

Menelusuri Kode Program dengan Cscope dan Ctags

Pernah merasa pusing saat memprogram ribuan baris? Atau suatu hari ingin menelusuri kode kernel yang jumlahnya sampai puluhan ribu baris? Cscope dan Ctags bisa jadi solusinya.

Mulanya saya juga berpikir “ribet sekali menelusuri kode program yang bejibun ini.” Memang ini masalah klasik, dari jaman mainframe sampai PC, dari jaman DOS sampai era Linux kernel 2.6. Sebenarnya masalah ini serupa (tapi tak sama) bagi programmer, yakni bagaimana menelusuri program secara cepat dan efisien. Proses yang tepat dalam menelusuri program ini berpengaruh pada kecepatan *development*, juga memudahkan penelusuran jika ditemukan bug. Memang kata “cepat” ini sangat relatif. Ada yang mampu secara intuitif mencari letak suatu *procedure*, ada yang mesti serius memelototi editor sambil tekan tombol *Page Up* dan *Page Down*. Ada yang enjoy (ini kalau di Linux) pake *grep* dan *cat*. Benar-benar banyak jalan menuju “Roma.”

Masalahnya sekarang, bagaimana kalau kode program itu sangat banyak, semisal kode kernel? Waktu saya mencoba menelusuri kode kernel Linux, saya sering bertemu suatu *struct* (record, istilah di Pascal) yang definisinya ada di file *header* lain. Ini masih belum seberapa. Ada juga pemanggilan fungsi atau prosedur pada direktori yang berbeda, dengan referensinya “tersembunyi” pada deklarasi header di dalam file header

lain (*rekursif declaration*). Lalu bagaimana? Menurut saya ada dua cara, kuatkan mental anda atau gunakan duet Cscope dan Ctags.

Sebagai contoh kasus dalam artikel ini, saya akan mencoba menelusuri kode kernel bawaan Red Hat 7.3 dengan editor Vim. Tools yang akan digunakan adalah Cscope dan Ctags yang disambungkan (atau di-binding) langsung dengan Vim. Oleh karena itu, kita langsung saja mencoba tools ini. Pertama download Cscope dari <http://cscope.sourceforge.net> di *section Download*. Ambil versi terbaru (saat tulisan ini dibuat) yaitu versi 15.5. Penulis saat ini menggunakan versi 15.3 tapi cara penggunaannya sama saja. Untuk Ctags, ambil versi terbaru (saat artikel ini ditulis, versi 5.5.2) di <http://ctags.sourceforge.net>.

Letakkan kedua file ini di suatu direktori, misal di `/usr/src`, lalu unpack dengan perintah tar.

```
# tar xzvf ./ctags-5.5.2
# tar xzvf ./cscope-15.3.tar.gz
```

Segera configure, kompilasi, dan instalasi.

```
# cd cscope-15.3
# ./configure --prefix=/usr/local/cscope
```

```
# make && make install
```

```
# cd ../ctags-5.5.2
```

```
# ./configure --prefix=/usr/local/ctags
```

```
# make && make install
```

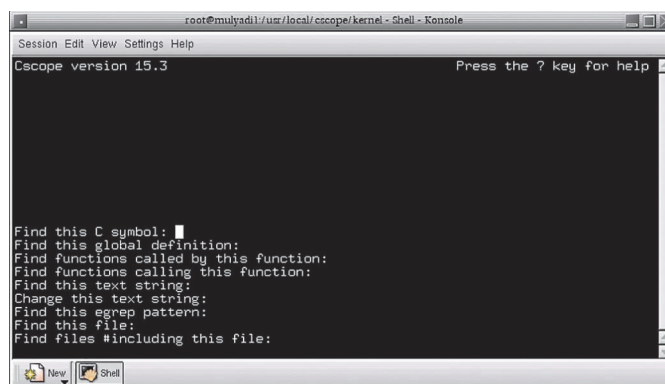
Buat symbolic link dari masing-masing *executable* ke `/usr/local/bin` agar memudahkan eksekusi program.

```
# ln -s /usr/local/cscope/bin/cscope /usr/local/bin/cscope
# ln -s /usr/local/ctags/bin/ctags /usr/local/bin/ctags
```

