ICS 第 14 次小班课

2022-12-26

第二章 # 整数

- 大小端表示

  IP地址

  单字节对象大小端表示都一样！C标准字符串的存储不受大小端影响！

- 补码数 x 的负 -x：~x + 1
- 补码表示范围的不对称性：

  #define INT_MIN (-INT_MAX-1)

  x 是 int，x > 0 || -x >= 0 永真吗？N　　x < 0 || -x <= 0 永真吗？ Y

  Int 的典型值范围? [-2^31, 2^31 - 1]，或[-2147483648, 2147483647]

- 算术右移 vs 逻辑右移，以及x86-64汇编指令 shr, sar 与其的对应关系。

  对左移来说 shl, sal 功能等价

- 有符号数除以2的幂和算术右移：

  x类型为int, x>>4 与 x/16 等价吗？x类型为 unsigned, x >> 4 与 x/16等价吗？

  取整方向不同；都是向下(或者说向零)取整

- 运算中的隐式类型转换：

  有符号和无符号同时参与运算？尤其注意它们是 < 或 > 的操作数的情况。

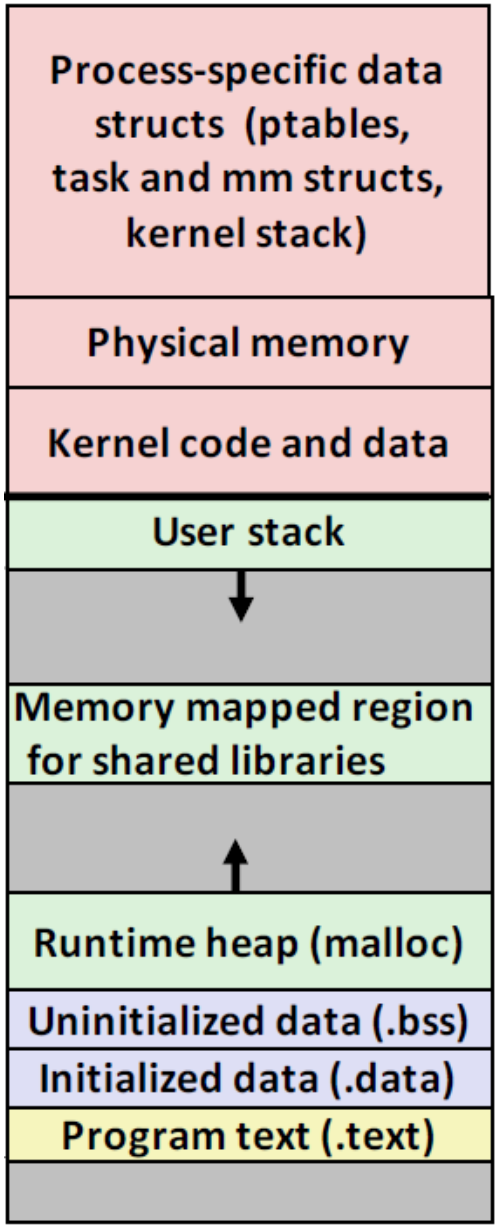
  Buggy: return sizeof(str_a) – sizeof(str_b) > 0; // compare the length of two strings

第二章 # IEEE 754 浮点数

- float：1 + 8 + 23
- double：1 + 11 + 52
  - 指数位的 bias ?
    - float: 127, double: 1023
- 规格化 vs 非规格化
- 浮点运算律
  - 可结合吗？为什么？
  - 可交换吗？为什么？
  - 单调性是指？
- 转到浮点：向偶数舍入
  - 不能被float精确表示的最小正整数是？
    - 2^24+1 或者 16777217
  - double 能精确表示int所能表示的所有数？
    - 正确
- 转到整数：向零舍入
  - int x = -1.7; -> x = -1
- NaN / INF参与运算
  - NaN == NaN -> 0
  - INF – INF ; INF + INF -> NaN ; INF

第三章 x86-64汇编

- 过程栈帧的结构
  整型参数多于 6 个时的压栈顺序？
- 整型参数不多于6个时的寄存器传参惯例？
  %rdi, %rsi, %rdx, %rcx, %r8, %r9 或者其不同长度的寄存器的相应命名
- 数据对齐:
  栈上传递的参数：对齐到8的倍数
  常规：对齐到基本数据类型的倍数
  struct的大小计算问题
- X86-64惯例 之 写入一个寄存器的低4字节会把高4字节置零
- Imm(rb, ri, s)
  movl $0x400000, %eax 和 movl 0x400000, %eax ?
  一个装载立即数，一个从内存中取数
- 条件码的使用
  jle 的跳转条件为？
  (SF ^ OF) | ZF
- 直接跳转 vs. 间接跳转
- CISC vs. RISC



