Lab2: 消息机制与设备管理

徐栋

南京大学，计算机科学技术系

学号：**121220312**，邮箱：**dc.swind@gmail.com**

时间：**2014.6.28**

摘要：

消息机制即使用消息通信，发信进程把 **Message** 结构体投递到邮箱中，就可以立即去做其他的事情；收件进程会访问他的信箱，每当发现有 **Message** 时，就取走 **Message**，否则则阻塞自己。消息机制的这些实现过程对我们是透明的，另外消息机制除了通信功能还提供了自然的同步手段。在 **Lab2** 中，我们使用 **Lab1** 的框架，在分时多线程的基础上，通过消息来进行进程之间的通信，通过消息机制来实现临界区的管理。

关键字：

消息机制；分时多线程；操作系统；**send**；**receive**；临界区管理；

1 实验介绍

在实现了分时多线程后，我们的“操作系统”已经初具规模，我们可以创建多个线程并可以看到其交替运行。这时我们自然会想再多做些什么，让我们的操作系统更加完善，更加像一个操作系统 (具有更多的功能)。我们自然而然的想到了创建的多线程间的通信。那么本实验的目的就是实现消息机制，提供消息传递功能并在此基础上实现临界区的管理。而在熟悉操作系统中断的基础上，实现这一切并不困难。

2 实验目标

1. 利用 **lock**、**unlock**、**sleep**、**wakeup** 等函数实现 **PV** 信号量操作。
2. 因为 **Lab1** 中没有使用 **ListHead** 结构，所以需要熟悉并在 **Lab2** 中加以使用。
3. 通过对中断机制的了解，在此基础上结合 **PV** 实现消息机制。
4. 通过测试验证消息机制的正确性。
5. 加入并调试提供的时钟和终端驱动程序，使其可以正确运行。
6. 培养阅读文献、搜索资料、独自学习的能力。

3 涉及知识

3.1 信号量

信号量是一种变量类型，用一个结构型数据结构表示，有两个分量：一个是信号量的值，另一个是信号量队列指针。除了初始赋值之外，信号量仅能由同步原语 **P** 和 **V** 对其进行操作，**PV** 操作的不可分割性确保执行时的原子性及信号量值的完整性。

3.2 消息机制

消息机制是一种间接的通信方式，进程之间发送或接收消息通过一个共享的数据结构——信箱进行，消息可以理解为信件，每个信箱都有唯一的标识符。当两个以上的进程拥有共享信箱时，他们就能进行通信。与信号量类似，其通过 **send**、**receive** 两个原语进行消息发送与接收。

4 设计思路及具体实现

4.1 一些必要的准备及思路说明

首先 **PV** 操作需要在信号量上操作，而信号量有一个队列存储阻塞在这个信号量上的线程。所以在 **Thread** 结构体中加入了 **ListHead semq;** 同时为了方便，在 **Thread** 结构体中加入了 **Semaphore sem;** 当 **PV** 操作时，直接操作 **Thread** 的 **sem**，在 **PV** 中将 **semq** 挂在信号量的队列中。（同时我也尝试直接使用 **sleep** 和 **wakeup** 控制 **send**、**receive** 的唤醒与阻塞）同时，在设计 **send**、**receive** 时，每个线程可能接收到多封 **Mail**，这些 **Mail** 的读取是要严格按照发送顺序进行的，而公共邮箱并不提供记录这个顺序的功能，所以在 **Thread** 结构体中，加入了 **ListHead mailq;** 维护一个收到邮件顺序的队列。

4.2 PV

实验已提供，不再说明。

4.3 send

我申请了一段空间作为公共邮箱来进行邮件的存储转发。**send** 与 **receive** 一样最基本的是要 **lock();** 关中断的情况下执行，首先进行判断是否存在可用的信箱。不可以返回错误，可以则继续执行。然后判断是否存在收件人，如果不存在直接 **return**，即相当于不处理无效发送。然后找到空信箱进行分配，将 **message** 拷贝，然后将信箱地址加到收件线程的信件链表中。这里是为了使得收件人收取的信件顺序是正确的。最后对收件人进行 **wakeup** 操作，如果收件人当前正在等待这信件而阻塞，则唤醒它。