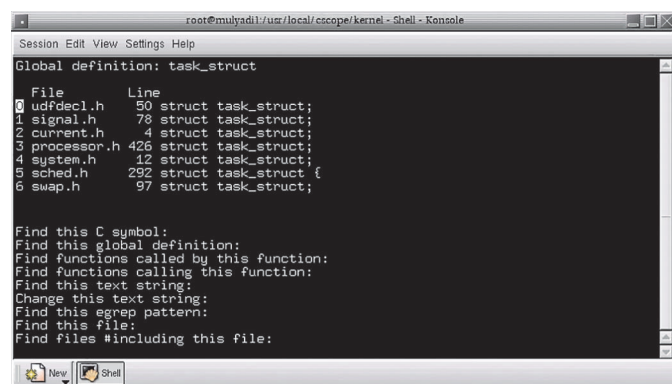
Pindahkan juga beberapa man file agar memudahkan jika nanti anda ingin membaca setting cscope/ctags lebih lanjut.

```
# mv /usr/local/ctags/man/man1/ctags.1 /usr/man/man1/
# mv /usr/local/ctags/man/man1/etags.1 /usr/man/man1/
# mv /usr/local/cscope/man/man1/cscope.1 /usr/man/man1/
```

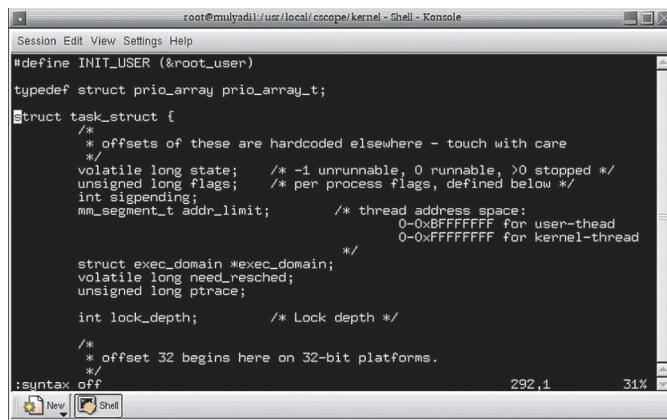
Sekarang kita fokus dulu ke Cscope. Langkah berikutnya yang harus kita lakukan adalah membuat daftar file untuk di-in-



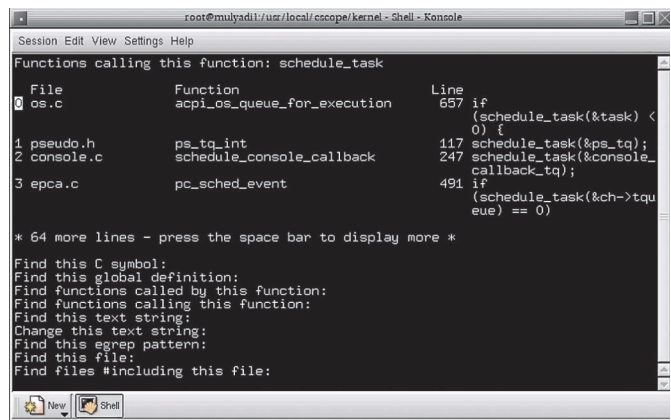
Gambar 1. Tampilan awal interface cscope.



Gambar 2. Hasil pencarian task_struct.



Gambar 3. Cscope memanggil Vi pada posisi deklarasi task_struct.



Gambar 4. Hasil pencarian fungsi yang memanggil schedule_task.

dex oleh Cscope. Langkah ini tidak mutlak dilakukan, namun sangat berguna untuk mempercepat proses pencarian. Untuk membuat daftar file di kernel tree, anda bisa gunakan shell script berikut ini.

```
#!/bin/bash
LNX=/usr/src/linux-2.4/
cd /
find $LNX \
  -path "$LNX/arch/*" ! -path
"$LNX/arch/i386*" -prune -o \
  -path "$LNX/include/asm-*" !
  -path "$LNX/include/asm-i386*"
  -prune -o \
  -path "$LNX/tmp*" -prune -o \
  -path "$LNX/Documentation*"
  -prune -o \
  -path "$LNX/scripts*" -prune
  -o \
  -path "$LNX/drivers*" -prune
  -o \
  -name "*.chxsS*" -print >
/var/cscope/kernel/cscope.files
```

