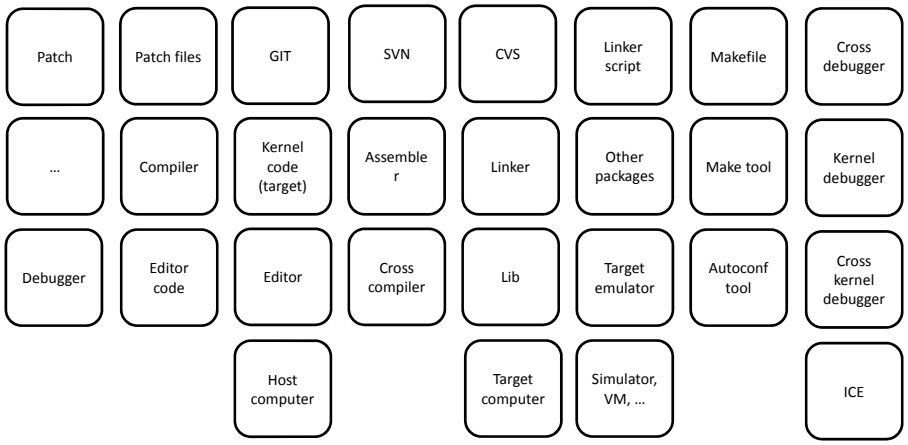
Operating System Design and Implementation

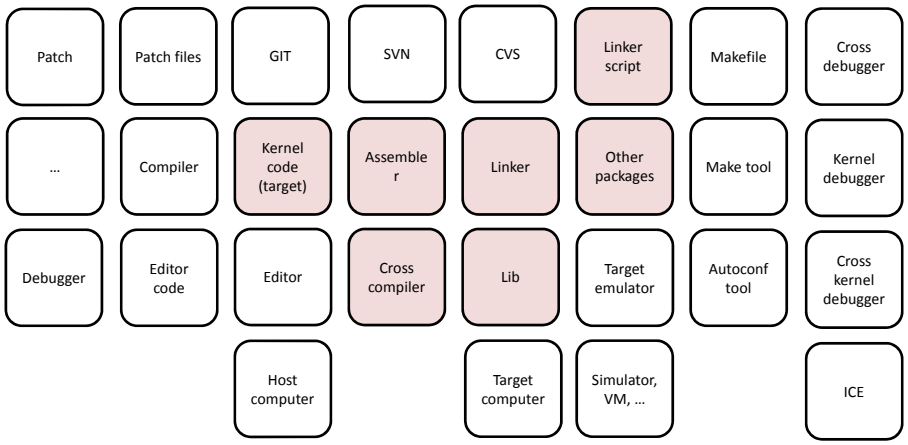
Getting started with kernel and kernel debugging

Shiao-Li Tsao

A Big Picture



Zoom In Compiling/Linking

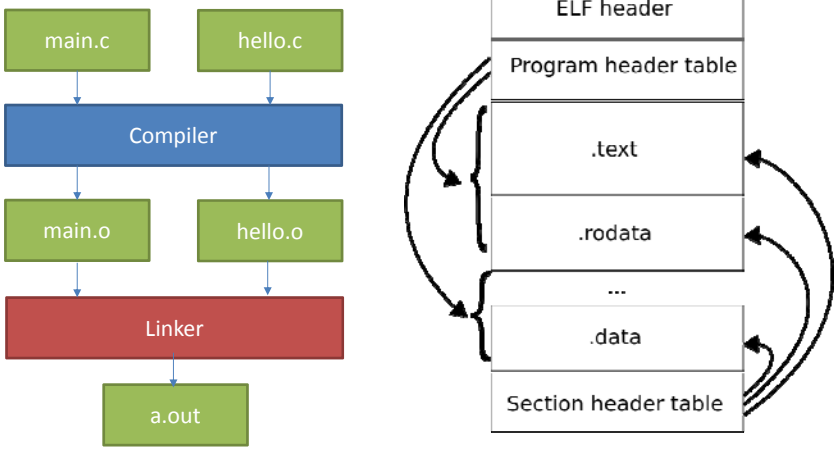


Linking

```
// main.c
//
void sayHello();

char str[6] = {'h', 'e', 'l', 'l', 'o', 0};

int main(int argc, char *argv[])
{
  sayHello();
  return 0;
}
```

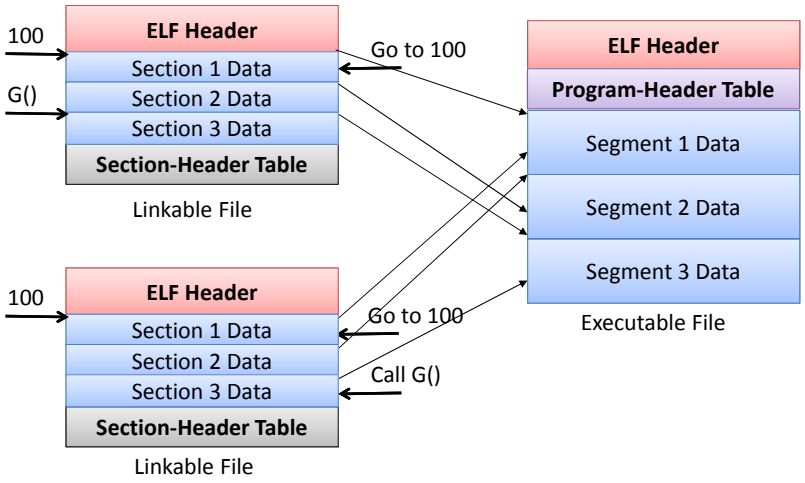
```
// hello.c
//
#include <stdio.h>

extern char str[6];

void sayHello() {
  printf("%s\n", str);
}
```

