ICS 2019、2020 期末考题分析

王畅

2021 年 12 月

注意：水平有限，不保证正确，祈请原谅。

2020 年

选择题

1. C。A 是正确的，不能把 attack lab 和 bomb lab 混淆了。C 的错误原因在于可以支持可变栈帧，编译时不能确定大小，必须使用帧指针。

2. B。计算知该缓存有 4 组，每组 2 行 4 字节，所以容量为 32 字节。每行需要 4 字节存储块的内容，1 位有效位和 2 位 tag。鉴于课本没有讲解 cache physical size 的计算方法，而且实际上 cache line 可能还有一些额外的 flag bit，因此 1 项后半部分实属超纲。但额外的 3 位无论如何也占不到 2 字节，因此可判断 1 错误。3 说反了，B 的格式导致相邻的缓存块进入相同的组中，会影响空间局部性比较好的程序（顺序访问），这样一来 4 错误（参看书 P432）。最后来计算 2，注意到访问 1、7、8 之后，三个缓存块会进入 A 中的不同组，而 B 则会发生一次替换（8~11 和 0~3 在同一组），后来再访问就会发生 miss 而 A 会命中，所以 2 正确。综上 2 正确。

3. A。取 `x=INT_MIN` 即可。B 正确，因为 `~x+~y+1=~x+~y+2-1=-(x+y)-1=~(x+y)`。讨论 `x, y` 的大小关系立刻知道 C 是正确的。由于 >> 是向下舍入的，所以 D 正确。

4. B。送分题，`float` 不能精确表示绝对值 $2^{24}+1$ 及以上的整数。

5. D。送分题，RISC 中寄存器一般更多。

6. D。指令长度为 6 字节，内存地址需要用基址加上偏移计算。

7. D。只有编译时打桩需要访问源代码，A 错误。B，可执行目标文件中仍然会有 `ABS` 节（文件名）。程序的入口是 `_start`，后者跳到题干所说的位置，C 错误。D 是正确的（书 P489 原话），但是 `-fPIC`（大写）选项更好。

8. B。请注意编译器看不到两边 `x` 类型的不同，因此不会报错。链接器会将 `x` 初始化为 0，在 `f1.c` 中编译器生成的代码视其为浮点而在另一边视其为整数，所以自增结果是一个很小的浮点数，转为整数是 0。

9. B。注意 `printf` 是行缓冲，缓冲区会被复制，最后都一次性输出。另外 `fork() && fork()` 是父进程再 `fork()`，子进程不 `fork()`，而 `fork() || fork()` 则是子进程再 `fork()`，父进程不 `fork()`。故实质上一次产生 3 个进程，所以答案为 $3^2 \times 2 = 18$。（如果不考虑缓冲区则要去掉第二次产生的 6 个净拷贝，答案应为 12。）

10. C。送分题，外部 I/O 会触发中断，CPU 执行完当前指令之后可能会去处理该中断。

11. A。送分题，属于书上原话。

12. D。缺页是故障，这种故障需要重新执行引发故障的指令；系统调用是陷阱。二者都是同步的异常，而且处理完毕是从内核态回到用户态，会处理信号。

13. B。A 显然是正确的。B 我们实验过，一般这种读取对于用户来说是允许的，参看书 P511。C 一般同学不会实验过，需要推理一下，因为参数列表和环境变量实际上是程序自己处理的，而且 shell lab 中重定向是 shell 帮忙完成的，故推测 C 正确。D 是陆老师著名的“灵魂出窍”例子，即 `printf` 在信号处理程序中可能导致死锁。

