System Call operation (syscall)

describe the steps of a system operation.

1. Syscall invoked via a special CPU instruction that triggers a software trap (software interrupt)
   ex. int 0x80: (used to invoke syscalls in intel-compatible processors)
   - lcall7: "execution domain specific" ← iBCS (Application binary interface)
   - lcall27: ← Solaris/x86 binaries
   - SYSENTER/SYSEXIT:
     Alternative ~~sys~~ software interrupt method for system call entry mechanism ~~implemented~~ available on PentiumII+ and compatible processors
2. Process making the syscall is interrupted.
3. Information needed to continue its execution later is saved (Process state saved)
4. Processor switches to higher privileged level
5. Processor determine the service being requested by the user-mode by examining the processor state and/or its stack
6. Execute the requested system call operation

when a system call is performed, what happens with the interrupt?

7. Process making the syscall may be "put to sleep" if the syscall involved blocking I/O (Any time you have access to I/O)

8. When syscall complete process is "woken up" (if needed)
(Being born from block state to the ready state. One in ready state the CPU process will pick the process)

9. Original process state is restored.
(Process saved will be taken back)

10. Processor switches back to lower (user) privilege level

11. Process return from sys call and continue execution

- When you make a normal function call, you save address on the stack, then you jump

- System calls are similar but there are additional steps, invocation of special instructions that variate between OSs

what happens to the process that makes a system call?

what is the difference between a syscall and a normal function call?

[redacted] (self-Study)

Intermediate Library to Invoke Syscalls

- To make it easier to invoke system calls, OS writters normally provide a library that sits between progrms and syster calls interface.
  ex: • libc: Overview of standard C libraries
  • glibc: GNU C libraries
- This library provide wrapper routines
  ex: truncate(): invokes the underlying "truncate" System call
- Wrapper function is thin, doing little work other than copying arguments to the rigth registers before invoking the system call, and then setting errno appropriately after the system call returns. (sometimes they do extra work before invoking system call)
- Wrappers hide the low-level detail of:
  - Preparing arguments
  - Passing arguments to kernel
  - Switching to supervisor mode (executing the index instruction done by wrapper level rutine)
  - Fetching and returning results to application (Return back to user program. transition between kernel space to user space
- Helps to reduce OS dependency and increase portability of Programs.

Where does the library sits?

What is a wrapper function?

What kind of help provide a wrapper function

Implementing System Calls

- uname -r [redacted]
- /proc/cpuinfo

Steps in writing a system call

<- arch/x86/ia32/ia32entry.S (for 32bit)

1. Create an entry for the system call in the kernel's sys_call_table.
   - User processes trapping to the kernel (through SYS_ENTER or int 0x80) find the syscall function by indexing into this table
   - arch/x86/include/asm/unistd_32.h
     #define __NR_foo
   - arch/x86/include/asm/unistd_64.h
     #define __NR_foo 295
     __SYSCALL(__NR_foo, sys_foo)

2. Write the system call code as a kernel function
   - Note: be careful when reading/writing to user space
   - Use copy_to_user or copy_from_user routines

   syst-call w/no arg and return value (void)

   syscall with one primitive argument

3. Generate/Use a user-level system call stub
   - Hides the complexity of making a system call for user applications

   Definition of stub: article containing only few sentences of text with is too short regarding the coverage of a subject

SYS_ENTER/SYS_EXIST Method

- Newest and faster method to make syscalls
- Faster technique to enter and exit kernel mode that "int 0x80" software interrupt

- Pthony: A name for some commands to be executed when you make an explicit request
  - used to avoid conflic with a file of the same name
  - used to improve performance.

Ask what stub means

Which are the steps to implement a syscall

What is sysenter/sysexit methods?

What is Pthony and what is it used for?

KERNEL Modules

What are the kernel modules?
What do they do?

