Project4 Synchronization Primitives and IPC设计文档

中国科学院大学

袁峥

2017.11.27

# do\_spawn, do\_kill和do\_wait设计

（1）do\_spawn的处理过程，如何生成进程ID

在初始代码中kernel.c中有一个变量spawn\_times，可以每次调用do\_spawn时将该变量加一，并以此生成进程ID。

do\_spawn首先先调用ramdisk.c中的函数来获取do\_spawn调用时所传文件名参数对应的任务类型(thread/process)和入口地址(entry\_point)，然后调用initialize\_pcb函数并结合spawn\_times作为pid来初始化任务的pcb表项，最后再将该任务放入就绪队列。

（2）do\_kill的处理过程。如果有做bonus，请在此说明在kill task时如何处理锁

do\_kill主要的任务就是找到需要杀死的任务的pid所对应的pcb表项，并在其中修改相关，如将status修改为EXITED等。

这里主要涉及到的问题是设计到同步原语的设计，当一个进程被阻塞在锁、条件变量、信号量、屏障等的等待队列中时，那么该进程被杀死时，也应该将其从对应的等待队列中删去。另外，如果杀死进程时该进程持有锁，那么杀死时也应该把该锁释放。因此在pcb中需要增加5个队列，分别对应该进程所持有的锁、正在等待的锁、正在等待的条件变量、正在等待的信号量和正在等待的屏障，在杀死该进程时，应该将其在对应资源的等待队列中删除。

这里也就包含了bonus所要求的内容。在进程获得锁的时候，同时将该获得的锁放入该进程pcb中的持有的锁队列，在释放锁的时候同时在该队列中将该锁删除。在杀死进程时，需要查看该队列是否为空，如果不为空，那么需要将该队列中的每一把锁都取出并释放。

（3）do\_wait的处理过程

每次进入先遍历一遍pcb表，查看等待的pid的进程是否已经退出，如果退出，那么退出do\_wait，否则每遍历一遍pcb表后do\_yield，下次进入时重新遍历pcb表，直到所等待的进程已退出。

（4）设计或实现过程中遇到的问题和得到的经验

首先在do\_kill的时候第一次编写的时候没有考虑到锁、条件变量、信号量和屏障的相关处理，在测试邮箱的时候才发现这一些资源在进程杀死时也需要处理，这才更新了do\_kill函数。

另外，其实目前的设计中pcb表项在被do\_kill释放后不会再进行复用，这样很浪费空间，这主要是由于在初始代码的pcb结构中，每个表项中没有有效信号，而每个进程进行pcb初始化时都是利用的其pid所对应的表项。更优的设计应该是在pcb结构中每个表项增加一个有效信号，在初始化时先全部变为无效。每次需要初始化一个表项时，应找到第一个无效的表项并在该位置存放进程相应的pcb内容，并在退出时需要将该表项的有效信号赋为无效，这样可以真正实现pcb表项的复用。

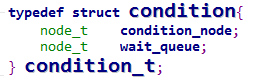
# 同步原语设计

（1）条件变量、信号量和屏障的含义，及其所实现的各自数据结构的包含内容

1、条件变量

含义：当进程不满足对应条件时，需要将其阻塞，直到其他进程使得该条件满足时，再将该进程唤醒。

数据结构：



函数：

a、void condition\_init(condition\_t \* c)中初始化wait\_queue队列。

b、void condition\_wait(lock\_t \* m, condition\_t \* c)中，首先释放锁，并将当前进程阻塞，下次执行到该进程时再获取锁。

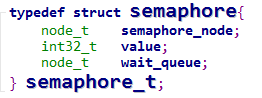
c、void condition\_signal(condition\_t \* c)中，唤醒wait\_queue队列中的一个阻塞进程。

d、void condition\_broadcast(condition\_t \* c)中，唤醒wait\_queue队列中的所有阻塞进程。

2、信号量

含义：表示拥有资源的数量，当资源大于0时，可以使用，当资源小于等于0时，如果还需要申请该资源，那么当前进程需要先阻塞，直到资源大于0时再将其唤醒。

数据结构：



value表示初始时的资源数。

函数：

a、void semaphore\_init(semaphore\_t \* s, int value)中先将value赋为初始的资源数，并初始化wait\_queue队列。