Linking



By Surueña - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=2922583

Linking

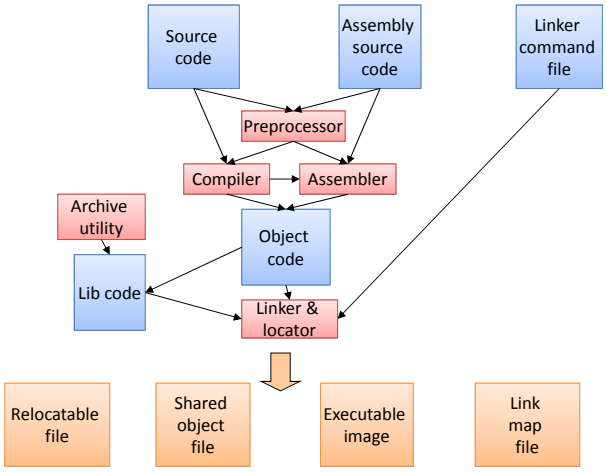


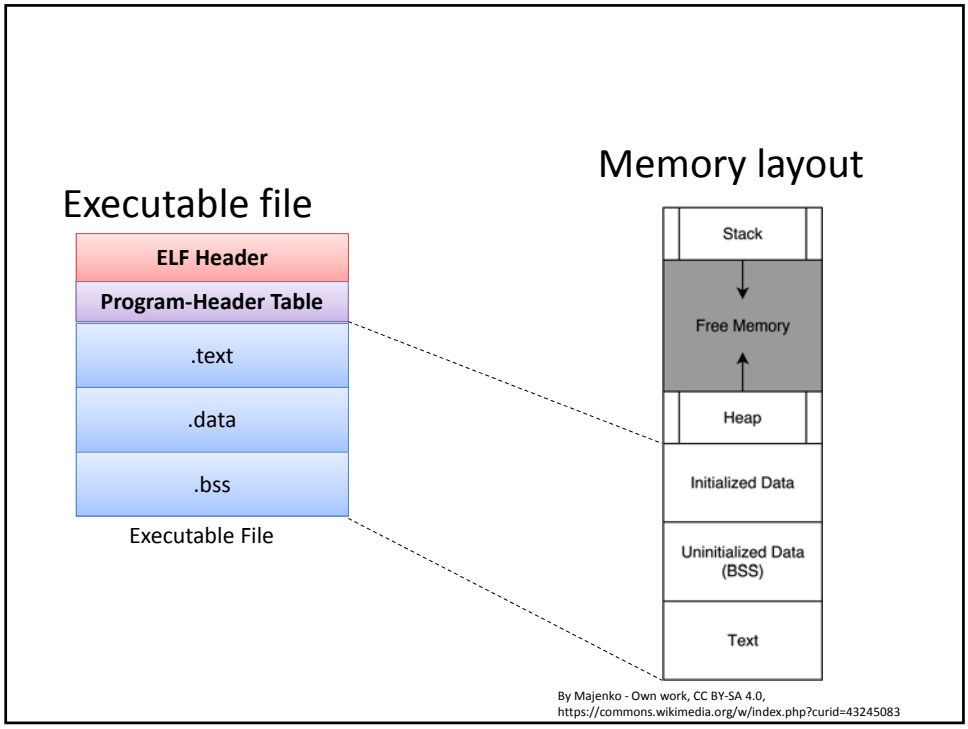
Linking

- Please review system programming and compiler if you are not familiar with below terms
  - Static linking
  - Dynamic linking
  - Relocations
  - Symbol table
  - Share library
  - Linking and loading

How to compile kernel codes

- Creating an executable image





Assembly, Machine code, Memory dump

Assembly ->

```
Dump of assembler code for function system_call:
   0x000076d8 <+0>:     cmp    $0x47,%eax
   0x000076db <+3>:     ja     0x76c8 <bad_sys_call>
   0x000076dd <+5>:     push   %ds
   0x000076de <+6>:     push   %es
   0x000076df <+7>:     push   %fs
   0x000076e1 <+9>:     push   %edx
=> 0x000076e2 <+10>:    push   %ecx
   0x000076e3 <+11>:    push   %ebx
   0x000076e4 <+12>:    mov    $0x10,%edx
   0x000076e9 <+17>:    mov    %edx,%ds
   0x000076eb <+19>:    mov    %edx,%es
   0x000076ed <+21>:    mov    $0x17,%edx
   0x000076f2 <+26>:    mov    %edx,%fs
   0x000076f4 <+28>:    call   *0x1a020(,%eax,4)
   0x000076fb <+35>:    push   %eax
   0x000076fc <+36>:    mov    0x1b140,%eax
   0x00007701 <+41>:    cmpl   $0x0,(%eax)
   0x00007704 <+44>:    jne    0x76ce <reschedule>
   0x00007706 <+46>:    cmpl   $0x0,0x4(%eax)
   0x0000770a <+50>:    je     0x76ce <reschedule>
End of assembler dump.
```

