期末往年题勘误、详解

by Arthals

整理自我的树洞 5833467 与 piazza。感谢树洞同学和 zzs 助教的帮助。

1 2015

1.1 第一题

第 17 问，废题，现代 Web 中 GET/POST/PUT 等都可以用于获得动态内容。

1.2 第二题

第（8）空，应该填写十进制 40，而不是 0x40。

1.3 第三题

第 4 问，应该在 AB 之间也插入一个寄存器。

1.4 第四题

如果你认为 a[0] 应当是 *(0xdeadbeefcafebffe)，那么你的理解有误。

你可以使用 union 结构体来验证这个重复定义的全局变量。

1.5 第六题

第一问题出错了，TLBI 完全没考虑，你当 TLBI 不存在去寻址；第二问解答说的挺好的，第三问解答说的也还行，主要是你要知道一个 PTE 是 4B，然后 0x27*4 得到 0x9C，最顶上那个 C 是因为这是虚存；最后一小问和 17 年虚存一样，修改了页表项需要让上一级缓存 (也就是 TLB) 失效，否则下次访问命中 TLB 会从中取出已经过期的值

所谓大页，就是你可以理解为我不需要更低一级的页表了，我保证我的上一级页表每一项都可以映射到连续的大页上。或者你可以理解为低一级页表中的所有项目都是连续的，那么这个低一级页表当然就不再需要了。

1.6 第八题

答案有误，两个 mutex 信号量的初值应当都为 1，且 PB 应当如下：

```
PB() {
    while(1) {
        P(&full1); // full1--
        P(&mutex1);
        // 从 Buff1 中取出一个记录
        // V 操作可以挪到最后，因为此时持有 mutex1 保证不会有 PA 来竞争
        V(&mutex1);
        V(&empty1) // empty1++
        // ---- ///
        P(&empty2); // empty2--
```

```
            P(&mutex2);
            // 将记录放入 Buff2
            // 这里顺序无所谓，V 操作不阻塞
            V(&mutex2);
            V(&full2); // full2++
        }
    }
```

可以看做是两个队列问题，因为题目说明了只有 PA PB PC 三个线程，不存在 PB 和 PB 之间的竞争。

2 2016

2.1 第一题

第 2 问，隐藏位就是规格化数里大伙都有的 1，这里的意思是你要以这个 1 为基准，上下加减 M 表示数，再乘以 E 阶码位。考虑正常的话，尾数位的映射范围 1~2，现在就是 1/2~2，相当于很多数的阶码位、尾数位其实可以有了两种选择（即两种表示方式），不像 IEEE 那样充分利用了。比如，D 的指数位是 7，-bias 之后是 4，所以指数就是 16。然后补码那块，你看现在是 10101，那么有-x=~x+1，你假装他是一个整数，取非搞出来 01010，加一搞出来 01011，这个是 11，也就是 -x，那么 x 就是 -11，你再考虑这里的末位代表 2^-4（注意有一位符号位，即第一位 1 不算成 1/2，第二位 1 才是），所以尾数就是 1 - 11*(2^-4)，你再拿这个乘以 16，应该就对了，我觉得这里挺扯淡的倒是（

第 10 问，A 是不会有信号的。在网络那一章中，只有链接异常断开会导致 SIGPIPE。A 中描述的“新连接到达监听端口”不会导致操作系统发送信号给进程，而是通过网络编程接口（如 POSIX 的 socket API）提醒应用程序有新的连接。这通常是通过 I/O 多路复用（如 select、poll、epoll 等系统调用）来实现的，其中应用程序会监控一个或多个网络端口的状态，当有新连接到来时，操作系统会唤醒正在等待的程序，而不是发送信号。

第 13 问，这里的所有符号都对应 bit 数。

第 20 问，关键点在于，printf 的时候因为这个 j 没有 volatile 定义，所以你考虑汇编的时候，这里肯定就是一个寄存器了，上面读完是啥就是啥，而寄存器是个线程上下文里的东西，所以你 V(&s) 之后到 printf 之间哪怕被切走，切回来的时候也是会恢复的，所以不会因为在别的线程里加了 j，我这里 printf 就会知道哦原来这个 j 变了，而是依旧恢复成切走的时候的 j（寄存器版本）。所以这题前三个都是可能的，只有 D，因为线程锁保证了这个 j 一定会加到 3，而对应的线程的打印一定能打出 3，所以 D 是错的。

2.2 第六题

第 2 问，权限解读如下：

- r = read
- w = write
- x = execute
- s = shared
- p = private (copy on write)

第 3.1 问，D 选项，COW 写时复制和页表无关。

3 2018

3.1 第一题

第 3 问，对于 C 选项，直接考虑冷不命中次数就知道是对的；对于 B 选项，如果数组刚好比 cache 总大小大一点点，LRU 会不断产生 miss 从而不如其他算法。

