OSLab4 同步机制

实验报告

吴澄杰　　151220122　　151220122@smail.nju.edu.cn　　2017年5月28日

一、实验进度

此次实验，我完成了所有要求内容。我实现了内核级的多线程，并且封装为wthread函数。由于在更改实验要求之前，我已经实现了进程间的具名信号量，因此我一共实现了进程间具名信号量，线程间匿名信号量，共2套同步机制。此外，我设计了两个测试样例。一个是生产者和消费者问题，另一个是测试多个线程（进程）竞争有限资源时的情形。

二、实现细节

1、用wthread函数封装多线程的实现

我实现了内核级的多线程，并且提供了wthread函数供调用。函数头文件在 **lib／include/wthread.h** 中，相应的内核函数实现在 **kernel/process/call.c** 中。为了实现方便，每个函数都被分配了一个系统调用号，会直接用系统调用执行相应内核函数，故其实主要实现工作是在内核函数中的。

（1）**wthread_create (wthread *thread, void *func, void *arg)** 函数会创建1个新的进程，func为入口函数地址，arg为传入该函数的参数。内核函数 **wthread_create_kernel (void *func, void *arg, wthread *thread)** 进行的工作和创建一个新进程类似，不同之处在于对数据和代码进行浅拷贝，只分配新的内核栈和用户栈。此外，要将陷阱帧中的 eip 设置为入口函数的地址，同时将传入参数 arg 提前放到用户栈的相应位置中。实现细节在代码中都已经加了注释。

（2）**wthread_exit()** 函数会结束当前线程。内核函数 **wthread_exit_kernel()** 进行的工作和进程退出类似，不同之处在于只释放自身独自占有的用户栈区域，共享的程序代码、数据区域则不做释放。

（3）**wthread_join(wthread *thread)** 函数会让当前线程阻塞直到thread线程退出后才会继续运行。具体实现在 **lib/wthread.c** 中。关键点是，每个thread创建时都会初始化一个名为 has_exited 的信号量，初始值为0。在 **wthread_exit_kernel()** 中，会对该信号量进行一个V操作。任何 join 住该线程的线程只要P该信号量即可。在该线程退出之前，P操作无法完成，因此就成功被阻塞在了该信号量上。该线程退出之后，P操作可以完成。同时，为

了出理多个线程join同一个线程的情况，**wthread_join** 中在P该信号量后还要进行V操作，使得其他的阻塞线程也可以继续进行下去。

2、线程间匿名信号量

函数原型在 **lib/include/semaphore.h** 中，相应内核函数实现在 **kernel/process/semaphore.c** 中。用户函数基本上直接进行系统调用了，具体实现都在相应内核函数中。

（1）**sem_init** 函数其实就是一个系统调用，直接陷入内核态并调用了相应的内核函数 **sem_init_kernel(semaphore *sem, int value)**（以下的函数都是此种情况）。该函数设置了信号量的初始值，设置等待队列为空，并且设置它正在使用。

（2）**sem_destroy** 函数销毁信号量。**sem_destroy_kernel(semaphore *sem)** 函数将该信号量设为不可用，同时设置等待队列为空，值为0。

（3）**sem_wait** 函数是P操作。**sem_wait_kernel(semaphore *sem)** 函数首先检查当前信号量是否可用。若可用，再判断当前值是否大于0。若是，则将信号量的值减1后直接返回，不被阻塞；反之，则将自己加入到该信号量的等待队列中，并且放弃自己的运行。

（4）**sem_post** 函数是V操作。**sem_post_kernel(semaphore *sem)** 函数首先检查当前信号量是否可用，若可用，再判断当前值是否大于0或等待队列是否为空。若是，则将信号量的值加1后直接返回；反之，则唤醒等待队列中第一个等待的线程，将其加入就绪队列。

3、进程间具名信号量

我同时实现了进程间具名信号量。函数原型在 **lib/include/semaphore.h** 中，相应内核函数实现在 **kernel/process/semaphore.c** 中。在使用信号量前，需要用 **sem_open** 函数用一个编号向内核申请一个信号量，并且赋初值。内核会返回该信号量的地址。**sem_wait** 与 **sem_post** 和匿名信号量是通用的。**sem_close** 函数则是向内核申请关闭该信号量。由于每个进程都能够和内核通信，因此，此种在内核中实现的具名信号量就能够实现进程间的通信。这一点在我设计的实验中有体现，详见下文。

三、测试样例

在OSLab工程目录下有测试样例sem，sem/sem.c是测试样例的源代码。提交的文件已经默认链接sem测试样例而不是游戏了。若要修改，可在工程目录下的Makefile第12行中修改，会有相应的注释给予帮助。**为了在两个测试样例中切换，sem.c中使用了条件编译。可注释且仅注释掉在sem.c中14、15中的一行，从而执行相应的测试样例。**文件中有相应注释提供帮助。

