

Random Stuff

The blog is mostly about some Random Stuff that i have tried to do as part of Hobby Projects or as Self Initiatives. Through this my aim is to inspire and inculcate the Open Source sharing spirit in each one of Discussions and Thoughts most welcomed.

[Analysis & Implementation of Mandelbrot Sets and Julia Fractals on Raspberry Pi using IPython](#)

[Writing a Linux Device Driver for VS-1003/VS-1053 Audio Codec on Raspberry Pi 3 with Linux Kernel 4.4+](#)

[Programming Interrupts in Raspberry Pi using a simple Kernel Character Device Driver](#)

[Interfacing nRF24L01+, 2.4Ghz Radio/Wireless Transceiver with RaspberryPi2, using SPI.](#)

[Interfacing RaspberryPi with DS1307,I2C based Real Time Clock.](#)

[Interfacing a Serial SD Card Logger with RaspberryPi.](#)

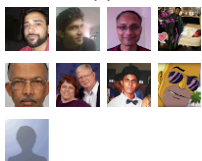
[DSP using RaspberryPi2 : Analysis and Implementation of 8x8 DCT using ARM NEON Assembly on RaspberryPi2](#)

[A Simple Real-Time Signal Plotting Application for Raspberry Pi using Cairo.](#)

[Adding a Simple System Call in Raspberry Pi](#)

Followers

Followers (9)



Blog Archive

▼ 2010 (3)

▼ April (3)

[Go Back To Main Page](#)

About Me

Rajiv

[View my complete profile](#)

Adding a Simple System Call in Raspberry Pi

The system call provides an interface to the operating system services.

Application developers often do not have direct access to the system calls, but can access them through an programming interface (API). The functions that are included in the API invoke the actual system calls. By using certain benefits can be gained:

- Portability: As long a system supports an API, any program using that API can compile and run.
- Ease of Use: Using the API can be significantly easier, than using the actual system call.

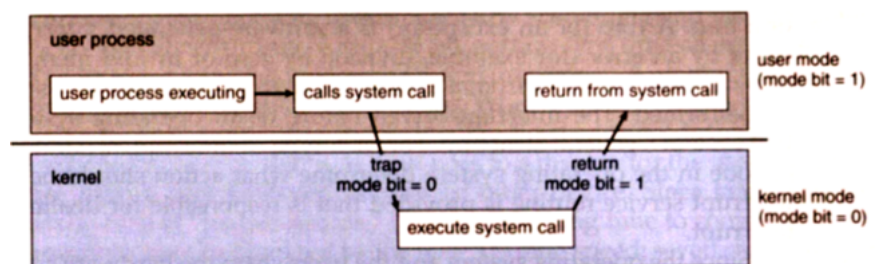


Fig1. How System Call works between the User Space and the Kernel Space.

Consider, the example of a simple system call, `write()`, called from an User Process. The User Process calls `write` which invokes the Kernel through an INT, interrupt instruction, where the write operation is conveyed as an `syscall[]` table. The file descriptor argument contains, among other things, a MAJOR Device number so that the Kernel function can access the driver's file operations structure. The Kernel calls the driver's write function, which copies user space buffer into the kernel space.

An Example: Invoking a given System Call

Now, coming to Raspberry Pi, which is a Broadcom SOC, BCM 2835, based on ARM Processor. Every System Call is having an entry in the System Call Table. The Index is an Integer value which is passed to the Register R7, in case of ARM processors, R0, R1 and R2 are used to pass the arguments of the System Call. The instruction, SWI, which is now known as SVC, which is a Supervisor Call, is used to jump to the Privileged Mode, to invoke the Kernel. The number embedded in the instruction, is used to refer to the Handler.

```
svc #0
```

