

第2章

1. 字节 = 8位的2进制块，每字节由唯一地址标记

2. 每台计算机都有一字长，指明指针数据标记。称大小虚拟地址：字长 w 位，范围 0~2^w-1

C 声明	32	64	unsigned 字长
char	1	1	
short	2	2	不变
int	4	4	
long	4	8	
float	4	4	
double	8	8	
void *	4	8	

3. 布尔位级运算：~、&、|、^ (掩码) 运算结果：0或1、11...1 (结果0x0/0x1)

4. 加减法优先级高于移位运算

5. W位数据类型 移位 K >= W, 取余

6. |TMmin| = |TMmax| + 1

7. signed OP unsigned 算术将有符号参数转化为无符号数 (包括 > <)

8. 无符号数求反

$$-u_w x = \begin{cases} x, & x=0 \\ 2^w - x, & x>0 \end{cases}$$

补码的非

$$-t_w x = \begin{cases} TM_{min}, & x = TM_{min} \\ -x, & x > TM_{min} \end{cases}$$

9. 补码加法与无符号位级表示相同 先2进制加再截断

10. 乘法：先算10进制正确结果 → 转2进制 → 截断

11. 算术左移 (有符号除法)

$$(x < 0 ? x + (1 < k) - 1 : x) >> k$$

偏置 biasing 修正舍入

无符号除法直接是算左移。

对有符号数 X + ~X = -1 = 0xFF...FF

12. IEEE

$$V = (-1)^s \times M \times 2^{-E}$$

S 符号、M 尾数 1~2-E (规格化) 或 0~1-E (非规格化) E 阶码

$$E = e - Bias, (e 为 exp 无符号整数)$$

$$Bias = 2^{exp-1} - 1$$

exp	frac	Bias
Float	8位	23位
double	11位	52位

exp	frac	M	E
规格化	非全0	\	1+f
非规格化	全0	\	1-Bias
00	全1	0	\
NaN	全1	\neq 0	\

非规格化：逐渐下溢、分布均匀接近0 到规格化平滑转变

13. 向偶数入 (向最接近值舍入)

① 先找最接近的值进行舍入

② 中间值：向偶数舍入 (避免统计偏差)

14. (int) float/double 向零舍入

第3章

.C → 预处理器 (cpp) → 编译器 (cc)

→ 汇编器 (as) → 链接器 (ld) → out

i: 对 #include 等替换，仍文本文件

s: 汇编语言文件

d: 2进制文件、可重定位目标文件

o: 可执行目标文件

gcc { -E
-S filename.c -o { .i
.s
.o
.out } }

机器级编程、两种抽象重要

< 指令集体系结构 / 指令集架构 (ISA) 地址 >

1. 程序员可见寄存器

① 程序计数器 PC、X86-64 用 %rip 表示

- ② 16个整数寄存器：③ 条件码寄存器：
- ④ 向量寄存器 (存放一个或多个整数/浮点数)
- 2. X86-64 虚拟地址 64位，目前实现中高16位 D，操作系统管理虚拟地址空间，翻译成处理器内存中的物理地址。
- 3. 以 . 开头的行都是指导汇编器和 ld 工作的伪指令
- 4. 字 = 2字节 = 2*8 = 16位 2进制。
- 5. 1个 X86-64 中央处理器单元 (CPU) 有 16个 64位通用寄存器。

0 %rax %eax %ax %al 返回值
3 %rbx %ebx %bx %bl 被调用者保存
1 %rcx %ecx %cx %cl 第4个参数
2 %rdx %edx %dx %dl 第3个参数
6 %rsi %esi %si %sl 第2个参数
7 %rdi %edi %di %dl 第1个参数
4 %rsp %esp %sp %pl 被调用者保存
%r8 %r8d %r8w %rb 第5个参数
%r9 %r9d %r9w %r9b 第6个参数
%r10 %r10d %r10w %r10b 调用者保存
%r12 %r12d %r12w %r12b 被调用者保存

6. 寻址方式

立即数寻址 Iimm \$Iimm

寄存器寻址 R[rax] ra

绝对寻址 M[Lmm] lmm

间接寻址 M[R[ra]] (ra)

[基址+偏移量]寻址 M[Lmm+R[rb]] lmm(rb)

变址寻址 (rb, ri) / lmm(rb, ri)

比例变址寻址 lmm(rb, ri, s)

rb 基址寄存器：rb 变址寄存器，均 64位

s 比例因子，必须 1、2、4、8

7. movq 源 32位补码立即数，符号扩展至64

movabsq (传递绝对四节) 以任意 64位立即数作源操作数、只能以寄存器为目的。

8. movz、movs、源只寄存器/内存 → 寄存器

movz (0 填充余字节): b → w/1/2, w → l/2

movs (符号扩展): b → w/l, g, w → l/2, l → g

cqtz: %eax(符号扩展) → %rax

* b 字节, w 字, l 双字, q 四字

9. 间接引用指针 [放在寄存器]、*xp.