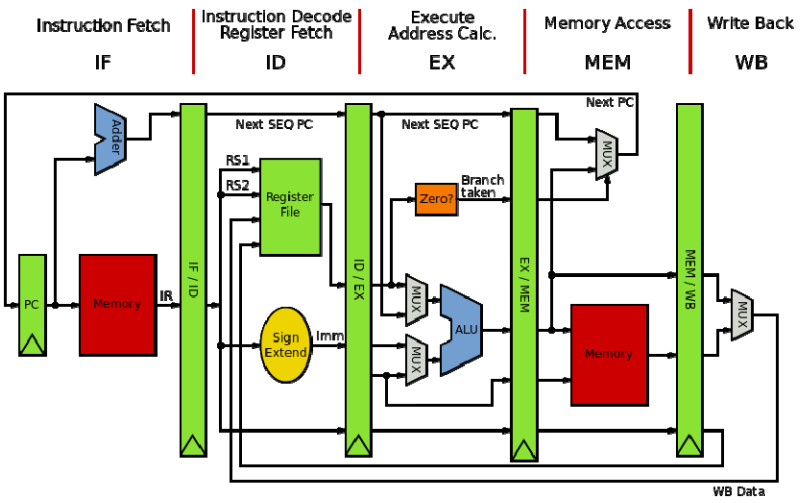
Memory dump

|

v

```
0x76d8 <system_call>:   131     248     71      119     235     30      6       15
0x76e0 <system_call+8>: 160     82      81      83      186     16      0       0
0x76e8 <system_call+16>:        0       142     218     142     194     186     23      0
0x76f0 <system_call+24>:        0       0       142     226     255     20      133     32
0x76f8 <system_call+32>:        160     1       0       80      161     64      177     1
0x7700 <system_call+40>:        0       131     56      0       117     200     131     120
0x7708 <system_call+48>:        4       0
```

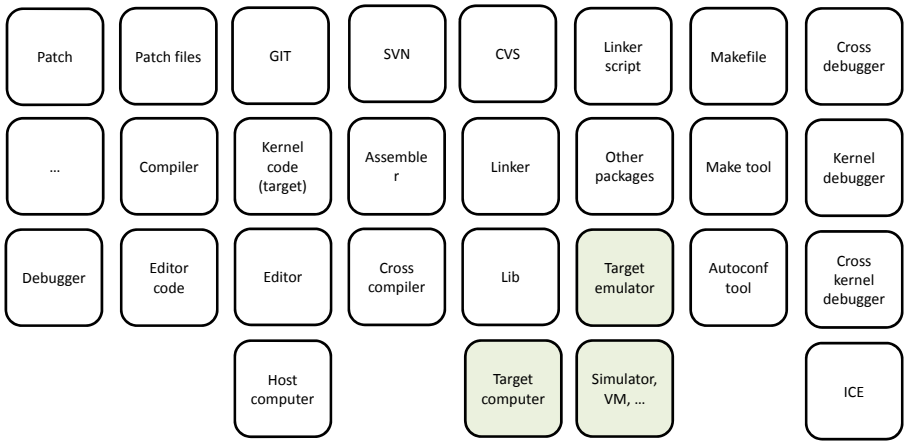
MIPS Core Dataflow Model



By Inductiveload - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=5769084

11

Emulator/Simulator/VM

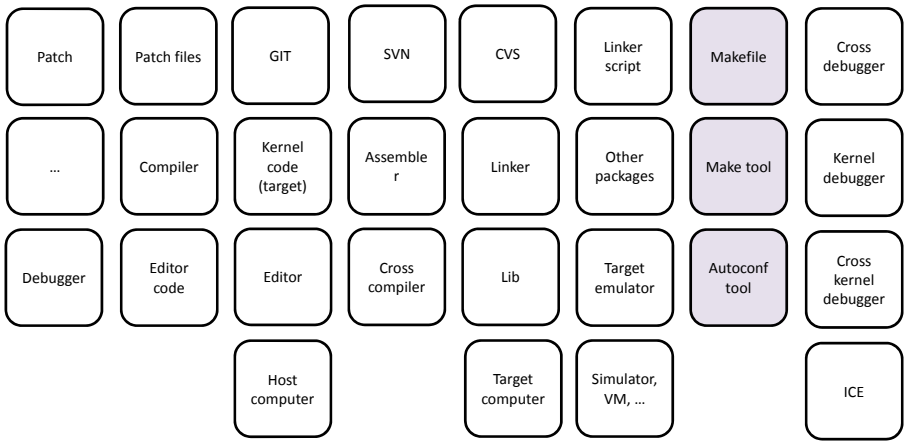


Hardware

- How to develop an operating system for a new processor
  - Simulator vs. emulator vs. virtual machine
- Simulator
  - A program to reproduce the behavior of a computer system based on an abstract model
- Emulator
  - Hardware or software or both that duplicates (or emulates) the functions of one computer system (the guest) in another computer system (the host), different from the first one, so that the emulated behavior closely resembles the behavior of the real system (the guest)[1]
- Virtual machine
  - A virtual machine (VM) is a software implementation of a machine (i.e. a computer) that executes programs like a physical machine[2]

[1,2] *http://en.wikipedia.org/*

Make/Makefiles



How to compile kernel codes

- 17,090 – The number of files in Linux 2.6.11
- 37,626 – The number of files in Linux 3.2
- How can I find and compile my network interface card driver among 100 network interface card drivers?
- Shall I recompile again when I modify one file?
- How can we produce kernel image ?

