CS230/330: Project 3
I/O Devices, Memory Management

Handed out: February 5, 2002
Due: Friday, February 15, 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 start implementing a variety of system calls. This assignment should be extremely straightforward if your memory management and context switching code is working.

2 Instructions

1. Finish Project 2. If you are having trouble getting this to work, it is critical that you see us as soon as possible. Here is a short check-list:

   - Make sure the kernel has its own page table in region 0.
   - Make sure every process has its own PCB structure including the idle process.
   - Make sure every PCB has a pointer to a unique Region 1 page table.
   - Make sure your PCB has a CPU context frame structure. This structure must be at least MAX_CPU_CONTEXT_FRAME bytes in size.
   - When context switching, make sure you copy the entire CPU context frame. Just copying the PC and SP is not enough to do context switching. Please refer to the big yalnix handout.
   - Make sure you flush the TLB whenever you update a page table.

   DO NOT PROCEED BEYOND THIS STEP UNTIL YOU HAVE PROJECT 2 FULLY WORKING!.

2. Implement a queue structure and create the kernel ready queue. Put init1 and init2 on the ready queue and modify the TRAP_CLOCK handler to context switch by removing the current process and putting it onto the end of the ready queue. Then context-switch to the next process on the ready queue. If the ready queue is empty, context switch to the idle process.

3. Implement process memory management and the Brk() system call.

4. Modify your kernel so that processes can grow their stack. Note: when the stack grows beyond a page boundary, the process will generate a TRAP_MEMORY exception. Your kernel should catch this and allocate more memory to the process for its stack.

5. Implement the DisplayWrite() system call.
6. Implement the PenRead() system call.
7. Implement the SerialWrite() system call.
8. Implement the SerialRead() system call.
9. Implement the GetPid() system call.
10. Implement the Delay() system call.
11. Implement the Exit() system call. This should terminate a process by releasing its PCB structure, memory, and CPU context. The process will never run again.
12. Modify your kernel so that when there are no more user processes, it terminates by calling Halt(). The idle process is not a user process (so it doesn't count here).

3 Specific Details and Suggestions

- 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 big handout for some details of this.
- Try to keep your implementation as simple as possible.
- You will need to use queues in other parts of the kernel. Try to make your ready queue implementation generic so that you can reuse it in other parts of the kernel.
- Think about how you are going to test each part of the kernel. Make sure you test each component before proceeding to the next.

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 kernel on a variety of init1 and init2 test programs. These tests will be provided to you in advance to help with your testing. However, in order to run these tests, your kernel must be able to run programs and context switch. Get project 2 working!

5 Preview

Project 4 implements the rest of the kernel including Fork(), Exec(), Wait(), Yield(), and Pipe(). If you are bored, feel free to work ahead.