PintOS System Calls assignment

Navneel Singhal
2018CS10360

Part A.

Q1.

The output of the program was empty.
The complete output is as follows:

```
Copying ../examples/test-syscalls to scratch partition...
qemu-system-i386 -device isa-debug-exit -hda /tmp/E_Nct0Nm2S.dsk -m 4 -net none -nographic -monitor null
WARNING: Image format was not specified for '/tmp/E_Nct0Nm2S.dsk' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
PiLo hda1
Loading.....
Kernel command line: -q -f extract ls run tl
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  13,094,400 loops/s.
hda: 9,072 sectors (4 MB), model "QM00001", serial "QEMU HARDDISK"
hda1: 196 sectors (98 kB), Pintos OS kernel (20)
hda2: 8,192 sectors (4 MB), Pintos file system (21)
hda3: 95 sectors (47 kB), Pintos scratch (22)
filesys: using hda2
scratch: using hda3
Formatting file system...done.
Boot complete.
Extracting ustar archive from scratch device into file system...
Putting 'tl' into the file system...
Erasing ustar archive...
Files in the root directory:
tl
End of listing.
Executing 'tl':
Execution of 'tl' complete.
Timer: 61 ticks
Thread: 30 idle ticks, 31 kernel ticks, 0 user ticks
hda2 (filesys): 44 reads, 199 writes
hda3 (scratch): 94 reads, 2 writes
Console: 853 characters output
Keyboard: 0 keys pressed
Exception: 0 page faults
Powering off...
```

Q2.

There are 4 syscall macros – syscall0, syscall1, syscall2 and syscall3, where the number corresponds to the number of arguments provided to the syscall other than the syscall number (so the arguments passed to the macros are syscall number and the arguments to actually be passed to the syscall). Each syscall does the following:
1. Push arguments onto the stack in reverse order.
2. Push syscall number onto the stack.
3. Call the interrupt 0x30 (which is used for all syscalls)
4. Restore the stack by moving the stack pointer back to where it was before the pushing onto stack took place.
5. Store the answer in retval.
There is memory clobbering done in the inline assembly to ensure that the read/write on memory locations other than input and output are consistent by flushing the register contents to memory, and not allow reordering memory accesses.

Q3.

Implementation

Q4.

Implementation

Part B.

Q1.

For a thread A to wait for another thread B, we can do so using a semaphore (initialized to 0) as follows:
1. In thread A, call sema_down on the pointer to the semaphore.
2. In thread B, call sema_up on the pointer to the semaphore.

Q2.

Implementation.

Q3.

Implementation.