Lab 2 Overview

Section 3: 4/18/19

Spinlocks

- Multiprocessor: Loop until exclusive lock is acquired
- Uniprocessor: Disable interrupts on single processor (xk)
- xk spinlock interface
  - inc/spinlock.h
  - kernel/spinlock.c
- What are the pros/cons of spin locks?
- **Warning:** If you try schedule while holding a spinlock, the scheduler will panic. Disabling interrupts and scheduling another process is a surefire way to never get a timer interrupt.

Synchronization Functions

- Main API for process control: wakeup/sleep
  - Helper function: wakeup1 (find all sleeping processes and wake up on channel)
- Relevant files
  - inc/proc.h
  - kernel/proc.c
- Possible from the time of “waking up” the condition is no longer true. (Mesa monitors)
- With the code snippet below, once the while loop is exited, the thread has guaranteed mutual exclusion and you KNOW the condition is held.

```
while(!CONDITION) {
  sleep(channel, &mylock);
}
```

Sleeplocks

- Use the interface (sleep/wakeup) from the previous slide. But the channel is the address of the lock.
- On `acquiresleep`, the waiting thread sleeps on the address of the lock, setting the state of the current process to `SLEEPING`. It won't get scheduled until there is an opportunity to grab the lock.
- On `releasesleep`, the thread holding the sleeplock wakes up all the processes waiting on the channel of the lock. This will set all the processes sleeping with `chan` value `&lock` to be `RUNNABLE`, these processes will wake up, check the lock condition, fall through if true, otherwise sleep again.
- XK sleeplock interface
  - inc/sleeplock.h
  - kernel/sleeplock.c
- What are the pros/cons of sleep locks?

More on Locks and Synchronization

See Chapter 5 in Operating Systems: Principles and Practice

fork()

Creates a new process by duplicating the calling process. Returns 0 in child, and child PID in parent.

What does this entail? What needs to be created and what/how do we copy parent process state?

wait()/exit()

wait() - Waits until a child process terminates and returns that child’s PID.

exit() - Halts program and reclaims resources consumed by the program.

What are some edge cases to consider when it comes to managing process resources?

pipe(pipefds)

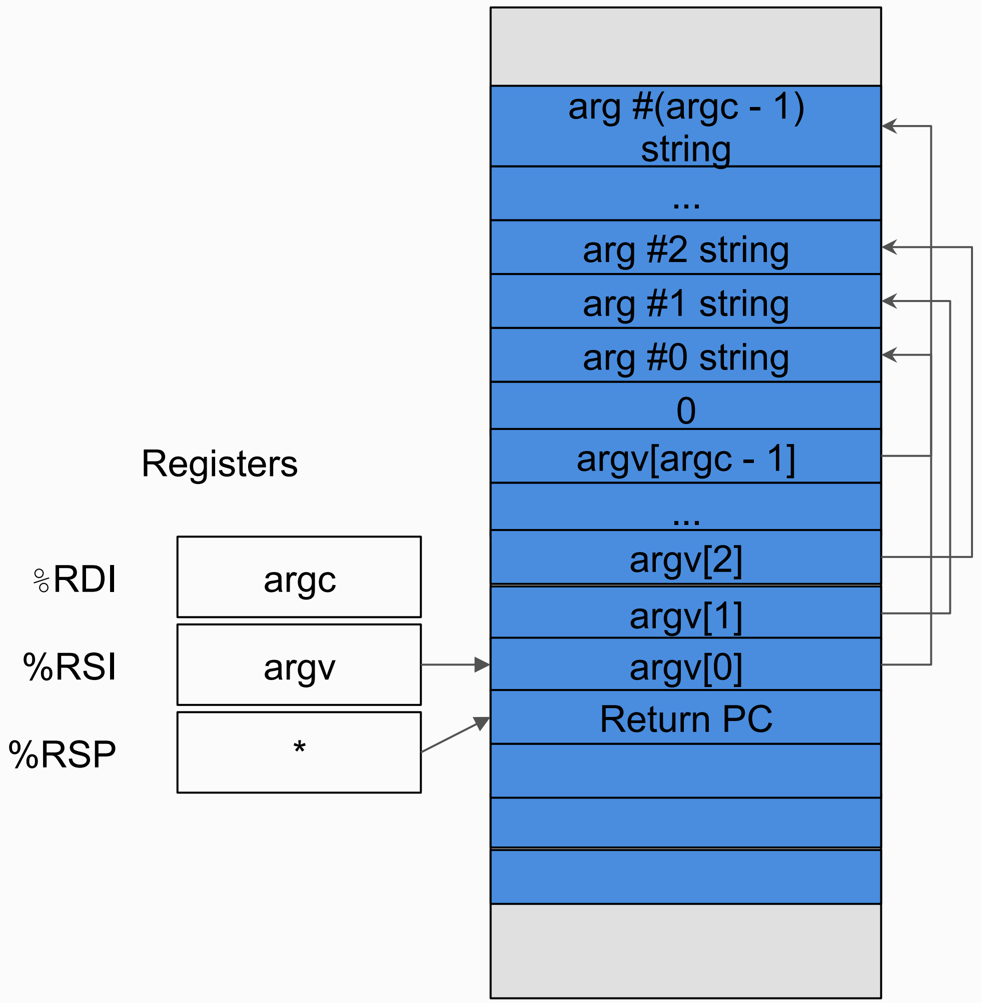
Creates a pipe (internal buffer) for reading to/writing from.

From user's perspective: Two new files will be allocated, one will be the "read end" (not writable), and one will be the "write end" (not readable).

exec(progname, args)

Replaces the process state by executing the given program using the given arguments. Before running the process you’ll need to carefully set up the user process stack, the register state and program arguments.

What are some challenges in this?



Return PC can be garbage data (like 0xfff..f) because we have no function to return to.

The first argument is an integer count of the number of pointers in the second argument. This is bounded by MAXARG.

The second argument should be a pointer to an array of pointers to the actual values. We put them on the user stack because that's the only space we have to put values.

Use vspacewritetova to export values to a page table that is not currently installed. And remember, when the process is ready to run, the virtual address space needs to be installed, with vspaceinstall(myproc());.

Reminder: Design Document

Due Friday**(4/19)**:

- This is for you, whatever will prepare you for success should be on the document.
- It will be hard the first time knowing what to include, that’s ok. You will learn from the earlier labs to (hopefully) become more successful in later labs.
- Office hours are a great time to talk about design. It’s easier to see your approach in words instead of spread throughout many files.
- Use lab/designdoc.md as a reference on what to include in your design docs!