



Project 1

Fast correction



Project #1

- waitpid vs wait
 - WEXITSTATUS
 - Memory leaks !
 - fflush after the >, before the fork
 - Why? Because the children and the parent can run in any order
- When piping a list of commands to your shell, some flushed in bad order

The leftovers

- cd alone goes back to \$HOME
- Space arguments under quotes ("")

Project #1

- Check your indentation, choose a style
 - Just after a little googling for tools :
 - cppcheck
 - Check for common errors
 - indent
 - Auto indent
 - uncrustify
 - C, C++, C#, D, Java and Pawn source code beautifier
 - vera++
- Fork + exec to test if program exists is way much costlier than stat (or other syscall to check for file existence)



Programming in the Linux Kernel

Kernel sources organization

- arch/x86 has every hardware-specific code for the 32-bit Intel processors
 - other arch/xxx directory can be ignored
- include the include files needed to build the kernel code
- init the initialization code
- mm the memory management code
- drivers Code needed to communicate with external devices
- fs file system code
- kernel contains the main kernel code
- net networking code.
 - Not the drivers, which are in driver... IP stack, TCP, ...

More on <http://www.tldp.org/LDP/tlk/sources/sources.html>

Find something

- launch **cscope -R** in your kernel directory,
- use "**Find this C symbol**" to locate any *use* of a function/structure field/macro/variable ...
- use "**Find this global definition**" to locate where the body of the function/structure declaration/... sits.
- enjoy the other features, too :)

or...

- Use an IDE like [Eclipse](#) with the C/C++ plugin

In fact, it's complimentary. I use the twos

Macros

Kernel programmers love macros. Macros are replaced by some piece of code by the preprocessor, and does not suffer from “stack overhead” of functions.

```
if (unlikely(PageDirty(page))) {  
    ...  
}
```

Kernel
C
Code

unlikely (and the contrary, *likely*) is a simple macro, which optimise code for the “else” branch of a condition (no jump).

You may encounter some functions that doesn't exist anywhere... Like *PageDirty*

```
PAGEFLAG(Dirty, dirty) → Will make PageDirty Available
```

Kernel
C
Code

They are defined by another macro, but never explicitly defined :

Define

For the same reasons, they love define. The .config file is a big “definer”.

You’ll encounter some things like this :

```
#ifdef CONFIG_SMP
    struct plist_node pushable_tasks;
#endif
```

Kernel
C
Code

It means that if CONFIG_SMP is enabled in the config file, the code is taken into account, if not it’s like if it never existed

```
host % cat .config | grep CONFIG_SMP
CONFIG_SMP=y
```

Terminal

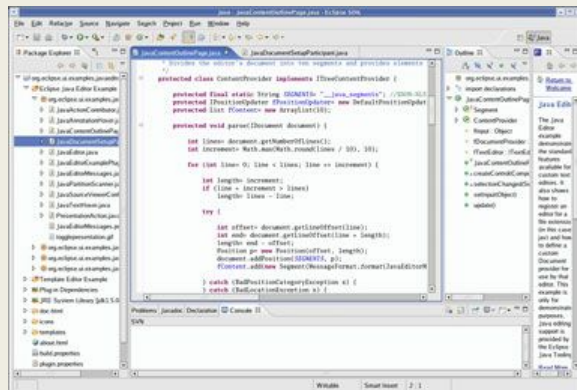
How to know if it’s activated ?

IDE

As said, any file editor like gedit (or vim/nano/emacs if you like to suffer) will work. Helping you with cscope in another window is a good idea.

But you may benefit from an IDE. I personally (and it's probably not the best) use **Eclipse**. You can “CTRL-CLICK” a function/macro/variable like *PageDirty* to jump to its definition.

It's really up to you.



Working with friends

- Don't even try Dropbox...
- SVN is probably the easiest.
- GIT is more powerful.

You can use montefiore's GitLab server on <http://gitlab.montefiore.ulg.ac.be> to have a free GIT repository ! Just create an account, a group, invite your collaborator to the group, and create a project inside your group namespace.