10. 算术逻辑操作

leaq S, D < LS 加载有效地址

INC D 加1 DEC D 减1

NEG D 取负 → NOT D 取补 ~D

ADD SUB IMUL XOR OR AND +S,D

+ - * ^ | & B D < DOPS

SAL SHL SAR SAL +K,D

<< >> >>L 算术逻辑 D < D 移 K 位

leaq 没有引用内存，而是将有效地址导入目的操作数

11.

imulq S, RC[%rdx]:RC[%rax] ← S × RC[%rax] 有符号

mulq S, RC[%rdx]:RC[%rax] ← S × RC[%rax] 无符号

cqto RC[%rdx]:RC[%rax] ← 符号扩展 (RC[%rax])

idivq S, RC[%rdx]:RC[%rdx]:RC[%rax] mod S 有符号

divq S, RC[%rax]:RC[%rdx]:RC[%rax] ÷ S 无符号

12. 条件码

CF 进位 ZF 零 SF 符号 OF 溢出

最近操作最高位进位，可检查无符号溢出

13. CMP S1, S2 (S2-S1)

TEST S1, S2 (S2&S1)

14. JMP *%rax: %rax 中值作目标

JMP *(%rax): %rax 中值为地址在内存中值作目标

15. SWI 8跳转表在 .rodata 区

16. 过程调用返回机制。P 调用 Q，返回 P。

① 传递控制。PC 设为从代码起始地址，返回寄存器为 P 调用后一指令地址。

② 传递数据，一或多参数，一个返回值 (value)

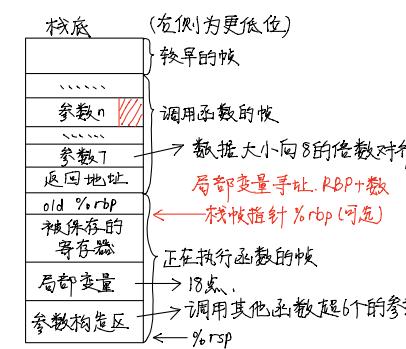
③ 分配和释放内存，开始时可能需为局部变量分配空间，返回前释放。

17. 移位控制

call Label 直接调用

call *Operand 间接调用

ret 从过程调用返回。



18. 栈上的局部存储

局部数据必须放在内存中的情况：

① 寄存器不足以放下全部本地数据。

② 局部变量用地址运算符 '%'，须产生地址。

③ 局部变量是数组或结构，通过引用访问。

19. X86-64 寄存器使用惯例。

被调用者保存寄存器: %rbx, %rbp, %r12~15

调用者保存寄存器: 其他除了 %rsp。

(被调用者任意使用，责任不在被调用者)

20. & D[i][j] = x0 + L(C, i+j)

21. 定长数组：优化 (数组引用转换成了指针间操作)

变长数组：参数 n 在参数 A[n][n] 前增加 3 寄存器的使用；用乘法指令对 i 伸缩 n 倍，而不是 leaq。

22. struct rect *rp.

rp → width 等于 (*rp).width

*rp.width 编译器译为 (*rp.width)

23. C 对数组引用不进行任何边界检查。

局部变量和状态信息存在栈中，越界数组元素的操作会破坏栈中状态信息，重新加载寄存器或执行 ret 指令，错误缓冲区溢出：栈中分配字符串数组来保存字符串，但字符串长度超出数组分配的空间。

gets / strcpy / sprintf / strcat

输入字符串包含可执行代码字节编码 — 攻击代码 (e.g. 覆盖返回地址)

24. ① 栈随机化：栈位置在程序每次运行都有变化，分配一段 0~n 字节之间的随机大小空间 (n 大足够变化，n 小不至于浪费空间)

地址空间布局随机化 ASLR

② 栈破坏检测：栈保护者机制来检测缓冲区越界 (金丝雀 / 哨兵值)

段寻址从内存读入 / 存放在特殊段

标志为“只读” / 检测是否改变。

③ 限制可执行代码区域

25. 结构体对齐：单个结构体内最大基本对象的倍数。K 字节基本对象地址也是 K 的倍数。

26. 变长栈帧 push %rbp (基指针)

mov %rsp, %rbp.

leave < mov %rbp, %rsp

pop %rbp

27. 流水线冒险

① 停待类避免：处理器停止一条或多条指令

技术：把一组指令阻塞在它们所处阶段，插入气泡。(允许其他指令继续通过)

条件：(源) srcA, srcB 处于译码阶段

目的：dstC, dstM 处于执行/访存/写回阶段

② 转发类避免：将结果值从一个流水线阶段传到较早阶段

③ 挂起转移结合 (加载互锁 + 转发技术) 平处理加载 / 使用数据冒险

控制冒险：当处理器无法根据处理器的当前指令来确定下一条指令地址时。

第2章 Y86-64 指令集	
halt	00
nop	10
rrmovq RA, RB	20 FAB
irmovq V, RB	30 FB + 8字节 V
rmmovq RA-D(RB), RA	40 FAB > +8字节 D
mrnmovq D(RB), RA	50 FAB
DQ RA, RB	bfa FA FB
jxx Dest	7fa + 8字节 Dest
cmovXX RA, RB	2fa FA FB
call Dest	80 + 8字节 Dest
ret	90
pushq RA	A0 TA F
popq RA	B0 FA F

