Project4 IPC and Process Management 设计文档

中国科学院大学

2014K8009929022 孔静

2014K8009929003 尚耔彤

2016.11.23

1. do_spawn, do_kill 和 do_wait 设计

（1）do_spawn 的处理过程，以及 spawn 和 fork 的区别

do_spawn 从磁盘指定位置处加载开始一个新的进程。如果磁盘指定位置不存在想要找的程序名则返回-1，如果没有空余 PCB 来供加载的新程序使用，则返回-2，如果成功加载并开始新的程序，则返回此程序的 pid，注意要在 pid 计算时保持原子操作。初始化新加载的程序并将其加入就绪队列。将优先级加入到总优先级中，以完成调度方法。

spawn 与 fork 不同的是 fork 函数是通过系统调用创建一个与原进程几乎完全相同的进程，相当于复制，除了在父进程中，fork 返回新创建子进程的 pid，在子进程中返回 0，如果出现错误则返回-1。

（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 先检查所要 kill 的 PCB 是否存在，如果不存在，则结束并返回-1，如果存在并且不是自己本身的话，就将其从所在队列删去，并检查它拥有的锁/管程/屏障/信号量，做相应的释放操作。

讲道理，为了做 bonus，增加了锁表，以及对进程开放的锁的 init/acquire/release 初始化（获取锁的 id）/获得锁/释放锁的这三种操作的接口函数，以及在系统初始化的时候对锁表进行初始化的操作，有了锁表的话，去检查进程/线程拥有的锁，就可以遍历锁表，一

旦发现某个锁的主人是要被杀死的 PCB，就将其释放，是最靠谱的做法（写了信箱之后，对信箱的操作是在 PCB 里记录了一个跟信箱总数一样数量的数组标志是否打开了对应的信箱，kill 之后会对其进行使用人数-1）。但是一开始没考虑锁表，想的是在 PCB 里面记录一下所拥有的锁，但又不知道最多能有几把锁，开头考虑用结点做，每增加一把锁 PCB 存锁那边增加一个结点，还在初始化的时候让那个用以记录的结点自己连自己，形成与 ready_queue 那样，作为标志，其他记录的锁/管程/屏障/信号量/就一个个插在这个结点后面。想法很美好，但是因为是新定义了一种结点类型（比较傻，没有利用现有结点 node_t），之后想到可以改，就又懒得改了（毕竟我们两个都有数据库实验），所以最后只是记录了一把锁，一个管程，一个信号量，一个屏障，kill 的时候，会去检查被 kill 的 PCB，对这 4 个进行释放操作。

（3）do_wait 的处理过程

do_wait 函数实现了将一个进程阻塞，直到某个它等待的进程完成。如果 do_wait 等待的进程不存在或者等待的进程是自己，则返回-1。不然则将进程的状态置于阻塞态，将其压入阻塞队列中，并调度下一个进程。这里听取了老师的意见，在每个 PCB 中增加了一个 wait_queue，do_wait 哪个 PCB 就放到哪个 PCB 的 wait_queue 里去。每个被杀/自行终止的时候，会检查该 PCB 的阻塞队列，并释放其中的 PCB，回到 ready_queue.

（4）设计或实现过程中遇到的问题和得到的经验（如果有的话可以写下来，不是必需项）

以后一定要考虑全面了再下手做！尽量避免，因为后面有更好的想法，但是之前写的东西，导致修改很麻烦。

主要的问题是（还记得的问题，太久远了），调试过程中，三次 kill 了拿锁的进程，然而卡在第二次，即第一次能成功将锁的阻塞队列中的进程释放，第二次不行，说明第二次

kill 的时候，拿锁 PCB 的进程并没有记录那把锁，查找原因最终发现信箱的锁会覆盖之前的锁（这就是只能记录一把锁的漏洞，但不想改了，数据库还等着写 12306 呢），然后考虑到持有信箱锁的时候并不会被 kill，所以在信箱那边记录了获取信箱锁之前记录一下锁，释放锁之后，恢复了锁。

2. mailbox 设计

（1）mailbox 的结构

```
typedef struct message{
    char msg[MAX_MESSAGE_LENGTH];
}message_t;


typedef struct mailbox{
    char name[MBOX_NAME_LENGTH + 1];//ensure the last is '\0'
    message_t box[MAX_MBOX_LENGTH];
    int use_num;
    int msg_count;
    lock_t l;
    node_t send_wait_queue;
    node_t recv_wait_queue;
    int write_place;
    int read_place;
}mailbox_t;
```

mailbox 的结构包含有它的名字，一块信息数组表示有界缓冲区，计数的变量，用

于判断何时重新初始化邮箱以及判断邮箱是否已满，一把互斥锁来控制访问，接受和发送阻塞队列来管理相应原因被阻塞的进程，写地址和读地址来保障发送数据和接收数据在缓冲区被顺序压入或取出。