Hence, as an example, say, we want to invoke a System Call to print "Hello World\n". The System Call Index to invoke is #4. Thus, the code will be something like,

```

        .arch armv6
        .section      .rodata
        .align 2
.data
HelloWorldString:
        .ascii "Hello World\n"
.LC0:
        .text
        .align 2
        .global main
        .type main, %function
main:
                                mov r7, #4
                                mov r0, #1
                                ldr r1,=HelloWorldString
                                mov r2, #12
                                svc #0
                                @ Need to exit the program
                                mov r7, #1
                                mov r0, #0
                                svc #0

.L3:
        .align 3
.L2:
        .size main, .-main
        .ident "GCC: (Debian 4.6.3-14+rpi1) 4.6.3"
        .section      .note.GNU-stack,"",%progbits

```

In, the above example, #4 is passed to register R7, which will invoke the System Call corresponding to index 4 o Call Table. Compile the above program on your RaspberryPi, using,

```
#gcc -o executable_name file_name.s
```

And execute the executable,

```
./executable_name
```

Now, check 'dmesg' using,

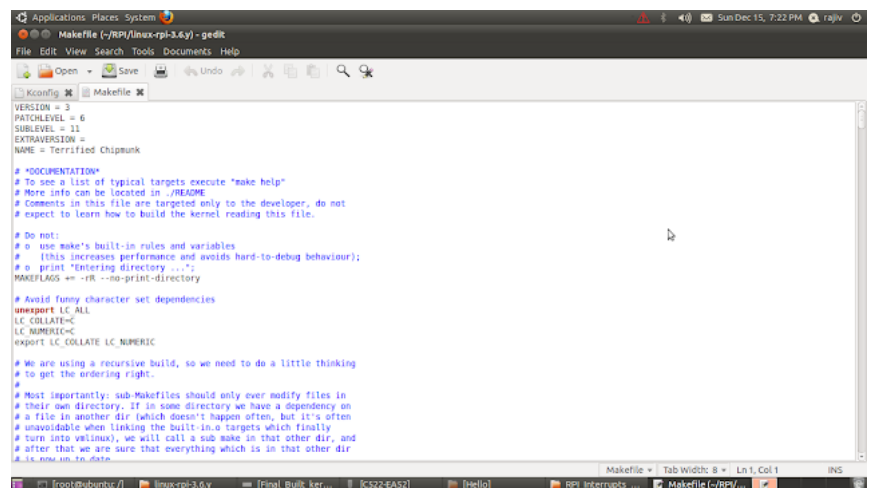
```
#dmesg | tail
```

The above command will display the message 'Hello World', which means, that we are able to successfully invoke Call.

Using the above example, we can Invoke a given System Call, from the User Process. In the next part, below, we w we can register a System Call with the Kernel. And, later, we will invoke the registered System Call, from the U:

Registering a new System Call with the Kernel

In this section, we will see, how we can register a new System Call with the Kernel, for RaspberryPi. For, t know, how to cross-compile the Linux Kernel for our RaspberryPi Platform. The Kernel version can be found by e command, `uname -a`. Now, the version of the Kernel shown as the output of the command `#uname -a` must match with of the Kernel source that we are trying to cross compile to Register our System Call. By looking into the 'Make the Kernel Source directory, we can get that info, as shown below:



Kernel Makefile through which we can know kernel version, here for me the RPi was having a version 3.6.11, and the same can be found h

So, we have our Kernel Source. Now, the following files need to be considered to add a New System Call to Raspbe:

- /arch/arm/kernel/calls.S
- /arch/arm/include/asm/unistd.h
- /arch/arm/kernel/sys_arm.c
- /include/linux/syscalls.h

In the first file, '/arch/arm/kernel/calls.S', add a new entry with the new System Call name to any entry, havi 'sys_ni_syscall'. I will call my new system call, as, 'sys_MyHelloWorld' and add it to 59th Index. Thus, choose the file, calls.S having the entry,

```
CALL(sys_ni_syscall)
```

and change it to,
CALL(sys_MyHelloWorld)
I choose, to give the name, 'MyHelloWorld', any other name would be fine too.