Variabel LNX bisa anda ganti dengan sembarang path tempat anda meletakkan source code kernel atau *project* anda. Jangan lupa tanda "/" di akhir path. Output file bisa anda letakkan di manapun, tidak harus di /var/cscope/kernel, karena ini hanya contoh. Nama file pun tidak harus cscope.files, tapi untuk memudahkan percobaan kita gunakan nama ini karena nama ini otomatis dikenali sebagai daftar file yang akan di-index oleh Cscope. Opsi "-prune" berarti kita tidak melakukan recursive lookup ke dalam subdirektori. Ini digunakan karena file-file yang diperlukan ada di direktori level pertama. Extension yang diperlukan adalah list-

ing C (*.c dan *.h) serta beberapa statement assembly (*.s, *.S, dan *.x).

```
Simpan file script di atas, misal dengan
nama "generate.sh" di direktori
/var/cscope/kernel.
# mkdir -p /var/cscope/kernel
# chmod a+x ./generate.sh
```

Lalu jalankan script "generate.sh". Tunggu beberapa saat (tergantung kecepatan komputer anda) dan terciptalah file "cscope.files" di direktori /var/cscope/kernel. Isi dari file ini kurang lebih sebagai berikut.

```
[root@mulyadi1 kernel]# head -20
./cscope.files
/usr/src/linux-2.4/
Documentation/DocBook/procfs_
example.c
/usr/src/linux-2.4/
Documentation/networking/
ifenslave.c
/usr/src/linux-2.4/abi/cxenix/
pathconf.c
/usr/src/linux-2.4/abi/cxenix/
misc.c
/usr/src/linux-2.4/abi/cxenix/
signal.c
/usr/src/linux-2.4/abi/cxenix/
stubs.c
/usr/src/linux-2.4/abi/cxenix/
sysent.c
/usr/src/linux-2.4/abi/cxenix/
utsname.c
/usr/src/linux-2.4/abi/ibcs/
sysent.c
/usr/src/linux-2.4/abi/isc/
sysent.c
/usr/src/linux-2.4/abi/isc/misc.c
/usr/src/linux-2.4/abi/sco/
```

```
ptrace.c
/usr/src/linux-2.4/abi/sco/
ioctl.c
/usr/src/linux-2.4/abi/sco/misc.c
/usr/src/linux-2.4/abi/sco/mmap.c
/usr/src/linux-2.4/abi/sco/
secureware.c
/usr/src/linux-2.4/abi/sco/stat.c
/usr/src/linux-2.4/abi/sco/
statvfs.c
/usr/src/linux-2.4/abi/sco/
sysent.c
/usr/src/linux-2.4/abi/sco/
tapeio.c
```

Setelah anda cek entry file cscope.files, berikutnya lakukan *indexing* (tetap di direktori /var/cscope/kernel):

```
# cscope -b -k
```

Tunggu beberapa saat dan akan menghasilkan file cscope.out. File inilah yang merupakan database untuk pencarian fungsi atau variabel oleh Cscope.

Sekarang kita test dulu fungsi dasar Cscope dengan menggunakan *interface* pencarian *built in*, dengan mengetik perintah berikut.

```
# cscope -d
```

Opsi -d mencegah Cscope melakukan indexing ulang (karena anda sudah membuat index sebelumnya dan tidak melakukan perubahan apapun di source code). Anda akan menemui tampilan seperti gambar 1.

Sebagai latihan, kita mulai dengan mencari definisi fungsi *schedule*. Misal sekarang kursor berada pada baris "Find this C symbol". Tekan [Enter] sekali, maka kursor akan

berpindah ke “Find this global definition”. Sesuai namanya, baris “Find this global definition” akan mencari seakurat mungkin suatu definisi, bukan sekadar deklarasi suatu fungsi atau struct atau variabel. Misal dalam hal ini kita cari deklarasi *task_struct*, maka isikan *task_struct* lalu [Enter], anda akan mendapat tampilan seperti gambar 2.