第 6 问，执行过程中没有发生从未打开的描述符进行复制，所以错。实际测试：可以向一个未打开的描述符进行复制、从一个未打开的描述符复制（即未打开的描述符是 dup2 的第一个参数）都是合法的（但是后者一旦开始写就会报错）。

第 7 问，D 选项没说是 LIFO 还是按照地址排序，所以不确定。

第 11 问，D 选项。题目里的意思应该是，用 socket 创建的 fd 在连接后是可以用 unix 标准 io 读写的。

第 13 问，A 选项，可能使用了全局变量。

3.2 第二题

明显答案错了，少了 CD,FG 两个插入，这种题的普遍做法是，找到一条边最少的，逐次枚举，一个技巧是任何一条通路上的寄存器数量是相同的。

3.3 第四题

为什么 PartB 产生了 +-+--+..这样的结果?

父子进程是分叉后单独打开的文件，由于并发，所以父进程给子进程发信号的时候子进程可能还没把 fd1 改掉，所以打印第一个+，然后：

```
-
++
 --
  +++
   ----
     ++++
...
=======
+-+--+---
```

3.4 第五题

第 2 问。

此问详细解答如下：

MMU 的 TLB 没有命中，找到一级页表项 0xC3F50D，由于一级页表 4KB 对齐，所以这个地址的低 12 位 0x50D 是一级页表项索引偏移，所以一级页表基址是 0x67F000；读出来 0x80AA32C4，8 代表最高位为 1，也就是有效，说明二级页表已经缓存在主存内，而读出来的 0x2C4 是一级页表项中存的对于二级页表的控制信息。所以二级页表基址为 0xAA3000，这里也可以由二级页表是 1024 个 PTE，说明二级页表项地址的最低至少 10 位为权限位，上取整到 4 的倍数 12，得到最低的 0x2C4 为控制信息。然后后两次写入，第一次为真正的写入物理内存，第二次为修改二级页表项目。所以根据物理内存地址 0xC3F50D，由于页大小是 4KB，所以低 12 位为 VPO/PPO，所以物理页基址为

0xC3F000；然后因为修改了物理页面，所以要修改对应的二级页表条目，标识 A 位引用与 D 位修改，所以写入地址 0xAA3AD0 的 0xAA3000 是二级页表基址，0xAD0 是二级页表项索引偏移，之所以写的还是 0x80C3F110，纯属巧合，说明运气很好，最高的 8 是有效位，0xC3F 是物理页表基址，0x110 是权限控制信息。然后根据如上过程，b 是指针，它的值其实就是这个过程中的虚拟地址，VPO 是 0x50D，VPN2 是二级页表偏移索引 0xAD0/4=0x2B4，注意只要低 10 位 1010110100，VPN1 是一级页表偏移索引 0x0E8/4 = 0x3A，注意这里每个 PTE 仍是 4B，这和页表项目数量无关，只和地址空间位数有关，也说明一级页表没有满其实，仍然注意只要低 6 位，111010，然后拼起来得到 1110101010110100，即 0xEAB4，这就是 VPN（总的），所以虚拟地址是 VPN+VPO = 0xEAB4AD0。

完成写之后该项 TLB 内容为?

```
0x00ba4227 & (0xFFFFFFFF - ((1 << 12) - 1)) | 0x27 | (1 << 6) == 0x00ba4067
              ^--低12位取0                     ^       ^
                                               权限rw   |
                                                       Dirty bit
```

第 3 问，pid 和 cnt 在一个页里，你考虑第一次分叉，分叉时，子进程写时复制改 1 页；二次分叉，两个新的子进程写时复制都要改 1 页...所以是 1+2+4+8=15 次

3.5 第七题

第 4 问，信号量也可以等于 2，本质上是大于等于 2 的时候这个锁就失效了，相当于只有剩下的一个互斥锁（

3.6 第八题

1：显然这是是代码段之上的数据段，必然可读可写，但是因为 COW 机制，所以是 private 的，这样以后才会触发保护故障，所以 rw-p

2：动态链接器本身也是共享文件，所以是 ld.so

3：这里是打开的文件头，题目说 MAP_SHARED 就是暗示的这个，你看他的权限位是 rw-s

4：显然是栈

5：单进程的实际可用虚拟地址空间为 6500KB，所以多进程的时候就是 10*6MB=60MB，上取整到 A

6：单进程实际使用虚拟地址空间 RSS 为 1600KB，不同进程因为 cow 写时复制机制的存在，所以代码段和共享库基本是共享的，只有自己的堆和栈是私有的页，而 heap 和 stack 的 size 都为 132KB，所以十个进程就是差不多 260KB*10= 2MB，再加上单进程固有的 RSS 1600KB≈2MB，所以整体 RSS 差不多就是 4MB，考虑到铁铁不可能选择 2MB 版本的，肯定是上取整到 6MB，选 C

7：多进程共享空间基本上就是 6 中说的 6MB，除给每个进程之后就是 600KB，所以选 D

8：多线程的时候，虽然共用虚拟地址空间，但是对于每个线程来说，他自己的又映射了一层虚拟内存，所以 VSS 总可用等价于多进程的时候，也就是 5 中的情况（这个不太确定）

