CS230/330: Project 4
Booting to a shell

Handed out: Friday, February 15, 2002
Due: Friday, March 1, 11:59 p.m.

1 Assignment

Now that you've got most of your I/O devices, context switching, and program loading to work, it's time to finish off the kernel by supporting processes, interprocess communication, and other features. By the time you finish this, your kernel will be able to run a simple shell and run all sorts of interesting programs.

2 Instructions

1. Finish Project 3 and Project 2 if applicable.
2. Implement the Fork() system call.
3. Implement the Wait() system call.
4. Implement the Exit() system call.
5. Implement the Exec() system call.
6. Modify your kernel so that it only loads a single process called 'init' instead of the bogus 'init1' and 'init2' scheme from Project 3. To test your kernel, try writing a simple init program like this:

```
#include "yalnix.h"

int main() {
   char *args1[] = { "init1", "init1", 0 };
   char *args2[] = { "init2", "init2", 0 };
   int status;
   if (Fork() == 0) {
       if (Exec(args1[0],args1) < 0) {
          printf("Couldn't run init1\n");
       }
       Exit(1);
   }
   if (Fork() == 0) {
      if (Exec(args2[0],args2) < 0) {
          printf("Couldn't run init2\n");
```

```
        }
        Exit(2);
    }
    Wait(0, &status);
    Wait(0, &status);
    Exit(0);
}
```

See if you can run any of the project3 tests as before.

7. Implement PipeCreate(), PipeRead(), and PipeWrite().

8. Implement SemaCreate(), SemaDestroy(), SemaSignal(), and SemaWait().

3 Specific Details and Suggestions

Most of this project should be extremely straightforward if everything in Project 3 is working. Here are a few hints and suggestions.

- To implement Fork() you need to make a copy of the calling process. This means you will need to allocate memory and use memcpy() to copy the data from all of the pages from the parent to the child. The only problem is that you can't map the parent and the child process in virtual memory at the same time. You'll need to devise a clever page table trick to do the copy.
- Each process should record the process ID of its parent process. Note: recording a pointer to the parent process is a bad idea—what happens if the parent dies before the child?
- To implement Wait() and Exit(), each process will need to have a queue of dead children (i.e., zombies). When a process calls Exit(), it is placed on the zombie queue of its parent. If the parent is dead, the process is simply destroyed and the exit code ignored.
- To implement Exec() you will need to modify your program loading code. Pay careful attention to error handling and remember that Exec() always returns to the calling process with an error if the new program couldn't be loaded for some reason. To coordinate program loading, you might want to implement a loading queue.
- Semaphores are extremely easy to implement. Pipes are not.

4 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 8.**