回收

zombie (PCB占用资源, 无法由自己回收) 信号

orphan 子进程还在运行时, 父进程终止 ⇒ 被pid 1 收养 (会wait)

Inter-Process Communications (IPC)

① message-passing: 易实现, 但开销大, 要内核空间支持

② shared memory: 对user方便, 开销小, 有安全?

direct / indirect, synchronous / asyn…, automatic / explicit buffer

③ signal ④ pipe {ordinary 无名, fork继承; named}

parent fd0 / fd1 → child fd0 read end / fd1 write end 单向!

⑤ Client-Server: socket, RPC, Java RMI

分时系统: 快速响应用户

同时性, 多路性, 独立性, 及时性

Thread

Thread: a basic unit of execution within a process.

有自己的 thread id, pc, regs, stack; 共享 code / data / heap / signals / open files,

优势: economy (创建, 切换上下文 cheap); resource sharing (no IPC, 并发); responsiveness (等待事件线程); scalability (多核机器)

缺点: 隔离弱

挑战: 数据依赖, 同步; 切分activities; 线程间负载均衡; split data; test / debug

user, kernel thread:

① many-to-one: 内核无法发挥多核优势; 一旦一个线程block, 其他也会

② one-to-one: ✓ 简单, 没有①的?, 没有①快.

③ many-to-many: 折中, m-n, 需要时再分配; 但太复杂

④ two-level: 可选② or ③

Thread Library: C/C++: pthread, openmp; Java Thread → user

Linux thread: LWP (Light-weight process), clone() 创建进程/线程

不区分PCB, TCB, task_struct

① 一个线程, pid = thread ID ② 多个线程, set clone_vm → pid = leading thread ID

not share (mm_struct share), tgid pid

Scheduling

Scheduling: OS决定给哪个ready进程运行, how long;

mechanism: dispatcher (上下文切换)

I/O-bound: 等IO; CPU-bound

① non-preemptive (cooperative): 进程一直占CPU直到自己放弃

② preemptive: CPU决定每个进程跑多久, 可以强制中止

调度目标: max cpu utilization; max throughput (单位时间多少进程结束); min turnaround time (一个进程创→结); min waiting time (ready的时间); min response time → 创 → 第一次响应

ready queue: 所有ready状态的进程

wait queue: 等待这个事件的进程. (一个事件一个queue)

Algorithm (Policy):

① FCFS (First-Come, First-Serve) Convey effect: 快车可能在慢车后 ⇒ yield()

② SJF (Shortest-Job-First): 最小平均等待时间

非抢占 vs 抢占式 (SRTF, Remaining Time) 但无法知道剩余时间

$T_{n+1} = \alpha t_n + (1-\alpha) T_n$

③ Round-Robin: 抢占式, time-sharing, 每个进程最多一个固定时间片, 用完回到队末排队

与SJF相比: wait time↑, response time↓ 且 no starvation. (bounded)

quantum大: overhead低, 但↓response; 小则相反

④ priority: 数小也可以表示高优先级, 也会有starvation, 可以 aging

⑤ multilevel queue: 一个队列用一种调度, 队列间也需调

⑥ multilevel feedback queue: 进程可在队列间迁移 e.g. Q0 RR(8), Q1 RR(16), Q2 (FCFS) 优先级高 时间片用完 ↓ 低

可以做的很通用

Thread scheduling ① PCS: 每个进程分到时间片, 再内部对线程进行调度 ② SCS: 所有线程一起调度

multi-processor: ① 所有thread在同一个ready queue中 ② 每个处理器有自己私有的线程队列

负载均衡: ① push migration: core工作多, 推给其他core ② pull ~: core工作少, 从其他拉任务

SMP: Symmetric multi-processor: 每个处理器 self-schedule

I/O bound 进程 often interactive, 优先级高 (快响应)

Synchronization

race condition: 多个进/线程访问, 操作了同一个数据, 且结果依赖于访问特定的次序.

critical section (如修改公用数据), 只能有一个进程处于这里; 其他进程要在 entry section 询问是否能进入; critical 运行结束后进入 exit section, 释放许可.

```
while(1){
  entry
  critical
  exit
  reminder
}
```

solution to CS:

① Mutual Exclusion 互斥访问, 至多一个进程在CS.