- Allow code to be added to the kernel, dynamically
- Only those modules that are needed are loaded
- Enables independent development of drivers for different devices.
- Can insert modules while the system is working. This make kernel faster since it is smaller, and only load modules when needed

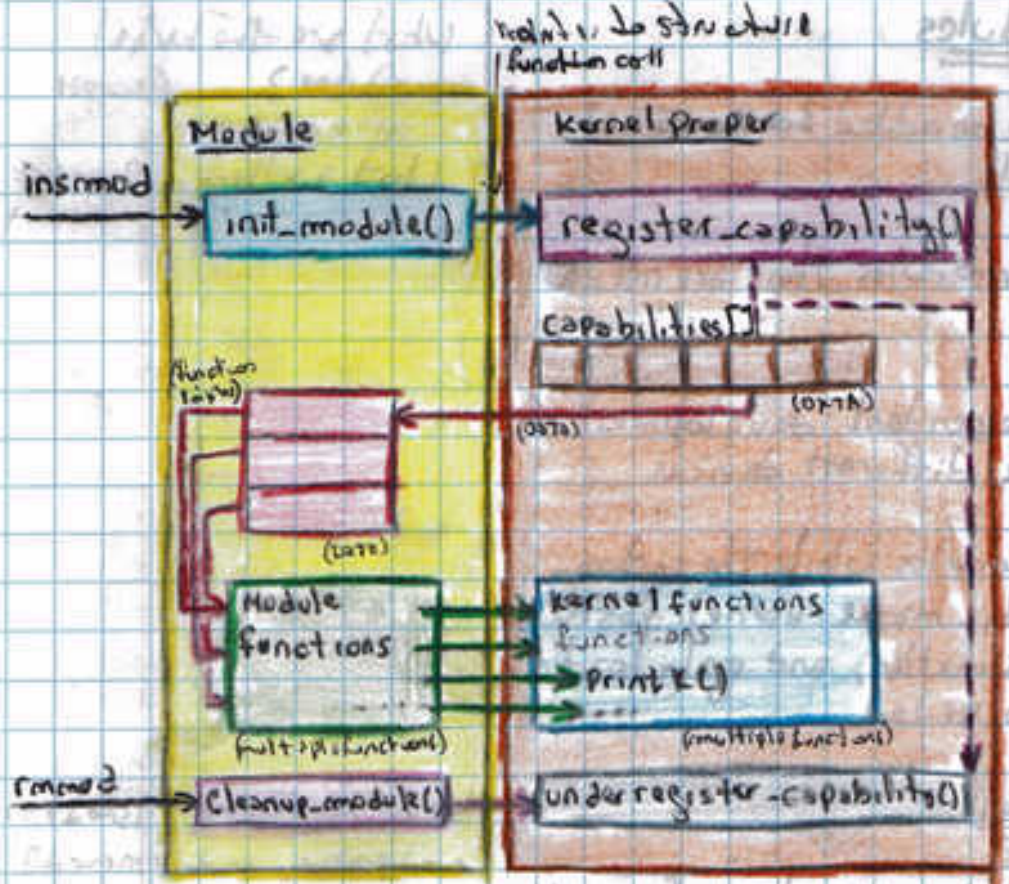
Module Control

What is lsmod, lsmod, modprobe, and rmmod?

- insmod: comand does insertion of a module into the linux kernel
- rmmod: command does removal of a module from the linux kernel
- lsmod: command shows the status of modules in the linux kernel
- Ext3 is a module (for example)
- You can compile some things as modules, some are going to be used in boot time to access to hardware, some will be used in the ram disks
  - Ramdisk will hold all the modules required in run-time ex. the disk driver or network driver in boot time.
  - the boot time speed is almost the same at time.

Where the modules are loaded to?

44 Workings of a Generic Module



- init_module(): introduce a new module to the system on which init_module will register some capabilities ("Hero I am, here's what I can do")

- register_capability(): Puts passing arguments (given) from init_module into an internal data structure. Then the system defines a protocol for how applications get access to the information in the capabilities data structure through system calls.

- Some functions are exported by the kernel's scheduling functions. Exporting means, that if some added third party modules, they can see these modules and call back and forward.

How does a generic module works?

← init_module call some register_capability function passing an argument a pointer to a structure containing pointers to functions within the module

What is init_module()?

What is register_capability?

What does it means a function being exported?

[redacted] scheduling

Do modules need to be running?