14. B。根据 Core i7 的用法，每张页表恰好有一页，每个页表项 8 字节，所以每个 `VPN` 都是 $14 - 3 = 11$ 位，于是四级 `VPN` 和 `VPO` 一共 58 位。

15. B。第二句话表明题目中的内存是按字节寻址的。首先计算 `VPO` 为 10 位，每个 `VPN` 都是 8 位，恰好是 34 位虚拟地址。最好情况是 1 MB 虚拟地址映射到连续的页，即占有 1024 页，所以需要 1024 个三级页表条目（占有 4 页），1 个二级页表条目（占 1 页）和 1 个一级页表条目（占 1 页），即需要 6 KB。最坏情况是每字节都恰好映射到不同的页面，也就是占有 1024×1024 个不同的页面（可以实现）。这些页面可以用尽全部 256×256 张三级页表（比如每 16 页用一个三级页表），所以需要 $256 \times 256 + 256 + 1 = 65793$ 页，即 65 973 KB。

16. B。和物理地址无关，`VPO` 为 13 位，`VPN` 均为 10 位，所以需要三级页表。

17. D。根据题目的描述就可以推断是 A 位和 D 位。

18. A。未解之谜。如果将共享区域理解为 `MAP_SHARED` 则 D 错误；否则共享库也是一种共享区域，会发生 COW。如果参照 Wikipedia 上的解说，COW 是一种技术，用户可以自行实现，那么对等线程之间也可能发生 COW（例如 `string y=x` 之后修改 `y`，C++ 中用 COW 实现）。A 可能是更好的选择。

19. A。其他段中都可能有指针，其引用的内存需要管理。代码段中则没有这种问题。

20. C。服务器端的进程通过 `accept` 来建立已连接套接字。D 应阻塞在 `read`，参看课程投影片。

21. D。送分题，根据这个命令的字面意思也能看出获得的是主机名，参数 `-i` 表示返回点分十进制的 IP 地址。网络字节顺序和大端法是一回事。

22. C。a) 是共享的（在数据段），b) 也是共享的（栈是私有的），c) 是共享的，属于书上原话，或者从同一进程的概念中读解出，d) 是特别的上下文，是私有的。所以有 3 个。

23. C。此题意义不大。主要的困难在于最终的正确顺序并不是确定的一个全序，所以需要找一个“最优”的全序。大概估计一下，L_2, U_2, S_2 和 L_4, U_4, S_4 之间不能有重叠，L_1, U_1, S_1 和 L_3, U_3, S_3 之间亦然，所以可能是

$$L_2, L_1, U_1, S_1, L_3, U_2, S_2, L_4, U_4, U_3, S_4, S_3.$$

数出逆序数为 12。实际上，合法的全序只有十几种，可以枚举验证答案正确性。

24. B。注意主线程如果调用诸如 `pthread_exit` 的函数退出，是不会引发其他线程退出的，所以 A 为一错误描述。但是如果主线程调用了 `exit`（或者 `return`），则整个进程都会结束，自然其他线程也结束。

25. B。首先 1 和 2 分别获得 a 和 c 的锁，然后 2 再获得 b 的锁，1 和 3 分别等 d 的锁和 a 的锁。此时如果 2 释放 c 和 b 的锁，1 和 3 无论如何都将出现互相等待（a、d）的情况，因此发生了死锁。

解答题一

1. 10 ns。送分题。

2. 60。题目中为一个 8 重循环，具体执行了什么不重要。每次循环出现数据冒险一次，故需要 7 个周期。最后分支预测错误有 2 个周期的惩罚，但是已经算在 4 个 trailing cycle 中了，所以需要 60 个周期。

3. 47；5；96。两个缓存都是 5 位偏移，10 位 tag（这里可能应该认为地址空间是 16 位）。因此指令缓存只会不命中一次，剩下 47 次都是命中的。数据缓存每块可以放下 4 个长整数；但注意起始位置不是 32 字节对齐的，所以会有 3 次不命中，剩下 5 次都是命中的。不命中一共额外造成了 9×4 周期的惩罚，所以需要 96 周期。

解答题二

1. 答案如下：

符号	`.symtab` 有条目?	符号类型	定义符号的模块	节
`buf`	有	外部	`m.o`	`.data`
`bufp0`	有	全局	`swap.o`	`COMMON`
`count`	有	局部	`swap.o`	`.bss`
`func`	有	全局	`swap.o`	`.text`
`temp`	无	/	/	/