② Progress 空闲让进: 没有线程执行CS时, 应从申请进入的线程中选一个允许执行.

③ Bounded waiting: 一个进程申请进入CS后, 必须有限时间内许可进入.

user Peterson's solution: 只适用2个进/线程, 设Load/Store是atomic

```
flag[0] = [1] = FALSE;
P0: do{ flag[0] = TRUE;
        turn = 1;
        while(flag[1] && turn==1);
        critical section
        flag[0] = FALSE;
        reminder section
    }while(true);
P1: do{ flag[1] = TRUE;
        turn = 0;
        while(flag[0] && turn==0);
        CS
        flag[1] = FALSE;
        RS
    }while(true);
```

但是现代处理器会重排指令. ×

硬件支持: ① Test-and-Set ② Compare-and-Swap 可能 busy wait

atomic variable

Mutex: acquire() 获得锁; release() 释放锁 (spinlock) 同时进CS 原子

用法: while(1){ acquire lock; CS; release lock; RS }

acquire可用cmpswap实现 while( ); 要busy waiting

yield() 就不用busy wait, 但要上下文切换

semaphore: wait()/P(); signal()/V()

初值表示允许多少人

```
wait(S){
  while(S<=0);
  S--;
}
signal(S){
  S++;
}
```

counting ~; binary ~: S只能为0/1 (互斥锁)

用waiting queue 来去掉busy wait: 每个semaphore有一个waiting queue (链表) 挂起进程

① block: 把进程放到对应queue ② wakeup: 从wait queue中取出一个放到ready queue 唤醒进程

实现: wait(S){ S->val--; if(S->val < 0){ add to S->list; block(); } } 记录型信号量 (默认)

signal(S){ S->val++; if(S->val <= 0){ remove P from S->list; wakeup(P); } }

用spinlock 保证op原子性

在CS段没有busy wait (在wait, signal仍有)

Mutex vs. Semaphore

① Pros: no blocking; cons: waste CPU on looping 适合短CS

② Pros: no looping; cons: context switch is time-cons… 适合长CS

deadlock: 大家都拿不到锁; starvation: 有进程不…

Priority Inversion: 低优先级拿到了高优先级的锁. sol: inheritance 优先级

Bounded-Buffer Problem 初值 代表多少个资源？

producer:
do{ //produce
wait(empty-slots);] 不能交换
wait(mutex);
add into buffer
signal(mutex);
signal(full-slots);] 可以交换
}while(TRUE);

consumer: 表明目前buf中个数，初值为0
do{
wait(full-slots);
wait(mutex); 控制对buf访问
//remove an item from buf
可以交换[signal(mutex); 初值为N
signal(empty-slots);
}while(TRUE)

mutex初始为1，full-slots：0（buf里已有元素）
empty-slots：N

Readers-Writers Problem

writer:
do{
wait(write);
// write shared data
signal(write);
}

reader: do{
wait(mutex);
readcount++;
if(readcount==1) wait(write);
signal(mutex);
...
wait(mutex);
readcount--;
if(readcount==0) signal(write);
signal(mutex);}

mutex初始为1，write初始为1.
（用来保护count，count初始为0）
这是Reader-First实现；还有Writer-First.

Dining-philosophers

sol：①只允许同时拿两只筷子 ②奇数号先拿左，偶数号先拿右

DeadLock

4 conditions ① Mutual exclusion ② Hold and wait
③ No preemption ④ Circular wait

Resource graph：Pi表示进程，Ri表示资源 edge ①request Pi→Rj ②assignment Ri→Pj
有环不一定死锁，但死锁一定有环。

prevention：① sharable ✓，non-sharable ✗ ②一次性申请所有资源，申请时不能有其他资源 ⇒ 利用率低，会有饥饿 ③ 让所有资源释放，再分配（抢） ④ 给资源一个优先级排序，只能从高到低取（有时不适用） 分别破坏四个条件

avoidance 要额外信息
保证不会处于circular-wait状态（资源分配状态：available，allocated，maximum demand resources）
safe state：<$P_1, P_2 \cdots P_n$> 对P_i，P_i的request可以由空闲+之前释放的资源满足.
safe state保证没有死锁，unsafe不一定死锁

single-instance：resource-allocation alg 每个资源只有1份
$P_i \dashrightarrow R_j$（虚线）表示想要（事先声明）claim/request/assignment edge
①. P_i request R_j时 claim→request ② R_j分配给P_i request→assign
③. R_i被P_i释放 assign→claim. no cycle→safe
request只能在（②之后不形成环）的时候才分配 要反向