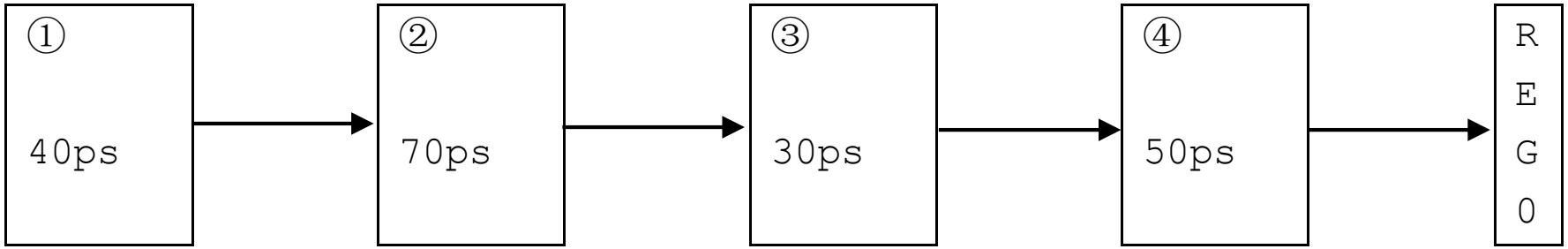
还有一个.rodata没画出来，在.text和.data之间

.rodata 最常见的内容之一：printf和scanf的格式控制串

第四章 Y86-64 简易处理器

- 流水线的通用思想
  - 吞吐量 vs. 延迟
  - 流水线的周期受什么限制？
    - 最慢的那一个流水级的延时
  - CPI, IPC (进程间通信的那个缩写也叫 IPC)
    - CPI: Cycles per Instruction
  - GIPS: Gillion Instructions per second

如下图所示，①~④为四个组合逻辑单元，对应的延迟已在图上标出，REG0为一寄存器，延迟为20ps。通过插入**额外的2个**流水线寄存器REG1、REG2（延迟均为20ps），可以对其进行流水化改造。改造后的流水线的吞吐率最大为_____GIPS，此时每条指令的延迟为_____ps。



吞吐率(GIPS) = 1000÷最慢的一级延迟(ps) = 1000÷100 = 10GIPS
延迟 = 最慢的一级延迟×流水线级数 = 100×3 = 300 ps

第五章 优化技术

- 循环展开技术，循环展开尾部的处理
- 关键路径：未完成一系列运算需要的耗时最长的依赖关系的那条路径。它制约着总体运算完成的性能。

第六章 高速缓存

- 高速缓存基于局部性原理提高系统的性能
  时间局部性 vs. 空间局部性
- 缓存术语：组、行（块）、路、直接映射、k路组相联、全相联

➢ 一个容量为8K的直接映射高速缓存，每一行的容量为32B，那么它有_____组，每组有_____行。

直接映射高速缓存：E=1，每组有1行
每组的大小为 32B × 1 = 32B
组数 = 8K / 32B = 256组

➢ 一个容量为16K的4路组相联告诉缓存，每一行的容量为64B，那么一个16位地址0xCAFE应映射在第_____组内。

每一组容量为64B × 4 = 256B，由于总容量为16K，因此组数为64组
块偏移的长度 = log 64 = 6比特，组索引的长度 = log 64 = 6比特
0xCAFE = (1100 101011 111110) 2 ， 组号是绿色部分，即 43

第六章 # 存储层次结构

- 不同介质的物理属性和访问特点

D 触发器	CMOS
SRAM	六个晶体管 / bit
DRAM	一个晶体管 + 一个小电容 / bit -> SDRAM, DDR-N
SSD	Flash
磁盘	Disks coated with magnetic recording material

  所谓信息密度，是指单位面积能表示的位数的大小
- 随机访问存储器

  支持任意的合法访问序列，且单次延时不受地址的影响

  e.g. 磁盘(机械硬盘) 不是随机访问存储器

  一般来说，地址连续的一串访问的性能都比随机的一串访问要高
- 非易失性
- VMA技术的主要思想
- Intel Core i7的存储层次结构

  L1 L2 L3高速缓存的组织形式和作用
- 存储层次结构

  形成原因和内容

  广义地，为什么可以把虚存看成磁盘的高速缓存？

第七章 链接

- 编译过程

  顺序？ A.编译 B.链接 C.预处理 D.汇编

  CABD

  简述每个阶段最主要的工作？e.g. 预处理: 头文件展开 + 宏展开

  编译：以模块为单位，将高级语言翻译成可读的汇编代码

  汇编：以模块为单位，转译成机器机器识别的机器码，构造可重定位文件

  链接：合并目标模块的节，进行符号解析、重定位

- 符号表中的符号 vs. 程序中的变量

```
int main(){
    static int x;
}
```

```
static int x;
int main(){
    ...// Codes
}
```

- 全局/局部/外部符号、强弱符号

  函数声明默认是 extern 的！如果本模块中已有定义，则为强符号。

第七章 链接

- 区别

```
extern int x;
int func(){
    ...// use x
}
```

```
int x;
int func(){
    ...// use x
}
```