Workings of a generic module (continue)

- Modules are a bunch of code that doesn't need to be running
- If you decide the modules are not required by the system then remmod, then the kernel will clean, if not done it may call function that doesn't exist in the kernel and crash the kernel

Which are the different device classifications? describe each of them

Device Classification

- Character (char) devices: the way in which an application program gains access to the services of a character device driver is through the open() system call.
  - a serial port you can type bytes in and read a character
  - Block devices:
    - Difference with character is that you can move back and forth within the stream of characters
    - Data of block devices is usually transferred in one or more blocks at a time and pre fetched and cached
    - Block devices generally have a filesystem associated with them whereas do character devices don't
- Network devices:
  - they differ in that they handle "packets" of data for multiple protocol clients rather than a "stream" of data for a single client
  - it required a different interface between kernel and the device driver

Character Devices

- Abstraction: stream of bytes
- Support: open(), close(), read(), and write()
- Typical ex: /dev/console, /dev/ ttyS0
- Oddball ex: video frame grabbers: devices that captures individual, digital still frames from an analog video signal or a digital video stream
- At difference to a block device, we cannot go back from within a stream of characters. For example: in the serial device we can't go back to reread a character

Describes character device

Block Devices:

- Abstraction: array of storage blocks
- Support: open, close(), read(), write(), seek(), mount
- Typical Example: /dev/hda, /dev/sda, /dev/cdrom
- Block devices can go back and forth in a stream of characters

Describe a block device

Network Devices:

- Abstraction: Data packets
- Support: API for sending and receiving packets
- Support: multiple protocol and streams on one device
- typical example: eth0, eth1,
- Network devices usually have no corresponding filesystem representation in Linux
  - ex: no /dev/eth0 in filesystem
  - Instead use shin/ ifconfig to change network devices

Describe a Network Device

[redacted] self-study

Device classification

- Devices that don't fit in memory: char, block, or network devices

  - USB
    - Arguably a category by itself
    - other devices operation can be emulated on top of USB such as keyboard, mouse, printer, scanner, hard disks, cameras, flash drives, ect
  - SCSI - Like USB, SCSI can be used in a wide range of devices
    - most commonly used for hard drive and tape storage
    - Also to scanner, CD-rom, printers, etc
  - Firewire:
    - Most recent serial bus interface standard
    - replacing SCSI in many applications due to lower cost
    - ex. Camcorders, ipods
  - I2O: - typically for low-end embedded systems
    - very simple bus specification

Filesystem modules

- Are software drivers
- Are not device drivers
- serve as a layer between user API and block devices
- Are intended to be device independent
- generally Block devices have a filesystem associated with them whereas character devices don't
  /proc filesystem serves as a window into the kernel and its device drivers

48 # Implementing Kernel Modules

Kernel Modules

- Allow code to be added to the kernel, dynamically
- Only those modules that are needed reloaded
- Reduce kernel size
- Enables independent development of drivers for different devices

What are the advantages of modules?

Module Control

- Insmod ____.ko
  - Inserts a module (dynamic linker, call system module)
  - Internally, makes a call to sys_init_module
  - Call vmalloc() to allocate kernel memory (Allocate enough space for memory for module)
  - Copies modules binary to memory
  - Resolves any kernel references (eg: printk) via kernel symbol table (there is a table called symbol table with all the exported functions and kernel variables it will check any call called by the functions provided by the kernel)

What does insmod do?

- modprobe ____.ko
  - Same as insmod, except it also loads any other modules that ____.ko references
  - Since you may have a perfect match with the table for the module, the module may fail. Mod probe take care of inter modules dependencies.
    - If you want to look at inter modules dependencies, use lsmod. The modules name on the left are the one using all the modules on the right side.

What does modprobe do?

[redacted] self study

Module control (continue)

- rmmod ____.ko
  - remove a module
  - fails if module is still in use
    - need to remove all the dependencies before being able to remove the module
- lsmod
  - tell what modules are currently load
  - Internally read /proc/modules.

What does rmmod do?

What does lsmod do?

Thing to remember

- Modules can call exported kernel functions such as printk, kmalloc, ect
  - therefore cannot call functions that are not exported by the kernel
- Modules should not include standard libraries such as stdio.h, stdlib.h, ect...
  - NOTE: Very careful to not having bad pointers, if kernel access to invalid memory then kernel crashes
- Segmentation fault may be harmless in userspace but a kernel fault can crash the entire system
- Version Dependency:
  - Module should be recompiled for each version of kernel that is linked to.
    - If you compile in one version of the kernel, it may not work with another version of the kernel.

Name some kernel functions.

Do modules include standard libraries

What happens when there is a segmentation fault at kernel level?