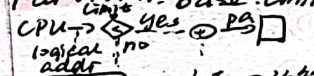
multi-instance：Banker's alg
available：还没被分配的空闲资源 max：P_i需要的资源 R_j
allocation：已经分配给P_i的资源 need：P_i还需要多的一份
选一个need < available的进程分配，运行后释放

Deadlock Detection

single-instance：wait-for graph. 只有$P_1 \cdots P_n$. $P_i \to P_j$ if P_i wait P_j
$O(n^2)$ 有cycle ⇒ deadlock
多实例：banker.如果找不到任何安全序列 ⇒ not safe / deadlock
recovery：① 终止死锁进程（all / 一个个放弃直到消除死锁）
② 资源抢占：选victim，回滚（要注意starvation，不要总抢同一个P_i）

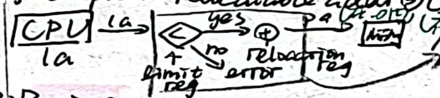
Main Memory

Partition：base.limit registers □ base+limit / ≤base
要求：① protection ② fast execution ③ fast context switch



Fixed：把内存切为等长pieces → Internal Fragmentation
Variable：基于进程需求，动态分配 → External（FF，BF，WF）
└可以通过compaction

Segmentation
logic addr：<segment-number, offset> array
segment table内是 limit，base，权限（number做索引）STLR STBR
Address Binding ①源代码：symbolic 依然有外部碎片
②compiler：relocatable addr ③ Linker：absolute addr



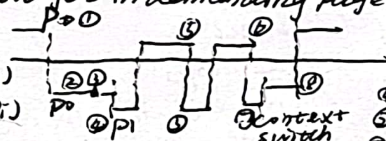
MMU memory management unit 地址翻译+保护

Paging：
①物理内存：帧frame（e.g. 4k） ②逻辑内存：页page 无外部碎片，有内部碎片（最后一个frame）
page table [p | d] ← 用来索引页表. 页表放在内存里.
PTBR（base）指向页表，PTLR标明页表大小 ⇒ 每次指令要两次内存访问
TLB（translation Look-aside Buffer）：全相联，既存page number，又存frame number.（64~1024 entries）
上下文切换时要切页表，flush TLB.
EAT（Effective Access Time） hit ratio × hit time + miss ratio × miss time（TLB+MEM / TLB+MEM×2）
内存保护：valid bit 页在logic addr中，可以访问？
page sharing：可以把重入代码（reentrant）共享，只读，可映射到不同P_i
页表结构：页表要求物理内存连续
2-level：[p1 | p2 | d] 10 10 12
hashed page table：[p | d] → hash表 → 存page＋frame number 指向下一个
Inverted pgtbl：索引是pa，整个sys只有一张表 每次遍历页表，找到entry（存pid与va）. ↓memory ↑time
Swapping：暂时将内存中frame存到backing store（disk）上，需要再次执行时再调回（la一样，pa不同）

Virtual Memory：可以只把程序一部分放入

Demanding paging：brings a page into memory only when it is demanded.
page fault：由MMU发出，OS处理，在user访问无效地址时触发
task_struct → mm_struct → vma-area-struct { vm_start, vm_end }
① 地址超过vma范围，or 权限不对 ⇒ OS kill P_i
② else分配空闲帧，映射页（pgtbl），restart inst. free-frame list（disk）
不在内存中

Lazy swapper：never swaps a page into memory unless it will be needed
pre-paging：预先分配一些页，↓page fault. if unused, waste↑
stages in demanding pages（worse case）



① MMU：page fault
② pgtbl interrupt handler
③ read disk optimized
④ P0 sleep，context switch
⑤ interrupt：disk I/O
⑤ handler：mark P0 ready
⑥ timer interrupt
⑧ fix P0 pgtbl. return to user.