- 下面的（全局区）代码片段会导致链接错误的是哪些？有风险的是哪些？

`int a = 0;`	`int a = 0;` X
`int a = 0;`	`double a;` Risky
`double a = 0;`	`static int a = 0;`

`int a = 0;`	`int a;`
`int a;`	`double a;` Risky
`static int a = 0;`	`static int a = 0;`

- 存档文件：只写一次一定够吗？ N
- .o文件只写一次一定够吗？ Y
- 节、伪节
- 段 != 节：数据段、代码段
- 重定位：注意是相对地址还是绝对地址

第八章 # 进程控制

- 并行 vs. 并发 concurrent vs. parallel
- 进程 vs. 线程
  进程：独立的逻辑流和私有地址空间
  线程：在同一进程内实现多个逻辑流，它们共享进程的地址空间和进程级系统资源
- 创建进程：fork 产生的父子关系、树状结构
- 进程号（getpid、getppid）、进程组号（getpgrp）
  参数是0或者负数的含义
- 子进程结束后立即被父进程回收吗？-> wait
- 子进程结束时没被父进程回收，父进程就结束了，会发生什么？ -> 子进程挂到init下
- 子进程不被及时回收的后果？会占用打开文件等系统资源吗？
  不被及时回收会导致进程ID池、PCB等系统资源被占用，但是不会占用文件资源或者内存！exit退出时已经释放了这些资源！

第八章 # ECF

- 异常的种类

异常的种类	是同步（Sync）的吗？	可能的行为？		
		重复当前指令	执行下条指令	结束进程运行
中断 Interrupt			✓	
陷入 Trap	✓		✓	
故障 Fault	✓	✓		✓
终止 Abort	✓			✓

- 系统调用（[哪一类异常?]） vs. 函数调用
  例如，额外开销：权限、用户/内核态、etc.

第八章 # 信号

- 两个不能被捕获也不能被忽略的信号
  注意 SIGTSTP != SIGSTOP, SIGINT != SIGKILL
- 信号的显式阻塞 vs. 隐式阻塞
  为什么需要隐式阻塞？-> 防止无休止的信号嵌套
- 使用 sigprocmask屏蔽信号、避免race
- 利用sigsuspend等待信号：

```
void sigchld_handler(int s) {
    int olderrno = errno;
    pid = Waitpid(-1, NULL, 0);
    errno = olderrno;
}
```

➢ 下面的等待SIGCHLD的方法正确吗？各有什么问题？

```
while(!pid)
    ;
```

Busy waiting, 浪费CPU计算资源

```
while(!pid)
    pause();
```

Missed signal / race condition

```
while(!pid)
    sleep(1);
```

OK，但是很慢

第八章 # 信号 & 用户层异常

- errno 的使用和维护

  例如，waitpid循环地回收子进程时，假设某次因为没有子进程而出错返回-1。
  退出时应当检查errno是否为ECHILD，确定出错原因。不是则应当报错。

- 非本地跳转

```
int setjmp(jmp_buf j)
void longjmp(jmp_buf j, int i)
```

原理：

1. 从jmp_buf中恢复上下文（一般寄存器、PC、栈指针）
2. 将%eax的值设为 i
3. 将PC的值设为jmp_buf中读出的值

只能跳转到已经调用但未结束的函数。

第十章 I/O与文件操作

- 不同的I/O函数：read/write、scanf/printf
  - printf：可重入？异步信号安全？线程安全？ N N Y
  - 在主程序中未屏蔽信号，且在信号处理函数中也调用了printf可能会发生？
    - 死锁，printf用一个全局锁保护缓冲
  - 例如，waitpid循环地回收子进程时，假设某次因为没有子进程而出错返回-1。
  - 退出时应当检查errno是否为ECHILD，确定出错原因。不是则应当报错。
- 缓冲
  - （默认）stdout带行缓冲、stderr不带缓冲
  - 刷新 stdout 缓冲的条件：
    - 1. 缓冲区满 2. 遇到换行 3. 程序正常退出 4. 手动刷新
- 三级文件表：描述符表、打开文件表、v-node表
  - 哪些表是所有进程共享的？后两张
  - 描述符：默认打开3个（012）
  - 如何增加打开文件的refcnt？
    - open两次同一个文件？N
    - fork？ Y
    - I/O重定向：dup2（注意参数顺序） Y
  - 由内核保证打开文件表中偏移关于读写操作的原子性
- 文件及时关闭：见后文echo服务器