You can directly fork my git repo at <http://gitlab.montefiore.ulg.ac.be/info0940/kernel-4.4.50> which contains the kernel-4.4.50 sources, along with a tag "4.4.50" to make the creation of patches easier.

Adding changes to git

Of course the patch will work if you committed your branch before...

After each minor changes in the files :

- Use *git add FILE* to add the files that you changed
- Use *git add --patch FILE* to add only some parts of the changes done in the file
- Use *git commit* when all changes are “added”

OR

- Use *git commit --interactive*

- Try to type a good commit message, explicitly telling what you changed
- (use *git push* to send the changes to your colleagues, which will receive them with *git pull*)

VCS Warning

**DO
NOT**

**Include compiled, binary files
into your VCS, especially
`gitlab.montefiore.ulg.ac.be` !**

Patch with GIT

If you use GIT for development, making a patch is easier.

Add all the clean linux sources files to your repo before doing anything else with *git add -A* . Create a tag “4.4.50” before any modifications (This is already done if you forked my GIT project).

- To see the differences with the clean sources, use *git diff 4.4.50*
- *git format-patch 4.4.50* will create one patch per commit, and can even directly send them by e-mail.
- To have a patch compliant to “patch -p1” and make only one patch file :

```
host % git diff 4.4.50 HEAD > g42-s01-source.patch
```

Terminal

In other terms, while format-patch is good practice, avoid it for this course as I cannot automatically find if your patch is a git patch or a good old diff -burN

Debug by printing

- `dump_stack()` will show you all the calls which led where it is called.
- `printk()` will display a line in the syslog. It's the same arguments than syslog. It may be a good idea to launch a virtual `$ tail -f /var/log/syslog` in a terminal before doing your tests in the kernel, to see what is going on.

```
host % man tail
```

```
TAIL(1)
```

```
User Commands
```

```
TAIL(1)
```

```
NAME
```

```
tail - output the last part of files
```

```
[...]
```

```
-f, --follow[={name | descriptor}]
```

```
output appended data as the file grows; -f, --follow, and --follow=descriptor are equivalent
```

```
Terminal
```

- `printk(KERN_CRIT "bla bla bla");` will display the line in the current terminal, as if it was a very critical error (usefull for debug)
→ More info : <http://www.makelinux.net/ldd3/chp-4-sect-2>