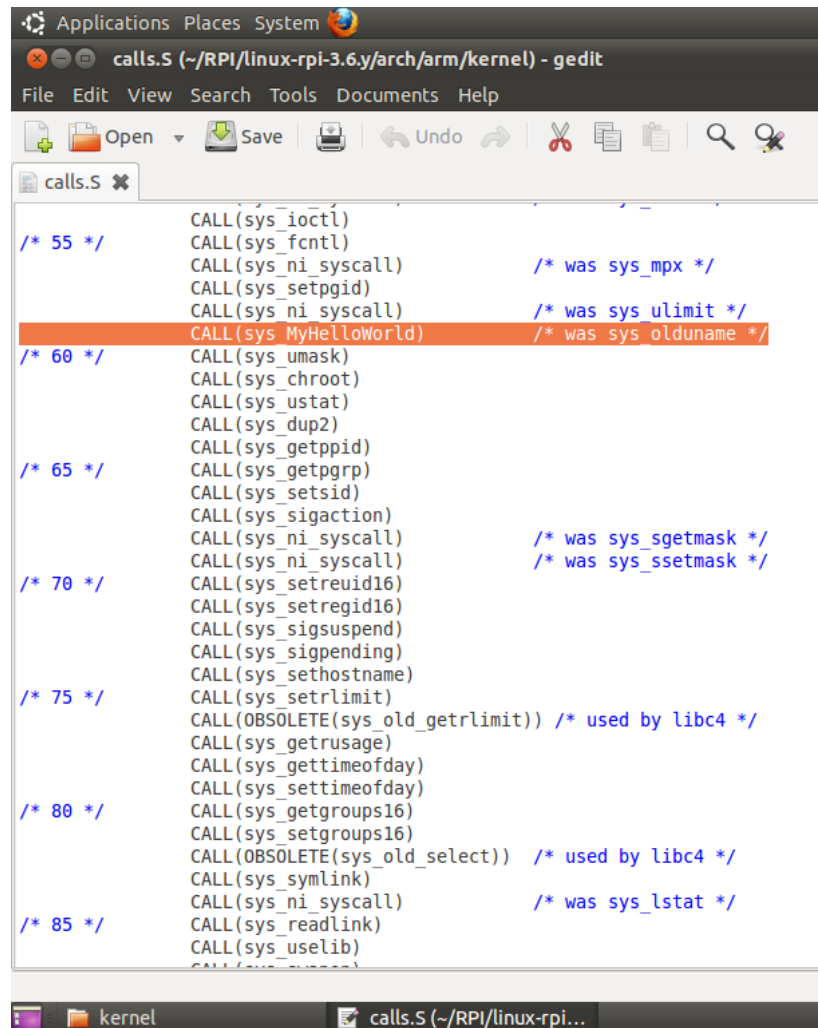


Fig2. Add the System Call entry in 'calls.S'

In the second file, '/arch/arm/include/asm/unistd.h', contains the mapping with respect to the System Table Base memory mapped address value. The Index that we chose in calls.S, would act as Offset for the same in the Ke Space of the System Call.

Here, at Index 59, we see that, it is commented as something follows:

```
/* 59 was sys_olduname */
```

which will be modified as below:

```
#define __NR_MyHelloWorldKernel          (__NR_SYSCALL_BASE+ 59)
```

