Overview: For this final project you will implement one or more system calls in the iPodLinux kernel and provide user-level test program(s) for testing your system calls.

This project must be done individually, without assistance from anyone else but me. You are free to use any textbooks or online resources that you may find, but you are not allowed to copy code, and you must cite any resources that you used.

Minimum Requirements: To meet the minimum requirements for this project, you must implement a system call that mimics the `ps` (process status) command — man `ps` for more information.

More specifically, for all processes currently executing in the system, for each process you must list the process id, user id, group id, process start time, and process name. Recall that the process image struct `task_struct` is located in `include/linux/sched.h`. Much of the information to be reported can be found therein, including cycling through the list of active processes. You may choose, but are not required, to sort the output by process id.

Your output must be returned to the user-level program through a single char* parameter. For successful completion of the system call, it is the user's responsibility to have allocated sufficient space for the buffer into which your system call will place its output. However, you are required in your system call to check if sufficient space exists at the user level before you attempt to write to the user level (i.e., use `verify_area`). You system call must include an appropriate return value. Definition of this return value is at your discretion, but generally 0 indicates success and -1 indicates failure.

Upon successful completion of the system call, the user level program should be able to print the contents of the string parameter directly and show well-formatted output. In other words, it is up to your kernel-level system call to handle formatting in the string returned to the user-level. The user-level program must be able to invoke your system call similar to the following:

```c
int returnValue = syscall(SYS_NUM, psString);
if (!returnValue)
    printf("%s", psString);
```

with resulting well-formatted output that looks something like the following:

```
pid  uid  gid  Time  Name
 0    0    0     0    swapper
 1    0    0     25   init
 2    0    0     26   keventd
   ...  
41    0    0    13553  userTest
```

(Note that, for iPodLinux, all processes will be run as root, so the user id and group id will always correspond to root — i.e., zero.)

Facts You Need To Know:

- A pointer to the current process is readily available (see `include/asm-arch/armnommu/current.h`).
- To allocate space within the kernel, use `vmalloc` (see `include/linux/vmalloc.h`). For a relatively readable article on Linux kernel memory allocation, see [http://www.linuxjournal.com/article/6930](http://www.linuxjournal.com/article/6930).
• Be very careful to match data types. Gaffes that you can get away with in user space can wreak havoc when done in kernel space. To interpret many of the named data types, see include/linux/types.h and include/asm/posix_types.h.
• To test your user-level program, use the File Browser (accessible from the top-level podzilla menu). Navigate to your program, press and hold the center button, and select Execute from the next level menu. Choose Read Output at the next level — if you see lots of error messages, you have likely mismanaged memory at the kernel level.
• Following is a very handy link for searching for items within the source code: http://lxr.linux.no/linux-old+v2.4.24/ This resource may not match our source code exactly, but it works much better than grep.

Possible Extra Credit (in the projects category):
• Enhance your system call by displaying meaningful information garnered from the inode, dentry, and/or file objects discussed in Chapter 12: The Virtual Filesystem of Robert Love’s book. Note that for a given dentry object, the d_path routine in include/linux/sched.h will write a full pathname into a buffer. Note that this stuffs the information into the end of the provided buffer.
• Enhance your system call by printing the user name and group name instead of ids. Take a look at /etc/passwd and /etc/group on one of the Linux machines for an idea on how to get at this information. You may have to create/modify the existing files on the iPod. In addition to man pages, the following link may be a helpful resource for you on this: http://yolinux.com/TUTORIALS/LinuxTutorialManagingGroups.html
• Enhance your system call by displaying elapsed time rather than process start time.
• Provide a separate system call that displays segment information similar to that you encountered in Nachos. The following link will be a helpful resource http://linuxgazette.net/112/krishnakumar.html

If you attempt any of these, be sure to clearly describe what you’ve done in your README.

Write-Up: For this project, submit a brief README that describes your system call(s), its(their) parameters and usage, your user-level test programs, and citations of any references you use. Drop this README in the topmost linux-2.4.24-ipod2/ directory before gzipping for submission.

Submission: Create a gzipped tarball of your iPodLinux kernel:
    tar -czvf iPodLinux-ibarnett-submit.tgz linux-2.4.24-ipod2/
and submit the tarball to the ~lbarnett/dropbox directory on the Linux machines. (Of course, substitute your name, so I can tell whose tarball is whose.)

Your work on this and all projects in this course is subject to the conditions of the Honor Code as described in the course syllabus.