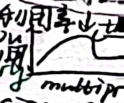
EAT：(1-p) × mem + p × (page fault + swap page out + swap page in + inst restart)
p：page fault rate；swap page out：optional

COW（copy-on-write）：fork后父子进程共享，Pi要修改page时，再分配页 ⇒ effecient process creation

Page Replacement：vm可以大于pm
① FIFO：更多的frame空间可能带来更多的page fault，Belady's Anormaly
队列模拟，hit不改变位置
② optimal：理想alg 用来比较其他alg效果
③ LRU：no Belady's Anormaly
counter-based 存一个时间戳，每次访问更新
stack-based：一页被访问时，移到栈顶
开销大，有LRU Approximation：NRU（Not Recently Used）
① reference bit：最开始设为0，页被访问了就设为1.
替换时选ref=0的.（not know order）
② Additional-Reference-Bits：一个页用一个字节（8 bits），一个时间周期后，右移1bit，高位=1（if used）. 值最小的就是最久未使用页面.
③ Second-chance：FIFO with ref bit（clock）counter
从头开始，ref=0 ⇒ replace；ref=1，set to 0，move to next page
④ Enhanced Second-Chance
使用引用位，修改位（ref，modify bit）
淘汰次序（0,0）⇒（0,1）⇒（1,0）⇒（1,1）
④ Counting-based Page Replacement 记录每个页ref次数
LFU（Least Frequently Used）replace最小counter
MFU replace 最大counter.
Page Buffering Algorithm：keep a pool of free frames
通过被置换页的缓冲，有机会找回被换出的 a list of modified frames

Allocation of Frames
equal / proportional allocation
global / local replacement 可以抢其他P_i的
①+ⓑ ②+ⓐ ②+ⓑ 达到一定... 一直...

Thrashing：进程一直在换进换出页. CPU利用率



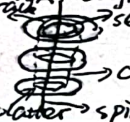
sol：① local page replacement
② work-set Model ∵ Σ locality > memory
Δ = 最近Δ个page ref，WS页集合. WSS_i size of P
$D = \Sigma WSS_i$；若 D > 可用帧数 m ⇒ 挂起某些进程
找work set：interval timer + ref bit
③ page fault frequency：过一个unit会time
设一个上.下界. 若 > upper：分配更多帧
也可以撤销部分P_i < lower：释放部分帧

DS 秦嘉俊 3210106182

page size：碎片⇒小；页表⇒大；resolution⇒小；I/O开销⇒大；缺页异常次数⇒大；locality⇒小；
TLB size⇒大　　实际：grow over time.
TLB reach = TLB size × page size　　要减少碎片
I/O interlock：锁住页，不被换出去。　　pgtbl只针对user space
allocate kernel memory：内核中要为不同ds分配内存，
有些kernel memory需要physically contiguous. e.g. device I/O
Linux Buddy System：每次分配2^n大小内存，
对物理连续内存平分成两段直到合适的大小(恰好>request size)
且次小且可以迅速组装成大内存，释放后合并；但会有内部碎片.
Slab Allocation：a cache of objects
最初是若干空闲对象，需要分配时则从中取并mark.
已经存满used了，从物理连续页上分配新slab. 无碎片.
可以快速满足request

mass Storage Structure

Disk：logic blocks(sectors)最小传输单元.



