CS230/330: Project 4
Everything

Handed out: Thursday, February 15, 2001
Due: Friday, February 23, 11:59 p.m.

1 Important Note

Due to the limited time remaining, you do not have to implement swapping or support the flash card device. Sigh.

2 Assignment

Now that you've got memory, processes, and all of the hardware devices fully working, it's time to finish off the kernel by supporting interprocess communication and threads. Most of this should be fairly straightforward.

3 Instructions

1. Finish Project 3 and Project2 if applicable.
2. Implement the Register() system call.
3. Implement the Send() system call.
4. Implement the Receive() system call.
5. Implement the SemaCreate() system call.
6. Implement the SemaDestroy() system call.
7. Implement the SemaSignal() system call.
8. Implement the SemaWait() system call.
9. Implement the RFork(FORK_THREAD) system call.

4 Specific Details and Suggestions

- To implement the Register(), Send() and Receive() system calls, your kernel will need some way to look up process IDs and service identifiers. In addition, each process will probably need to have some kind of queue of incoming messages or processes that are trying to send messages.

- Implementing SemaCreate(), SemaDestroy(), SemaSignal(), and SemaWait() should be fairly simple. A semaphore is nothing but an integer counter with a process queue attached to it.
- To implement RFork(FORK_THREAD), you will need to address a number of somewhat nasty design issues.
  - Your kernel must allow processes to have a shared page table (e.g., multiple processes with exactly the same page table or identical page table mappings).
  - Each thread needs to have its own dedicated stack.
  - When a thread calls Exit(), the kernel should release its stack pages and return a status code to the parent as normal. However, the shared page table should not be destroyed until all associated threads/processes have terminated.
  - System calls such as Brk() should affect all processes and threads with the same page table (the process heap is a shared resource).
  - Thought problem: What happens when a thread calls RFork(FORK_PROCESS)?
  - Thought problem: When the parent process exits, should all child threads also terminate?

5 Logistics and Grading

This project must be handed in via CVS. Make sure that all of your source files, Makefile, and README files have been committed to the repository. If working in a group, the project should be committed to the repository located in `/u5/groups/groupname`. Make sure your project name is "yalnix".

To test correctness, we will run your `yalnix` executable on an init process and shell that we will provide. This will then be used to run a number of test programs to exercise your process management system calls.

The grading on this part of the project is only a final check before the final project due date. **The entire kernel project is due on Friday, March 2.**