5. pushq %rsp 压入旧值

popq %rsp 置为内存读出的值

6. 状态码 Stat

值	名字	含义
1	AOK	正常操作 → 程序继续执行
2	HLT	处理器执行 halt 指令
3	ADR	非法地址读/写
4	INS	非法指令

7. SEQ 硬件结构

① 取指：PC 作地址，指令内容读取指字节

PC 增加器计算 valp。

② 译码：从寄存器文件两个端口 A、B 读出 valA 和 valB。

③ 执行：算术/逻辑单元 ALU

CC: ALU 计算。

④ 写回：内存或寄存器写回

⑤ 读取：内存读取指字节

时钟寄存器 = PC, CC

随机访问寄存器：寄存器文件、指令内存、数据内存。

8. 流水线

流水线提高了系统吞吐量、只不过会增加延迟，还会增加硬件 (流水线寄存器)。

吞吐量由花费时间最长阶段决定。

延迟：从头到尾执行一条指令所需时间。

9. 流水线局限性

① 不一致划分，运行时钟的速率由最慢阶段决定。总延迟 = 阶段 × 最慢延迟。

② 流水线过深，收益下降。

阶段数增加，由于流水线寄存器的延迟，延迟总阶段周期比例增加，吞吐量增加效果不好。

③ SEQ = 1，创建状态寄存器来保存一条指令执行过程中计算出来的信号，PC 更新阶段放在开始而不是周期末，PC 不再保存在寄存器中。

11. PIPE = 1: SEQ + 插入流水线寄存器、信号重排。

F: 保存程序计数器的预测值。

D: 位于取指和译码阶段之间。

E: 位于译码和执行阶段之间。

M: 位于执行和访存阶段之间。

W: 位于访存阶段和反馈路径之间。

12. 预测下一个 PC

取指是条件分支指令，要几个周期后，通过执行阶段，才能知道是否要选择分支。

ret 指令，访存阶段，才能确定返回地址。

对 call 和 jmp 无条件转移，下一条指令是 valC。

对条件转移 valC / valP，采用分支预测技术，选择分支，valC 60% 成功率。

没选择分支 valP

13. 流水线冒险

数据冒险

① 停待类避免：处理器停止一条或多条指令

技术：把一组指令阻塞在它们所处阶段，插入气泡。(允许其他指令继续通过)

条件：(源) srcA, srcB 处于译码阶段

目的：dstC, dstM 处于执行/访存/写回阶段

② 转发类避免：将结果值从一个流水线阶段传到较早阶段

③ 挂起转移结合 (加载互锁 + 转发技术) 平处理加载 / 使用数据冒险

控制冒险：当处理器无法根据处理器的当前指令来确定下一条指令地址时。

阶段	计算	halt	nop	OP9 RA, RB	rmmovq RA, RB	irmovq V, RB	rmmovq RA, DR8	mrmovq DR8, RA	pushq RA	poplq RA	JXX Dest	CN1 Dest	ret
取指	icode:ifun RA:RB VA:C VA:P	M1[PC1]	M1[PC1]	M1[PC1] M1[PC+1]	M1[PC2] M1[PC+1]	M1[PC1] M1[PC+1] M8[PC+2]	M1[PC1] M1[PC+1] M8[PC+2]	M1[PC1] M1[PC+1]	PCT+2	PCT+2	M1[PC1] M1[PC+1]	M1[PC1] M1[PC+1]	M1[PC]
译码	valA, srcA valB, srcB	PC	PCT+1	PCT+2	PCT+2	R[RA] R[RB]	R[RA] R[RB]	R[RA] R[RB]	R[RA] R[RB]	R[RA] R[RB]	R[RA] R[RB]	R[RA] R[RB]	R[RA]
执行	valE Cond. codes			valB OP valA Set CC	D+valA	D+valC	valB+valC	valB+valC	valB+(-8)	valB+8	Cond < Cond (CC, ifun)	valB+(-8)	valB+8
访存	Read/Write					M8[valE]=valA	valM< M8[valE]	M8[valE]=valA	valE< M8[valA]			M8[valE]< valP	valM< M8[valA]
写回	E port, dstE M port, dstM					R[RA] < valE	R[RB] < valE	R[RA] < valM	R[RA] < valE	R[RA] < valM		R[RA]< valE	R[RA]< valM
更新IPC	PC	valP	valP	valP	valP	valP	valP	valP	valP	valP	PC < Cond ? valC : valP	valC	valM

4. 流水线控制逻辑

- ① 加载/使用冒险：从一条内存中读出一个值的指令和一条使用该值的指令之间，流水线必须暂停一个周期。
- ② 处理 ret：流水线暂停直到 ret 指令到达写回阶段。
- ③ 预测错误的分支：在分支逻辑发现不应选择分支之前，分支目标处的几条指令已进入流水线，必须取消这些指令，并从跳转指令后那条指令开始取指。
- ④ 异常：当一条指令导致异常，禁止后面指令更新程序可见状态，且异常指令到达写回阶段时，停止执行。

第7章

1. 静态链接器以一组可重定位目标文件和命令行参数作为输入，生成一个完全链接的、可加载运行的可执行目标文件作为输出。

2. 链接器两个主要任务：

- ① 符号解析：每个符号对应一个函数、一个全局变量或一个静态变量。
- ② 重定位：符号定义与一个内存位置关联起来，修改所有对这些符号的引用（链接器使用汇编器产生的重定位条目）。

3. 共享目标文件：特殊类型的可重定位目标文件，可以在加载或运行时被动态地加载到内存并链接。

4. ELF 可重定位目标文件格式：

- ① ELF：可执行可链接格式。
- ELF 头：以一个 16 字节的序列开始，这个序列描述了生成该文件的系统的字的大小和字节顺序。剩下部分包含帮助链接器语法分析和解释目标文件的信息。

-text：已编译程序的机器代码。

-rodata：只读数据，printf 格式串、switch 块句跳表。

-data：已初始化的全局和静态变量。

-bss：未初始化或初始化为 0 的全局和静态变量。仅位标注，不占据任何实际的空间。

-symtab：一个符号表，存放程序中定义和引用的函数和全局变量信息，不包含局部变量且（或）。

-rel.text：一个 text 节中位置的列表，链接器链接重定位时需修改。

调用外部函数或引用全局变量的指令需修改，本地函数指令则不需要。（可执行目标文件不需要重定位信息，忽略）

-rel.data：被模块引用或定义的所有全局变量的重定位信息。

-debug：调试符号表。

-line：一个字符串表，包括 symtab 和 debug 节中的符号表，以及节头部中的字符串。

节头部表：不同节位置和大小，其中目标文件中每个节都有一个固定大小的条目（entry）。

描述文件的节，前面全为节。

5. 特殊伪节，在节头部表中无条目。

① ABS：不该被重定位的符号。

② UNDEF：未定义符号，本模块引用，却

在其他地方定义的符号。

③ COMMON：未初始化的全局变量。
(bss：未初始化的静态变量，初始化为 0 的全局或静态变量)