Pada bagian atas, anda akan menemukan hasil pencarian. Pada komputer penulis ditemukan 6. Di sini terlihat jelas, yang kemungkinan adalah deklarasi *task_struct* adalah di nomor 5, maka tinggal tekan keyboard angka 5. Seketika Cscope akan memanggil Vi (editor default sesuai variabel lingkungan EDITOR) dan display akan “meloncat” ke deklarasi *task_struct*. Tekan [Esc] [:] [q] [Enter] untuk keluar dari Vi.

Bagaimana? Mulai merasakan kehendakan Cscope? Kita coba tipe pencarian lain. Misal kita ingin mencari prosedur apa saja untuk memanggil fungsi *schedule_task* (ini suatu fungsi untuk memasukkan tugas ke dalam antrian scheduler kernel). Jika anda sebelumnya berada pada baris hasil pencarian *task_struct*, tekan [Tab]. Tombol [Tab] digunakan untuk berpindah antara baris hasil pencarian dan baris fungsi pencarian. Lalu tekan kursor panah atas atau bawah dan arahkan ke “Find functions calling this function”. Ketik *schedule_task* dan [Enter]. Kali ini anda akan mendapat cukup banyak hasil.

Pada komputer penulis mendapatkan 64 referensi. Gunakan tombol ‘+’ dan ‘-’ untuk bergerak maju dan mundur melihat seluruh hasil pencarian. Anda akan menemui kata-kata “Press space bar to display the first line again” jika anda sudah mencapai akhir dari pencarian. Tekan Space bar untuk kembali

ke awal. Ambil salah satu hasil dan seperti tadi tekan angka yang sesuai. Misal penulis pilih pada referensi file *sys.*, maka tampilan akan berpindah ke Vi dan meloncat ke pemanggilan *schedule_task*.

Setelah selesai mencek dengan editor (Vi), keluar dengan [Esc] [:] [q] [Enter].

Barangkali anda berpikir, bolak balik dari *interface cscope* ke Vi, sepertinya tidak efisien. Benar! Karena itu sekarang saatnya kita coba integrasikan Vi/Vim dengan Cscope. Di sini diasumsikan anda menggunakan Vim versi 6.1-2 bawaan Red Hat 7.3. Jika anda belum menginstall Vim, lakukan instalasi paket-paket RPM berikut:

```
# rpm -Uvh /lokasi/RPM/distro/
anda/vim-common-6.1-2.i386.rpm
# rpm -Uvh /lokasi/RPM/distro/
anda/vim-enhanced-6.1-2.i386.rpm
# rpm -Uvh /lokasi/RPM/distro/
anda/vim-minimal-6.1-2.i386.rpm
```

Pada distro Red Hat, lokasi RPM bisa anda temukan di direktori RedHat/RPMS.

Kemudian download file http://cscope.sourceforge.net/cscope_maps.vim. http://cscope.sourceforge.net/cscope_maps.vim.

1. Copykan ke home direktori user yang akan menggunakan vim untuk penjelajahan.
2. Cek apakah anda sudah memiliki file *.vimrc*

```
# ls -al ~/ | grep -i vimrc
```

3. Jika ternyata sudah ada, sisipkan baris berikut di *.vimrc*:

```
source cscope_maps.vim
```

4. Jika belum ada file *vimrc*, untuk praktisnya rename saja file *map* menjadi *vimrc*,

karena pada intinya file *map* vim juga berisi syntax yang sama dengan perintah-perintah *vimrc*:

```
# mv ~/cscope_maps.vim
~/vimrc
```

Awas, jangan sampai tanda “.” ketinggalan !