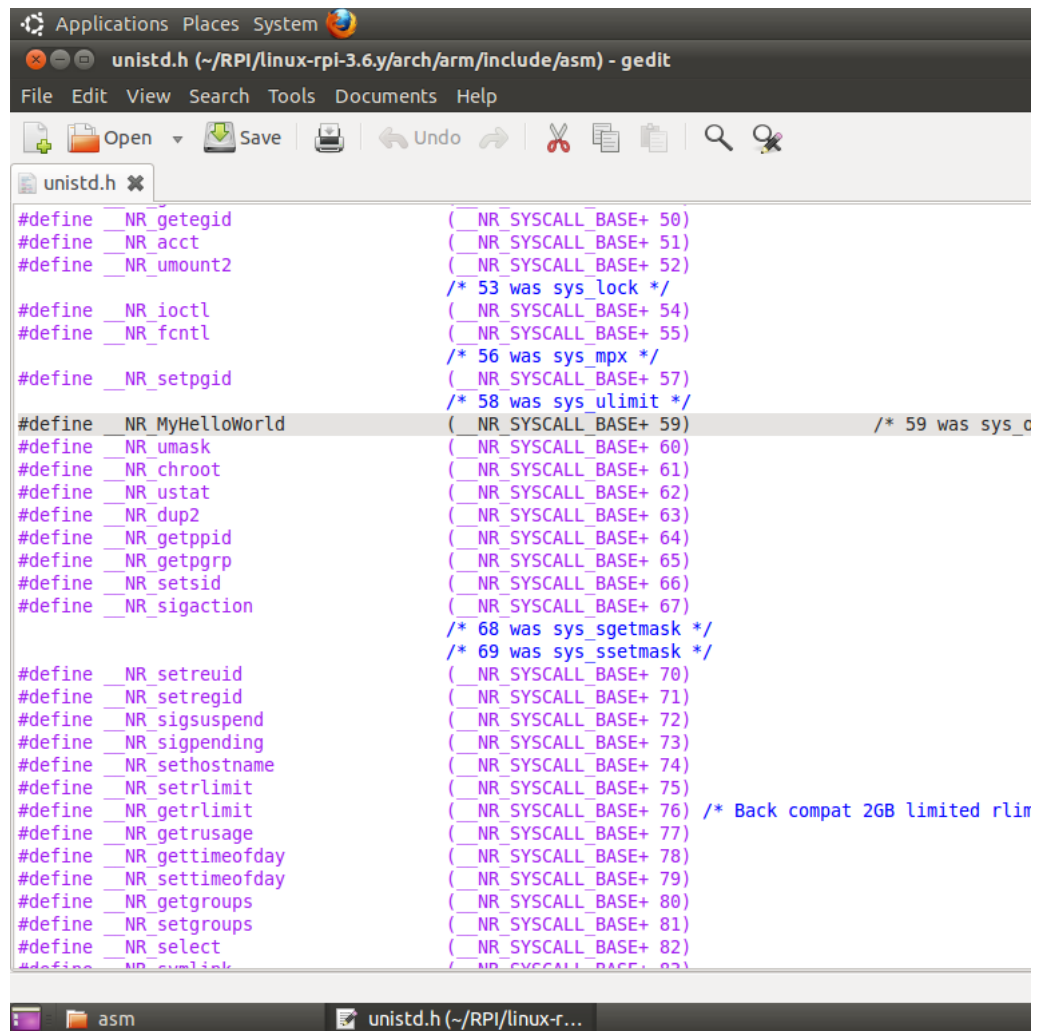
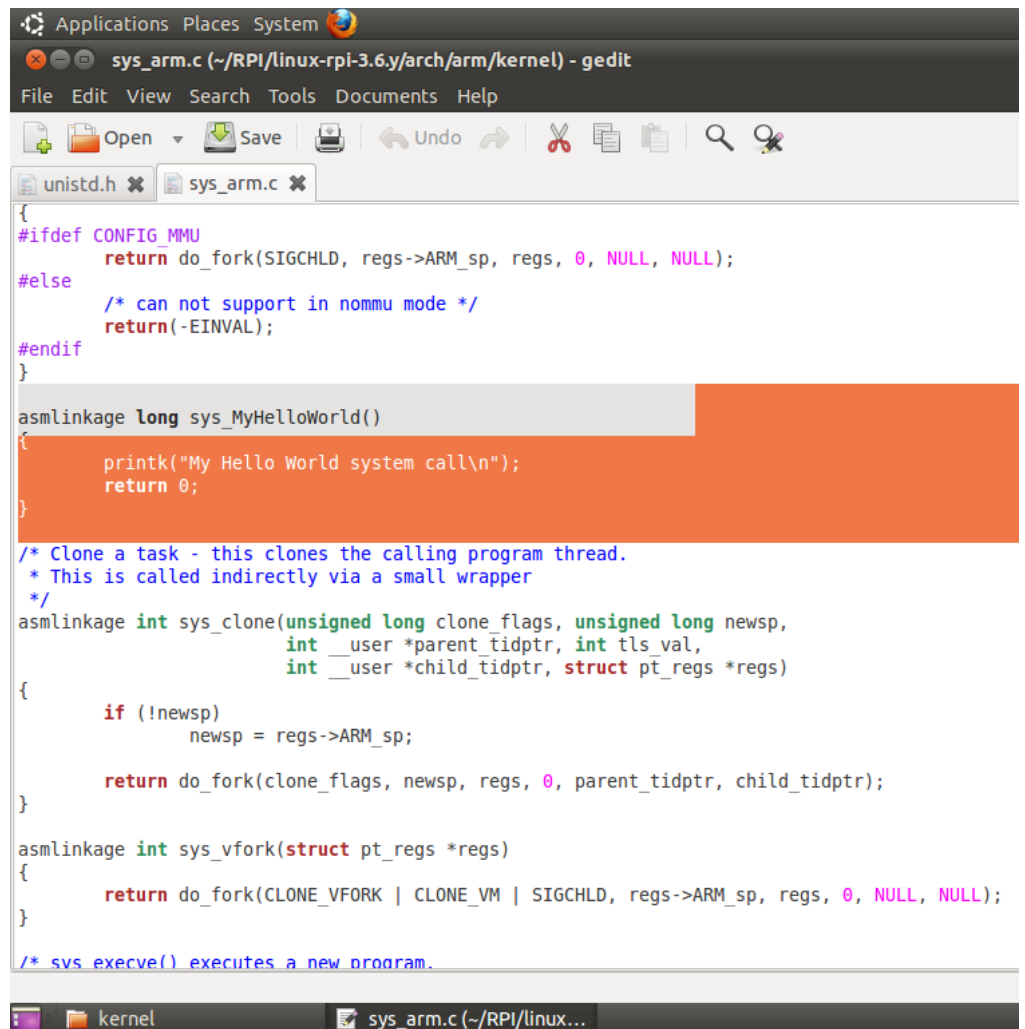