* 只有可重定位目标文件才有这些伪节。

6. 解释多重新全局符号：

弱：已初始化全局变量（包括 0）、函数体弱、未初始化全局变量、函数声明。

7. 静态库：

编译系统将所有相关的对象打包成一个单独的文件，称为静态库。
构造输出的可执行文件，复制静态库里被应用程序引用的目标模块。
在 Linux 系统中，静态库以一种称为存档的特殊文件格式存储在磁盘中，（一组连接起来的可重定位目标文件集合，头部、a）。

8. 重定位：

- ① 重定位节和符号定义：链接器将所有类型的节合并为同一类型的新的聚合节。将运行时内存地址赋给新的聚合节。完成后每条指令和全局变量都有唯一的运行时内存地址了。
- ② 重定位节中的符号引用：链接器修改代码节和数据节中对每个符号的引用，使之指向正确运行时地址。

9. 重定位条目

代码重定位条目：.rel.text

已初始化数据重定位条目：.rel.data

10. 重定位符号引用

```
refptr = s + r.offset
if (r.type == R_X86_64_PC32) {
    refaddr = ADDR(s) + r.offsets
    *refptr = (unsigned)(ADDR(s.symbol) +
    r.addend - refaddr);
}
if (r.type == R_X86_64_32)
    *refptr = (unsigned)(ADDR(s.symbol) +
    r.addend)
```

① PC 相对引用

```
e: e8 00 00 00 00 00 00 00 00 00 00 00
    .callq 13 <main+0x13>
    f: R_X86_64_PC32 sum -0x4
    r.offset r.type rsymbol r.addend
```

ADDS(S) 节运行时地址，由 rsymbol 节确定 ADDS(r.symbol) 符号运行时地址

call：下一条指令 + 填入数 = 函数起始

mov：下一条指令 + 填入数 = 后面地址

② 重定位绝对引用

11. ELF 可执行目标文件：

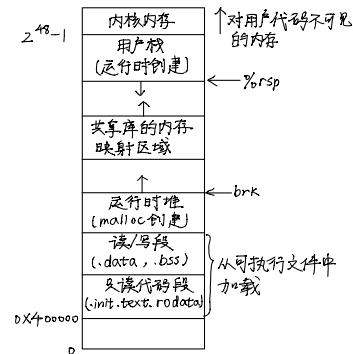
将连接的 ELF 头文件映射到内存段 {段头部表} → 只读内存段（代码段）
到运行时 {init, .text, .rodata, .data, .bss, .symtab, .debug, .line, .strtab} → 读/写内存段（数据段）
插述目标 {节头部表} → 不加载到内存的符号表和调试信息

ELF 头描述头文件的具体格式，还包括程序入口 init 节定义了一个函数 init，程序初始化代码会调用它。完全链接，无 rel 节。

12. 可执行目标文件的加载。

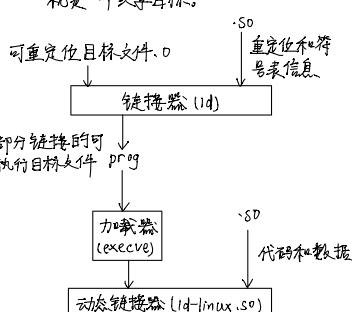
Linux 通过调用 execve 函数来调用加载器，loader 将可执行目标文件数据和代码从磁盘复制到内存，跳转到程序入口运行。将程序复制到内存并运行的过程叫加载。

13. Linux x86-64 系统，代码从地址 0x400000 处开始，后面是数据段、堆、共享模块、栈。从 2⁴⁸ 开始，为内核代码数据保留。



14. 静态库：定期维护和更新，显式地将程序与更新了的库重新链接；函数代码会复制到每个运行进程的文本段中，内存浪费。

共享库、目标模块，在运行或加载时可以加载到任意的内存地址，并和一个在内存中的程序链接；该过程为动态链接，由一个动态链接器程序执行。共享库又称共享目标，(.so / DLL)。动态链接器本身也是一个共享目标。



15. 共享库：允许多个正在运行进程共享内存中相同的库代码。可加载而无需重定位代码称为位置无关代码。

16. 库打桩：允许程序员拦截对共享库函数调用，取代代码执行自己的代码。编译 -fPIC 参数 /链接 -fwrapv /运行

3. 异常类别（用户→内核模式唯一方法）

中断 异步 I/O 设备信号 下一条

陷阱 同步 有意异常 下一条

故障 同步 潜在可恢复错误 当前

终止 同步 不可恢复错误 不返回

4. 异常与调用区别

① 异常跳转至处理程序前，压栈的是当前或下一条指令

② 窃取处理器状态压栈

③ 若控制从用户进程转到内核，则压入内核栈

④ 异常处理程序运行在内核模式下

5. 中断：硬件中断，e.g. 网络驱动器，磁盘控制器和定时器芯片，Ctrl+C。处理器芯片引脚发信号，异常号发系统总线。

6. 陷阱

用途：在用户程序和内核之间提供了一个像过程一样的接口，系统调用。

向内核请求服务：read / fork / execute / exit 处理器：特殊指令 syscall。

7. 故障：由错误引起，可能被故障处理器修正

e.g. 缺页异常

8. 终止：致命错误，常硬件错误，e.g. DRAM 或 SRAM 位被损坏 / 机器检查

9. 除法错误：除 0，或结果对目标操作数太大。除法错误（故障）

一般保护故障：引用未定义虚拟内存区域，或试图写只读文本段，Linux 不恢复段故障。

10. syscall

Linux 系统参数通过寄存器传递 %rax 调用号，最多包含 6 个参数：%rdi, %rsi, %rdx, %r10, %r8, %r9 返回 %rcx 与 %r11 换锁环，%rax 返回值 %rax 从 -495 到 -1 表明错误，errno。

