Operating Systems

Practical Coursework

Tom Spink

January 2016

Coursework

Two distinct parts:

- Shell (**due:** Thursday 4th February 16:00)
- Kernel Module (**due:** Thursday 17th March 16:00)

Coursework

Two distinct parts:

- Shell (**due:** Thursday 4th February 16:00)
- Kernel Module (**due:** Thursday 17th March 16:00)

Shell

Due: Thursday 4th February 16:00

- Improve your C coding skills
- Understand some of the services provided by the OS to application developers

Kernel Module

Due: Thursday 17th March 16:00

- Directly interact with OS processes
- Understand Linux scheduler & process control

C Programming

- Use an IDE to help you
  - Netbeans
  - Eclipse
- Use the `man` command for help with syntax
- Use Google for your problems/issues (not for solutions!)
- Use classmates/piazza for general programming discussion

Standard Library IO Functions

- `printf`: Prints out a formatted string to the console
- `scanf`: Reads a formatted string in from the console
- `fgetc`: Reads a single character from a file stream
- `fgets`: Reads line of text from a file stream

- Special file streams are `stdin`, `stdout` and `stderr`

Standard Library String Manipulation

Strings are `NULL`-terminated character arrays

- `strlen`: Returns the length of a string.
- `strcpy`: Copies a string onto another string.
- `strcat`: Concatenates one string onto another.
- `strcmp`: Compares two strings (returns zero if they are equal).
- `strtok`: Iteratively return tokens from a string.
- `strsep`: Destructively return tokens from a string.

You MUST make sure the buffers backing your string are big enough for what you are trying to do, or use limiting functions.

Memory Allocation/Manipulation

- Stack allocation: Allocated on function entry, and automatically freed on function exit (but limited in size).
- `malloc`: Allocates a block of memory of the given size, without zeroing it.
- `calloc`: Allocates and zeroes a block of memory, for an array.
- `free`: Deallocates a block of memory previously allocated with `malloc` or `calloc`.
- `memcpy`: Copies a region of memory from one location into another.
- `memset`: Fills in a region of memory with the given byte.

Standard Library Process Control

- `fork`: Duplicates the calling process and execution continues in both the original (the *parent*) and new (the *child*) process.
- `exec`: Replaces the calling process *image* with a new process *image*.
- `wait`: Blocks the calling process until a child process raises a signal or terminates.
- `signal`: Defines a signal handler function to be called when the given signal is raised.

UNIX signals

- Are delivered asynchronously to the process, in response to some event that requires action.
- Default signal handler usually results in the process terminating.
- Majority of signals (with the notable exceptions being `SIGKILL` and `SIGSTOP`) can be trapped and handled.

Clarifications

- You are allowed to use `-lreadline`
  e.g. `gcc -Wall -lreadline myshell.c`
- You may (but are not required to) handle filenames with spaces in them.
- Any questions?

Clarifications

- You are allowed to use `-lreadline`
  e.g. `gcc -Wall -lreadline myshell.c`
- You may (but are not required to) handle filenames with spaces in them.
- Any questions?

Linux Kernel

- The Linux Kernel is of a monolithic design.
- It can be dynamically extended by the use of kernel modules.
  - Device Drivers (storage devices, USB devices, sound cards, graphics cards).
  - Algorithm Implementations (MD5, SHA{1,256,512}, GZIP).
  - Filesystems (EXT3, EXT4, BTRFS, NTFS, VFAT)
- Kernel features can be compiled in or out, or compiled as modules during the kernel build process.
- Kernel modules can also be compiled as standalone modules, in their own source tree.

Kernel Modules

- They have a unique name (the name of the module file, without the extension).
- The are deployed as a single, normal ELF binary file, with the extension `.ko`.
- Kernel modules must be compiled against the headers for the targeted kernel version.

- `make -C /lib/modules/`uname -r`/build M=$PWD`
- Need a `Kbuild` file to tell the Kernel build scripts how to build the module (i.e. what files the module is composed of)

Module Loading/Unloading

- `insmod` command will load a kernel module from a `.ko` file.
- `rmmod` command will unload the named kernel module.
  - You can (*try*) and forcibly unload a misbehaving module with `rmmod -f`.
  - If the process is hung and not killable, then you're out of luck and will need to reboot.
- `modprobe` command will load a kernel module from a name, by searching the system kernel module directories.