GDB

If you got an error with a dump_stack like :

```
[ 368.815072] BUG: unable to handle kernel NULL pointer dereference at 00000024
[ 368.815244] IP: [] __schedule+0xc0/0x8e0
[ 368.815397] *pde = 1f040067 *pte = 00000000
[ 368.815572] Oops: 0000 [#1] SMP
[ 368.815711] Modules linked in: ip6table_filter ip6_tables ebttable_nat ebtables ipt_MASQUERADE iptable_
_comntrack ipt_REJECT xt_CHECKSUM iptable_mangle xt_tcpudp iptable_filter ip_tables x_tables bridge stp l
pnsnouse parport_pc serio_raw i2c_piix4 lp parport usbhid hid e1000
[ 368.816452]
[ 368.816571] Pid: 1707, comm: main Tainted: G      W   3.2.35 #25 innotek GmbH VirtualBox/VirtualBox
[ 368.816758] EIP: 0060[<c14cf50>] EFLAGS: 00010046 CPU: 0
[ 368.816758] EIP is at __schedule+0xc0/0x8e0
[ 368.816758] FAX: 00000000 EBX: 00000000 ECX: 00000000 EDX: dd125840
[ 368.816758] ESI: dd125b64 EDI: dd1258d0 EBP: db731e68 ESP: db731dec
[ 368.816758] DS: 007b ES: 007b FS: 00d8 GS: 00e0 SS: 0068
[ 368.816758] Process main (pid: 1707, ti=db730000 task=dd1258d0 task.ti=db730000)
[ 368.816758] Stack:
[ 368.816758] db731e1c c127dca0 00000001 00000010 dc189df0 c17b1140 df2ebd58 00000001
[ 368.816758] c17b1140 dfbe4140 dd1258d0 00000001 db731e9c c127d9ea 00000250 00000000
[ 368.816758] 90000002 ffff00ff 00000001 00000000 dcc73000 c14f4fe0 00000001 00010000
[ 368.816758] Call Trace:
[ 368.816758] [] ? soft_cursor+0x190/0x220
[ 368.816758] [] ? bit_cursor+0x51a/0x550
[ 368.816758] [] schedule+0x35/0x50
[ 368.816758] [] schedule_timeout+0x21d/0x290
[ 368.816758] [] ? _raw_spin_lock_irq+0x17/0x20
[ 368.816758] [] ? start_flush_work+0x7f/0xc0
[ 368.816758] [] ? flush_work+0x18/0x30
[ 368.816758] [] ? default_spin_lock_flags+0x8/0x10
[ 368.816758] [] n_tty_read+0x1d4/0x6b0
[ 368.816758] [] ? tty_ldisc_try+0x37/0x50
[ 368.816758] [] ? wake_up_process+0x20/0x20
[ 368.816758] [] ? is_ignored+0x40/0x40
[ 368.816758] [] tty_read+0x6d/0xb0
[ 368.816758] [] ? is_ignored+0x40/0x40
[ 368.816758] [] ? tty_fasync+0x30/0x30
[ 368.816758] [] ufs_read+0x80/0x150
[ 368.816758] [] sys_read+0x3d/0x80
[ 368.816758] [] syscall_call+0x7/0xb
[ 368.816758] [] ? scrdown+0x60/0xc7
[ 368.816758] Code: e8 c6 25 00 00 8b 45 ac 8d b0 98 02 00 00 8b 00 85 c0 74 11 89 e0 25 00 e0 ff ff f6
85 c9 74 95 0b 45 a8 ff d1 8b 45 a8 8b 40 04 85 c0 0f
[ 368.816758] EIP: [] __schedule+0xc0/0x8e0 SS:ESP 0068:db731dec
[ 368.816758] CR2: 0000000000000024
[ 368.816758] ---[ end trace 8a1b55a5b6964664 ]---
```

__schedule+0xc0

GDB (con'd)

- Find the file containing the function with cscope
- Launch GDB on the binary file (binary.o)
- Type list *(function+offset)

```
host $ cd kernel/  
host $ gdb sched.o  
GNU gdb (GDB) Fedora 7.6.1-41.fc19  
(...)  
Reading symbols from (...)done.  
(gdb) list *(_schedule+0xc0)  
0x18150 is in _schedule (kernel/sched.c:3232).  
3229 /* assumes rq->lock is held */  
3230 static inline void pre_schedule(struct rq *rq, struct task_struct *prev)  
3231 {  
3232     if (prev->sched_class->pre_schedule) //The error was a NULL pointer...  
3233         prev->sched_class->pre_schedule(rq, prev);  
(gdb) quit
```

Terminal

It's too slow

In order

- Do not compile in virtual machine
- Are you using `make -j(Ncores + 1)` to compile on all cores?
- Use `rsync -ruv` instead of `scp` for remote copy : it will only copy file that have changed
- Do not rebuild modules if you don't change modules ! Juste “`make ARCH=i386 bzImage -j5`” or vice-versa...
- Make shell script to compile your kernel, export variables, install, push files by ssh ... and probably another script in the virtual machine to copy thoses files, update `initramfs`, update `grub`, ... A recompilation should take two or three shell lines from you. I personally pushed the vice to making bash alias which are “s” to launch `sshfs` and log, “k” to update the kernel and “l” to launch my test program...
- Buy a SSD
- Buy a new processor (Core i5 or i7), but the SSD will make the greatest price/speed tradeoff

Beware of errors !

- Your kernel compiles
 - All .o object files are compiled
- You modify a .c source file and introduce an error
 - All object files are re-generated, except yours
- As the kernel compilation print thousand of lines, you don't see the error
- The kernel is built using the old .o file
- The kernel seems to work...

But if you do a “make ARCH=i386 clean”, you won't be able to compile again as your old .o file is not there anymore

Thus, I won't be able to compile...

→ **0 / 20**



Happens
every year !

PC 3

Additional time to compile the kernel, run it,
and start project 3 !