CS230/330: Project 2.3
Booting to the Shell

Handed out: Friday, May 7, 1999
Due: Friday, May 21, 1999, 11:59 p.m.

1 Assignment

The purpose of this project is to enhance your Yalnix kernel so that you can run programs from the shell just like you can under Unix. To do this, you will need to complete the system calls that you have not been required to implement yet (in particular: `Fork`, `Exec`, `Wait`, `Register`, `Send`, `Receive`, `Reply`, `CopyTo`, and `CopyFrom`). You may also need to extend some of the system calls that you already support (in particular, `Exit`). You may also need to extend the functionality of some of the kernel subsystems that you have already written such as the scheduler and memory manager.

Here are some steps to get you started:

- Implement `Fork()`
- Write an `init` process that prints a "hello" message and then forks itself. Have the parent and the child print different messages and then exit. Modify the `KernelStart` function to only load the single `init` process, not both `init1` and `init2`.
- Implement `Exec`.
- Modify the `init` process so that the child process, after printing out a message, `Exec`'s some other process that prints out yet another message and then exits.
- Implement `Wait`.
- Modify the `init` process so that the parent process waits for the child to exit, at which time it prints out yet another message that includes the exit code of the child and then quits.
- Once you get this working, you are getting very close to completion. Write an `init` process that forks an idle process and the shell (provided by us). At this point, you can remove the in-kernel idle process.
- Implement `Register`, `Send`, `Receive`, and `Reply`.
- Implement a test program that uses the basic IPC operations and can be run from the shell prompt (e.g., run a program that forks a copy of itself and has the child `Register` with the kernel so that the parent can `Send` it a message to print out).
- Implement and test `CopyTo` and `CopyFrom`.
- Congratulations! You have just finished Yalnix.

2 Logistics and Grading

As usual, 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 `/u1/projects/groupname`. Make sure your project name is "yalnix".

In addition to providing a new set of test programs, we will re-run all of the tests used in the grading of project2.2. If you lost points for that stage of the project, you will earn half of the points deducted back if your project 2.3 kernel passes *all* of the 2.2 tests.

This part of the project is worth 30% of the final project 2 grade. It will be graded on correctness (80%) and style (20%). To test correctness, we will run your `yalnix` executable using our own `init` process, which will fire up an idle process and a shell. We will then run a variety of tests on top of this shell that will add increasing amounts of "stress" to your system (e.g., Fork'ing an Exec'ing lots of times, firing up a mix of big and small programs, etc...). Make sure that every system call is supported and that you test all of the inputs from the user program for correctness, etc.

Note: To receive a passing grade in CS230, your kernel *must* be able to boot up a shell prompt and run a minimal set of well-behaved programs! If your kernel cannot do this, you will receive an incomplete until you get it working at least that much.

The style portion of your project grade will be based on the quality of your documentation, both internal (comments) and external (README), as well as the cleanliness of your design and use of efficient algorithms.

What on earth is Project 2.4? Stay tuned...