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

两个进程（生产者和消费者）同时使用同一块缓冲区，当缓冲区为空时消费者无法从中获取信息，缓冲区为满时生产者无法向内加入信息。

在 mailbox 设计中，首先需要加互斥锁，避免不同进程同时对缓冲区内数据进行操作。如果发现信息的数量已达到信箱所能容纳的最多信息数量 do_mbox_is_full，则将发送 mail 的进程压入到 send_wait_queue 中使其阻塞，直至有进程接收信箱中的信息时将阻塞队列中的信息释放。如果发现信箱为空 do_mbox_is_empty，则将接收 mail 的进程压入到 recv_wait_queue 中使其阻塞，直至有进程再发送信息时将其从阻塞队列中释放。

（3）设计或实现过程中遇到的问题和得到的经验（如果有的话可以写下来，不是必需项）

完全不记得这里有遇到问题。

3. 键盘中断处理设计

（1）本次作业中的键盘中断处理开发主要解决什么问题？

实现 irq1 键盘中断，该中断会接受来自键盘的数据，然后缓冲到 mailbox 中。irq1 调用 keyboard_interrupt，scan_ii 判断是不是 normal_handler，如果是则放入缓冲区，

缓冲区为空则阻塞获取信息的程序，缓冲区为满则放弃信息，退出临界区，结束中断。如果是控制类的按键输入，特殊处理。如果是字母类输入，利用 mailbox 机制，等待 getchar 来取。

（2）你是如何解决键盘中断的 producer-consumer 问题？

与解决 mailbox 的生产者消费问题一样运用了互斥锁和阻塞队列。区别只在于 producer 只有一个键盘中断，而很多 consumer 需要获取信息，和当信箱满的时候 producer 会被放入阻塞队列，而键盘中断不会，而是丢弃该内容。

（3）设计或实现过程中遇到的问题和得到的经验（如果有的话可以写下来，不是必需项）

印象里，因为写的时候，没有好好在不同的头文件放置应该放的东西，导致编译的时候互相都不认识，最后我偷懒，在很多文件里增加了不同名字功能相同的函数 orz。

4. 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

static int do_spawn(const char *filename)：do_spawn 从磁盘指定位置某个名字处加载其开始一个新的进程。

static int do_kill(pid_t pid)：实现了 kill 一个进程，并且释放它所有的锁/管程/屏障/信号量。

static int do_wait(pid_t pid)：实现了将一个进程阻塞，直到某个它等待的进程完成。

void init_mbox(void)，void init_mbox_one(mbox_t i)：初始化信箱缓冲区及阻塞队列。

mbox_t do_mbox_open(const char *name)：创造一个名字为 name 的 IPC 的队列(mailbox〕并返回〔返回一个指向该 mailbox 的整数〕，在这里返回的为 mailbox 的编号，如果该队列已经存在，则队列的使用计数加 1。即允许多个 task 访问同一 mailbox。

void do_mbox_close(mbox_t mbox)：减少指定 mailbox 的引用计数。当没有 task 使用该

mailbox 时，则对其进行回收初始化。。

int do_mbox_is_full(mbox_t mbox)：如果指定的 mailbox 为满的，则返回 1，否则返回

0。注意此处需处理阻塞队列。

void do_mbox_send(mbox_t mbox, void *msg, int nbytes)：向 mbox 指定的 mailbox 发送 buf 指定的长度为 size 的信息。如果该 mailbox 已经满了，则该操作阻塞。每次发送的信息长度不超过 MAX_ MESSAGE_ LENGTH。注意此时需处理阻塞队列。

void do_mbox_keyboard(mbox_t mbox, void *msg, int nbytes)：接收由键盘中断发来的信息并压入信箱处理。注意此时需处理阻塞队列。

int do_mbox_is_empty(mbox_t mbox)：如果指定的 mailbox 为空的，则返回 1，否则返回

0。注意此处需处理阻塞队列。

void do_mbox_recv(mbox_t mbox, void *msg, int nbytes)：向指定的 mailbox 里面接收 size 大小的数据到 buf 中。注意此处需处理阻塞队列。

void unblock_mailbox_wait(node_t * wait_queue)，void block_mail(node_t *wait_queue)：处理生产者消费者问题，阻塞与释放进程。

int lock_sheet_init():初始化锁表

int lock_init_do(void):获取未使用的锁 id

void lock_acquire_do(int):获取锁

void lock_acquire_do(int):,释放锁

参考文献

[1] [单击此处键入参考文献内容]