position time(random-access)
access time = seek time + rotational latency + 传输时间 transfer
磁头到目标扇面　转到目标扇区 rpm×60
→min seek time
Disk Scheduling：由firmware(disk controller)负责
① FCFS 先来先服务，公平，无无限等待；但不是最快
② SSTF：shortest-seek-time first：优先处理离当前磁头最近的请求. 会有starvation. 且时间不是最优(与SJF不同)
平均响应时间↓，吞吐↑；但要提前计算出seek time，且alg复杂
③ SCAN(elevator)移到disk的一头(0)，扫到另一头，到达另一头改变方向，继续处理. 高吞吐，响应时间方差低，且平均；如果刚好错过，就需要等待很久
④ C-SCAN(Circular)从一端到另一端，随后立刻回头，返回路上不服务(把cylinder看作circular list) 相比SCAN等待时间更均匀uniform
⑤ LOOK/C-LOOK：不达到头，而是最远的服务请求.
减少了extra delay(不必要的遍历. 到end)
SSTF：common. default. 若I/O少，FCFS/SSTF即可，
heavy I/O时要(C)-LOOK. 若SSD，则FCFS(不用seek)
表现依赖于请求类型和数量，请求依赖于文件分配策略. alg应模块化，以随时更换
→nonvolatile
SSD(Solid-State Disk)比HDD more reliable，与磁盘比寿命短
容量小，速度快、无arm不用转(no seek. rotate) FCFS ✓
无数量erase(先放电)⇒去家
magnetic Tape磁带：容量大但慢，若随机访问要倒带 对应d>顺序访问
一般顺序访问. 用作备份. 使用前要先格式化/分扇区
Disk management：① physical formatting 将磁盘划分为扇区才能读写 ② logical：创建文件系统. 为了提升效率把块集中一起成为cluster 引导(初始化bootstrap) OS

RAID(Redundant Arrays of Independent Disks)
① Data Mirroring ② Data Striping ③ ECC(Error-Code Correcting) - Parity Bits　　文件结合
RAID 0 数据分散存储；RAID 1 主+备份磁盘；RAID 2 bit stripes data bit-level，用Hamming code纠错 RAID 3 纠错码存在一个单独的盘里；RAID 4. 从3中bit→block；RAID 5 检错码分散到不同盘里；RAID 6：P+Q冗余，纠错码

I/O System

hard link not new inode，只是++ref；symbol link会new，但源文件删除后失去意义
I/O系统组成：PC BUS I/O系统；主机I/O系统
· I/O方式：polling/interrupt；寻址方式：direct I/O inst / mem-mapped I/O
busywait，若设备快✓　　要上下文切换，若频繁效率高✗
中断还可用于 ① protection error ② page fault ③ 软中断(syscall)
④ 管理内核的控制流
数据块
DMA：在DMA开始传输时，主机向内存写入DMA命令块. 后CPU做自己的. DMA操作内存总线写/读内存. disk
I/O devices：block I/O(read, write, seek)；character I/O(stream) clock keyboard
memory-mapped files access；network socket　　Linux：IOCTL()
Application IO Interface：设备驱动提供API操纵IO设备 分类方式
设备分为 character-stream or block. sequential or random access.
synchronous or asynchronous；sharable or dedicated
speed of operation(快中慢)；read-write，read only，write only
块设备：包括disk drivers，一般R.W.S. RAW IO or fs访问. 内存映射文件访问✓
字符设备：键盘鼠标串口. get/put. 库函数有缓冲. 编辑 的按行访问
同步I/O可分为 ① blocking I/O：进程挂起直至IO完成. 某些需求仅能以满足 ② nonblocking：立即返回
异步：IO与进程同时进行. 异vs非阻：nonblocking读数据可以少于所要求的，但异步需完整执行.
buffer：解决设备速度不匹配，块大小不匹配；支持copy semantics

File System Interface

可以有extended file attributes
File 文件是存储介质上的连续逻辑空间.
File Attributes：① name ② identifier：unique tag num
③ type ④ location ⑤ size ⑥ protection ⑦ time, date and user identification. 这些信息保存在目录结构中→disk.
File Operations：create, open, read/write, close, delete, truncate, reposition within file(seek)
其他op可以由上面的op组合实现
Open files：① 有个open file table记录打开的文件 ② file pointer 指向上次R/W位置 ③ file-open count 记录文件打开数量，=0时可以关闭，系统可以移除条目 ④ disk location of the file
⑤ access rights
File Structure：① no structure：a stream of bytes/words
② simple record str - 记录文件结构：lines, fixed/variable length
③ complex structure：formatted document, relocatable load file
可以插入适当控制字符，用第一种方法模拟②③. 由OS. 程序决定
Access Methods：① sequential access：in a predetermined order
② Direct access：可跳到任何位置，也称为随机访问
File Type：executable, object, src code, batch, text, word processor, library, print or view, archive, multimedia
识别文件类型：① 文件名一部分(file extension)
② magic number(在文件开头)