9：RSS 显然因为多线程共用虚拟地址空间，基本等价于单个进程的 RSS，也就是 1600KB。所以选择 D

4 2019

4.1 第一题

第 14 问，为什么阻塞在 `read` 而不是 `connect`？请注意，`connect` 并不是在 `accept` 之后才返回的。

连接成功意味着客户端发起的连接请求已经被服务端接受，此时 TCP 三次握手过程已经完成。在编程层面，客户端的 connect 函数会在连接成功时返回 0。所以只要服务端使用了 `listen`、当前连接数没超过 `listen` 的 backlog、网络畅通，那么客户端的 connect 就一定会返回 0。

服务端的 accept 函数用于**从监听队列中接受一个新的连接请求**。一旦客户端的 connect 请求到达服务器，服务器的操作系统会处理这个连接请求，完成三次握手过程，并将这个连接放入等待服务端 accept 的队列中。当服务器调用 accept 函数时，它会返回一个新的套接字描述符，这个描述符代表了与特定客户端的连接。

因此，从客户端的角度看，connect 返回 0 就表示连接成功。而服务端需要调用 accept 来正式接受这个连接，然后才能开始通过这个新的套接字与客户端进行通信。

4.2 第五题

第（1）问，题目有错，虚拟地址大小为 28 位。

4.3 第七题

为什么读写问题要一直 while？原因是要通过两个 while 每次循环之间的竞争来模拟读者一直读写者一直写。

5 2020

5.1 第一题

第 7 问，关于 D 选项，可能有些同学读书认为此时 ABS 节应当也不存在，但是书上说的是静态链接后的情况。你可以认为，在默认的编译选项下得到的可执行文件是会有这些伪节的。或者参考我的理解，我觉得书上说的是完全连接的可执行文件，你搞出来的是部分链接的可执行文件，其中还有一些共享库的条目？

第 18 问，感觉 AD 都有问题。

第 25 问，选项除 D 外均会死锁。因为线程 1 和线程 3 的 a，d 锁的获得顺序刚好相反。

5.2 第三题

④ 处怀疑是答案给错了，根据汇编看似乎应该是 count 才对，也就是 0x00dedd38（count 的地址，初始化置零）

5.3 第六题

第 1 问：首先你需要知道为什么会死锁？要是所有人都同时获得了 UP 的锁，那么 DOWN 的就一个都获取不到，就寄了。

然后如何保证不会发生这种情况？就需要规定获得锁的顺序，必须先获得大的，再获得小的（你可以自己思考一下正确性的原因）。所以答案是 1221 或者 2112.

第 2 问：实际上是用 value 存储你模拟的 sem 的值，用 mutex 保护 value，用 zero 实现等待。如果 value<0 了那么 P 操作就必须等待，所以 C 应该是<.

对于 F，如果 value 本来就<0，那么就需要唤醒一个 P 阻塞的线程。自增后<条件变为<=，故也填写<=.

具体的条件判断可以使用边界值判断法，就是考虑 value 为-1,0,1 三种特殊值就可以了。

6 2021

6.1 第三题

Part A，seed。老师想考的是非全局的局部静态变量（函数内的 static）会被重命名，加一个.x 的后缀，所以是否，后来讨论了之后还是认为有。

Part B，`<main+0x1f>` 处的偏移量计算：

一个简单的计算是，设 `pnt` 的地址是 `addr(pnt)`，那么我们要求的相当于是 `addr(pnt) - addr(.bss)`，你需要深入理解这个相对重定位的含义，为什么要有一个 addent？你考虑读完 1c 这条指令（即其进入 D 译码阶段时），PC 已经到下一条指令（23）处了，所以要加一个 addent = -4 字节来抵消这个偏移（也即 23 - 1f），使得读入“指针”的位置是正确的。

于是，有：addr(.bss) + 0xc = addr(pnt) - 4，所以偏移量是 0xc + 4 = 0x10。

Part C，既然这个函数是可以被换成自己指定的代码段的，那么肯定运行在用户态下。否则就构成越权了。

6.2 第六题

2.b，如果是 if，有：

```
P(&mutex);
if (items == BUF_SIZE) {
    num_waiting_producer++;
    V(&mutex);
    P(&sem_waiting_producer);
    P(&mutex);
}
```

本质是第 30 行 P(&sem_waiting_producer); 和 31 行 P(&mutex); 之间可能被其他进程捷足先登。

而不是你出了 while 循环之后的问题，出了 while 循环时你已经持有锁 mutex 了。

7 2022

7.1 第三题

第 3 问，interp（最前面的一节）放路径，也就是 ascii 码，解读出来就是动态链接器的路径。

第 4 问，lt 其实给出了 plt0 了，就是 printf 前面那个，那个 push 和 jmp 明显是 plt0，所以是 1