送分题。注意外部符号要填定义的模块中的节。

2. `0x00002020, 0x000000be, 0x0000a000, 0x0000900d, 0x00decade`。

   1 处，`ADDR=1000, offset=a, addend=-4`，结果为 `302e-1000-a-4=0x00002020`。3 处类似，结果为 `10e4-1000-22-4=0x000000be`。4 处结合汇编代码知道目标符号是 `buf` 的地址移入，所以是 `0x0000a000`（绝对重定位）。9 处对应 `bufp1`，下一条指令地址是知道的，可以不用重定位算法，所以是 `a250-1243=0x0000900d`。11 处对应 `count`，下一条指令地址也是知道的，所以是 `dedd38-125a=0x00decade`。具体填要注意小端法和立即数格式。

解答题三 4 行，内容为

```
a b c 0
a b a 1
a a b 1
a a a 2
```

此题直接画进程树，在结点旁边标注父进程和子进程，并写出状态变化就可以。考点为每次父进程 `open` 时，指针都重新回到文件开头；`dup` 则保持文件指针为同一个；`count` 四个进程相互独立。注意（相对的）父进程总是用 `wait(NULL)` 等待子进程，所以输出的顺序一定是子子、子父、父子、父父。

解答题四 答案如下：

n	缺页次数	dTLB 失效次数					
		1	2	3	4	5	6
8	1	1	1	1	1	1	1
16	2	2	2	2	2	2	2
64	24	24	24	24	24	24	24
512	1536	266436157	268661500	252182591	252182591	235241284	521528

题目看起来非常复杂，实际上分析一下就可以发现没有多少耦合，数据也凑得比较好，很容易拿到大部分分数（但是此题意义不大，而且没有考察虚拟内存的精华）。

不难看出，缺页次数只和矩阵占的页数有关（无论怎么访问），而 TLB miss 等价于为一个缓存块 4 KB，64 行的全相联高速缓存的 miss 次数。这样一来，数据占据页面的个数，也就是缺页次数为 $\lceil 3n^2 \times 8/2^{12} \rceil$，分别为 1、2、24 和 1536。由于 TLB 是全相联的，只有满了之后才会发生替换，所以前三行的答案已经全部得出（缺页次数 = TLB miss 次数 = 页面个数）。

$n = 512$ 时的 TLB 行为模拟比较简单，但仍然有一定的工作量，这 1536 个页面每页恰好对应矩阵的一行，所以 TLB miss 的次数就转变为对行访问的分析，由于全相联缓存的行可以放在任何位置，所以简单替换就可以，通过计算可以得出答案。

解答题五

1. 死锁。可以看出，这实际上就是哲学家就餐问题，如果所有的同学同时拿起左边那个同学手机 `P(UP(i))`，就发生死锁。

2. 分别填 `0` 和 `0`。观察代码知道 Dave 的方法每次只有一位同学去找洞主，也就是说 26 个同学轮流去验证。由此知道只需要一个互斥锁即可。

3. ①②②①。根据 1 问的例子可以知道依赖关系成环是最大的问题。因此我们可以让除一位同学之外的同学正常工作，而那位同学则反过来工作，不难证明这个体系无死锁且无竞争（事实上，还有一种方法，即要求单号同学先拿左边的手机，而双号先拿右边的手机）。

4. 分别填 `1`、`0`、`<`、③、②、`<=`。根据代码容易看出 `mutex` 保护结构体，`zero` 是一个“等待信号量”，所以计数降到 0 以下时需要等待，由此发现其初值应该是 0。请注意 ③、② 不能反，否则会造成死锁。

2019 年

选择题

1. A。唯有这一项会先算 `d+a`，可能发生不结合的问题。

2. B。数据冒险一般是指当前指令没有写回，而下几条指令在流水线就已经要用到的情况。

3. A。除法向零取整，移位向下取整；2 相当于给负数添加一个修正，3 相当于转成正数后模拟除法行为。所以 `a=-2016, b=-2016, c=-2016`。

4. C。CISC 变长指令，有不少指令的长度是要比 RISC 短的。当代手机常常采用 ARM 架构，这是 RISC；实际上对能耗要求高或者结构简单的设备一般都用 RISC。

5. B。DRAM 常常组织成一个矩阵族，整行访问时效率比较高，B 错误。SSD 设备擦写的时间会比读取要高一个数量级。一般来说，SRAM 使用比较多的晶体管，而 DRAM 晶体管很少，主要基于电容，SSD 是闪存，晶体管也不会很多，因此后二者的存储密度相对高。