11. 进程：一个执行中程序的实例。系统中的每个程序都运行在某个进程中，两个关键抽象：独立的逻辑控制流和私有的地址空间。

12. 上下文：程序正确运行所需状态：包括内存中数据和代码、栈、通用目的寄存器、PC、环境变量以及打开文件描述符的集合。

上下文就是内核直接启动一个被抢占的进程所需状态。

调度、调度器

13. 上下文切换

① 保存当前进程上下文

② 恢复某个先前被抢占的进程被保存的上下文

③ 将控制权传递给这个新恢复的进程。

14. 系统调用错误处理：返回 -1，设 errno, strerror(errno);

15. 进程 ID：getpid 调用进程 PID getppid，父进程 PID

16. 进程状态

① 运行：CPU 执行 / 等待被执行会内核调度

② 停止：进程执行被挂起且不会被调度。

SIGSTOP, SIGSTP, SIGTTIN, SIGTTOUT 信号，停止 / SIGCONT 信号，开始运行

③ 终止：信号默认行为是终止进程 / 主程序返回 / exit 函数，以 status 退出状态。

第8章 异常控制流 ECF

1. ECF 发生在计算机系统各个层次

① 硬件层：硬件检测到的事件会触发控制突然转移到异常处理程序。

② 操作系统层：内核通过上下文切换控制一个用户进程转移到另一个进程。

③ 应用层：进程发送信号到另一个进程，接收者会将控制突然转移到它的一个信号处理程序。

2. 处理异常硬件和软件（操作系统）合作

异常号：处理器 / 操作系统内核分配

17. fork

子进程得利与父用户级虚拟地址空间相同(独立)的副本,包括代码、数据、堆、共享库及用户栈,打开任何文件描述符相同的副本,读写父进程打开的任何文件。

18. 回收子进程

子进程终止,被保持在一种已终止状态(僵死进程)直到被父进程回收。父进程终止,内核安排init进程成为孤儿进程父亲,回收僵死子进程(消耗系统内存资源)。

19. waitpid(pid_t pid, int *status, int options)

① options

默认为0,挂起调用进程的执行,直到等待集合一个子进程终止。

WNOHANG:不挂起当前进程,等待集合任何子进程没有终止,立即返回0。

WUNTRACED:子进程之一终止或停止。

WCONTINUED:挂起子进程终止或被停止进程收到SIGCONT开始执行。

② pid>0,特指子进程

pid=-1,全部子进程

③ status若非零,waitpid会在status中放上关于导致返回的子进程状态信息, status是status指向的值。

④ 若子进程返回-1, errno设置为ECHILD
被信号中断,返回-1, errno设置为EINTR
wait(&status)=waitpid(-1,&status,0)

20. unsigned int sleep(unsigned secs)

时间到返回0
信号中断返回 被挂起
剩下休眠秒数

21. execute(*filename, *argv[1], *envp[1])

当前进程上下文中加载运行一个新的程序,一次调用从不返回。(只有出错才返回到调用程序)

argv以null结尾指针数组。
argv[0]可执行目标文件名字
envp以null结尾指针数组。
每个指针指向一个环境变量字符串
控制块基准主函数
argc在%rdi / argv在%rsi
argc表示argv非空指针数。

22. 信号

① 发送信号:内核通过更新目的进程上下文的某状态、发送一个信号给目的进程
<内核检测到系统事件
<进程调用kill函数
*可以自己给自己发。

② 接收信号:目的进程捕获内核通知以某种方式对信号发送作出反应(忽略/终止/通过)
通过信号处理器程序捕获信号)

SIGFPE 浮点异常
SIGILL 非法指令
SIGSEGV 非法内存引用
SIGINT 前台进程组 Ctrl+C
SIGKILL 终止
SIGCHLD 子向父已经终止或停止
SIGSTP 前台进程组 Ctrl+Z(挂起)
SIGALARM alarm 内核 secs 秒发送自己

23. 待处理/阻塞信号

待处理信号:发出而未被接收
一种类型至多一个待处理信号,不会排队
除非被丢弃
阻塞信号:可发送,不被接收。
内核为每个进程在pending位向量中维护待处理信号集合,blocked阻塞信号。

24. 进程组 ID, getpgid 函数

父子进程同一组

```
int setpgid(pid_t pid, pgid)
pid 进程的进程组设为 pgid。
pid = 0, 当前进程。
pgid = 0, 进程 pid 的 PID 作进程组 ID
/bin/kill -9 进程组 ID (SIGKILL)
int kill(pid_t pid, sig) 若 pid > 0,
进程 pid 所在进程组的每个进程发送 sig;
pid = -1, 当前组; pid < 0,
进程组 |pid|
```

25. 接收信号

内核把进程从内核模式切换到用户模式,检查进程未被阻塞的待处理信号的集合。空,内核将控制传递到进程。

是转控制流下一条指令,非空,内核选择集合中某信号 K(always MIN) 强制进程接收。

26. 信号默认行为

- ① 进程终止
- ② 终止并转换内存
- ③ 挂起直到 SIGCONT 重启
- ④ 处理该信号

27. signal 函数修改和信号相关联的默认行为。

```
signal(int signum, handler)
handler = SIG_IGN, 忽略类型 signum
= SIG_DFL, 依循默认
否则 handler 是用户定义的函数地址作信号处理器
```

