LINUX KERNEL COMPILATION &
SYSTEM CALL IMPLEMENTATION TUTORIAL
(for Ubuntu 8.04 and i386 architecture machines)

--- version 3.1 date: 09.10.2008 ---

I. How to Compile a Kernel: The Ubuntu Way (Ubuntu 8.04)

NOTE: Accessing some parts of the system, installing system wide programs, such as a kernel, requires root privileges. Do not forget to add `sudo` to the beginning of most commands mentioned in the instructions below, which will grant you root priviledges during the execution of the next command.

1) Update your package database:

```
aptitude update
```

2) Install all needed packages for kernel complation:

```
aptitude install kernel-package libncurses5-dev fakeroot wget bzip2
```

3) Download the kernel sources to `/usr/src`

```
cd /usr/src
```

```
wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.20.15.tar.bz2
```

```
tar xjf linux-2.6.20.15.tar.bz2
```

```
ln –s linux-2.6.20.15 linux
```
(form symbolic link for linux-2.6.20.15)

```
cd /usr/src/linux
```

4) Configure the kernel by copying configuration file from the host system

```
cp /boot/config-2.6.24-19-generic   ./.config
```

```
make menuconfig
```

- Linux kernel configuration menu appears.
- Select "`Load an Alternate Configuration File`".
- `.config` shall appear, as the name of the configuration file you wish to load, in the dialog box that appears next. Press `<Ok>`.
- (Optional) Make your configuration choices browsing through the configuration options.
- When you are finished, select `Exit` and answer the question "`Do you wish to save your new kernel configuration ?`" with `Yes`.

5) Build the kernel by executing the following two commands:

Note: Current directory must be `/usr/src/linux`

Note: If you are planning to add a new system call to the kernel, you will need to compile the kernel again. To avoid compiling the kernel two times, see section II for additional steps and start the compilation after performing those steps.

```
make-kpkg clean
```

```
fakeroot make-kpkg --initrd –-append-to-version=-custom kernel_image kernel_headers
```

After `–-append-to-version=` you can write any string that helps you identify the kernel, but it must begin with a minus `(-)` and must not contain whitespace.

After having entered the last command, the kernel compilation starts. Depending on your kernel configuration and your processor speed, kernel compilation may take some hours.

6) Install the new kernel:

```
cd /usr/src
```

```
ls –l
```

```
dpkg –i linux-image-2.6.20.15-custom_2.6.20.15-custom-10.00.Custom_i386.deb
```

```
dpkg –i linux-headers-2.6.20.15-custom_2.6.20.15-custom-10.00.Custom_i386.deb
```

After having built the kernel successfully, you shall find two `.deb` packages in the `/usr/src` directory The `.deb` package `linux-image-2.6.20.15-custom_2.6.20.15-custom-10.00.Custom _i386.deb` contains the actual kernel and the `.deb` package `linux-headers-2.6.20.15-custom_2.6.20.15-custom-10.00.Custom_i386.deb` contains files needed if you want to compile additional kernel modules later on.

NOTE: Even after installation of the .deb packages under /usr/src .deb packages `linux-image-2.6.20.15-custom_2.6.20.15-custom-10.00.Custom _i386.deb` and `linux-headers-2.6.20.15-custom_2.6.20.15-custom-10.00.Custom _i386.deb` shall remain. Thus, you can now even transfer the two `.deb` files to other Ubuntu systems and install them there exactly the same way, which means you do not have to compile the kernel there again.

7) Check `/boot/grub/menu.lst` to find the new two stanzas for your new kernel:

```
title    Ubuntu, kernel 2.6.20.15-custom
root     (hd0,0)
kernel /boot/vmlinuz-2.6.20.15-custom root=/dev/sda1 ro quiet splash
initrd   /boot/initrd.img-2.6.20.15-custom
```

```
title    Ubuntu, kernel 2.6.20.15-custom (recovery mode)
root     (hd0,0)
kernel   /boot/vmlinuz-2.6.20.15-custom root=/dev/sda1 ro single
initrd   /boot/initrd.img-2.6.20.15-custom
```

To make your custom kernel boot by default, change the line

```
default    0
```

to

```
default    2
```

indicating third listed kernel is to be the default kernel to be booted. (Counting starts from zero)

8) Restart your system now.

```
sudo reboot
```

9) Make sure the “custom” kernel is selected during restart.

10) Check if the system is using the new kernel by typing:

```
uname -r
```

It should display:

```
2.6.20.15-custom
```

II. How to Add a System Call (for 2.6.x Kernels and i386 Type Architecture Machines)

Assume we'd like add a system call calles "mycall" that takes two integers as input and return their sum.

1) Add **".long sys_mycall"** at the end of the list in the file **syscall_table.S**.

---Full path for the file **syscall_table.S** is
`usr/src/linux/arch/i386/kernel/syscall_table.S`

NOTE: You can use the command `sudo gedit syscall_table.S` to edit the file **syscall_table.S**. Beware that if you do not have the root privileges, system will not allow you to edit any of the kernel files.

2) In file **unistd.h**

- Add `#define __NR_mycall <Last_System_Call_Num+1>` at the end of the list (e.g. If the number of the last system call `Last_System_Call_Num is` 319 then the line you shall add should be `#define __NR_mycall 320.).`
- Increment `NR_syscalls` by one (e.g. If before adding your system call total number of system calls is 320, then this will be `#define NR_syscalls 321.).`

---Full path for the file **unistd.h** is `/usr/src/linux/include/asm-i386/unistd.h`

3) Add the following line at the end of the file **syscalls.h**:

asmlinkage long sys_mycall(int i, int j);

---Full path for the file **syscalls.h** is `/usr/src/linux/include/linux/syscalls.h`

4) Add **mycall/** to **core-y** += in **Makefile**.

The line in the end shall look like:

```
core-y    += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ mycall/
```

---Full path for **Makefile** is `/usr/src/linux/Makefile`.

5) Create a new directory in `/usr/src/linux` and name it **mycall.**

6) Create a new file called **mycall.c** in `/usr/src/linux/mycall`**.** Contents of the file shall be as follows:

```
/*----------Start of mycall.c----------*/

#include <linux/linkage.h>

asmlinkage long sys_mycall(int i, int j)    {

     return(i+j);

}

/*-----------End of mycall.c-----------*/
```

***asmlinkage** is used to look for the arguments on the kernel stack.

7) Create **Makefile** in `/usr/src/linux/mycall`. **Makefile** shall be like:

```
########## Start of Makefile ##########

obj-y := mycall.o

##########  End of Makefile  ##########
```

8) Create the following userspace program to test your system call and name it **testmycall.c**. The contents of this file shall be:

```
/*---------- Start of testmycall.c File ----------*/

#include <unistd.h>

#include <stdio.h>

#define __NR_mycall <Last_System_Call_Num+1>


long mycall(int i, int j) {

     return syscall(__NR_mycall, i, j);

};

int main()    {

     printf("%d\n", mycall(10, 20));

     return 0;

}

/*---------- End of testmycall.c File ----------*/
```

9) Compile testmycall.c to test whether your system call works.

```
gcc testmycall.c –o testmycall
```

REFERENCES:

[1] How to Compile a Kernel: The Ubuntu Way, http://www.howtoforge.com/kernel_compilation_ubuntu

[2] Amit Choudhary, “Implementing a System Call on Linux 2.6 for i386”, http://tldp.org/HOWTO/html_single/Implement-Sys-Call-Linux-2.6-i386/

*** Internet is the ultimate resource for the solution of the problems you come across to during system call implementation and kernel compile procedures.