6. C。实际上对符号谈重定位是不正确的，应当对引用谈重定位，例如全局变量如果初始化为值且不使用，则不需要重定位。需要留意，如果是同一 C 语言源文件中定义的函数则一般不需要重定位。

7. C。除法错误是不可恢复故障，是同步的；I/O 中断是异步的，一般会执行完当前指令，再去处理，返回就不需要再处理了；缺页异常当然需要重新执行遇到问题的访存指令；时间片到中断属于时钟中断，属于异步异常。

8. 未解之谜，CD。并行一定并发。

9. C。`fork` 返回两次，`setjmp` 可返回多次，`longjmp` 和 `execve` 不返回。

10. A。高速缓存是物理寻址的，所以一定不用刷新。用户态和内核态的转换不会改变内存映射，而上下文切换会改变虚拟内存到物理地址的对应关系，所以 TLB 在上下文切换时要刷新。

11. C。VPO 为 11 位，页表均占一页，则 VPN 都为 8 位，所以映射满 48 位地址空间至少需要 5 级页表。

12. D。直接模拟就可以。

13. B。最好情况是高速缓存和 TLB 都命中，不需要访问内存；最坏情况是都不命中，需要访问四级页表项以及主存中的数据，一共 5 次。

14. B。参看课程投影片，迭代服务器的第二个客户端会在 read 阻塞。

15. C。网络字节序规定为大端序，在不同的主机中可能需要转换。

16. A。文件名和参数应该用 ? 分隔。

17. D。2 和 4 都是很明显的，6 仍然是陆老师著名的“灵魂出窍”例子，即 printf 在信号处理程序中可能导致死锁（加锁后打断，在信号处理程序再调用即死锁，因为需要信号处理程序返回后才能解锁）。

18. B。保持跨越多个调用的状态的函数不是线程安全的，而 C 项是隐式可重入函数，如果合理使用就是线程安全的。

19. C。否则会在对等线程的赋值和主线程的 accept 之间引入竞争，如果赋值在 accept 之后完成，描述符值就错了。

20. D。D 是超纲选项，sem 型指令会阻塞整个进程，而 pthread_mutex 型指令只会阻塞相应线程，后者的效率更高。A、B、C 都是容易看出正确的，所以可以用排除法做出。

解答题一

1. 8，送分题。

2. 48 周期，即 8×6。

3. 50 周期。因为每个缓存块可存 8 个长整数，但起始地址不是 64 字节对齐，所以会发生 2 次 miss。

4. 42 周期。因为只有前两个 addq 可以一次执行完，这样能节省 8 周期的时间。

5. 26 周期。指令顺序可重新安排为

```
.L2:
    movq    (%rdi), %rbx
    movq    8(%rdi), %rdx
    addq    %rbx, %rax
    addq    $16, %rdi
    addq    $2, %rcx
    addq    %rdx, %rax
```

```
        cmpq    $8, %rcx
        jl      .L2
```

此时每次循环只需要 6 周期，一共 4 次循环，即 24 周期，补上两次 miss 的惩罚得到 26 周期。

解答题二

1. 答案如下：

符号	.symtab 有条目?	符号类型	节	强弱符号
bufp1	有	全局	.data	强
buf	有	外部	UNDEF	弱
bufp0	有	全局	COMMON	弱
temp	无	/	/	/
count	有	局部	.bss	非强非弱