28. 信号阻塞

- ① 隐式阻塞机制:内核默认阻塞任何当前处理器正在处理信号类型的待处理信号。
- ② 显式: sigprocmask 函数和其辅助函数,明确规定阻塞和解除选定信号。

```
int sigprocmask(int how, *set, *oldset)
SIG_BLOCK: set 添加到 blocked
SIG_UNBLOCK: blocked 中删去 set
SIG_SETMASK: blocked = set
若 oldset 非空, blocked 保存在 oldset 中
sigemptyset: set = φ
sigfillset: set = all
sigaddset: signum 加入 set
sigdelset: set 中除 signum
```

29. 非本地跳转:直接直接从一个函数转到当前正在执行函数, setjmp 和 longjmp 提供。

```
int setjmp(jmp_buf env)
在 env 缓冲区保存当前调用环境(后面 longjmp 使用, 返回 D.(PC, %rsp, 通用目的寄存器)。1 次调用多次返回)
void longjmp(env, int retval)
从 env 缓冲区恢复调用环境,触发一个从最近一次初始化 env 的 setjmp 调用的返回。1 次调用从不返回
```

第6章 存储器层次结构

1. 随机访问存储器 RAM

SRAM: 双稳态存储单元,有电则保持值,无电后可恢复稳定值。
DRAM: 干扰敏感,无法恢复;光载导致电压改变,必须用其读出刷新。
传统 DRAM: d 个超单元,每个 m 个 DRAM 单元, r 行 c 列阵列, rc=d。

DRAM 和 SRAM 易失性:断电丢失信息
PROM 烧录一次
EPROM 可擦写可编程 ROM
EEPROM 电子可擦除 ROM
闪存:非易失性存储器 基于 EEPROM → 固态硬盘 SSD

2. 磁盘

磁盘容量 = 盘片数 × 表面数 (2) × 道道数 × 平均扇区数 × 字节数 (单位: B)

$$B = \frac{KB}{2^10} \times MB \times GB$$

访问时间 = 寻道时间 + 旋转时间 + 传送时间

$$= T_{\text{seek}} + T_{\text{rotation}} + T_{\text{transfer}}$$

$$= \sim + \frac{1}{2} T_{\text{max rotation}} + \sim$$

$$= \sim + \frac{1}{2} \times \frac{1}{\text{RPM}} \times \frac{60\text{s}}{1\text{min}} + \frac{1}{\text{RPM}} \times \frac{1}{\text{平均扇区数}} \times \frac{60\text{s}}{1\text{min}}$$

3. 连接 I/O 设备 I/O 总线

4. 局部性:倾向于引用邻近于其他最近引用过的数据项的数据项,或者靠近引用过的数据项本身。(局部性原理)
时间局部性:引用过一次的内存位置很可能在不足的将来又被多次引用。
空间局部性:引用过一次的内存位置很可能在不足的将来引用附近一个内存位置。
计算机会层次,从硬件到操作系统、到应用层都利用 3 层局部性。

5. 步长为 1 的引用模式(顺序引用模式)
步长增加、空间局部性下降
② 重复引用相同变量程序有良好时间局部性
③ 取指令,循环在好的时空局部性,循环体越小越好

6. 存储器层次结构

L0: 寄存器
L1-L3: 高速缓存(SRAM)
L4: 主存(DRAM)
L5: 本地二级存储(本地磁盘)
L6: 远程二级存储(分布式文件系统、Web 服务器)

7. 高速缓存 cache

K+1 层的存储器被划分为连续的数据对象组块(chunk),称为块(block),每个块都有唯一地址或名字,固定/可变大小
数据总以块大小为传送单元。

8. 缓存命中: K+1 层数据对象在 L 层中缓存有缓存不命中: 从 K+1 层取块,若 L 层满替换或驱逐某个牺牲块(替换策略控制)
只要不命中,执行放置策略
① 强制不命中或冷不命中,冷缓存(为空),
② 冲突不命中,限制性的放置策略,多个下层块映射到同一区域
③ 容量不命中: 工作集大小超过缓存大小

9. CPU 和主存之间差距大,CPU 寄存器文件和主存之间插入了 SRAM 高速缓存存储器,称为 L1 高速缓存。

10. 存储器地址 m 位。

$S = 2^s$ 个高速缓存组
每组 E 个高速缓存行
每行 $B = 2^b$ 个字节的数据块
每行 t 个有效位
每行 $t = m - (b+s)$ 个标记位

地址: t 位标记, S 位组索引, b 位块偏移
 $m-1 \quad CT \quad C1 \quad CO \quad 0$
高缓存大小 $C = S \times E \times B$

11. 直接映射 E=1

有效位设置 + 高速缓存行标记位与地址标记位匹配

12. 标记位和索引位连起来唯一标识内存中的每个块
多个块会映射到同一个高速缓存组
映射到同一个高速缓存组的块由标记位唯一地标识

13. 斜率组相联,每组 E 行
 $E = C/B$, 全相联, $S = 1$ (地址无索引位)
行替换 < 最不常使用(LFU)
最近最少使用(LRU)

14. 索引位为何在中间

若用高位索引,一些连续内存块会映射到相同高速缓存块,对于空间局部性良好的程序,高速缓存使用效率低。而中间位索引,相邻块映射到不同高速缓存行。

15. 高速缓存的写

已缓存的字 w(写命中)在高速缓存更新了它的 w 副本之后,怎么更新 w 在层次结构中紧挨着的低一层副本。
① 直写立即(简单,每次引起总线流量)
② 写回:替換算法导致更新了的过时块才写(显著减少总线流量,但增加了复杂性,为每个高速缓存行维护一个修改位,表明是否修改过)

16. 写不命中处理

① 写分配:加载相应低一层中的块到高速缓存中,然后更新(利用局部性,每次不命中都会传递)
② 非写分配:直接把字写到低一层去
* 直写 —— 非写分配
写回 —— 写分配 △