Make and Makefile

- Make: utility to provide a convenient facility to build, install, and uninstall projects
- Makefile: script file for make to compile and link programs

Make and Makefile

```
target ... : prerequisites ...
        recipe
        ...
        ...
```

```
edit : main.o kbd.o command.o display.o \
       insert.o search.o files.o utils.o
        cc -o edit main.o kbd.o command.o display.o \
                   insert.o search.o files.o utils.o

main.o : main.c defs.h
        cc -c main.c
kbd.o : kbd.c defs.h command.h
        cc -c kbd.c
command.o : command.c defs.h command.h
        cc -c command.c
display.o : display.c defs.h buffer.h
        cc -c display.c
insert.o : insert.c defs.h buffer.h
        cc -c insert.c
search.o : search.c defs.h buffer.h
        cc -c search.c
files.o : files.c defs.h buffer.h command.h
        cc -c files.c
utils.o : utils.c defs.h
        cc -c utils.c
clean :
        rm edit main.o kbd.o command.o display.o \
           insert.o search.o files.o utils.o
```

http://www.gnu.org/software/make/manual/make.pdf

Make and Makefile

```
objects = main.o kbd.o command.o display.o \
          insert.o search.o files.o utils.o

edit : $(objects)
        cc -o edit $(objects)
main.o : main.c defs.h
        cc -c main.c
kbd.o : kbd.c defs.h command.h
        cc -c kbd.c
command.o : command.c defs.h command.h
        cc -c command.c
display.o : display.c defs.h buffer.h
        cc -c display.c
insert.o : insert.c defs.h buffer.h
        cc -c insert.c
search.o : search.c defs.h buffer.h
        cc -c search.c
files.o : files.c defs.h buffer.h command.h
        cc -c files.c
utils.o : utils.c defs.h
        cc -c utils.c
clean :
        rm edit $(objects)
```

```
objects = main.o kbd.o command.o display.o \
          insert.o search.o files.o utils.o

edit : $(objects)
        cc -o edit $(objects)

main.o : defs.h
kbd.o : defs.h command.h
command.o : defs.h command.h
display.o : defs.h buffer.h
insert.o : defs.h buffer.h
search.o : defs.h buffer.h
files.o : defs.h buffer.h command.h
utils.o : defs.h

.PHONY : clean
clean :
        rm edit $(objects)
```

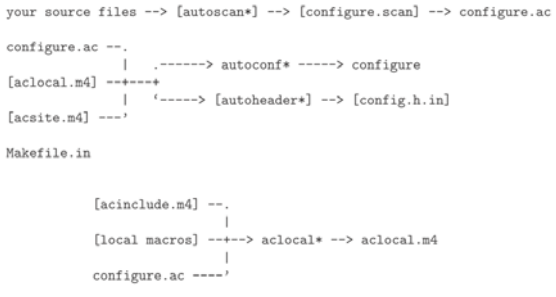
http://www.gnu.org/software/make/manual/make.pdf

Make and Makefile

- Variables and settings
  - make config
  - make menuconfig
- Phony targets
  - make all
  - make clean
  - make depend
  - make install
  - make uninstall

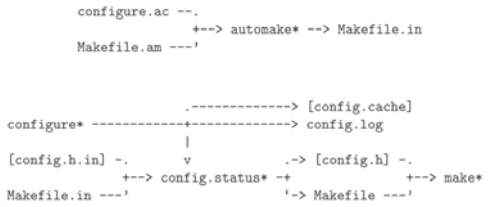
The portability problem

- Hardware differences
- OS differences
- Compiler differences



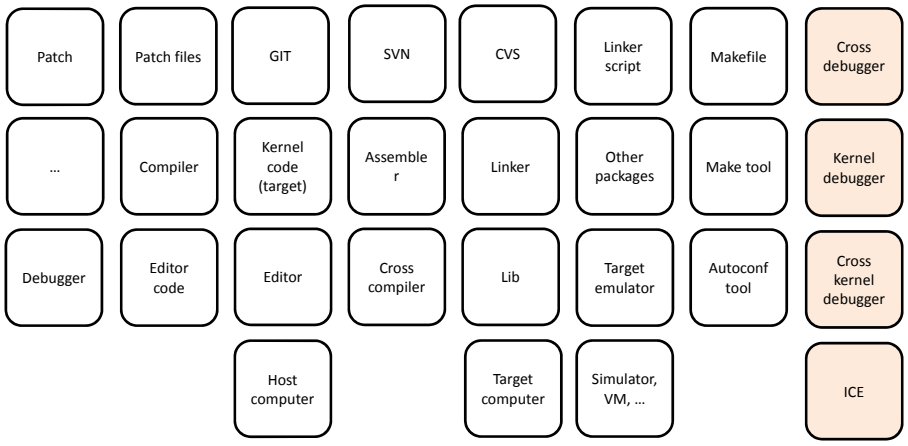
Autoconf Manual - The GNU Operating System