   送分题。注意外部符号要填 foo.o 中的节，extern 的变量严格说不能区分强弱，但是如果要问则回答“弱”。

2. 未知；bufp1.foo。弱弱选弱，强弱选强。

3. 0x000000be, 0x0000a000, 0x0000900d, 0x0000babe。

   1 处，ADDR=f28, offset=16, addend=-4，结果为 1000-f28-16-4=0x000000be。2 处结合汇编代码知道目标符号是 buf 的地址移入，所以是 0x0000a000（绝对重定位）。6 处对应 bufp1，下一条指令地址是知道的，可以不用重定位算法，所以是 a050-1043=0x0000900d。9 处对应 count，下一条指令地址也是知道的，所以是 cb24-1066=0x0000babe。具体填要注意小端法和立即数格式。

解答题三

1. 3, 4。注意 0, 1, 2 是标准 I/O 流占用的。

2. 4, 5。因为 fd2 已经被关闭，其描述符可复用；另外 fd3, fd4 虽然指针相同，但是描述符的号码是不同的。

3. 0, 1, 2, 3。父进程等待子进程结束，所以子进程先输出，另外父子进程的 count 当然是独立的。

4. 2019PKUCS。对 f1.txt 的写有三处，其中 20、21 行是独立的，得到 2019PKU，最后只有子进程再次写文件，此时 fd1 的指针在最后，即 2019PKUCS。

5. PKUICS2019。鉴于 dup 产生 fd4，而父子进程的文件描述符指向同样的打开文件表条目，所以对 f2.txt 的写本质上只有一个指针，因为父进程等待子进程结束，所以先输出 PKUICS 再输出 2019。

解答题四 此题我们课上讲过了，原题有几个数据错误，条件也不够。摘取正确题目的分析如下：

1. 4096；6；10。送分题。

2. `0xEAB450D, 0x67F000, 0xAA3000, 0xC3F000`。

   考虑父进程执行 `*b` 的写时的读写问题。在解析 `*b` 时，第一步当然是找出一级页表项，因此从地址 `0x67F0E8` 读出 `0x80AA32C4` 就是从一级页表中，物理地址 `0x67F0E8` 的位置，读出一级页表项的内容 `0x80AA32C4`。根据题目对 PTE 结构的描述，我们知道 `0xAA3` 是二级页表页的页号，即 `0xAA3000` 是二级页表的起始物理地址。由于一级页表是 4 KB 对齐的，所以一级页表项的物理地址就是 `0x67F000`。

   为什么会写两次？显然，我们知道 COW 机制在起作用，这段内存首先要被复制。`0x80C3F110` 这个数字当然是精心设计的，必然暗藏玄机。首先，一次写肯定是写 `*b`，那么另一次写呢？注意这次写也是父进程，因此和子进程无关。由于 COW 后内存映射需要修改，结合后面也有类似于 `0xC3F...` 的物理地址，我们可以推定这个实际上就是 COW 后，`b` 对应的二级页表项。这样我们马上知道，其二级页表项的物理地址在 `0xAA3AD0`，而 `b` 指向的真实物理地址是 `0xC3F50D`（页号 `0xC3F`）。这时我们就可以计算出 `b` 的值（即虚拟地址）了。一级页表项的偏移是 `(0x67F0E8-0x67F000)/4=0x3A`，二级页表项的偏移是 `(0xAA3AD0-0xAA3000)/4=0x2B4`，`VPO=0x50D`，所以虚拟地址是它们的拼接。注意不是直接将 16 进制拼接，虚拟页号为 `11 1010 10 1011 0100=0xEAB4`，所以 `b=0xEAB450D`。

   此题页表项中的最低 12 位是何含义，属于未解之谜。

解答题五

1. `bind, connect, accept, writen, readlineb, readlineb, writen`。容易读解，但注意 RIO 包的 API 名称不能记错。

2. `bind, listp, listenfd`。

解答题六

1. `7, 14, 7, 2, 0`。这时两个读者都在正常读，因为是读者优先，所以写者在 14 行阻塞等待，写锁的值是 `0`。

2. `14, 7, 3, 0`。完全同理。

3. 会出现竞争的错误。例如两位读者同时进入，由于 `readcnt` 的写不是原子的，可能导致进入后 `readcnt=1`，当一位读者离开后，写者就错误进入了。