① 较长传送时间
② 与处理该方式对称,利用局部性

17. L-cache 只保存指令的高速缓存
D-cache 只保存数据的高速缓存
统一的高速缓存(都保存)

不同访问模式来优化 L2,
块大小相联度和容量可不同
不同的高速缓存也确保了数据访问和指令访问不会形成冲突不命中
代价可能引起容量不命中增加。

18. 性能影响

不命中率 = 不命中数 / 引用数量
命中时间:从高速缓存传递一个字到 CPU 时间。
组选择,行地址,字选择时间
不命中处罚:额外时间
① 高速缓存大小:大,提高命中率,运行时间 ↓
② 块大小:大,空间局部性好,块大行数少,访问时间局部性比空间更好的程序员的命中率
③ 相联度:高 E 降低冲突不命中抖动可能,但成本高、复杂,增加命中时间,不命中处罚
④ 写策略:直写易实现,独立于高速缓存的写缓冲区。在下层,越可能写回(传递时间)

第9章 虚拟内存 VM

1. 虚拟内存三个重要能力

- ① 将主存看成一个存储在磁盘上的地址空间的高速缓存,在主存中只保存活动区域,并根据需要在磁盘和主存之间来回传送数据。
- ② 为每个进程提供了一致的地址空间,简化内存管理。
- ③ 保护进程地址空间不被其他进程破坏。

2. 物理地址 PA 虚拟地址 VA
地址翻译需 CPU 硬件和操作系统间紧密合作,内存管理单元 MMU

3. 虚拟地址位数 n, 虚拟地址数 $N = 2^n$
最大可能虚拟地址 $N-1$

4. 虚拟内存分割为虚拟页 VP, 大小 P = 2^p 字节
物理页 PP 大小也为 P 字节(页帧)
虚拟内存存储于磁盘
虚拟页的三种状态:

- ① 未分配: VM 系统未分配(或例建)的页,无任何数据关联,不在磁盘空间。
- ② 缓存的: 页缓存在物理内存中已分配页
- ③ 未缓存的: 未缓存在物理内存中已分配页

5. 页表页相联,每组 E 行
 $E = C/B$, 全相联, $S = 1$ (地址无索引位)
行替换 < 最不常使用(LFU)
最近最少使用(LRU)

6. 物理内存
虚拟地址 → 物理页号
物理页号 → 物理页框
物理页框 → 物理页
物理页 → 物理内存
物理内存 → 物理内存(磁盘)

7. 缺页: DRAM 缓存不命中
处理: 调用内核缺页异常处理程序,选择 DRAM 中一个牺牲页,修改牺牲页 PTE,再将新页复制到内存,更新 PTE,异常处理器返回时重装启动导致缺页的指令。

8. 虚拟内存作内存管理工具
操作系统为每个进程提供独立页表,多 VP 可映射到一个共享 PP

- ⑨ 简化链接：进程内存映像使用相同基本格式
 ⑩ 简化加载：双向内存加载可执行文件和共享对象文件、为代码段和数据段分各自虚拟，标记未被缓存，页表条目指向目标文件适当位置
 ⑪ 简化共享：
 ⑫ 简化内存分配：没必要分配连续的物理内存页面（分散）

10. 内存保护工具

页表增加许可位。

SUP (内核模式) READ, WRITE
 违反许可条件、触发保护故障
 也 Linux 错误

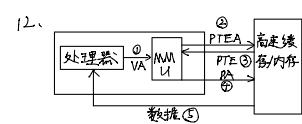
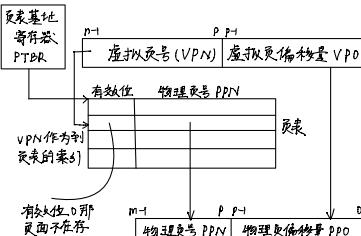
11. 地址翻译

$N = 2^m$ 虚拟地址空间数量

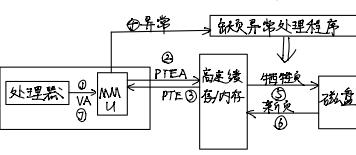
$M = 2^n$ 物理地址空间数量

$P = 2^p$ 字节，页面大小

PTE 数 = 虚拟页数 = $\frac{N}{P}$



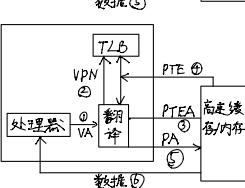
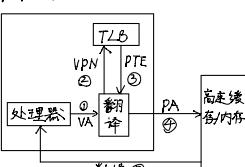
- ① 处理器生成一个虚拟地址，传递给 MMU
 ② MMU生成 PTE 地址，从 ... 请求得到它
 ③ ... 向 MMU 返回 PTE
 ④ MMU 构造物理地址，传递给 ...
 ⑤ ... 返回请求的数据字给处理器



- ①-③.
 ④ PTE 有效位 0，MMU 触发异常，传递 CPU 中控制到操作系统内核中缺页异常处理程序
 ⑤ ... 确定出物理内存中牺牲页，若已修改，换出磁盘
 ⑥ 缺页处理程序调入新页面，更新 PTE
 ⑦ 返回原来进程，再执行导致缺页指令

13. TLB 翻译后备缓冲器

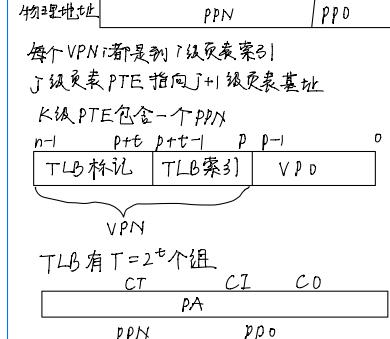
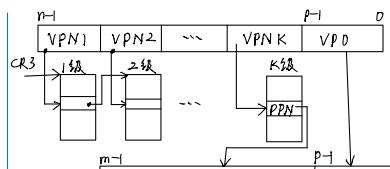
从 MMU 中关于 PTE 的小缓存