5. Pastikan file *.vimrc* dan/atau file *map* bisa dibaca dan dieksekusi.

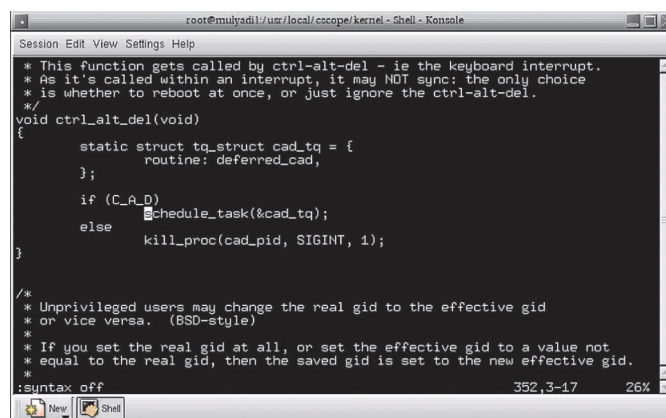
```
# chmod 755 ~/.vimrc
# chmod 755 ~/cscope_maps.vim
```

Berikutnya, perhatikan sejenak isi dari *cscope_maps* ini (berikut cuplikannya):

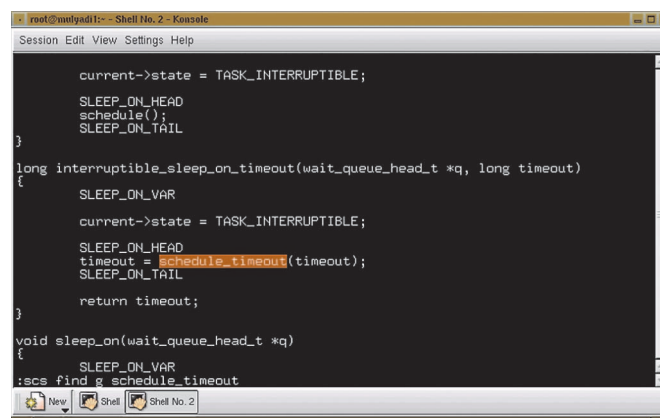
```
(baris 40) " add any cscope
database in current directory
(baris 41) if
file_readable("cscope.out")
(baris 42)   cs add cscope.out
(baris 43) " else add the
database pointed to by
environment variable
(baris 44)   elseif $CSCOPE_DB
!= ""
(baris 45)     cs add $CSCOPE_
DB
(baris 46)   endif
```

Di baris 44, ada variabel lingkungan *CSCOPE_DB* yang bisa anda set untuk menunjukkan lokasi data hasil *indexing* Cscope seperti yang telah kita lakukan di atas (dengan perintah *cscope -b -k*). Cara paling mudah menambahkannya adalah menyisipkan baris berikut di *.bashrc* (pada home direktori user yang ada edit file *.vimrc* nya):

```
export CSCOPE_DB=/var/cscope/
kernel/cscope.out
```

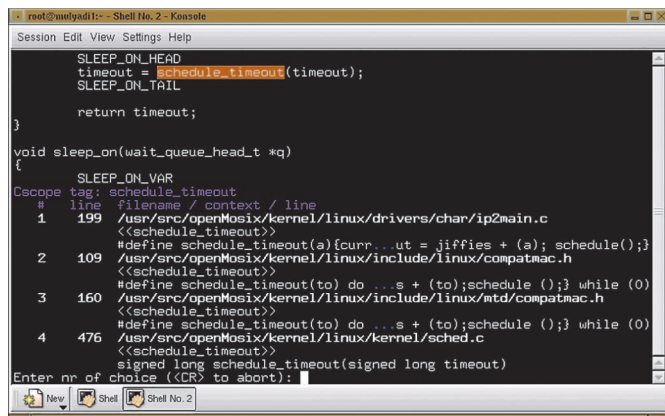


Gambar 5. Pemanggilan *schedule_task* di file *sys.c*.

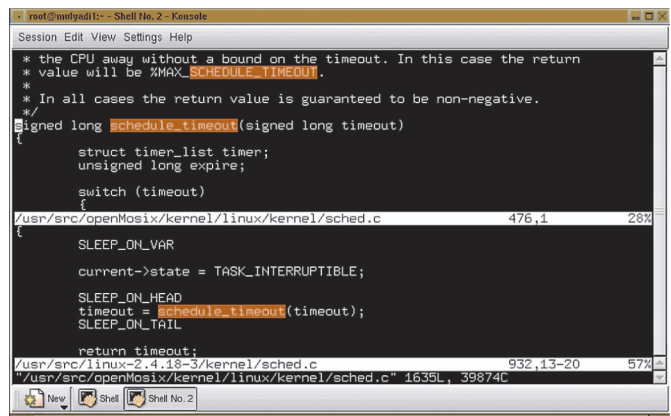


Gambar 6. Perintah lewat keyboard untuk pencarian fungsi di Vim.