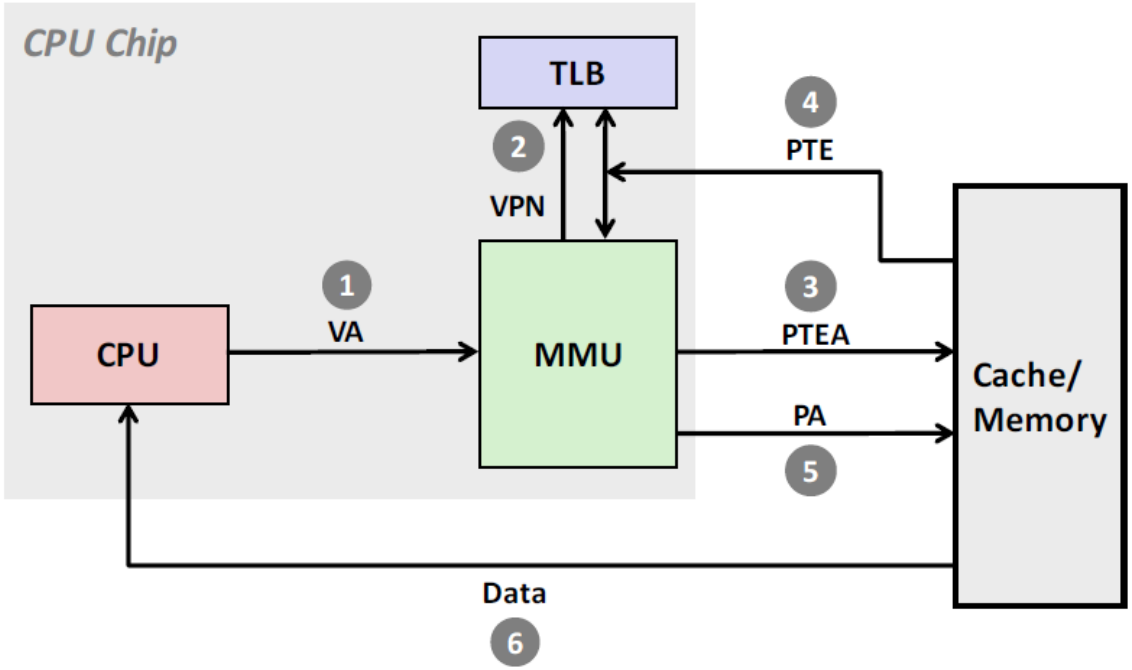
第九章 虚拟内存 # 1/3

- 考虑同时存在TLB和Cache下的命中和不命中的组合，共有四种情况。每种情况下的动作和次序？

  首先，为什么这四种情况都是可能发生的？

  TLB: 缓存PTE，Cache: 缓存指令和数据

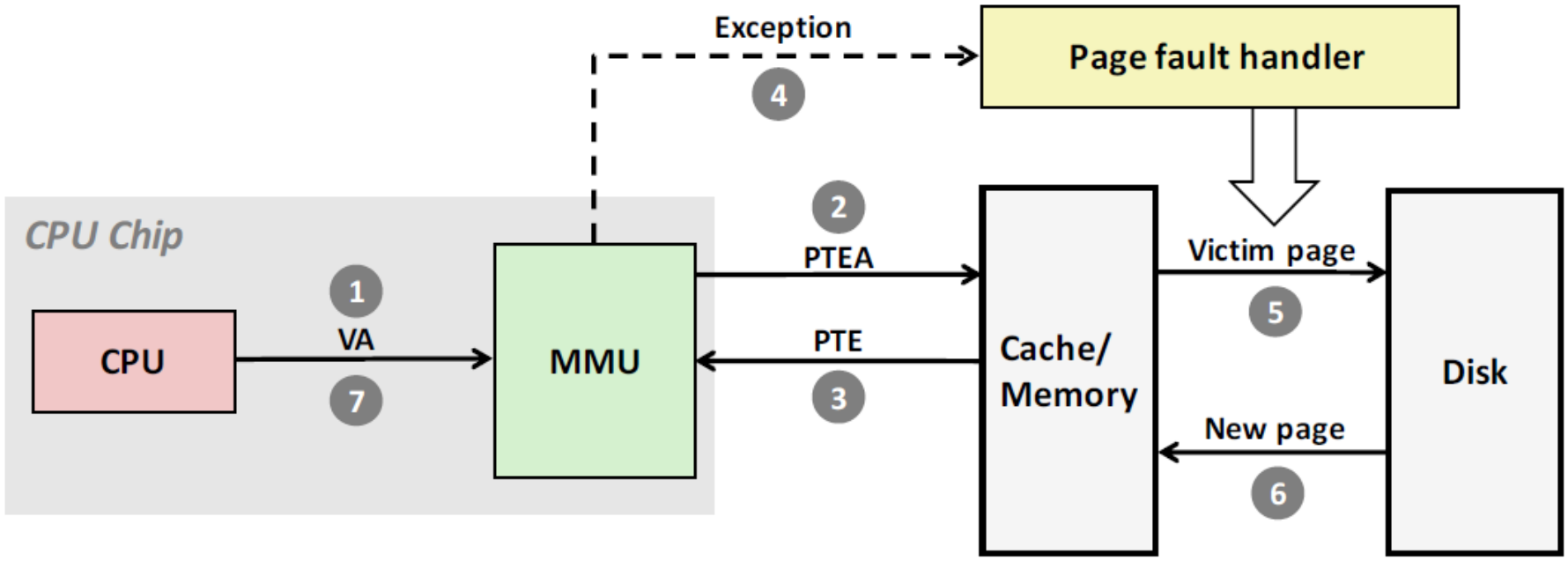
  例如，TLB 和 Cache 都不命中会怎样？



- 如果再考虑页不命中，会发生什么？

  下张 slides 展示了没有 TLB 时一个 PF 发生的场景

第九章 虚拟内存 # 2/3



- TLB命中，一定页命中吗？

  Yes, TLB命中表明PTE有效。有效的PTE(valid bit有效)指向的页一定在内存中。

- TLB命中，一定Cache命中吗？