4.4 receive

receive 同样在 **lock();** 关中断下执行，首先对信件链表进行顺序查找，找到想要收取的信件，将信件取出，归还信箱地址，将信箱从信件链表中删除。如果链表为空或者未找到信件，则需要执行 **sleep** 操作阻塞当前进程。

4.5 终端添加

4.5.1 关于终端的一些实现说明

很遗憾的，终端我并未完整的实现。最终我的终端是无法显示画面的，但是可以输入字符，光标可移动，可以在 shell 中输出输入的字符，可以进行 4 个 tty 的切换。但是会出现意外错误而中断。这里我的考虑是前面加入终端时存在问题而导致的。所以暂时也无法解决。在调试终端的过程中，我尝试过许多方法：

1. 最先考虑的是 send 和 receive 的实现有问题，而在检查中的确发现了一些小问题，比如说在 send 中我进行 V 操作唤醒进程，而在 receive 中如果没有找到信件则执行 P 操作阻塞。这样会出现先 V 几次，但找不到信件仍然无法立即阻塞的情况，虽然 receive 的进程仍然在死循环中，但是没有立即阻塞我依然觉得有些不妥，所以我又用 sleep 和 wakeup 实现了一下。这两种实现的结果是一致的，所以对解决终端问题并未起到帮助。

2. 将 message 的 src 在 send 中赋值 or 在 send 前赋值，这里我主要考虑的是在中断是否会发生一些和设计思路不同的事件，在 send 前赋值不再赘述，其正确性应该是没有异议的，但是在中断中赋值，我的考虑是设置一个全局变量，在 irq_handle 的开始赋值为 0，在 irq_handle 的结束赋值为 1，用来标识是否处于中断中，正确性我并未深究，因为在换成 send 前赋值后依然没有调出终端，所以我暂且搁置了这个问题。

3.dev_rw 中的 m 是 Message 还是 DevMessage，根据 JYY 学长提供的说明，是需要将 m 修改为 Message 结构体防止越界，但是我在修改后，ttyd 中无法正确的接收到 dev_id，所以我又将其调回 DevMessage。但是仍未解决问题。

4. 其中许多未定义的函数我给予了实现，仍未解决问题。

5. 其中 void update_sched(void); 函数中 need_sched 变量在整个工程中并未使用到，只是赋值为 True，不明白这里是否存在问题。

4.5.2 中断函数添加

即 add_irq_handle() 的实现，主要的功能是将指定函数添加到一个中断队列函数中，当发生指定中断时，依次执行这些函数。

4.5.3 memcpy、memset、strcmp

void memcpy(void *dest, const void *src, size_t count); 复制 src 的前 count 个字节到 dest 中。
void memset(void *src, uint8_t data, size_t count); 将 src 的前 count 个字节初始化为 data。
int strcmp(const char* str1, const char* str2); 将 str1 和 str2 进行逐字节比较，返回比较结果。
上述三个函数的实现均是使用 while 循环逐字节处理。

4.5.4 pa to va

进行物理地址到逻辑地址的转换，因为 bootload 中设置的起始位置为 0xC0000000，所以将物理地址减去 0xC0000000 即可。

5 问题及解决方法

Q1：开始代码移植过程中缺失的函数及变量。

A1：因为一开始错误的认为 Lab2 主要是实现消息机制，所以对于出现的这个问题非常浮躁，导致处理这个问题有些不调理，这可能就是最终无法正确调出终端的原因之一。最后经过查阅比如 memcpy 等函数的实现及与同学的交流，实现了这些函数。