The portability problem

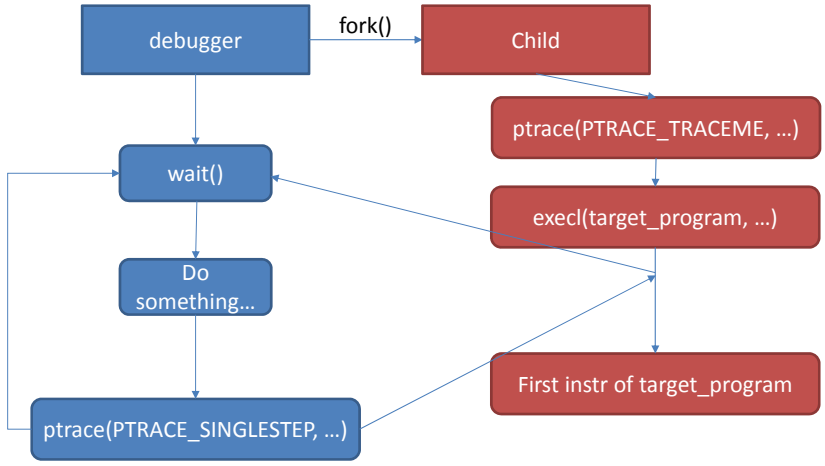


Autoconf Manual - The GNU Operating System

Debugging



How Debugger Works?



How Debugger Works?

Parent

```
void run_debugger(pid_t child_pid)
{
    int wait_status;
    unsigned icounter = 0;
    procmsg("debugger started\n");

    /* Wait for child to stop on its first instruction */
    wait(&wait_status);

    while (WIFSTOPPED(wait_status)) {
        icounter++;
        /* Make the child execute another instruction */
        if (ptrace(PTRACE_SINGLESTEP, child_pid, 0, 0) < 0) {
            perror("ptrace");
            return;
        }

        /* Wait for child to stop on its next instruction */
        wait(&wait_status);
    }

    procmsg("the child executed %u instructions\n", icounter);
}
```

Child

```
void run_target(const char* programname)
{
    procmsg("target started. will run '%s'\n", programname);

    /* Allow tracing of this process */
    if (ptrace(PTRACE_TRACEME, 0, 0, 0) < 0) {
        perror("ptrace");
        return;
    }

    /* Replace this process's image with the given program */
    execl(programname, programname, 0);
}
```

Source: http://eli.thegreenplace.net/2011/01/23/how-debuggers-work-part-1/

How Debugger Works?

- To understand how ptrace is implemented in concept on x86, please reference http://eli.thegreenplace.net/2011/01/27/how-debuggers-work-part-2-breakpoints

Kernel Debugger

- Why user debugger does not work anymore?
- How can we debug kernel?

- Kernel logs (discontinues logs)
  - Printk
  - Oops and Kallsyms
- Kernel debug supports
  - Kexec, kdump, SysRq
- Kernel hacking options
- Kernel debug tools
  - gdb, kgdb, kdb
- Profile
  - OProfile
- Trace
  - KFT, LTT/LTTng
  - Gprof
- Lock detection
  - Lockmeter
- Memory leaking
- Test equipment

Debugging and profiling device drivers

- printk()
  - Loglevels

Loglevel	Description
KERN_EMERG	An emergency condition; the system is probably dead
KERN_ALERT	A problem that requires immediate attention
KERN_CRIT	A critical condition
KERN_ERR	An error
KERN_WARNING	A warning
KERN_NOTICE	A normal, but perhaps noteworthy, condition
KERN_INFO	An informational message
KERN_DEBUG	A debug message typically superfluous

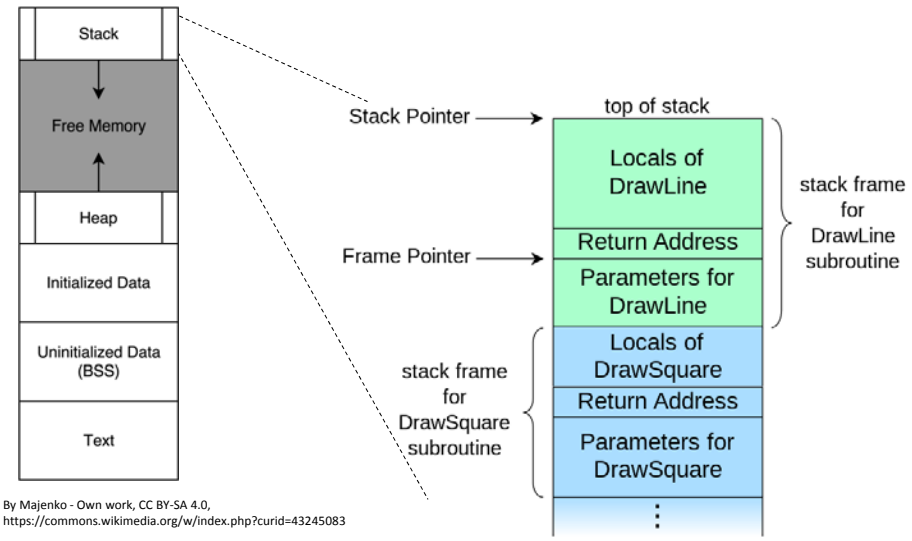
printk()