  No

- 采用四级页表地址翻译，TLB，Cache 和DRAM 的存储系统，访问一个单字节虚拟地址的内容至少访问内存几次？至多几次？

  至少0次，如果PTE被TLB缓存且data被Cache缓存；至多5次，取4次PTE，外加一次数据访存

第九章 虚拟内存 # 3/3

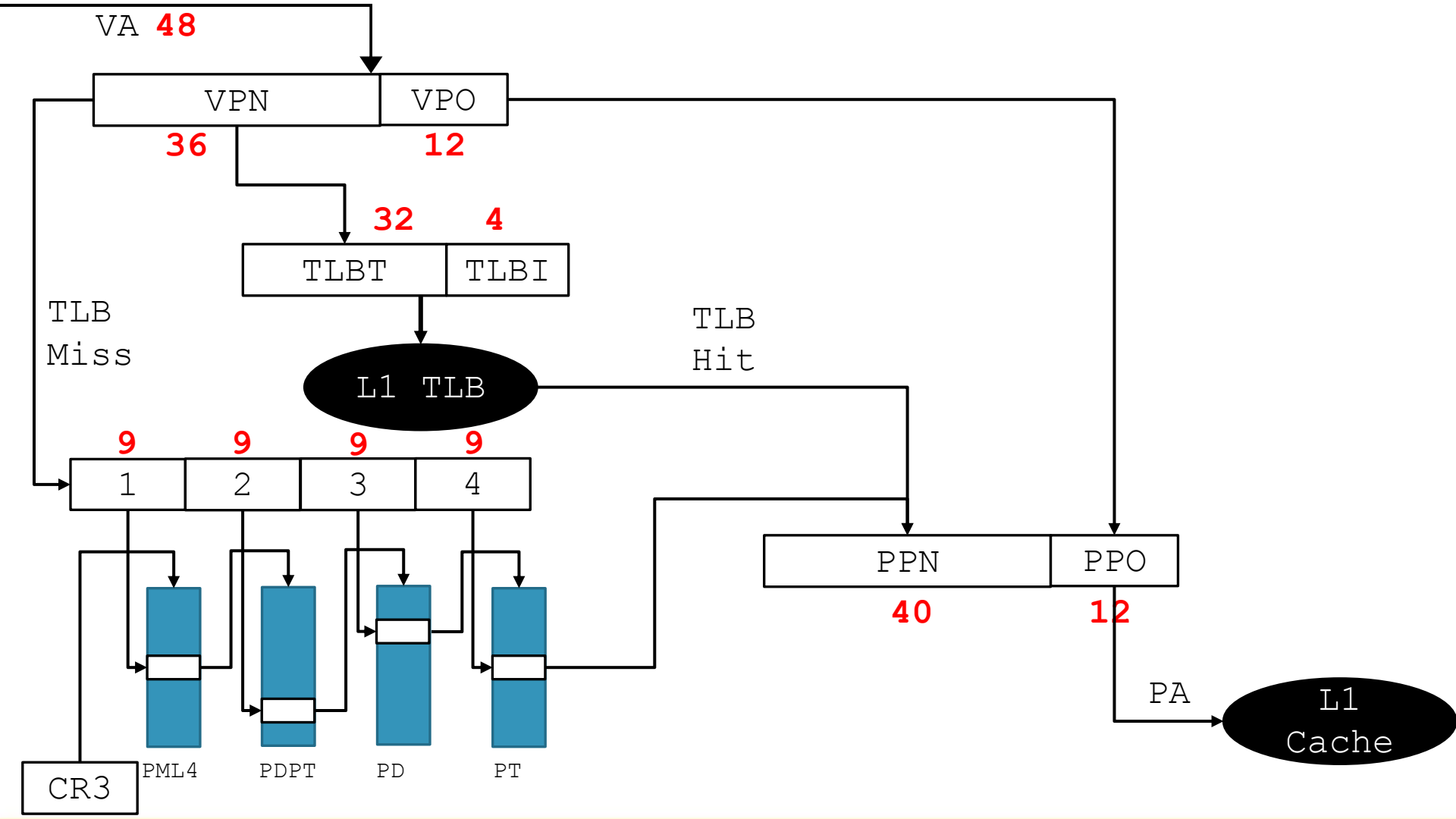
- IA32：页大小4KB、PTE条目4字节、每张页表占用一页、采用两级页表。虚拟地址空间是几位的？大页有多大？
  - 页大小4KB / PTE条目4 = 1024条目/页 → 每级VPN长度为10
  - 页大小4KB → （小页）VPO长度为12
  - 虚拟地址长度：10*2+12 = 32位
  - 大页大小：
    - VPN1→大页VPN1；
    - VPN2+VPO→大页VPO 10+12位→4MB

