Lecture 30: Sleep and wakeup in xv6

Mythili Vutukuru
IIT Bombay
https://www.cse.iitb.ac.in/~mythili/os/
Sleep and wakeup

• A process P1 in kernel mode gives up CPU to block on an event
  – Example: process reads a block from disk, must block until disk read completes
• P1 invokes “sleep” function, which calls sched() and gives up CPU
• Another process P2 in kernel mode calls “wakeup” when event occurs, marks P1 as runnable, scheduler loop switches in P1 in future
  – Example: disk interrupt occurred when P2 is running, so P2 handles the interrupt, and marks P1 as runnable
• How does P2 know which process to wake up? When P1 sleeps, it sets a channel (void * chan) in its struct proc, and P2 calls wakeup on same channel (channel = any value known to both P1 and P2)
  – Example: channel value for disk read can be address of disk block
• Spinlock protects atomicity of sleep: P1 calls sleep with some spinlock L held, P2 calls wakeup with same spinlock L held
  – Eliminating missed wakeup problem that arises due to P2 issuing wakeup between P1 deciding to sleep and actually sleeping
  – Lock L released after sleeping, available for wakeup
  – Similar concept to condition variables studied before
Sleep function

- Sleep calls sched() to give up CPU
  - Needs to hold ptable.lock
- Acquire ptable.lock, release the lock given to sleep (make it available for wakeup)
  - Unless lock given is ptable.lock itself, in which case no need to acquire again
  - One of two locks held at all times
- Calls sched(), switched out of CPU, resumes again when woken up and ready to run
- Reacquires the lock given to sleep and returns back
  - Code that invoked sleep with lock held returns with lock held again

```c
2871 // Atomically release lock and sleep on chan.
2872 // Reacquires lock when awakened.
2873 void
2874 sleep(void *chan, struct spinlock *lk)
2875 {
2876     struct proc *p = myproc();
2877     if(p == 0)
2878         panic("sleep");
2879     if(lk == 0)
2880         panic("sleep without lk");
2881     if(lk != &ptable.lock)
2882         acquire(&ptable.lock);
2883     release(lk);
2884     // Go to sleep.
2885     p->chan = chan;
2886     p->state = SLEEPING;
2887     sched();
2888     // Tidy up.
2889     p->chan = 0;
2890     if(lk != &ptable.lock)
2891     release(&ptable.lock);
2892     acquire(lk);
2893 }
```

- Acquire ptable.lock, release the lock given to sleep (make it available for wakeup)
  - Unless lock given is ptable.lock itself, in which case no need to acquire again
  - One of two locks held at all times
- Calls sched(), switched out of CPU, resumes again when woken up and ready to run
- Reacquires the lock given to sleep and returns back
  - Code that invoked sleep with lock held returns with lock held again
Wakeup function

- Called by another process with lock held (same lock as when sleep was called)
- Since it changes ptable, ptable.lock will also be held
  - If sleep lock is ptable.lock itself, then directly call wakeup1
- Sleep holds one of sleep's lock or ptable.lock at all times, so a wakeup cannot run in between sleep
- Wakes up all processes sleeping on a channel in ptable (more like signal broadcast of condition variables)
  - Good idea to check condition is still true upon waking up (use while loop while calling sleep)
Example: pipes

- Two processes connected by a pipe (producer consumer)
  - Common shared buffer, protected by a spinlock
- One process writes into pipe, another reads from pipe
- Reader sleeps if pipe is empty, writer wakes it up after putting data
- Writer sleeps when pipe is full, reader wakes it up when data is consumed
- Addresses of pipe structure variables are channels (same channel known to both)
Example: wait and exit

• If wait called in parent while children are still running, parent calls sleep and gives up CPU
  – Here, channel is parent struct proc pointer, lock is ptable.lock

```c
2706 // Wait for children to exit. (See wakeup1 call in proc_exit.)
2707 sleep(curproc, &ptable.lock);
```

• In exit, child acquires ptable.lock and wakes up sleeping parent

```c
2650 // Parent might be sleeping in wait().
2651 wakeup1(curproc->parent);
```

• Here, lock given to sleep is ptable.lock because parent and child both access ptable (sleep avoids double locking, doesn’t acquire ptable.lock if it is already held before calling sleep)

• Why is terminated process memory cleaned up by parent? When a process calls exit, CPU is using its memory (kernel stack is in use, cr3 is pointing to page table) so all this memory cannot be cleared until terminated process has been taken off the CPU
  – Parent code in wait is a good place to clean up child memory after child has stopped running
Summary

• Sleep and wakeup functionality in kernel for processes to wait for or signal each other
  – Similar to condition variables for synchronization of user space threads

• Examples of sleep/wakeup
  – Pipe reader and pipe writer processes
  – Parent sleeps for child to die, zombie child wakes up parent

• Code calling sleep and wakeup need to hold same lock, in order to avoid missed wakeup problem