CS230/330: Project 2.1
Getting Started on the Yalnix Kernel

Handed out: Friday April 9, 1999

Due: Friday, April 23, 1999, 11:59 p.m.

1 Assignment

The purpose of this project is to develop the initial pieces of the Yalnix kernel. To do this, you must produce a kernel that initializes all of its interrupt and trap vectors, recognizes system calls, and runs an idle process. This project does not require a lot of programming, but it does require a reasonable understanding of the overall project. Make sure you have read the Yalnix kernel handout a few times before beginning this assignment.

Specifically, you must do the following:

1. Form a group of up to three people and let us know who is in your group and what name you want the group to be given (up to eight letters). We will use this name to create a Unix group for your project. If you don't know what a Unix group is and how it pertains to file security, please refer to the information in the man pages for `chmod`, `chgrp`, and `newgrp`. **You must form your group by Friday, April 16th!**

2. We will create a directory and repository for use by your group. After this directory has been created, modify your `.cshrc` file (or other login files) to set the `CVSROOT` environment variable to your group project repository. Please see Dustin or myself if you have any trouble setting this up. If you are doing the project by yourself, you don't need to do anything for this step.

3. Create a project directory "yalnix" in your group directory and check it into CVS.

4. Copy the file `/u0/yalnix/sample/Makefile.template` to the `yalnix` project and rename it to `Makefile`.

5. Write a `kernel.c` file that defines a rudimentary `KernelStart()` function that simply prints "Hello Yalnix" and then calls `exit()`. In addition, write a function `SetKernelBrk(caddr_t addr)` that prints the message "SetKernelBrk not implemented yet" and returns. Modify the `Makefile` to create the yalnix executable from this file. Run "yalnix" to test your kernel. If it works, the "Hello Yalnix" message should appear.

6. Remove the call to `exit()` from `KernelStart`. Insert a call to `BogusInitMemory(pmem_size)` at the beginning of `KernelStart`, passing in the physical memory size that is passed to `KernelStart`. `BogusInitMemory` is a temporary hack to set up basic memory mappings for the kernel (this call will be removed later). Don't forget to include a prototype for it like this:

- `extern void BogusInitMemory(int)`

7. Write dummy trap (interrupt) handling functions that simply prints out a message indicating the type of interrupt and the current value of the program counter. Modify `KernelStart` to initialize the exception vector to point at these dummy routines. Note: don't forget to load the address of the exception vector into one of the special machine registers. The output of each interrupt handler should appear something like this:

   ```
   TRAP_CLOCK : pc = 3265d8
   TRAP_MEMORY : pc = 3265d8
   ... etc ...
   ```

8. Write a simple in-kernel idle process that prints a message stating that it is executing each time around the idle loop. It should look something like this:

   ```
   void KernelIdleLoop() {
        while(1) {
             write(1,"Idling indeed\n",14);
             Pause();
        }
   }
   ```

9. Get yalnix to jump to this function after you exit `KernelStart`. To do this, you will need to set the program counter (PC) field in the context structure (passed to the `KernelStart` routine) to point to your `KernelIdleLoop` function. In addition, you will need to set the stack pointer (SP) field to a suitable location. After setting these values, simply return from KernelStart. Now, you should see the timer interrupt calling your exception handler every second, and your idle loop running. Congratulations, you have just created the first Yalnix process!

10. To finish the project, implement dummy handling functions for all of the system calls required in the project (FORK, EXEC, etc...). These handler functions should do nothing more than print the name of the system call and return. Modify the trap handler for `TRAP_KERNEL` to invoke the appropriate system call handler. You should also check for invalid/unknown system calls and print an appropriate error message. Note : the system call numbers can be found in the file `/u0/yalnix/include/yalnixcalls.h`.

11. To test all of your trap vectors and system calls, you need to run your kernel using the special "yalnixping" command as follows (note: this command should only be used for this step of the assignment):

    ```
    % /u0/yalnix/bin/yalnixping yalnix
    ```

    This will "ping" your kernel with all of the required interrupts, traps, and system calls. If you have done everything correctly, your kernel will print a series of messages for each trap and system call. At this point you can relax–you have resolved all of the possible entry points to your kernel.

2 Testing

A reference kernel is available in

- /u0/yalnix/kernel/yalnix

You can run this kernel to view the desired behavior of what you will be implementing (note : you must run the reference kernel from your own directory). The correct output of the kernel ping test should appear something like this:

```
% /u0/yalnix/bin/yalnixping yalnix
*** Booting the kernel...
BogusInitMemory: pmem_size = 2097152 (00200000).
&end 0002ac58, &etext 000162e0, sbrk(0) 0002e000.
pmem_size: 2097152 (00200000).  Num pages: 256.
Making first 12 pages executable.
Loading initial kernel page table.
Starting Yalnix!

*** Pinging the interrupt vectors...
TRAP_CLOCK : pc = 12180
TRAP_ILLEGAL : pc = 12180
TRAP_MEMORY : pc = 12180
TRAP_MATH : pc = 12180
TRAP_TTY_RECEIVE : pc = 12180
TRAP_TTY_TRANSMIT : pc = 12180

*** Testing system calls...
TRAP_KERNEL : pc = 12180
Fork()
TRAP_KERNEL : pc = 12180
Exec()
TRAP_KERNEL : pc = 12180
Exit()
TRAP_KERNEL : pc = 12180
Wait()
TRAP_KERNEL : pc = 12180
Brk()
TRAP_KERNEL : pc = 12180
Delay()
TRAP_KERNEL : pc = 12180
TtyRead()
TRAP_KERNEL : pc = 12180
TtyWrite()
TRAP_KERNEL : pc = 12180
GetPid()
```

```
TRAP_KERNEL : pc = 12180
Register()
TRAP_KERNEL : pc = 12180
Send()
TRAP_KERNEL : pc = 12180
Receive()
TRAP_KERNEL : pc = 12180
Reply()
TRAP_KERNEL : pc = 12180
CopyFrom()
TRAP_KERNEL : pc = 12180
CopyTo()
TRAP_KERNEL : pc = 12180
Unrecognized system call!  13

*** Done with project2.1 test.

** Machine Halted **
```

3 Grading

This project must be handed in via CVS. Make sure you have committed all of the files needed to recreate your kernel including source code, header files, and makefiles. The Makefile should be build an executable called "yalnix." Furthermore, you should commit any external documentation you have written (e.g., a README file) and comment your code thoroughly and clearly. Make sure all team member names appear in the README file and all source files.

This part of the project will be graded solely on correctness. It is worth 20% of the final project 2 grade. The following grading criteria will be used:

- Printing the "Hello Yalnix" message (25%).
- Initializing the interrupt/trap vector correctly and printing the correct interrupt messages in the exception handler (25%).
- Initializing the idle process and having it print the correct messages (25%).
- Modifying the `TRAP_KERNEL` handler to invoke functions for all of the required system calls and printing the correct system call messages (25%).
- **If we cannot compile your kernel, you will receive NO CREDIT!**

A variable amount of credit will be subtracted for missed or spurious messages.