Xv6 Kernel Thread

by Zhang Xiaozheng

- 增加system call
  - 首先在syscall.h, syscall.c, user.h, usys.S中增加对应的system call并指定编号。
- 添加clone，join实现，主要在proc.c中实现
  - 首先修改proc.h中的pcb结构，由于在join的时候需要返回每个thread的user stack的地址，所以我们在pcb中增加一个字段来存储user stack
  - 在proc.c中增加实现clone
    - clone实现和fork类似，不同的是我们不能将父进程的pgdir复制给新建的进程，因为要共享内存。所以我们只是简单的赋值即可
    - clone中的user stack要进行初始化，我们要将参数和一个fake return address push到user stack中，首先进程中的stack地址应该是从高位向低位增长，所以我们先将栈指针移动到栈底。利用copyout()函数将参数复制到user stack中
    - 其次，clone返回以后，如果新建的这个线程被调度到，他会从eip寄存器中的地址开始执行，并使用地址为esp寄存器中的栈地址，所以将栈地址赋值给esp寄存器，将函数fcn地址赋值给eip寄存器
  - 在proc.c中增加实现join
    - join和wait类似，首先在ptable中寻找时，不能简单的判断父进程是不是当前进程，因为我们join是提供给线程使用的，单纯判断是否为父进程的时候，子进程也会被判断成功，所以还需要判断pgdir是否一样。
    - 其他只需要更改wait中的free部分，因为我们和父进程共享内存，不能free掉父进程内存
  - 同时由于可能出现子线程同时grow父进程内存空间的情况，所以需要在growproc中加锁，保证父进程内存正确
- 增加ticket lock和thread library
  - 首先在x86.h中增加fetch_and_add的原子性操作
  - 新建ticketlock.h文件，添加ticketlock结构体的定义
  - 在user.h中增加函数签名
  - 在ulib.c中实现函数
    - 增加一个结构体来保存所有线程的stack地址，分配的内存空间地址和该线程是否被占用
    - 实现thread_create时，需要注意malloc后判断地址空间是否对齐，没有对齐一页要对齐一页，然后遍历所有线程，找到可用线程，更改状态并赋值内存地址，stack地址，注意判断时要加锁，避免多个线程同时更改同一个。
    - 实现thread_join时，首先调用stack获取子线程的stack地址，然后和所有线程中的stack进行比较，如果相同说明找到了此线程，free掉分配的内存空间并返回。
    - lock_init, lock_acquire, lock_release实现见老师给出的参考材料