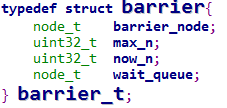
b、void semaphore\_up(semaphore\_t \* s)为释放资源，首先将value加一，如果此时value仍小于等于0，那么唤醒一个被阻塞的进程。

c、void semaphore\_down(semaphore\_t \* s)为申请资源，先将value减一，如果此时value小于0，那么将当前进程阻塞。

3、屏障

含义：在所有进程到达同一个地方前，先到达的进程先阻塞，等到所有进程到达时再全部唤醒，达到所有进程通过同一个指令的效果。

数据结构：



max\_n为该屏障规定的同时通过的进程数，now\_n为当前到达的进程数。

函数：

a、void barrier\_init(barrier\_t \* b, int n)中先将max\_n赋为n，表示该屏障要求同时通过的进程数，并将now\_n赋为0，表示当前没有进程到达，并将wait\_queue队列初始化。

b、void barrier\_wait(barrier\_t \* b)中首先判断max\_n是否等于now\_n，如果相等表示已经达到要求同时通过的进程数，因此将wait\_queue中的所有进程唤醒。否则将当前进程阻塞，并将now\_n计数器加一。

最后说明，上述三种同步原语的函数中都需要在进入前关中断，并在离开前开中断，保证操作的原子性。

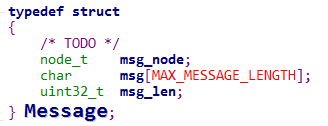
（2）设计或实现过程中遇到的问题和得到的经验

这三种由于在操作系统的课上介绍的比较详细，因此编写代码时逻辑比较清晰，也较为顺利。在过程中遇到过的问题是，一开始在信号量中的value变量的类型被定义成uint32\_t，导致在程序运行中发现进程不会被阻塞，在仔细检查代码后发现了这个问题。

# mailbox设计

（1）mailbox的数据结构以及主要成员变量的含义

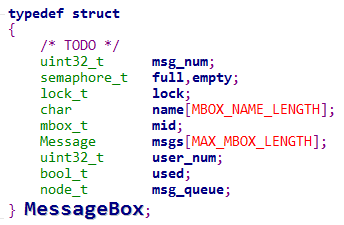
Message的数据结构：



msg[MAX\_MESSAGE\_LENGTH]为消息的内容

msg\_len为该条消息的长度

MessageBox的数据结构：



msg\_num为该信箱中的消息数

full和empty为表示信息空和满的两个信号量

lock为处理该信箱时保证原子性所拥有的锁

name[MBOX\_NAME\_LENGTH]为该信箱的名称

mid为该信箱的编号

msgs[MAX\_MBOX\_LENGTH]为存放消息的数组

user\_num为当前打开该信箱的进程数

used为表示该信箱当前是否被使用

msg\_queue为为了便于处理消息所使用的FIFO队列

（2）producer-consumer问题是指什么？你在mailbox设计中如何处理该问题？

信箱中的存入消息和取出消息类似于producer-consumer问题中的生产和消费。在mailbox中采用信号量来处理该问题。

a、void init\_mbox(void)函数中进行信箱数组的初始化，将每个信箱进行编号，并将状态设为未使用，使用计数赋为0，信箱名称设为空，并初始化信号量full和empty。

b、mbox\_t do\_mbox\_open(const char \*name)函数先检查当前是否有该名称的信箱，如果有，则将该信箱的使用计数加一并返回，否则找到一个未使用的信箱，并将该信箱的名称改为name，使用计数赋为1，并返回该信箱编号。

c、void do\_mbox\_close(mbox\_t mbox)函数首先将该信箱使用计数减一，如果此时使用计数为0，那么将该信箱的状态设为未使用，并清空里面的内容。

d、int do\_mbox\_is\_full(mbox\_t mbox)函数判断当前信箱中的信息是否已满。

e、void do\_mbox\_send(mbox\_t mbox, void \*msg, int nbytes)函数为向信箱中发送消息，首先将信号量full进行semaphore\_down，如果full为0则表示当前信箱已满，需要将当前进程阻塞。否则将msg的前nbytes放入信箱中的一条消息里，并放入msg\_queue队列，同时将信箱的消息数msg\_num加一。在退出前将信号量empty进行semaphore\_up，如果有进程被阻塞可以唤醒。另外，在整个函数的处理中，需要使用该信箱所拥有的锁，在函数开始获得，在函数返回前释放，保证操作的原子性。