- Log buffer
- Klogd
  - /proc/kmsg or syslog()
- Syslogd
  - Appends all the messages it receives to a file
  - /var/log/messages
  - /etc/syslog.conf
- Not for booting stage debugger (early_printk())
- Not for debugging non-console case (via serial port)
- Not easy to detect racing condition

Kernel debugging options

- Turn on in kernel hacking/linuxconfig
- BUG()/ BUG_ON() case oops (stack trace, error message dump to kernel)
- Panic() prints error messages and halts the kernel
- dump_stack()

Call Stack



By Majenko - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=43245083

By R. S. Shaw (R. S. Shaw) - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=1956587

ksymoops

```
NIP: C013A7F0 LR: C013A7F0 SP: C0685E00 REGS: c0905d10 TRAP: 0700
Not tainted
MSR: 00089037 EE: 1 PR: 0 FP: 0 ME: 1 IR/DR: 11
TASK = c0712530[0] 'swapper' Last syscall: 120
GPR00: C013A7C0 C0295E00 C0231530 0000002F 00000001 C0380CB8 C0291B80 C02D0000
GPR08: 000012A0 00000000 00000000 C0292AA0 4020A088 00000000 00000000 00000000
GPR16: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
GPR24: 00000000 00000005 00000000 00001032 C3F7C000 00000032 FFFFFFFF C3F7C1C0
Call trace: [c013ab30] [c0020744] [c001b864] [c0007e80] [c00061c4]
[c0007b84] [c0007bf8] [c0003ae8]
```

- Ksymoops + system.map+module information
- Linux 2.6.X uses kallsyms

ksymoops

```
Oops: Exception in kernel mode, sig: 4
Unable to handle kernel NULL pointer dereference at virtual address 00000001

NIP: C013A7F0 LR: C013A7F0 SP: C0685E00 REGS: c0905d10 TRAP: 0700
Not tainted
MSR: 00089037 EE: 1 PR: 0 FP: 0 ME: 1 IR/DR: 11
TASK = c0712530[0] 'swapper' Last syscall: 120
GPR00: C013A7C0 C0295E00 C0231530 0000002F 00000001 C0380CB8 C0291B80 C02D0000
GPR08: 000012A0 00000000 00000000 C0292AA0 4020A088 00000000 00000000 00000000
GPR16: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
GPR24: 00000000 00000005 00000000 00001032 C3F7C000 00000032 FFFFFFFF C3F7C1C0
Call trace:
[c013ab30] tulip_timer+0x128/0x1c4
[c0020744] run_timer_softirq+0x10c/0x164
[c001b864] do_softirq+0x88/0x104
[c0007e80] timer_interrupt+0x284/0x298
[c00033c4] ret_from_except+0x0/0x34
[c0007b84] default_idle+0x20/0x60
[c0007bf8] cpu_idle+0x34/0x38
[c0003ae8] rest_init+0x24/0x34
```

Kernel hacking options

- Some kernel hacking options are architecture-dependent
- CONFIG_PRINTK_TIME: show Timing information on printks
- CONFIG_DEBUG_SLAB: debug slab memory allocations
- CONFIG_DEBUG_SPINLOCK: finds lock-related problems
- CONFIG_MAGIC_SYSRQ: Magic SysRq key
- CONFIG_DETECT_SOFTLOCKUP: detect tight loops in kernel code that last for more than 10 seconds

Kernel hacking options

- CONFIG_DEBUG_SLAB/CONFIG_DEBUG_HIMEM/CONFIG_DEBUG_PAGE_ALLOC : help debug memory management problems
- CONFIG_DEBUG_STACKOVERFLOW: warnings if the available stack space falls below a threshold
- CONFIG_DEBUG_STACK_USAGE): adds stack space instrumentation to the magic Sysrq key output
- CONFIG_DEBUG_BUGVERBOSE: verbose BUG() reporting
- CONFIG_KALLSYMS: debug an "oops" message

gdb

- Compile kernel with –g flag
- gdb vmlinux /proc/kcore
- Cannot modify the kernel data
- Cannot single-step
- Cannot set breakpoint

kgdb

- http://kgdb.linsyssoft.com/
- Remote debug
- Kernel patched + gdb (over serial line)
- Full gdb functions