Cscope dan Ctags



Gambar 7. Hasil pencarian schedule_timeout di Vim.



Gambar 8. Vim menampilkan dua window untuk memudahkan crosscheck.

Kelak jika membutuhkan lebih dari satu database, tambahkan lewat `cs add` setelah baris `endif` (baris 46) di `.vimrc`, misalnya:

```
cs add /var/cscope/kernel2/
cscope.out
```

Sekarang percobaan dengan Vim bisa dimulai. Log out dan log in sebagai user (yang tadi telah dimodifikasi `.vimrc`-nya) lalu panggil vim sambil mengedit suatu file, misal:

```
# vim /usr/src/linux-2.4/kernel/
sched.c
```

Sebelumnya mungkin perlu penulis beritahu, dalam artikel ini tidak dijelaskan secara detail bagaimana mengoperasikan Vi. Jadi pembaca bisa memadukan dengan membaca fasilitas `help` di Vi. Mungkin di lain waktu akan dicoba menulis artikel mengenai Vi. Kembali ke Vim, coba cari kata “*interruptible_sleep_on_timeout*”, kursor akan berpindah ke deklarasi seperti berikut (tekan ‘n’ beberapa kali sampai menemukannya):

```
long interruptible_sleep_on_
timeout(wait_queue_head_t *q,
long timeout)
{
    SLEEP_ON_VAR
    current->state = TASK_
    INTERRUPTIBLE;
    SLEEP_ON_HEAD
    timeout = schedule_
    timeout(timeout);
    SLEEP_ON_TAIL
    return timeout;
}
```

Sekarang coba arahkan kursor ke `schedule_timeout` di dalam deklarasi `interruptible_sleep_on` ini. Lalu tekan [Ctrl] [SpaceBar], lalu lepas, secepatnya dan tekan [g].

Perhatikan, sejenak di layar terbawah Vim (baris perintah) ada tertera:

```
scs find g schedule timeout
```

Ini adalah perintah yang sebenarnya dikirimkan ke Vim, namun karena kita telah memasang file `mapping`, kita tidak perlu mengetikkan perintah yang panjang ini.

Mirip dengan hasil pencarian di dalam interface `cscope`, anda akan disuguhkan *window* hasil pencarian. Tekan angka ‘4’ dan Vim akan *split* tampilan menjadi dua *window*, satu masih menunjuk di `interruptible_sleep`, *window* satunya menunjuk ke `schedule_timeout`.

Nah, menyenangkan bukan? Anda bisa mempelajari isi fungsi `interruptible_sleep_on_timeout` sekaligus melihat isi `schedule_timeout`. Untuk berpindah antar *window*, tekan tombol [Ctrl] [W] diikuti [w] (hampir bersamaan). Untuk menutup salah satu *window*, tekan [Esc] [:] [q] [Enter].

Kita lanjutkan dengan mengonfigurasi Ctags untuk *indexing* kernel. Mungkin anda bertanya, “Buat apa Ctags, sepertinya fungsinya sama saja Cscope?” Jawaban ini benar, tapi alasan utama kita menggabungkan dua sistem ini (Cscope dan Ctags) adalah untuk memaksimalkan kinerja pencarian. Karena masing-masing tools ada kelebihan dan kekurangan dalam membuat database, maka cara terbaik adalah menggunakan dua database hasil Cscope dan Ctags.

Sekarang kita langsung membuat database Ctags. Mirip seperti Cscope, kita gunakan file `generate.sh` untuk menghasilkan daftar file yang akan di-index. Langkahnya

sebagai berikut:

1. Buat direktori untuk menampung database ctags, misal

```
# mkdir -p /var/ctags/kernel
```

2. Copykan file `generate.sh` ke direktori diatas dan ubah baris berikut :

```
-name “*.chxsS” -print >
/var/cscope/kernel/cscope.
files
```

menjadi

```
-name “*.chxsS” -print >
/var/ctags/kernel/kernel.file
```

Intinya nama output apapun bisa dipakai asalkan memudahkan identifikasi untuk input Ctags. Jangan lupa `chmod a+x ./generate.sh`.

