ELEC 377 – Operating Systems

Week 4 – Lab 2 Tutorial

Lab 2

- Modules
- Provide extensions to the kernel
- Device Drivers
- File Systems
- Extra Functionality

Lab 2

```
int init_module() {
   .. do initialization stuff ..
   .. tell the kernel what we do...
   return success or failure;
}

void cleanup_module(){
   .. remove structures from the kernel..
}
```

Lab 2

```
int init_module() {
    struct proc_dir_entry proc_entry;
    proc_entry  = create_proc_entry("lab2",0444,NULL);
    if (proc_entry == NULL) return failure
    proc_entry->read_proc = your read proc
    return success ;
}

void cleanup_module(){
    ... remove structures from the kernel..
}
```

File Access (From Disk)

c = getchar();

1 library asks the kernel for data from the disk using the read system call (asks for more data from current position)
2 system call handler calls kernel file system (procedure call)
3 Kernel is tracking current position in the file
4. Kernel file system module calls a lower level module to ask for the contents of a file starting from a given position (the current position)
5. low level module does disk io and returns data

File Access (From Disk)

c = getchar();

1 library asks the kernel for data from the disk using the read system call (asks for more data from current position)
2 system call handler calls kernel file system (procedure call)
3 Kernel is tracking current position in the file
4. Kernel file system module calls a lower level module to ask for the contents of a file starting from a given position (the current position)
5. low level module does disk io and returns data

File Access (From Disk)

sequence:

1. Ask low level for 1024 bytes starting from location 0.
2. Low level returns 512 bytes (bytes 0-511)
3. Ask low level for 1024 bytes starting from location 512
4. Low level returns 1024 bytes (bytes 512 to 1535)
5. Ask Low level for 1024 bytes starting from location 1536
6. Low level returns 309 bytes (bytes 1536-1844)
7. Ask Low level for 1024 bytes starting at location 1845
8. Return 0, no more bytes
9. kernel tells higher level process that the end of the file has been reached

Lab 2

Not getting data from the disk

- generating data on the fly
- sort of like web sites with dynamic content generation
- kernel asks for data in the virtual (non-existent) file starting from a particular offset
- when starting, the kernel asks for offset 0

- Large part of lab is research. Look at struct task_struct in sched.h, and read the comments
- /proc/ksym
- /boot/System.map

Linux Processes Scheduling

- Old Scheduling algorithm
  - ◊ Is the version used in your lab
  - ◊ 2.6 kernel uses a new O(1) scheduling algorithm with real time extensions
  - ◊ nr_running is now a routine, and is not in the export lists. nr_threads doesn't exist
  - ◊ Lab2 only works on unpatched 2.4 kernels with the old scheduling algorithm. So watch out if you do the lab at home!!

Lab 2

sched.h - definitions of scheduling structures

```
struct task_struct {
    …
    pid_t pid;
    …
    int nice;
    …
};
```

Lab 2

/proc protocol - Linux docs confusing

◊ *eof does not mean end of file, it means end of current request

◊ Program such as cat or more attempts to read a file
- calls read(fd,buffer,length) <- system call
- kernel calls proc file system reader
- proc file system reader call your read_proc function
- if your read_proc function returns less than the length of the system buffer, it is called again for more data.
- you return 0 to indicate end of file
- setting *eof to true means don't call read_proc again until kernel is called again.

Lab 2

/proc protocol – allows to handle breaks over buffer boundaries.

Have a /proc file that returns

Number of Running Process: xxx
Number of Running Threads:
PID UID Nice
1 YY XX

The boundary (max length) for a buffer might be in the middle of a line. You would have to write part of the info to the buffer and save the rest for the next call.

*eof and *start allow you to break the system call at a meaningful point in the file(*start= page; *eof=1);

Lab 2

Extra Info

– Some PCBs in the list are unused (pid == 0), skip them

– Sometimes the first PCB in the list has pid == 0, so may lead to infinite loop

```
if (file_pos == 0) {
    .. write header ...
   theTask = &init_task;
   lastTask = theTask;
}
```

- use do{}while(); loop to advance the task in the else

Testing

Nice varies from -20 (highest priority) to 19 (lowest priority)
What is the default priority?

The unix nice command allows to change the priority (use the command ‘man nice’).
Any user can start a command with a lower priority
Only root can give a higher priority
Start up some long running commands (e.g. an editor) using the nice command with various priorities so that the output (both your module and the ps command) have various priorities.