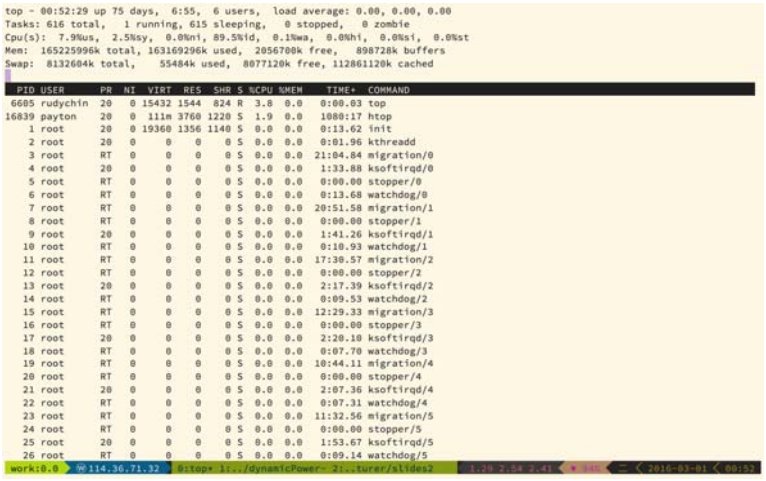
kdb

- http://oss.sgi.com/projects/kdb/
- built-in kernel debugger (not remote debugger)
- kernel patch
- support variable modification, breakpoints, and single-stepping, …

Linux Profile

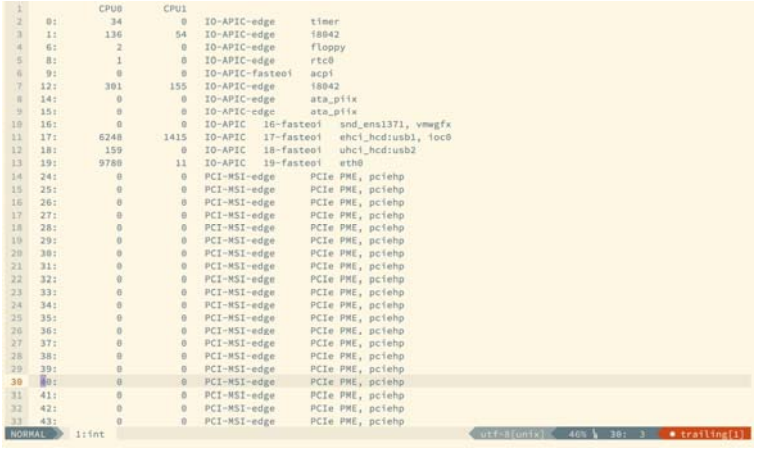
- Linux-built-in vs. 3rd party package
- Instrument vs. non-instrument
- Trace-based vs. Counting-based vs. Sampling based
- Kernel profiling vs. AP profiling

System Load Monitoring



System Load Monitoring

- /proc/interrupts



OProfile

- Can reference here for more info: http://oprofile.sourceforge.net/news/

KFT

- Be careful of interpreting results
  - Duration
    - Do not subtract interrupts and thread switching
  - Delta
    - The problem may be caused by child functions
- Good for debugging straight-line code
  - No block or lock by mutex/semaphores
- For more info, http://elinux.org/Kernel_Function_Trace

Gprof

- cc -g -c myprog.c utils.c -pg
- cc -o myprog myprog.o utils.o –pg

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
33.34	0.02	0.02	7208	0.00	0.00	open
16.67	0.03	0.01	244	0.04	0.12	offtime
16.67	0.04	0.01	8	1.25	1.25	memccpy
16.67	0.05	0.01	7	1.43	1.43	write
16.67	0.06	0.01				mcount
0.00	0.06	0.00	236	0.00	0.00	tzset
0.00	0.06	0.00	192	0.00	0.00	tolower
0.00	0.06	0.00	47	0.00	0.00	strlen
0.00	0.06	0.00	45	0.00	0.00	strchr
0.00	0.06	0.00	1	0.00	50.00	main
0.00	0.06	0.00	1	0.00	0.00	memcpy
0.00	0.06	0.00	1	0.00	10.11	print
0.00	0.06	0.00	1	0.00	0.00	profil
0.00	0.06	0.00	1	0.00	50.00	report

Linux Trace Toolkit Next Generation

- Reference here for more info, http://lttng.org/

Other useful information

- Linux Test Project
  - http://ltp.sourceforge.net/
  - a suite consisting of around 3,000 tests designed to exercise different parts of the kernel
- User Mode Linux
  - http://user-mode-linux.sourceforge.net/
  - lets you debug the kernel without "oops"ing the machine

lockmeter

- Lockmeter is a tool for instrumenting the spin locks in a multiprocessor Linux kernel
- http://oss.sgi.com/projects/lockmeter/

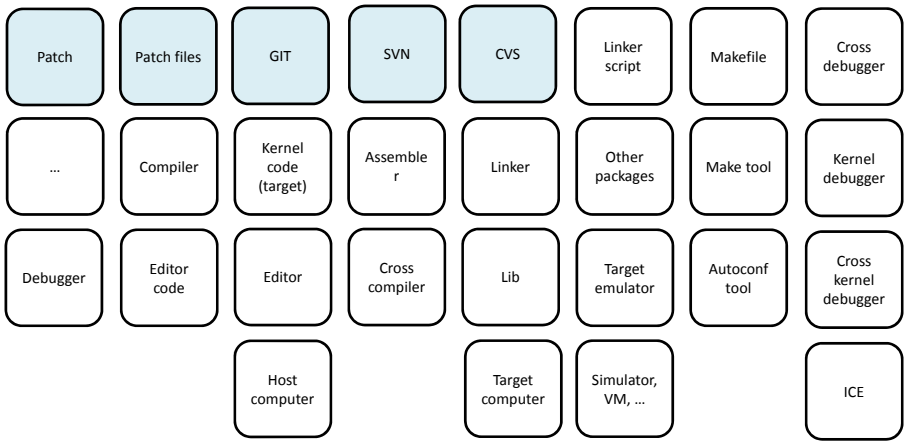
Memory leaking

- Kernel: kmemcheck
- http://free-electrons.com/kerneldoc/latest/kmemcheck.txt

Test equipment

- JTAG/ICE debugger
- Logical analyzer
- NIC
  - Sniffer
- USB
  - Protocol Analyzer
- …

