Project 3 Interactive OS and Process Management 设计文档

中国科学院大学

段江飞

2018 年 11 月 13 日星期二

1. Shell 设计

（1） shell 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）

1. 输入字符不能实时打印，而且按了回车也不打印，我在按下回车之后会打印命令，但是它一直不打印，后来发现是回车判断我用的是'\n'，但是串口里面回车是'\r'，而且打印的位置不对，实时打印要每读一个字符就打印一次。删除也是这样，退格在串口里是 0x7f，还是需要用%x 打印看一下。
2. 系统调用里面用 printk 打印，然后滚屏的时候，发现系统调用打印的内容不见了，是因为 printk 打印的内容没有写入缓冲区，滚屏是按照缓冲区进行的，所以出错，需要自己写一个不进入系统调用的 printf 来在系统调用里面打印。

2. spawn, kill 和 wait 内核实现的设计

（1）spawn 的处理过程，如何生成进程 ID

新进程的 ID 和 PCB 相关，在 PCB 数组中，占用第几个 PCB，其 ID 就是几。

```
new_pcb->pid = ((uint32_t)new_pcb - (uint32_t)pcb)/sizeof(pcb_t);
```

（2）kill 处理过程中如何处理锁，是否有处理同步原语，如果有处理，请说明。

在 PCB 中有一个记录该 PCB 持有的锁的数组，在 kill 时，会将该进程持有的锁全部释放，在 PCB 中还有一个等待该进程的等待队列，kill 时也会释放所有等待的任务，PCB 中有一个记录进程在那个队列的成员，将进程移除队列，然后回收栈空间和 PCB 表。

没有处理同步原语，因为继续处理的话还需要给 PCB 添加很多内容，这样造成操作系统很复杂，同步原语的处理我觉得应该由用户来小心使用。

（3）wait 的处理过程

修改当前进程的状态为阻塞状态，修改当前进程所在的队列为要等待进程的等待队列，然后将当前进程的 pcb 压进要等待进程的等待队列，然后做调度 do_scheduler，但是当前进程不进入就绪队列。

3. 同步原语设计

（1）条件变量、信号量和屏障实现的各自数据结构的包含内容

1. 条件变量

```
typedef struct condition{
    queue_t cond_queue;
} condition_t;
```

条件变量仅有一个队列构成。初始化时初始化改队列；等待的时候，先将锁释放，然后就当前进程调用 do_block 阻塞，当进程再次被唤醒时，会去获取锁；signal 的时候，调用 do_unblock_one 释放条件变量队列的一个任务；broadcast 的时候，调用 do_unblock_all 释放条件变量队列的全部任务。

2. 信号量

```
typedef struct semaphore{
    int semph;
    queue_t semph_queue;
} semaphore_t;
```

信号量的数据结构包括一个初始信号量的值和一个信号量的阻塞队列。初始化的时候，将信号量的值初始化为初始要设的信号量的值，并初始化信号量阻塞队列。在 down 操作时，如果信号量 semph 大于 0，则将信号量减 1，否则，将当前进程阻塞，调用 do_block；在 up 操作时，如果 semph_queue 队列不为空，说明有任务阻塞在该信号量上，则释放一个任务进入就绪队列，不修改 semph，否则，对 semph 加 1。

3. 屏障

```
typedef struct barrier{
    uint32_t barry_n;
    uint32_t waiting_num;
    queue_t barry_queue;
} barrier_t;
```

屏障的数据结构 barry_n 表示要屏障的任务个数，waiting_num 是当前以及阻塞的任务个数，barry_queue 是屏障的阻塞队列；初始化的时候，将 barry_n 初始化为指定值，waiting_num 初始化为 0，并初始化 barry_queue；在屏障的时候，对 waiting_num 加 1，和 barry_n 比较，如果达到了屏障任务个数，则释放全部阻塞任务，然后调度，否则，将当前任务阻塞，调用 do_block。

4. mailbox 设计

（1）mailbox 的数据结构以及主要成员变量的含义

```
typedef struct mailbox{
    char name[25];
    uint8_t msg[MSG_MAX_SZ];
    int msg_front, msg_rear;
    int used_sz;

    int occur;

    condition_t full;
    condition_t empty;
    mutex_lock_t mutex;
} mailbox_t;
```

每个信箱的名字是 name，信箱中的信箱用一个缓冲区 msg 表示，缓冲区的实现利用循环队列，msg_front 和 msg_rear 表示队头和队尾，used_sz 标记已经使用的空间的大小，occur 表示信箱的索引次数，没打开一次，都会加 1，只有索引次数为 0 时，才表示没有进程打开信箱，才可以关闭。信号量 full 和 empty 表示信箱中信息是否已经满或空，互斥锁 mutex 用于对信息的互斥修改。

（2）你在 mailbox 设计中如何处理 producer-consumer 问题，你的实现中是否有多 producer 或多 consumer，如果有，你是如何处理的，

首先，生产者和消费者问题需要一个公共缓冲区，缓冲区的建立利用信箱，早 open 一个信箱的时候，会遍历全部的信箱，看是否已经存在和要打开的信箱名字相同的信箱，如果存在，就在该信箱的索引加 1，返回信箱的地址，否则，找一个空的信箱，处理后返回信箱的地址，处理包括赋名字，修改索引值。当然，在 open 和 close 的时候，需要互斥修改，这个呼出依靠的是一个全局的信箱的互斥锁。

其次，生产者向缓冲区发消息，消费者从缓冲区取消息，对信息的操作需要互斥，因此首先需要获得要修改信箱的锁，然后判断信箱缓冲区的剩余空间的大小是否满足要发送的信息的大小，如果缓冲区大小不够，则需要等待条件变量 empty 的信号在进行写缓冲区操作，写过缓冲区之后，会发一个 full 的广播，将所有阻塞在的取信息的消费者释放。消费者的处理也类似，先去尝试获得锁，然后判断消息的大小是否满足，然后取消息，唤醒等待 empty 信号的任务。

最后，实现中解决了多消费者和多生产者的问题，在判断消息的大小是否满足要接受的消息大小或者缓冲区剩余空间大小是否满足要发送的信息大小时，用 while 进行判断，任务每次唤醒都会进行判断，根据判断结果决定是等待还是继续执行。

（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）

遇到最大的 bug：写好了信箱，然后去测试，发现就是卡住，没法执行下去，百思不得

其解，后来 gdb 跟发现卡在了 memcpy，怎么想觉得不会错啊，后来去看反汇编，发现每次拷贝 size 都是加 4，然后发现老师的 typedef unsigned uint8_t，顿时惊为天人，卡了好久的 bug 找出来一个，然后又陷入泥淖，在成功杀死一次之后，卡住了，我打印 pcb 发现找不到 pcb 在那个队列里，有找了好久，发现任务需要自己写初始化，我就直接在第一个任务开头进行初始化，然后任务杀死之后，每次重新执行都会初始化，导致挂在信箱信号量上的任务什么的丢丢失了…

最开始我以为我的循环队列有问题，就先写了一个 naïve 的直接 memcpy，然后后来测试我的循环队列的时候，发现正常运行十几次之后会卡死，我又百思不得其解，然后 gdb 跟啊跟，发现拷贝那有问题，但是我在写个 C 程序测试，没有问题，后来发现是我注释的时候少注释了一行…愚蠢背锅

5. 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

do_spawn 函数，初始化一个任务

do_kill 和 do_exit 函数，杀死任务和任务退出

尤其是 kill 和 exit 的时候，对 pcb 的处理

具体代码见代码文件(sched.c)