ECS 150 Programming Project #1

Due date: 11:59 PM, April 23rd, 2007

In this programming project you will learn how to modify and compile a new FreeBSD kernel. You will also gain experience with the "kld" (Kernel Loadable Module) interface and use it to dynamically add a new system call to the FreeBSD operating system. Instruction on these topics will be given during discussion section and links to additional reference material will be posted on the course website shortly.

You must use the version 5.4 kernel to do this programming assignment. Please submit all kernel source code files that you modify, the Makefile for your kernel module, and a README file indicating what does or does not work. If you are working in a group, make sure both partners are listed in the README file. Please do NOT hand in the whole source tree or generated binaries.

To package your submission, please run the following command:

```
tar vcfz proj1-name.tar.gz <list of files to submit>
```

name should be the last name of the person submitting the project. This will generate a compressed tarball called `proj1-name.tar.gz`. You can double check the contents of your tarball by typing

```
tar vtfz proj1-name.tar.gz
```

(note the 'c' in the original command has now been replaced with a 't'). This will give you a listing of the files in the archive. Once you are satisfied that all files have been included, mail your tarball to `roper@cs.ucdavis.edu` with a subject line of "ECS 150 Project 1." Indicate your name and (if applicable) the name of your partner in the email body. Only one submission is required per group.

Part 1: Process Structure Modification

For the first part of your project, you must modify the FreeBSD process structure so that the operating system will track two additional pieces of information for each process, a string and an integer. The process structure is defined as `struct proc` in `/usr/src/sys/sys/proc.h`. Add two new fields: `p_category` (an integer) and `p_message` (a string buffer of size 30).

Part 2: KLD Syscall Implementation

The second part of your project is to create new system calls to get and set the new `p_category` and `p_message` fields of the process structure. Specifically:

- `int setcat(pid_t pid, int cat);`
  This system call will set the `p_category` field to `cat` for the process with ID `pid`. Returns 0 on success, -1 on errors (e.g., invalid PID).

- `int getcat(pid_t pid);`
  This system call returns the `p_category` field for the process with ID `pid`. If `pid` is -1, returns the value for the current process.

- `int setmsg(pid_t pid, char* msg);`
  This system call will set the `p_message` string to the value pointed to by `msg`. Returns 0 on success, -1 on error (if the length of message exceeds the buffer size of 30, return -1 and take no other action). If `msg` is NULL, set the process' message to the empty string ("").

- `int printmsg();`
  If the current process has a non-null message, print it using the kernel's `uprintf()` function. (Note that the kernel's `printf` works a bit differently from a regular program's — the message will probably display on the console or in the system log, depending on how you have your system setup; `uprintf()` will print in the manner you expect.). If the `p_message` field for the current process holds the empty string, return -1 and don't print anything.