14. 多级页表减少内存要求

① 一级页表的一个 PTE 为空，相应二级页表根本不存在

② 只有一级页表才存在主存中，虚拟内存系统可在需要时创建、页面调入调出二级页表



每个 VPN 都是到 1 级页表索引

1 级页表 PTE 指向了 1 级页表地址

K 级 PTE 包含一个 PPN

TLB 标记 TLPB [TLB 索引] VPD

VPN

TLB 有 T=2^t 个组

CT CI CO

PPN PA PPD

15. 内存映射

虚拟内存区域与一个磁盘上的对象关联起来，以初始化这个虚拟内存区域的内容，这个过程称为内存映射。内存映射的两种类型之一：

① 普通文件：一个区域可以映射到一个普通磁盘文件的连接部分

文件区域分成大小的片，对虚拟页面初始化

② 匿名文件：内核创建，包含的全是二进制零

首次访问该区域的虚拟页会引发缺页异常 → 分割一个全零的物理页

一旦页面被修改，即和其他页面一样

16. 共享对象：任何写操作对其他映射了共享对象的进程可见，变化反映在磁盘原处对象上

私有对象：写操作对其它进程不可见，变化只反映在磁盘上

采用写时复制，共享同一物理副本直至一个进程执行了写，才创建一个新副本

17. 动态内存分配

显式分配区域：显示释放已分配块

malloc, free

隐式分配器：垃圾收集器、分配器检测

一个已分配块不再被程序使用释放

18. 碎片

内部碎片

① 维护数据结构产生的开销

② 增加块大小以满足对齐的约束条件

③ 显式地策略决定

外部碎片

空间内存结合起来足够满足一个分配请求，却没有足够大的独立空闲块能满足分配请求

20. 内存向图根节点、不在堆中，可以是寄存器栈里的变量、全局变量

21. 记录空闲块

① 隐式空闲链表，通过头部中的长度字段 — 隐式地链接所有块

② 显式空闲链表，在空闲块中使用指针

③ 分离的空闲列表，按大小分类，每个类/组使用一个空闲链表

④ 块大小排序：平衡树

22. 放置策略：首次走丢、下一次走配、最佳走配

合并策略：立即合，并延迟合，并

23. 垃圾收集：垃圾收集将内存视为一张有向

可达图，节点分为一组根节点和一组堆节点

每个堆节点对应一个已分配块，根节点包含

指向堆中指针，有寄存器、栈底或甚至是虚

拟内存中，读取数据区域的全局变量。

24. 隐式空闲链表

一个块由一个字的头部、有效载荷，以及可能的一些额外的填充组成的

25. 显式空闲链表

将堆组织成一个双向空闲链表，每个空闲

块主体中都包含一个前驱后继指针

3 2 1 0 头部

pred 指向

succ 后继

填充(可选)

块大小 | 块部

原来的有效载荷

26. 共享文件

不同进程共享一个互斥块 (锁住文件)

open 函数打开 filename 两次

第5章

编译器优化

提供从程序到机器的有效映射

寄存器分配、代码选择排序

消除死代码：消除轻微的低效问题

优化策略：

① 消除循环的低效率

· 代码移动、减少执行的效率 (公共运算)

· 复杂运算简化，用更简单的方法

替换昂贵的操作 (移位加减乘)

· 支持公用表达式 (重用表达式一部分)

② 减少跨程调用 (函数移到循环外)

③ 消除不必要的内存引用 (局部变量暴政)

④ 循环展开 —— 面向 CPU (可以避免每个)

⑤ 提高并行性，重新组织操作 循环的边界检查

⑥ 尽量缩短关键路径

⑦ 用条件表达式代替 if-else

⑧ 使用速度更快的 CPU 指令如 SSE

妨碍优化因素

① 函数调用

② 内存引用使用 (不同内存引用指向相同位置)

循环展开 (流水线 + 并行指令级)

① for (i=0; i<length; i++)

 x = x * op d[i];

② limit = length - 1;

 for (i=0; i<limit; i+=2)

 x = x * op d[i] * op d[i+1];

 for (i=1; i<length; i+=1)

 x = x * op d[i];

 // x = x * op d[0] * op d[1] * ... * op d[n-1];

③ 使用分离的累加器 (独立操作的流水)

for (i=0; i<limit; i+=2) {

 x0 = x0 * op d[i];

 xi = xi * op d[i+1];

 for (j=i+1; j<length; j++)

 x0 = x0 * op d[j];

 dest = x0 * dest;

分支预测

第10章

1. Linux 所有 I/O 设备模型化为文件

甚至内核也映射为文件

Linux 内核引出了简单、高级应用接口 Unix I/O

open, close, read, write, lseek (改变当前的文件位置)

2. 文件类型

① 普通文件：任意数据 (二进制和文本)

② 目录：一组链接的文件，文件名映射至文件

③ 套接字：易于进程间、网络通信。

④ 命名通道、符号链接、字符和块设备

3. 不足值

① 读时遇到 EOF

② 从终端读文本行

③ 读写网络套接字

④ 读磁盘文件，磁盘满或空间不足

⑤ 写磁盘文件，磁盘满或空间不足

4. 元数据：关于文件信息 (内核保存)

调用 stat 和 fstat 函数来访问

5. 插件共享 I (每进程共享)

打开文件集 II (所有进程共享)

v-node 表 III (所有进程共享)

FileA Terminal

文件句柄

文件大小

文件类型

文件权限

文件位置

引用数 = 1

文件类型

文件大小

文件类型

文件大小