3. Jalankan `generate.sh` dan lakukan *indexing*.

```
# cd /var/ctags/kernel
# ./generate.sh
# ctags -L ./kernel.file
```

Akan tercipta file “tags” di direktori `/var/ctags/kernel`.

4. Berikutnya tambahkan baris berikut di file `.vimrc` atau `cscope_maps.vim`.

```
set tags=/var/ctags/kernel/
tags
```

Idealnya letakkan perintah di atas setelah “`set cscopeverbose`”. Ini untuk memasukkan database Ctags tiap kali Vim anda jalankan. Anda bisa memasukkan sekaligus beberapa file tag dengan dipisahkan koma (“,”)

5. Edit “`set csto=0`” pada `.vimrc` dan

cscope_map.vim menjadi set csto=1. Ini untuk menset agar pencarian tag mendahulukan database Ctags, baru jika tidak ditemukan, digunakan database Cscope. Kalau dibalik, anda akan mendahulukan Cscope baru Ctags. Ini sebenarnya masalah selera saja, tapi untuk memaksimalkan fungsi keduanya, kita coba dulu setting di atas.

Kombinasi Vim, Cscope, dan Ctags sudah lengkap. Saatnya mencoba kemampuan ctags. Coba jalankan Vim berikut ini.

```
# vim /usr/src/linux-2.4/kernel/timer.c
```

Sekarang coba cari kata "spin_lock_irqsave". Anda bisa temukan di sekitar baris ke 180-185. OK, sudah ketemu? Kalau masih kesulitan, lakukan langkah berikut (fungsi search): Tekan [Esc] [Esc] [/] spin_lock_irqsave, artinya tekan [Esc] dua kali, ketik [/], lalu ketikkan spin_lock_irqsave.

Setelah menemukan yang anda cari, arahkan kursor agar pas berada di dalam kata "spin_lock_irqsave", lalu tekan [Ctrl] dan [I] (tombol Ctrl dan karakter 'I').

Pilih salah satu, misalnya di nomor 5. Begitu anda tekan [Enter], anda akan dibawa ke definisi spin_lock_irqsave. Jika kebetulan anda menemukan definisi lain yang ingin anda telusuri, arahkan kursor ke definisi tersebut lalu tekan [Ctrl] [I]. Anda bisa mundur ke langkah sebelumnya dengan [Ctrl] [T]. Pencarian berlapis ini disebut "tag stack" dan bisa dibayangkan seperti tumpukan atau stack.

Jadi secara umum (untuk bernavigasi di stack):

1. [Ctrl] [I] untuk mencari definisi tag. Jika ditemukan hanya satu hasil, Vim akan langsung "meloncat" ke definisi tersebut. Jika ada lebih dari satu, maka hasil pencarian akan ditampilkan dan anda bisa memilih angka hasil pencarian
2. [Ctrl] [T] untuk mundur ke posisi teks sebelumnya. Jika anda sudah berada di posisi paling awal, anda akan menemui teks "at bottom of tag stack". Jika anda belum melakukan pencarian tag, [Ctrl] [T] akan memunculkan peringatan "tag stack empty".

Barangkali anda ingin melakukan *split* window seperti saat kita mencoba Cscope. Anda punya dua pilihan:

1. Menggunakan "stjump" dan keyword yang akan dicari. Ketikkan [Esc] [:] stjump <keyword-yang-anda-cari>
2. keluar dulu dari Vi, lalu tambahkan baris berikut di .vimrc.

```
nmap <C-@>k :stjump <C-R>=expand("<Cword>")<CR><CR>
```

Perintah ini artinya membuat mapping/shortcut tombol [Ctrl] [Spacebar] [k] untuk melakukan "stjump" seperti point 1.