Q2：调试终端始终无法正确调出。

A2：这个问题最终页没有解决，具体的解决过程在 **4.5** 中已有说明。

Q3：**Git log** 发生错误。

A3：这个问题从 **Lab1** 开始出现，虽然得到解决，但是偶尔仍会出现，应该是我还没有真正理解出错的原理。**git** 中有一个 **master** 中的 **current** 指向当前版本，而当出现一些意外的情况时，**current** 往往会指向一个空文件。我开始将这个空文件删除。再 **git log** 的时候就会 **fatal**。因为它无法找到当前版本。最后我找到 **git log** 的记录，找到最后的一个版本，然后将 **current** 修改过去。

Q4：**send**、**receive** 实现过程中的一些逻辑上的细节。

A4：在多次测试检查后纠正。

6 总结

首先对没有达成最终目标表示遗憾，因为在考试周时间并不是很充裕，我决定在暑假的空余时间重新实现一下。不过在实现过程中，我对消息机制有了深入的体会，掌握了消息机制中 **send**、**receive** 的实现方法，并在实现过程中了解了容易出现的实现错误，这对我有很大的帮助。

除此之外是对整个学期操作系统实验课的一个小结，在这个不断经历着发布实验时有些许不情愿与完成实验后充满喜悦的学期中，我想我最重要的收获是培养了自己查阅资料、文档的能力，独立解决问题的能力。这些无疑成为我大学四年最宝贵的一部分，对于我的成长与发展也会起到巨大的作用。正如 **JYY** 学长说的那样，在培养这些能力的过程中，真的不知不觉中学到了很多东西，在阅读 **IA32** 手册、搜索资料的过程中，对中断机制的整个过程有了详细的理解和把握，对操作系统一些以前并不熟悉的功能的实现有了清晰的认识，对一个操作系统是如何运行的有了一个较为清晰的印象。就像之前只知道线程可以切换，但却对具体如何切换不甚理解。同时对之前不想用心的去学习的一些知识也进行了一定程度上的了解，比如说 **git**，**latex** 等。当真正自己面对一些问题后，就会千方百计的去了解它的原理，了解如何去修复、完成它，知识也就这样一点一点的累积起来。如果只是畏惧这些问题，那么最后收获的也就只有这份恐惧而已。

致谢

特别感谢 **JYY** 学长在这个学期的实验课中给予的正确引导！同时也感谢在整个学期的实验过程中一起交流，共同解决问题，共同探索知识的同学们！

参考

[1].**wikipedia-Git**
[2].**int 0x80**
[3].**IA32** 手册
[4]. 操作系统教程 (第四版)-孙钟秀主编

附：问题回答

1.Nanos 的消息机制依赖于一个消息队列，而这个消息队列是可能溢出的。为了避免这一问题，有些系统 (例如 **Minix**) 设计了完全同步的消息机制。当 **send** 发生时，若目标进程没有阻塞在 **receive**，则发送者会被阻塞；当 **receive** 发生时，若没有适当的消息，接收者也会被阻塞。这种消息机制确实不会带来缓冲区溢出的问题，但却也带来了诸多的麻烦，其中之一是把中断包装成消息将会很困难，其二是容易在系统中产生死锁。你能想明白这两个问题吗？

包装成中断很困难是因为中断是不可阻塞的、同时阻塞的话无法实现中断的功能，所以在发送者阻塞的实现上需要考虑更多的情况。产生死锁是因为这样实现的话必须考虑 **send** 和 **receive** 的顺序，如果 **receive** 在 **send** 的执行之后，那么就非常容易产生死锁。

2. 设备驱动程序可能正在完成一些相对耗时的操作 (如执行用户进程的读写请求)，此时到来了一个中断，那么它将会排在等待队列中，直到设备驱动程序处理完排队的请求后，才会处理它。如果希望提高中断响应的速率，你有什么好的解决办法？

提供缓冲区，提高设备与 **CPU** 的并行程度。用户进程将读写指定位置，设备驱动程序独立读写。