What do you have to do when you update the kernel?

50

- Take care of concurrencies. When in program regular program, you have to flow from the top to the bottom, however, in kernel prgor you may not have a sequencial execution
  - After a module is call, it will put information into memory so it can be use by the kernel any time
  - very careful with data structures, if both functions access to a data struct at the same time without herirarchy access, you may encounter a problem, that is why we use locks.

what would happen if more than one function would try to access the same data structure

Concurrency Issues

- Many processes could try to access your module concurrently so different parts of your module may be active at the same time.
  - the difference between simultaneous process and concurrenting process:
    Ex: a. let say you have one cpu, but you can run 100 processes that looks like they run at the same time, this is concurrency
    b. But simultaneous, therefore in a table of execution, two processes can be running at the same time in different cpus therefore simultaneous means at the same time (this is related with threads)

what is the difference between simultaneous are concurrent

- Device interrupts can trigger Interrupt Service Routines (ISR)
  - ISRs may access common data that your module uses as well (these can be tricky)

what is ISR?

Concurrency Issues (continue)

- kernel timers can concurrently execute with your module and access common data.
  - kernel's timers are normally timers scheduled to execute in certain time.
  - these kernel's timers can be trigged at the same time, so it can create some bad conditions especially with multi-processor system

What are the kernel timers?

How do the kernel timers work?

- You may have symmetric multi-processor (SMP) system, so multiple processors may be executing your module code simultaneously (not just concurrently) ~~therefore~~

What do your module have to make sure to do?

- therefore, your module (and most kernel code in general) should re-entrant
  - capable of correctly executing in more than one context simultaneously
  - the correctness of the function is not get affected because is called many times at the same time. Same piece of code can get called again from ~~the~~ a different context.

Can you have concurrency with a single CPU?

- Even with a single CPU you can have concurrency because you can have a CPU interruption at the same time have a I/O interruption.

52 Version Numbering

- Kernel components have inter-dependencies
- Version numbers help to trace these dependencies
- Well written modules have minimal dependencies
  - they should be backwards compatible with all versions.
- Linux kernel version numbers:
  - Major. minor. release (eg: 2.6.20)
  - Odd numbered minor are for development only
  - Even numbers are supposed to be stable enough for general use

Do kernels have inter dependencies?

How do the version numbers work?

GNU General Public License (GPL)

- Basis for all GNU software development, including linux
  - MIT licence, you can modified code and not released
- GLP2, if you take a little code and modified it to called your own, you must released.
- GLP Allow users to modify software as see the need
- GLP3 is more restricted
- "Infects" transitively through modification and distribution by multiple parties
- Device drivers need not be licensed under GPL, but the mainstream ones are

What is GNU?

What is GPL?

What is the difference between GPL, GPL2, GPL3 and MIT licence?

Memory Management (CHP 4)

- Ideally programmers want memory that is
  - large
  - fast
  - non volatile

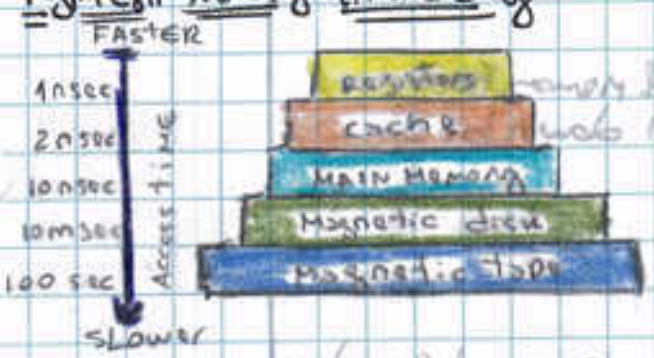
What does ideally programmers wants?

- Memory hierarchy
  - small amount of fast, expensive memory (cache)
  - some medium-speed, medium price memory
  - Gigabytes of slow, cheap disk storage

How is the memory hierarchy?

- Memory management handles the memory hierarchy

Typical Memory Hierarchy



Basic Memory Management

- Mono programming without swapping or Paging
  ↳ No memory abstraction - simple physical memory

Explain basic memory management

- Three ways of organizing memory:

Which are the three ways of organizing memory with basic memory management?

(a) OS at the bottom of memory ram
(b) OS at the top of memory ram
(c) OS device drivers at top of memory ram and rest of the OS in RAM below down below

- Generally only one process at a time can be running

How many process could be running generally?