f、void do\_mbox\_recv(mbox\_t mbox, void \*msg, int nbytes)函数为从信箱中接受消息，首先将信号量empty进行semaphore\_down，如果当前没有消息那么需要将进程进行阻塞。接着将信箱的消息计数msg\_num减一，并从msg\_queue中取出一条消息，将其前nbytes字节的内容赋给msg。在函数返回前对信号量full进行semaphore\_up，将之前因为邮箱满而没有成功发送消息的进程唤醒。在整个函数的处理中，也需要使用该信箱所拥有的锁，在函数开始获得，在函数返回前释放，保证操作的原子性。

（3）设计或实现过程中遇到的问题和得到的经验

在实现信箱功能的过程中，第一次采用的是条件变量来处理的，但反复调试后发现实现上较为负责，特别是在杀死进程时的资源释放上。后来改为使用信号量来控制相关变量，调试起来较为顺利，由此可见信号量使用起来比条件变量更加方便。

在三国游戏中，由于在杀死进程时，该进程可能被信号量阻塞，因此在do\_kill函数中需要处理信号量，更为全面的，需要处理锁、信号量、条件变量和屏障等资源，主要是如果该进程在对应资源的wait\_queue中，需要从其中删去，否则下次从wait\_queue中取出阻塞进程进行释放时会出现问题。这也是在第一遍编写do\_kill函数时考虑不全面，导致在三国游戏中出现问题后才想到的。

# 关键函数功能

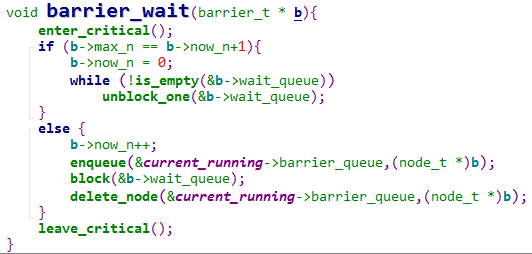
请列出上述各项功能设计里，你觉得关键的函数或代码块，及其作用

1. do\_kill(kernel.c)



该函数中包含了bonus所要求的锁的处理内容，同时还处理了杀死进程时信号量、条件变量及屏障的问题。具体实现来说，首先在pcb表中找到pid所对应的表项，并在wait\_queue和sleep\_wait\_queue队列中查到该进程，如果有则删除。再在拥有锁队列、屏障等待队列、信号量等待队列、条件变量等待队列及锁等待队列中，将相应的资源释放，在各自等待队列中删除该进程。最后将该进程的状态改为EXITED并退出。

2、barrier\_wait(sync.c)



该函数为屏障的实现函数，首先检查当前到达屏障的进程数是否等于max\_n，如果是则同时唤醒所有被阻塞的进程，结束一次屏障操作。否则将当前进程阻塞，等到满足屏障要求时再进行释放。

3、do\_mbox\_send(mbox.c)



该函数为信箱的发送消息函数，首先将信号量full进行semaphore\_down，如果full为0则表示当前信箱已满，需要将当前进程阻塞。接着获取处理当前信箱的锁，该信箱中的内容不同时被多个进程修改。再将msg的前nbytes放入信箱中的一条消息里，并放入msg\_queue队列，同时将信箱的消息数msg\_num加一。在退出前释放信箱的锁，并将信号量empty进行semaphore\_up，如果有进程被阻塞可以唤醒。

4、do\_mbox\_recv(mbox.c)



该函数为信箱的接受消息函数，首先将信号量empty进行semaphore\_down，如果当前没有消息那么需要将进程进行阻塞。然后先获取当前信箱的锁，以保证别的进程不会同时修改信箱。接着将信箱的消息计数msg\_num减一，并从msg\_queue中取出一条消息，将其前nbytes字节的内容赋给msg。在函数返回前释放当前信箱的锁，并对信号量full进行semaphore\_up，将之前因为邮箱满而没有成功发送消息的进程唤醒。