Sekarang lakukan lagi seperti pencarian tag, hanya saja kali ini lakukan dengan "stjump" atau shortcut kita yang baru. Setelah kita meloncat ke lokasi teks yang baru, otomatis window akan di-split menjadi dua. Kita bisa sekaligus melihat ke dua bagian (mirip dengan Cscope). Jika anda tidak suka dengan split secara horisontal, Cscope dan Ctags

bisa melakukan split window vertikal dengan bantuan Vim, caranya sebagai berikut.

1. Untuk Cscope, tekan [Ctrl] [Spacebar] [Spacebar] (artinya dua kali space bar secara cepat) diikuti [g].
2. Untuk Ctags, keluar dulu dari Vim dan tambahkan mapping berikut di .vimrc.

```
nmap <C-@><C-@>k :vert
stjump <C-R>=expand("<Cword>")<CR><CR>
```

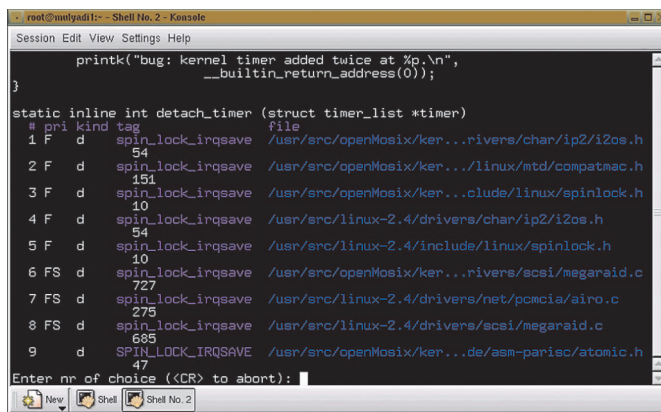
Jalankan lagi Vim, letakkan kursor di keyword yang dicari definisinya lalu tekan [Ctrl] [Spacebar] [Spacebar] [k]. Setelah memilih tag tujuan (atau langsung meloncat jika hanya ditemukan satu hasil pencarian), window akan displit secara vertikal.

Kedua cara di atas bisa diterapkan secara rekursif dan juga berlaku untuk split horisontal. Anda bisa juga mengkombinasikan split horisontal dan vertikal.

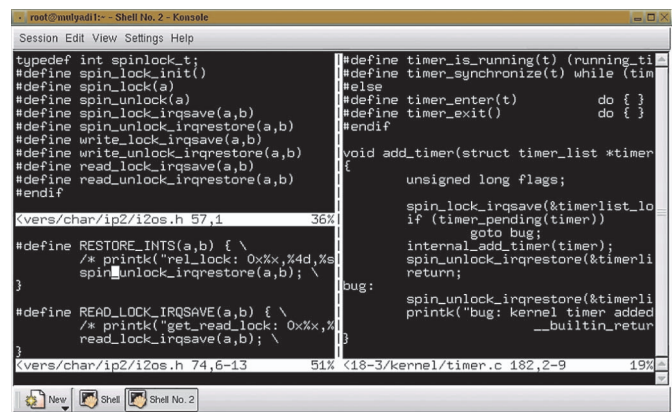
Untuk berpindah antar window yang "ruwet" seperti ini, gunakan [Ctrl] [W] diikuti panah kursor atas, bawah, kiri dan kanan. Fokus window akan berpindah sesuai arah kursor yang ditekan. Jika sudah selesai dengan satu window anda bisa tutup dengan perintah menekan [Esc] [:] [q].

Kesimpulannya, dengan Vim, Ctags dan Cscope anda bisa menjadi lebih cepat dan produktif dalam menelusuri suatu kode program yang sangat besar dan terpisah-pisah. Contoh dalam artikel ini bisa anda terapkan tidak hanya untuk *development* kernel, tapi juga projek lainnya. Selamat mencoba, variasi lain bisa anda perdalam sendiri dengan bekal tutorial ini dan mengamati contoh file mapping cscope.

Mulyadi S. (a_mulyadi@telkom.net)



Gambar 9. Hasil pencarian Ctags pada keyword spin_lock_irqsave.



Gambar 10. Contoh tampilan dengan split window di Vim.