CS230/330: Project 3
Booting to a Shell

Handed out: Thursday, February 8, 2001
Due: Friday, February 16, 11:59 p.m.

1 Assignment

Now that you've got virtual memory and context switching fully debugged and working perfectly, it's time to implement the rest of process management. Although this assignment is going to sound like a lot of work, it should be extremely straightforward if your memory management code is working.

2 Instructions

1. Finish Project 2 and the Project 2 supplement.
2. Implement the RFork() system call. RFork() has a number of options. For now, you only need to implement RFork(FORK_PROCESS) which works exactly like the normal Unix fork() system call.
3. Implement the Exec() system call. Hint: you already implemented 90% of it in load.c.
4. Implement the Wait() system call.
5. Implement the Exit() system call. When a process exits, you should collect its exit code and free its pages. However, the process does not officially die until the exit code is collected by the Wait() system call in the parent. Therefore, your kernel will need to maintain some kind of data structure in the PCB that keeps tracks all of the child processes that have called Exit().
6. Modify your kernel so that it loads a single process called 'init' at startup. Furthermore, you should modify your kernel so that it halts when there are no more processes to execute other than the idle process.
7. Implement the SerialRead() system call.
8. Implement the SerialWrite() system call.
9. Your Yalnix kernel should now be able to run a partial version of the Yash shell.

3 Additional Instructions

To implement threads (RFork(FORK_THREAD) and RFork(FORK_EXEC)), you will need to make some modifications to your kernel memory manager. Do not attempt to make these changes unless you're confident that everything else is fully working.

1. Meditate for a few moments, read the big Yalnix handout, take a deep breath and decide if your kernel is going to support threads or not. Although there will be a modest penalty for not implementing threads, I would advise you not to attempt this part of the project unless project 2 is fairly solid and you feel confident about the rest of the kernel project.
2. Modify your physical memory manager so that each physical page has a reference count. You will need to store the reference count somewhere in the kernel.
3. Modify GetFreePage() so that it returns a page and sets its reference count to 1.
4. Modify ReleasePage() so that it decreases the reference count and only puts the page back on the free list if the reference count reaches zero.
5. Implement ReferencePage(int pfn) which takes a physical page number and increases its reference count. This will later be used to implement page sharing.
6. Implement int RefCount(int pfn) which returns the reference count for a given physical page frame.
7. Modify the kernel so that the same user page table structure can be shared by more than one PCB. To do this, you will also need to add referencing counting to user page tables.
8. Think about how you might implement the RFork(FORK_THREAD) system call.

4 Specific Details and Suggestions

- To implement RFork() you will need to create a new user process and copy all of its memory (note: you should perform a simple copy–do not attempt to do anything fancy like page-sharing unless you are really on top of everything else). To make this copy, you will effectively need to have two different region 1 page-tables active (one for the parent process and one for the child process). Since it is impossible to map two page tables simultaneously, you will need to come up with some alternative scheme. One approach is to grab an unused page table entry in the parent process and use it temporarily to copy one page at a time to the child.
- To implement Wait() and Exit(), you may want to put some kind of process queue in your PCB structure. This queue could be used to hold PCBs for child processes that have called Exit() but which have not been collected by Wait() (Zombies).

- If a process calls Exit() and its parent process is dead, the exit code should be discarded and the PCB destroyed.
- To implement SerialRead() and SerialWrite() you will need to have a variety of process queues and data structures for holding sleeping processes and received input. Please consult the handout.

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.

This project has a relatively short due-date. Do not wait until the last minute to start.