- 请求零页：.bss区在内存中大小是否为0？ N
- mmap、munmap
  共享映射（MAP_SHARED）和私有映射（MAP_PRIVATE）的确切含义？
- 处理COW的基本流程：

1. CPU发现页表权限不对（没有写权限），因此引发保护故障（x86中统一为Page Fault）
2. 异常处理程序（内核模式）发现这是一个COW页，因此另开新物理页、将内容复制过去、将映射改为指向新页、设置页表权限为可读可写
3. 重新执行当前指令

页表条目地址 = 页表基地址 + VPNi × 条目长度

例：PTE条目地址 = PDE读出的PT基地址 + VPN1 × 8



第九章 # 动态内存分配器 & 垃圾回收

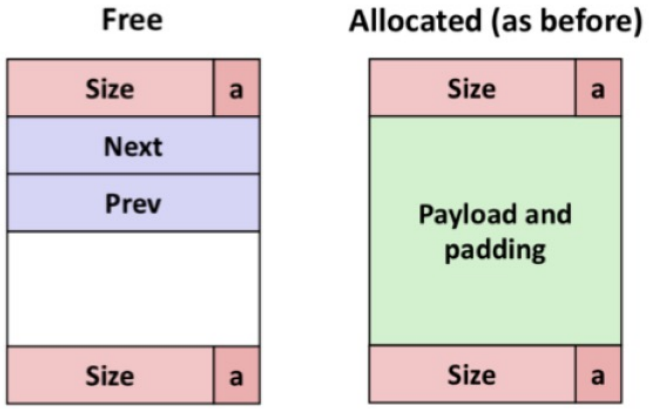
- 外部碎片、内部碎片
- 隐式链表、显式链表、分离链表、etc

- Java 和 C 中垃圾回收机制的不同和原因？
- C Mark-and-Sweep
  is_ptr() 的保守的回收策略

19期末

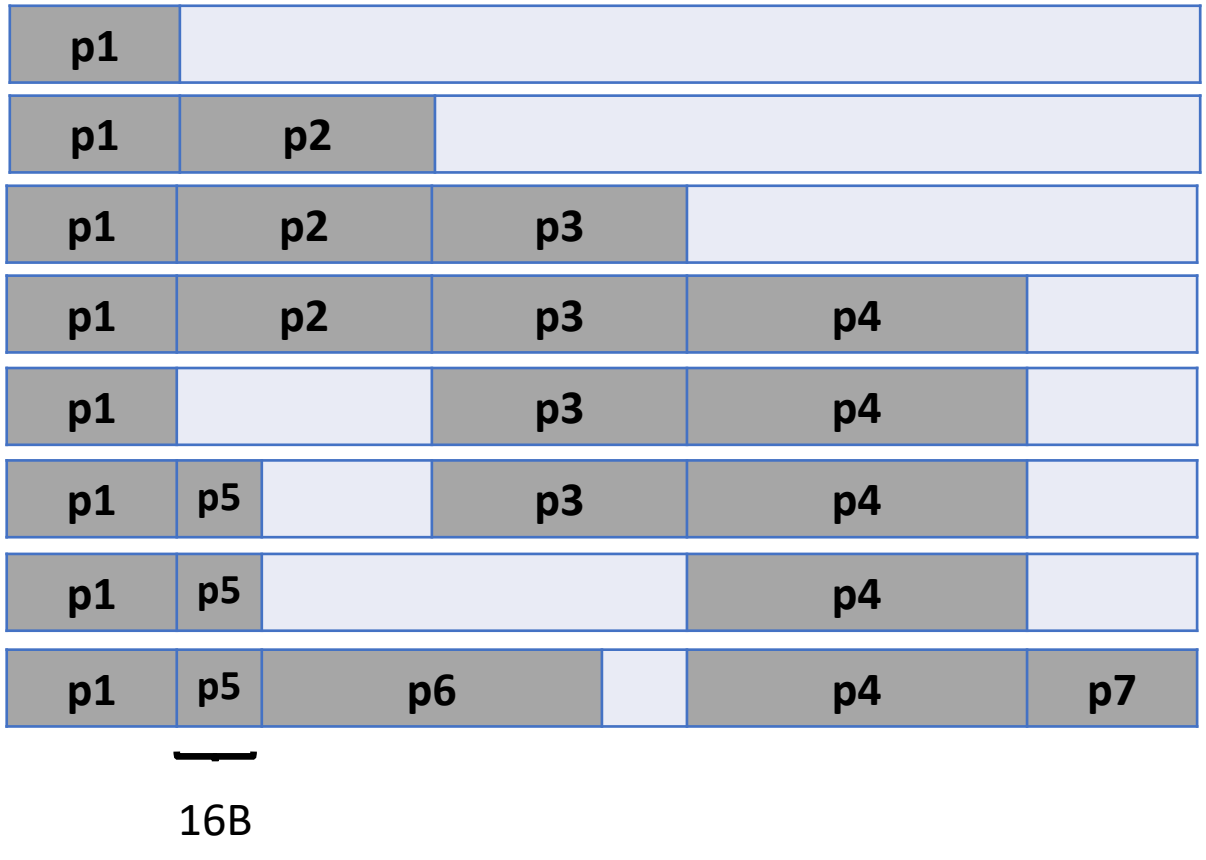
12. 现在有一个用户程序执行了如下调用序列

```
void *p1 = malloc(16);
void *p2 = malloc(32);
void *p3 = malloc(32);
void *p4 = malloc(48);
free(p2);
void *p5 = malloc(4);
free(p3);
void *p6 = malloc(56);
void *p7 = malloc(10);
```