54 (a) OS at bottom of memory in RAM (Random Access Mem)

- Used on mainframes and minicomputers

you have OS at the bottom in ram, what kind of computer would normally work with this memory?

(b) OS in ROM (Read-Only Memory) at top of memory

- Handheld computers and embedded systems

(c) Device drivers at top of memory (ROM), and the rest of OS at RAM down below

- Early personal computers

- One way to get some parallelism in a system with no memory abstraction is to program with multiple threads.
- limited use since people often want unrelated programs to be running at the same time
- Threads abstraction does not provide this

How is multiprogramming parallelism. Does it be one way to do parallelism in a system with no memory abstraction

[redacted] self study

Running Multiple Programs Without a Memory Abstraction

How can we run multiple programs without memory abstraction?

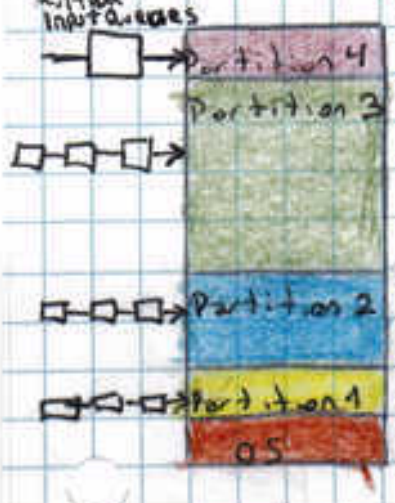
- OS has to save the entired content of memory to a disk file, then bring in and run the next program.
- As long as there is only one program at the time in memory, there are not conflicts (concept called swapping)
- Using special hardware it is possible to run multiple programs concurrently using a technique know as static relocation.
  - ↳ Static Relocation: When a program was loaded at address 16,384, the constant 16,384 was added to every program address during load process. While work is done quickly, it is not a very general solution and slow down loading

What is Static Relocation

Multiprogramming with Fixed Partitions

Explain the fixed multiple programming with fixed partitions

a) Separate input queues for each partition (MFT)



- Different size partitions
- Called cautious Multiprogramming with Fixed number of Tasks or OS/MFT
- Once partition has been a certain size then it remains at that size
- Job arrives it is placed in the input queue for the smallest partition that will accomodate it.

How do separate input queues work with fixed partitions

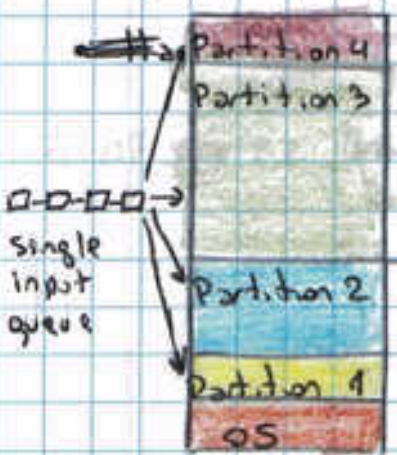
s6 Multiprogramming with fixed Partitions (continue)

- Drawbacks: (disadvantage)

1. As the partition sizes are fixed, any space not used by a particular job is lost
2. It may not be easy to state how big a partition a particular job needs
3. If job is placed in (say) queue three it may be prevented from running by other jobs waiting (and using) that partition

What are the drawbacks of separate input queues

b) Single input queue ~~(cont)~~



- Single input queue where all jobs are held
- When a partition is free, we search the queue looking for the first job that fits into the partition
- Alternative is to look for the largest job that fits into the partition:

What are the advantages and disadvantages of single input queue

→ advantage: not wasted large partition on a small job

→ Disadvantage: smaller jobs are discriminated against

Relocation and Protection

When introducing multiprogramming, two problems to address.

1. Relocation: When a program is run it does not know in advance what location it will be loaded at; therefore, the program cannot simply generate static addresses (ie. jump instruction). Instead, they must be made relative to where the program has to be loaded
   ↳ Address location of variables - code routines cannot be absolute.

   What is Relocation?

2. Protection: Once you have two programs in memory at the same time there is a danger that one program can write to the address space of another program which is dangerous and should be avoided
   ↳ Must keep a program out of other processes partition

   What is protection?

- Relative address in original program binary