disk divide partitions(minidisks) 一个fs可以有多个disks，一个disk可以有多个partitions，一个partition可以有自己fs.
要实现efficiency(快速定位)，Naming(不同用户命名)
目录操作：create/delete/search a file；list a directory，traverse the fs.
① Single-Level Directory：所有文件在同一目录中. naming grouping?
② Two-level：为每个用户独立目录. UFD(User File Directory)
搜索效率高，但无分组能力　　path
③ Tree-Structured 高效，可分组(解决了naming? grouping)
④ Acyclic-Graph：树基础上，允许共享子目录和文件
有dangling问题：删一个文件后其他链接成为dangling
sol：① 符号链接(soft Link) 删除链接不影响源文件，删除源文件会删掉all links ② hard link：ref count，=0时才删除文件. 快捷方式　　目录本身 不能跨fs
⑤ General Graph Directory：允许目录中有环，有garbage collection. 若没有外界目录指向环，回收. 每次加入链接都要检测.
FS Mounting 挂载：fs在访问前必须挂载?
File Sharing：共享文件要有保护. User ID/Group ID 允许某些用户，某些组访问. 分布式sys里可用网络共享.
Protection：file owner/creator应能控制文件可以被谁访问，做什么. type of access：read. write. append. execute；delete；list
给每个文件. 目录维护ACL(Access Control List)
细粒度控制，但构造、存list麻烦
access mode：R. W. Execute；3 classes of users，3×3 = 9 bit chmod. chgrp. owner/group/public
path name：absolute/relative(相对于当前目录pwd)
① touch <filename> 创建 ② rm < > ③ mkdir < >

File System Implementation

application programs
↓
logical fs ①
↓
file-organization module ②
↓
basic fs ③
↓
I/O control ④
↓
devices

① keep metadata，stores directory structure，FCB ~~(name, 所有权, permissions, ref count, 时间戳, 指向其他 pointer to data blocks)~~ 见后 FCB
input：path；output：logic blocks
② logical file blocks → physical translation
也管理free space
input：logic blocks；output：physical blocks
③ buffer. cache. 向合适的设备驱动程序发送一般命令即可对磁盘上物理块读写
④ device drivers + interrupt handler
分层↓复杂度，↓性能
On-disk structures：boot control block(optional)；volume control block；directory；per-file FCB
In-memory structures：mount table；directory cache；global open-file table；per-process open-file table；各种buffer. open()时先从sys wide table里找文件以确认文件是否正在使用：若有则在per-process table里new-entry 指向sys里的对应项. 否则在directory里找到文件，加载FCB到内存，放入sys表 再在per-process中指向

VFS (Virtual FS)
object-oriented，支持多种fs，用相同的syscall接口(API)调用不同fs的函数 (vfs_write 函数指针) f-op
在struct file里，indirect call

Directory Implementation

dir-entry：inode号，目录项长度，文件名长度，文件名
① Linear list ② hash table

Disk Block Allocation
① Contiguous，每个文件占据disk上一组连续的块 (连续也)
顺序访问快，只用维护起始block与长度；会带来碎片；文件增大要重新分配空间。
变种：Extent-Based System 空间不够了，再分配另一块连续空间。文件块头要加上指向下一扩展的指针 (头；块数)
② Linked Allocation：每个文件都是block的链表。只需一个起始地址。不会浪费空间，但不支持random access
block (指针(指)，数据)
③ Index Allocation：每个文件有自己的index block (第i entry 指向第i块 data)
里面存若干指针，指向数据块 支持 random/direct 访问
无外部碎片，索引块本身占用空间
可以2-level index 下文

Free-Space Management 位图；链表；分组；计数
① bit[i]=0 block[i]空闲；=1 占用
位图要额外空间
② 将空闲块用链表连接，并指向第一空闲块指针保存在disk特殊位置，并缓存在内存中。no waste of space 难得
③ 将n个空闲块地址存在第一个空闲块中，最后一块包含另外n个空闲地址 不易得到连续空间
④ 记录第一块addr与紧跟第一块的连续空闲块数量n