内存分配器内部使用**显式空闲链表**实现，按照地址从低到高顺序来维护空闲链表节点顺序。分配块格式参照上图。每个分配块 16 字节对齐，头部和脚部的大小都是 4 字节。分配算法采用**首次适配**算法，将适配到的空闲块的第一部分作为分配块，剩余部分变成新的空闲块，并采用**立即合并**策略。假设初始空闲块的大小是 1KB。那么以上调用序列完成后，分配器管理的这 1KB 内存区域中，**内部碎片**的总大小是 74B，链表里**第一个**空闲块的大小是 16B 。

```
void *p1 = malloc(16);
void *p2 = malloc(32);
void *p3 = malloc(32);
void *p4 = malloc(48);
free(p2);
void *p5 = malloc(4);
free(p3);
void *p6 = malloc(56);
void *p7 = malloc(10);
```



第十一章 网络：分层 & 协议

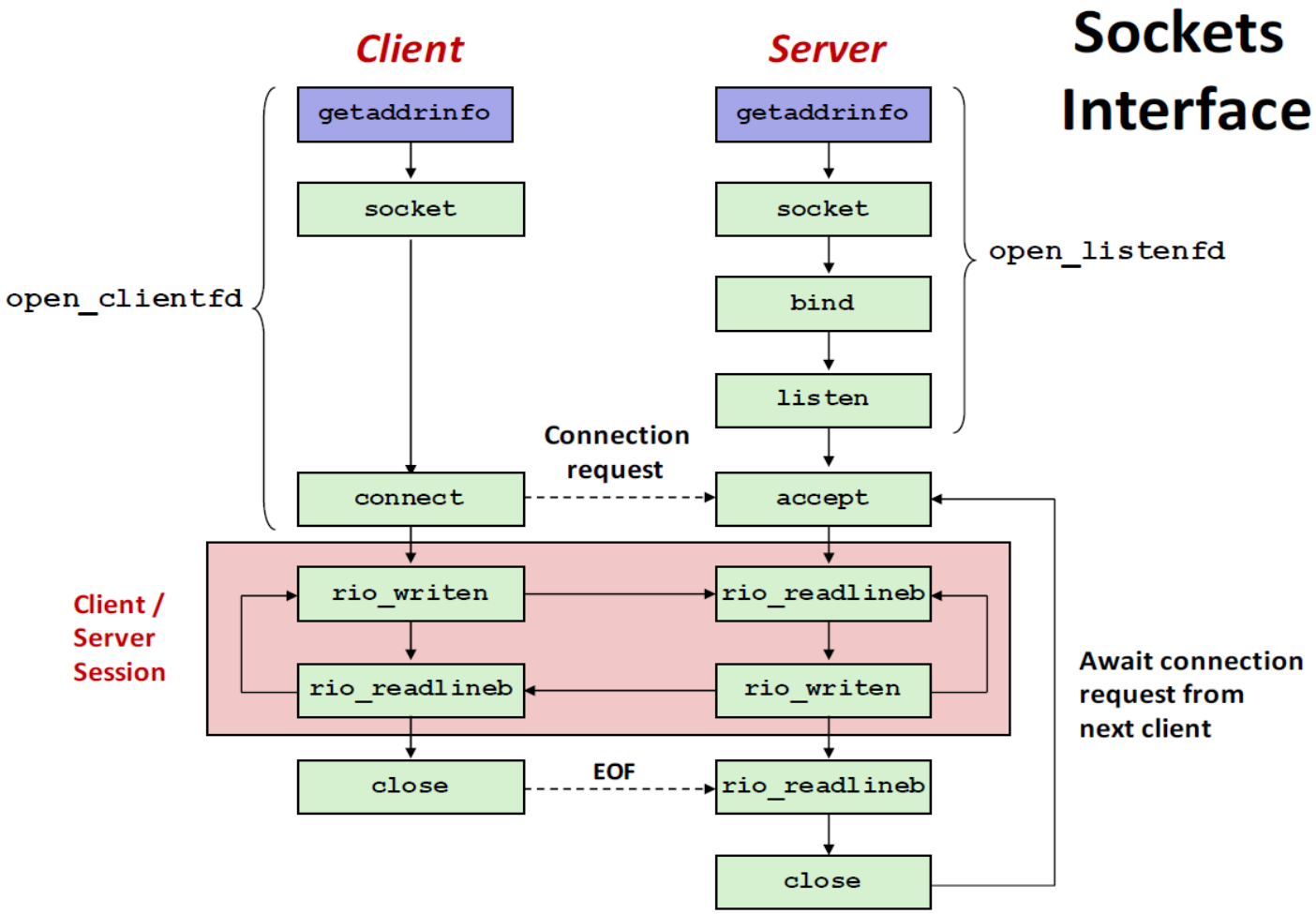
- Client-Server模型
  主机（Host）

- LAN、WAN
- IP地址、DNS。域名和IP地址的对应关系是什么样的？
- ABC类IP、特殊的IP地址
- HTTP协议、TCP/IP协议

- **以上内容请参考网络那节小班课的助教课件**

第十一章 网络编程：套接字

- 套接字地址 = IP地址 + 端口号
- 知名端口、临时端口



第十一章 Web服务和http协议

- URL格式
- CGI
- HTTP请求格式
- 使用telnet协议登录远程主机请求http服务：
  http request 和 response 的基本格式，关键字段的意义

均请见网络那节小班课的助教课件

第十二章 并发编程 # 1/2

- 线程模型、POSIX标准、pthread库

  哪些资源是同一进程的各线程共享的？通用寄存器、PC、文件描述符表、.text、地址空间、条件码、运行时栈、运行时堆

  -> 文件描述符、.text、地址空间、运行时堆

- exit、pthread_exit、return

1. 在除主线程外的其他线程中return时，将隐式地调用pthread_exit。 而当主线程return时，则会结束整个进程。
2. 为了使主线程结束时其他线程能够继续运行，主线程应该显式地调用pthread_exit退出。
3. 不管何时，调用 exit 都将导致整个进程的退出。

- 竞争、死锁、饥饿

  基本含义 与 常见的产生原因

- 同步机制：信号量及PV操作

  经典同步问题及问题产生的场景：

  生产者-消费者模型

  第一类/第二类读者-写者问题

  均请见上周小班课的助教课件

第十二章 并发编程 # 2/2

- Async-signal-safe thread-safe re-entrant (参考csapp3e / POSIX.1 的定义)
  - Async-signal-safe functions can be safely called from a signal handler, either because it is re-entrant, or because it cannot be interrupted by a signal handler.
  - Thread-safety: A function is said to be *thread-safe* if and only if it will always produce correct results when called repeatedly from multiple concurrent threads.

