CS230/330: Project 4
I/O Devices

Handed out: Friday, Monday, May 8, 2000
Due: Friday, May 19, 2000, 11:59 p.m.

1 Assignment

Now that you have virtual memory and processes working, you are in the home stretch. In this project, your task is to bring up the kernel I/O subsystem and to implement drivers for various I/O devices. Once you have this completed, your kernel will be able to boot to shell, run graphical programs, and do other interesting things. Although none of this is extremely difficult (not like memory management), this project involves a large number of interconnected components and a layered design. You should plan to start early.

You will be working on all I/O devices except the flashcard. Do not work on that until you get all of the other devices working first.

2 Instructions

1. If you are still working parts of project 2 and project 3, get that to work first.
2. If you haven't done so already, create a separate file that contains handler functions for all of the high-level I/O syscalls supported by your kernel. You have already written these handlers as part of project 1 so this part should be pretty easy.
3. For each of the display, pen, and serial devices create a separate file in which you will implement the low-level I/O operations for that device.
4. In each file created in the previous step, create a handler function for each operation that the device needs to support (for example, the LCD display needs to support the Open(), Write(), and Close() operations). The list of required operations can be found in the Yalnix handout. In each function, simply print a message and return. For example:

```
void DisplayWrite(struct cpu_context_frame *context) {
    printf("display: Write()\n");
    context->regs[0] = 0;
    return;
}
```

5. Create a virtual method structure that contains function pointers for all of the I/O system calls that must be supported by your Yalnix kernel. For example:

```
struct IOMethods {
    void (*open)(struct cpu_context_frame *);
    void (*close)(struct cpu_context_frame *);
    void (*read)(struct cpu_context_frame *);
    void (*write)(struct cpu_context_frame *);
    void (*seek)(struct cpu_context_frame *);
    void (*unlink)(struct cpu_context_frame *);
    void (*sync)(struct cpu_context_frame *);
};
```

Note: Your structure may vary depending on how you want to write the low-level functions (you can use different arguments if you want to).

6. In each of your device driver files, create a variable of type `IOMethods` that sets all of the above fields to the matching functions in your device driver. If the device doesn't support a particular operation, the field should be set to 0. For example:

```
static IOMethods DisplayMethods = {
   DisplayOpen,        /* open */
   DisplayClose,       /* close */
   0,                  /* read */
   DisplayWrite,       /* write */
   0,                  /* seek */
   0,                  /* unlink */
   0,                  /* sync */
};
```

7. Define a new data structure `OpenFile` that contains a pointer to an `IOMethods` structure, a reference count, and any other information you think is useful in your implementation. For example:

```
struct OpenFile {
     struct IOMethods *methods;
     int    refcnt;
     ... whatever ...
};
```

8. Modify each of the device-specific Open() functions to return an initialized `OpenFile` structure on success. For example:

```
struct OpenFile *DisplayOpen(struct cpu_context_frame *context) {
   struct OpenFile *of;
   ...
   of = (struct OpenFile *) malloc(sizeof(struct OpenFile));
```

```
    of->methods = DisplayMethods;
    of->refcnt = 1;
    return of;
}
```

9. Create an in-kernel array of `OpenFile` pointers to serve as the kernel open file table. The size of this array is `MAX_KERNEL_FILE` (found in `yalnix.h`).

10. Create an array of integer file descriptors in your process PCB structure. The size of this array is `MAX_USER_FILE` (found in `yalnix.h`).

11. Implement the Open() system call. Your implementation should look at the passed device name (e.g., "/dev/display", "/dev/pen", etc...) and call the corresponding low-level Open() function specific to the device. On return, you should take the returned `OpenFile` pointer and place it in any free location in the kernel open file table. Then, take the table index and place it into the first free location in the process file descriptor table. Finally, return an integer to the user that corresponds to the index in the user file descriptor table that was used.

12. Implement each of the high-level I/O syscalls so that they take an integer file descriptor (passed as the first argument) and call the corresponding low-level I/O operation. To do this, you will first need to take the file descriptor and perform a lookup in the process file descriptor table. You will then use this value to perform a lookup of the `OpenFile` structure in the kernel open file table. Once you have this structure, you will invoke the low-level function using the `IOMethods` table. For example:

```
void WriteHandler(struct cpu_context_frame *frame) {
    int fd;
    struct OpenFile *of;
    fd = (int) frame->regs[0];

    /* Lookup OpenFile in your data structures... place in of */
    ...

    /* Call the low level operation */
    if (of->methods->write) {
        (*of->methods->write)(frame);
    } else {
        frame->regs[0] = ERROR_NOSUPP;
    }
}
```

13. Once you have made it this far, you will have a substantial portion of the I/O system working. You should now be able to test out your kernel with a programs that open various devices and cause the low-level I/O operations for each device to be called. For example:

```c
#include "yalnix.h"

int main() {
   int f, g;
   f = Open("/dev/display");
   g = Open("/dev/serial");
   Write(f,0,0);           /* Should call DisplayWrite() */
   Write(g,0,0);           /* Should call SerialWrite() */
   Read(g,0,0);            /* Should call SerialRead() */
   Read(f,0,0);            /* Should return ERROR_NOSUPP */
   ... etc ...
};
```

14. Implement the Close() system call. This function should decrement the reference count in corresponding `OpenFile` structure. If the reference count reaches zero, the low-level close method should be called and the `OpenFile` structure destroyed.

15. Modify your Fork() system call so that it duplicates the file descriptor table of the parent and it increases the reference count of all open devices in the parent.

16. Modify the Exit() system call so that it closes all of its open files before it is destroyed. Note: this only decreases the reference count and destroys the `OpenFile` structure if the reference count reaches zero.

17. Implement the device driver for the LCD display device.

18. Implement the device driver for the pen input device.

19. Implement the device driver for the serial device.

20. Implement the Dup2() system call.

21. Implement the device driver for pipes and the Pipe() system call.

22. Implement non-blocking I/O.

23. Stare at your kernel in complete disbelief.

3 Specific Details and Suggestions

- There are a lot of places where things can go wrong in this part of the assignment. You may want to perform argument checking and other sanity checks in the high-level I/O interface so that you don't have to repeat a lot of work in the device drivers.

- You will need to use your process queue structure to implement a number of the devices. Specifically, processes will have to wait on queues while waiting for I/O to complete.

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 `/export/home/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 various parts of your kernel.

In order to receive a passing grade for the class, your kernel must be able to boot up and run a simple shell. If you feel that you are behind, please come and see us.