Page buffer：将文件数据作为页而不是磁盘块缓冲到内存

Recovery：consistency checking
backup

I/O请求：user program → syscall程序 → 设备驱动 → 中断处理程序 → 设备控制器
可抢占资源：CPU，内存
不可抢占

· user mode → kernel mode ① trap ② exception ③ interrupt
(syscall) (÷0，非法addr)
· Context-switch：save the state of current running process then restore。会影响reg，stack，memory，但不影响global variables
· process is dynamic，has life-cycle
高响应比调度 HRRN，基于FCFS，SJF，优先级 = (wait time + request time) / request time
综合考虑等待，执行时间，不会有starvation

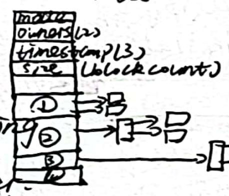
windows 调度

· 页表：每次访存需要两次访存(① 取出页表对应地址 ② 取出地址上的inst/data) 无论是否hit，访问TLB都要时间
可用TLB缓解 实际
· 若page reference string 长度为P，访问了n个不同的page，一个进程(m个frame)
则对于任何替换算法，至少page fault n次，至多P次
multiple-partition ⇒ 导致外部碎片
Page fault后，导致intr的指令会被再次执行。
· use virtual memory，at execution time address binding can be done
· Dynamic relocation relies on a relocation register.
· First Fit 空闲区按地址递增顺序连在一起。
· 若进程访问的页不在主存，且主存无空闲帧时，正确处理顺序：缺页中断 → 决定淘汰页，页面调出，页面调入
· SPOOLing 缓存设备的输出的缓冲，这些设备不能接受交错的数据流 (printers) ① 实现了virtual devices，不用外存 无法满足
· NTFS：windows的 在open的时候创建FCB
read file control information from outer storage into mem
· absolute path 从root directory开始
· FCB包括 file permission；file dates (create/access/write)，file owner，group，ACL；file size；file data blocks or pointers to file data blocks
· File access is protected by user access rights & file attributes
· Files on hard disks are accessed as units of blocks.

三级存储
· tertiary storage，一般用removable media 可移动介质 e.g. floppy disks；光盘 CDROMs；tape；DVD 没有hard disk 软盘
· RAID 0.2.4.5 中 RAID 5 high reliability inexpensively？
· A raw partition (swap space) is fastest.
· Linux treats I/O devices as special files
· Device driver presents a uniform device-access interface to I/O subsystem，much as system calls provide a standard interface between the app and os
· The I/O control of disk devices mainly adopt DMA.
· SSTF：at any moment 可以改变方向
扇区 sector
cylinder

段页式：先segment再paging，地址转换要三次访问主存(段表，页表，实际数据)

· read (syscall) 若文件不在内存，进程进入等待；参数不应包含文件名称 (从user → kernel)
· 一个信号量S，28次P，18次V后，S=0，又对S 3次V，则有0个进程在wait
· batch system 后台 cons：lack of interaction (劣势) 一般FCFS
前台交互式 RR
· Indexed Allocation Example
ext2-node：



① direct block：first 12 pointers
② 1 single indirect
③ 1 double indirect
④ 1 triple indirect
Block size = 512B，pointer = 4
max file sz = 512/4
$12 \times 512 + 128 \times 512 + 128^2 \times 512 + 128^3 \times 512$

· shared printer 正在工作，来了一个新任务，则your job will be spooled for printing in the order it arrives
bool
TestAndSet (boolen *target) { boolen rv = *target; *target = TRUE; return rv; } 原子
使用：bool lock = FALSE;
while (test and set (&lock));
CS
lock = FALSE;
RS
int
② CompareAndSwap (int *val, int expected, int new)
{ int temp = *value; if (*value == expected) *value = new
return temp; }
使用：while (compare_and_swap (&lock, 0, 1)) != 0;
CS； atomic
lock = 0； 但while()不能被中断
RS
· 10GB disk，一个簇 4KB，用位图，存放位图要多少 (1 bit) 10GB/4KB/4
· Access I/O device should be P
· 现代OS (windows/os，Linux，Ios) 不使用Banker (忽略deadlock不管)
· 使用VM，则地址binding发生在execution
complete time：absolute code；Load time：relocatable
· 同步：让权等待 (忙等待法，Test，Swap) 不能进入临界区的执行态进程立即放弃CPU (信号) 可能死锁 可能 Not must