xv6©-rev10

(Copyright Frans Kaashoek, Robert Morris, and Russ Cox.)

sleep syscall

Carmi Merimovich

Tel-Aviv Academic College

February 20, 2017

- The `sleep` system call suspends execution of the process for the number of ticks supplied by the argument.
- There is an argument!!
- It must be checked carefully!!
- The `sys_sleep` implementation is very simple:
  - It is assumed each clock tick declares an event with id `&ticks`.
  - `sys_sleep` waits for the `&ticks` event.
  - When sys_sleep resumes execution, it checks if it was suspended for long enough.
  - If not it returns to the event waiting.

Variables in `sys_sleep`

- `n`: Number of ticks to wait.
- `ticks`: Global variable containing the number of ticks from boot.
- `tickslock`: A spinlock protecting `ticks`.

sys_sleep

```
3815  sys_sleep(void) {
       int n;
       uint ticks0;

       if (argint(0, &n) < 0)  return -1;
       acquire(&tickslock);
       ticks0 = ticks;
       while (ticks - ticks0 < n) {
        if (myproc()->killed) {
         release(&tickslock);
         return -1;
        }
        sleep(&ticks, &tickslock);
       }
       release(&tickslock);
       return 0;
      }
```

trap() part 1

```
void trap(struct trapframe *tf) {
 if (tf->trapno == T_SYSCALL) {
  if (myproc()->killed)
   exit();
  myproc()->tf = tf;
  syscall();
  if (myproc()->killed)
   exit();
  return;
 }
```

- User mode might be loooong, hence the check on **myproc()->killed** before returing.

trap() part 2, controller interrupts

```
switch (tf->trapno) {
case T_IRQ0+IRQ_TIMER:
 if (cpuid() == 0) {
  acquire(&tickslock);
  ticks++;
  wakeup(&ticks);
  release(&tickslock);
 }
 lapiceoi();
 break;
case T_IRQ0+IRQ_IDE:
 ideintr();
 lapiceoi();
 break;
case T_IRQ0+IRQ_IDE+1:
 break;
```

```
case T_IRQ0+IRQ_KBD:
 kbdintr();
 lapiceoi();
 break;
case T_IRQ0+IRQ_COM1:
 uartintr();
 lapiceoi();
 break;
case T_IRQ0+7:
case T_IRQ0+IRQ_SPURIOUS:
 cprintf("cpu%d:␣spurious␣\
␣␣␣␣interrupt␣at␣%x:%x\n",
   cpuid(), tf->cs, tf->eip);
 lapiceoi();
 break;
```

trap() part 2, unexpected interrupt

```
default:
  if (myproc() == 0 || (tf->cs&3) == 0) {
    cprintf("unexpected trap %d from cpu %d \
         eip %x (cr2=0x%x)\n",
         tf->trapno, mycpu()->id, tf->eip, rcr2());
    panic("trap");
  }
  cprintf("pid %d %s: trap %d err %d on cpu %d "
      "eip 0x%x addr 0x% x      kill proc\n",
  myproc()->pid, myproc()->name, tf->trapno, tf->err
         cpuid(), tf->eip,
  rcr2());
  myproc()->killed = 1;
}
```

trap() part 3

```
  if (myproc() && myproc()->killed &&
            (tf->cs&3) == DPL_USER)
    exit();

  if (myproc() && myproc()->state == RUNNING &&
          tf->trapno == T_IRQ0+IRQ_TIMER)
    yield();

  if (myproc() && myproc()->killed &&
            (tf->cs&3) == DPL_USER)
    exit();
}
```