Initialisation/Destruction

- Modules are a service, they are loaded and stay resident until asked to do something.
- `module_init`: Defines the function that is called when the module is loaded.
- `module_exit`: Defines the function that is called when the module is unloaded.

- You cannot use the standard C-library - kernel modules are linked against the Kernel, not a C-library therefore C-library functions are not available.
- However, many (useful) C-library functions have been implemented for the Kernel - but not all.
- There isn't a `printf` - but there is `printk`.
- You cannot use floating point variables/arithmetic in Kernel code.
- You can only call exported Kernel functions.
- Memory allocation can be tricky, the closest approximation to `malloc` is `kmalloc` but if you need to allocate memory, try to allocate on the stack.
- Segfaults in your module will at best crash the process the Kernel is running in the context of, but could potentially crash the system if they occur in a Kernel thread.

- You cannot use the standard C-library - kernel modules are linked against the Kernel, not a C-library therefore C-library functions are not available.
- However, many (useful) C-library functions have been implemented for the Kernel - but not all.
- There isn't a `printf` - but there is `printk`.
- You cannot use floating point variables/arithmetic in Kernel code.
- You can only call exported Kernel functions.
- Memory allocation can be tricky, the closest approximation to `malloc` is `kmalloc` but if you need to allocate memory, try to allocate on the stack.
- Segfaults in your module will at best crash the process the Kernel is running in the context of, but could potentially crash the system if they occur in a Kernel thread.

- You cannot use the standard C-library - kernel modules are linked against the Kernel, not a C-library therefore C-library functions are not available.
- However, many (useful) C-library functions have been implemented for the Kernel - but not all.
- There isn't a `printf` - but there is `printk`.
- You cannot use floating point variables/arithmetic in Kernel code.
- You can only call exported Kernel functions.
- Memory allocation can be tricky, the closest approximation to `malloc` is `kmalloc` but if you need to allocate memory, try to allocate on the stack.
- Segfaults in your module will at best crash the process the Kernel is running in the context of, but could potentially crash the system if they occur in a Kernel thread.

- You cannot use the standard C-library - kernel modules are linked against the Kernel, not a C-library therefore C-library functions are not available.
- However, many (useful) C-library functions have been implemented for the Kernel - but not all.
- There isn't a `printf` - but there is `printk`.
- You cannot use floating point variables/arithmetic in Kernel code.
- You can only call exported Kernel functions.
- Memory allocation can be tricky, the closest approximation to `malloc` is `kmalloc` but if you need to allocate memory, try to allocate on the stack.
- Segfaults in your module will at best crash the process the Kernel is running in the context of, but could potentially crash the system if they occur in a Kernel thread.

- You cannot use the standard C-library - kernel modules are linked against the Kernel, not a C-library therefore C-library functions are not available.
- However, many (useful) C-library functions have been implemented for the Kernel - but not all.
- There isn't a `printf` - but there is `printk`.
- You cannot use floating point variables/arithmetic in Kernel code.
- You can only call exported Kernel functions.
- Memory allocation can be tricky, the closest approximation to `malloc` is `kmalloc` but if you need to allocate memory, try to allocate on the stack.
- Segfaults in your module will at best crash the process the Kernel is running in the context of, but could potentially crash the system if they occur in a Kernel thread.

- You cannot use the standard C-library - kernel modules are linked against the Kernel, not a C-library therefore C-library functions are not available.
- However, many (useful) C-library functions have been implemented for the Kernel - but not all.
- There isn't a `printf` - but there is `printk`.
- You cannot use floating point variables/arithmetic in Kernel code.
- You can only call exported Kernel functions.
- Memory allocation can be tricky, the closest approximation to `malloc` is `kmalloc` but if you need to allocate memory, try to allocate on the stack.
- Segfaults in your module will at best crash the process the Kernel is running in the context of, but could potentially crash the system if they occur in a Kernel thread.

- You cannot use the standard C-library - kernel modules are linked against the Kernel, not a C-library therefore C-library functions are not available.
- However, many (useful) C-library functions have been implemented for the Kernel - but not all.
- There isn't a `printf` - but there is `printk`.
- You cannot use floating point variables/arithmetic in Kernel code.
- You can only call exported Kernel functions.
- Memory allocation can be tricky, the closest approximation to `malloc` is `kmalloc` but if you need to allocate memory, try to allocate on the stack.
- Segfaults in your module will at best crash the process the Kernel is running in the context of, but could potentially crash the system if they occur in a Kernel thread.

Questions/Clarifications