SVN/CVS/GIT



How to maintain kernel codes

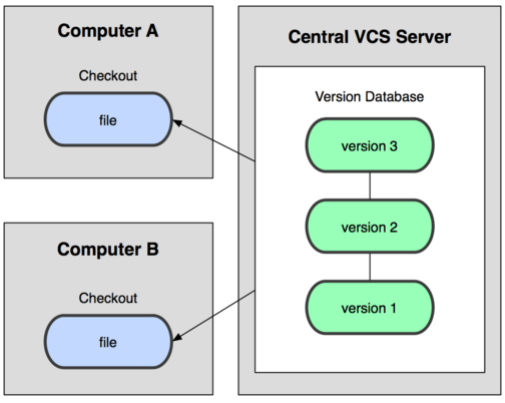
- Why version control?
  - Single developer
  - Multiple developers

Basic Work Cycle

- Update your working copy (check out your code)
- Make your changes (modify, add, remove, copy, move files/directories) on your working copy
- Review your changes you've made in your working copy
- Fix your mistakes (may start all over from unmodified state)
- Resolve any conflicts (merge others' changes)
- Publish (commit) your changes (lock and commit) Others can see your work, too!

Centralised Version Control

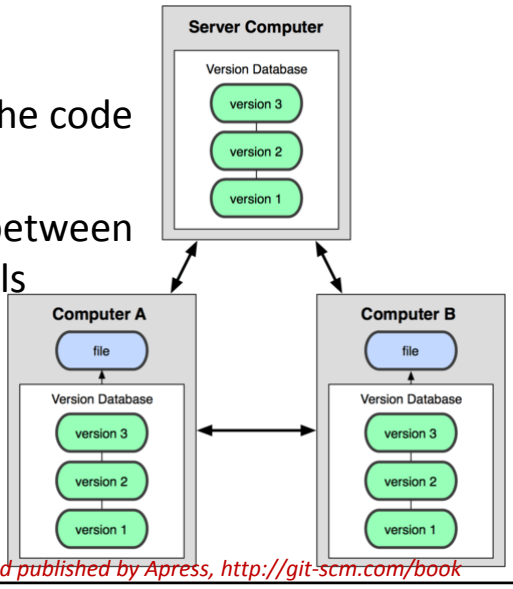
- One server holds the code base
- Clients access the server by check-in/check-outs
- CVS, SVN



Pro Git by Scott Chacon and Ben Straub and published by Apress, http://git-scm.com/book

Distributed Version Control

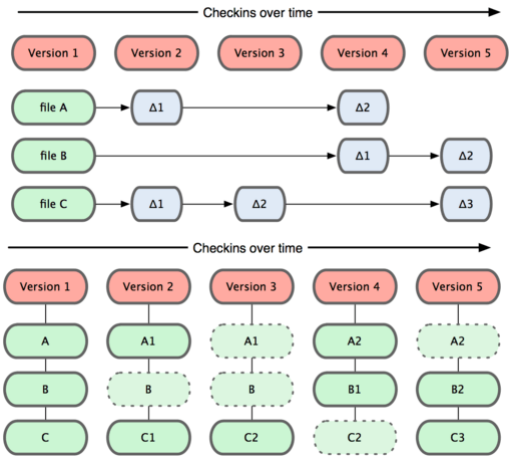
- Each client holds a complete copy of the code base
- Codes are shared between clients by push/pulls
- GIT



Pro Git by Scott Chacon and Ben Straub and published by Apress, http://git-scm.com/book

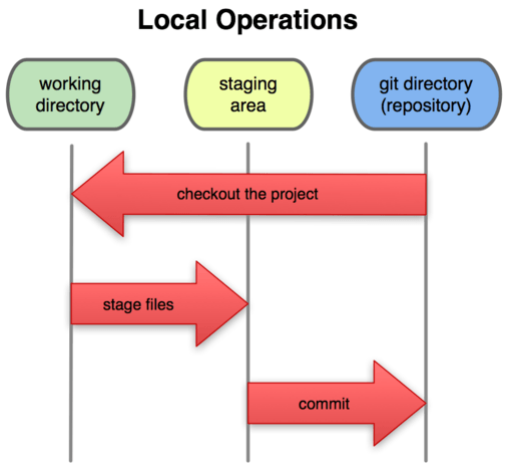
GIT

- Delta form vs. snapshot form



Pro Git by Scott Chacon and Ben Straub and published by Apress, http://git-scm.com/book

Local operations



Pro Git by Scott Chacon and Ben Straub and published by Apress, http://git-scm.com/book

Basic GIT workflow

- Init a repo
- Edit files
- Stage the changes
- Review your changes
- Commit the changes

56

Discussion on 3/1

- What happened from power-on to login ?