1、生产者消费者问题

实验有一些参数可以设置：T是每个生产者和消费者执行生产、消费操作的轮数，N是buffer的大小，PN是生产者的数量，CN是消费者的数量。在主函数 test_main 中，首先会

进行buffer和信号量的初始化，随后按照给定的数量创建生产者和消费者线程并且join住它们。最后销毁所有的信号量后退出。

生产者和消费者所做的工作是类似的：进行T轮迭代，每轮进行一次生产（produce）或消费（consume）操作。consume操作会取出buffer最后一个数并且返回它的值。produce操作会将生产出的数*i*存到buffer的下一个空闲空间中。为了保证不同的producer生产不同的内容便于实验展示，规定了若 x mod PN = k，那么x只能由k+1号生产者生产。

为了模仿真实应用，并且也为了提升实验展示效果，在生产和消费前都必须经过一个halt()函数。它的作用就是延长一下整个过程的运行时间。

```
Producer No.0 puts data No.0 into the buffer
Producer No.1 puts data No.1 into the buffer
Producer No.0 puts data No.2 into the buffer
        Consumer No.0 withdraws data No.2 from the buffer
Producer No.1 puts data No.3 into the buffer
        Consumer No.1 withdraws data No.3 from the buffer
        Consumer No.0 withdraws data No.1 from the buffer
Producer No.0 puts data No.4 into the buffer
        Consumer No.1 withdraws data No.4 from the buffer
Producer No.1 puts data No.5 into the buffer
Producer No.0 puts data No.6 into the buffer
        Consumer No.0 withdraws data No.6 from the buffer
Producer No.1 puts data No.7 into the buffer
        Consumer No.1 withdraws data No.7 from the buffer
        Consumer No.0 withdraws data No.5 from the buffer
Producer No.0 puts data No.8 into the buffer
        Consumer No.1 withdraws data No.8 from the buffer
Producer No.1 puts data No.9 into the buffer
Producer No.0 puts data No.10 into the buffer
        Consumer No.0 withdraws data No.10 from the buffer
Producer No.1 puts data No.11 into the buffer
        Consumer No.1 withdraws data No.11 from the buffer
        Consumer No.0 withdraws data No.9 from the buffer
Producer No.0 puts data No.12 into the buffer
        Consumer No.1 withdraws data No.12 from the buffer
Producer No.1 puts data No.13 into the buffer
Producer No.0 puts data No.14 into the buffer
        Consumer No.0 withdraws data No.14 from the buffer
Producer No.1 puts data No.15 into the buffer
        Consumer No.1 withdraws data No.15 from the buffer
        Consumer No.0 withdraws data No.13 from the buffer
Producer No.0 puts data No.16 into the buffer
        Consumer No.1 withdraws data No.16 from the buffer
Producer No.1 puts data No.17 into the buffer
Producer No.0 puts data No.18 into the buffer
        Consumer No.0 withdraws data No.18 from the buffer
Producer No.1 puts data No.19 into the buffer
        Consumer No.1 withdraws data No.19 from the buffer
        Consumer No.0 withdraws data No.17 from the buffer
        Consumer No.1 withdraws data No.0 from the buffer
```

代码中默认设置迭代次数（T）为10，buffer大小（N）为4，生产者和消费者数目（PN、CN）都是2。实验结果如图。若调整参数进行实验，需注意，在当前的设计下，若所有进程（线程）都被阻塞，则机器会停止运行并显示消息。

2、多个线程（进程）竞争有限资源问题

第二个测试样例模拟了多个线程（进程）竞争有限个数资源的情况。每个线程（进程）都会重复进行一个简单过程：申请资源（P操作），释放资源（V操作）。在迭代次数达到设定值时则结束运行并输出“OK”的信息。此测试主要是为了通过比较大的线程（进程）数量和迭代数量，测试线程、进程实现的正确性，线程（进程）间通信的正确性，和信号量中阻塞、唤醒功能实现的正确性。该测试首先会输出线程间匿名信号量的测试结果，其次会输出进程间具名信号量的测试结果。

四、实验心得

经过此次实验，我对进程（线程）间通信有了更好的了解，通过自己的实现，真正弄清楚了它其中可行的机制。此次实验基于前几次实验已经形成的框架，因此可以充分利用之前实现的框架。比如线程就可以完全模仿进程而稍作调整；信号量的等待队列的进入与离开就可以模仿内核中“休眠”队列的进入与离开来实现。这样也可以使得OS前后具有一致性，易于理解、管理与后续的实验进行。