    常见的四种非线程安全的原因：

    没有保护共享变量

    保存内部状态（e.g. rand） -> redesign arguments

    返回static变量指针（e.g. ctime）-> Lock & Copy to save

    调用了线程不安全函数
  - Re-entrancy: they do not reference *any* shared data when they are called by multiple threads.

- 明显可重入将保证线程安全以及异步信号安全。

- **研读几种 echo 服务器的源码！**尤其是其中 ticky 的地方。

Reminders

- 记得带上学生证，试卷上写清小班号 7 班。
- 在线答疑持续到考试正式开始前。

Good Luck & Best Wishes!

- This is only a foretaste of what is to come, and only the shadow of what is going to be.
  By Alan Turing
  这不过是将来之事的前奏，也是将来之事的影子。
- You are right now equipped with sufficient background knowledge of computer system, so move ahead into the broader and more specific area of the computing world, including hardware, computer system organization, networks, software and its engineering, security and privacy, human-centered computing, applied computing and so on.
- **Below are some references which you may find useful in the near future.**

- Operating System course page for 6.S081 Fall 2020 at MIT, along with **xv6** labs:
  https://pdos.csail.mit.edu/6.S081/2020/
- Operating System course page for 6.828 Fall 2018 at MIT, along with **JOS** labs:
  https://pdos.csail.mit.edu/6.828/2018/
- Accurate categorization of the computing world, ***ACM Computing Classification System***
  https://dl.acm.org/ccs
- Collection of useful CS learning sources
  https://www.zhihu.com/column/college-simulator
- The Missing Semester of Your CS Education
  https://github.com/missing-semester-cn/missing-semester-cn.github.io