Fig 3. Add Offset Entry corresponding to the Index in 'unistd.h' file

```
In the third file, '/arch/arm/kernel/sys_arm.c', add the system call handler code.
asmlinkage long sys_MyHelloWorld()
{
    printk("My Hello World system call\n");
    return 0;
}
```



```
{
#ifdef CONFIG_MMU
    return do_fork(SIGCHLD, regs->ARM_sp, regs, 0, NULL, NULL);
#else
    /* can not support in nommu mode */
    return(-EINVAL);
#endif
}

asmlinkage long sys_MyHelloWorld()
{
    printk("My Hello World system call\n");
    return 0;
}

/* Clone a task - this clones the calling program thread.
 * This is called indirectly via a small wrapper
 */
asmlinkage int sys_clone(unsigned long clone_flags, unsigned long newsp,
                        int __user *parent_tidptr, int tls_val,
                        int __user *child_tidptr, struct pt_regs *regs)
{
    if (!newsp)
        newsp = regs->ARM_sp;

    return do_fork(clone_flags, newsp, regs, 0, parent_tidptr, child_tidptr);
}

asmlinkage int sys_vfork(struct pt_regs *regs)
{
    return do_fork(CLONE_VFORK | CLONE_VM | SIGCHLD, regs->ARM_sp, regs, 0, NULL, NULL);
}

/* sys execve() executes a new program.
```

Fig 4. Add the System Call Handler code in 'sys_arm.c' file

And, in the last file, '/include/linux/syscalls.h', add the following above as System Call Handler prototype.
asmlinkage long sys_MyHelloWorld(void);

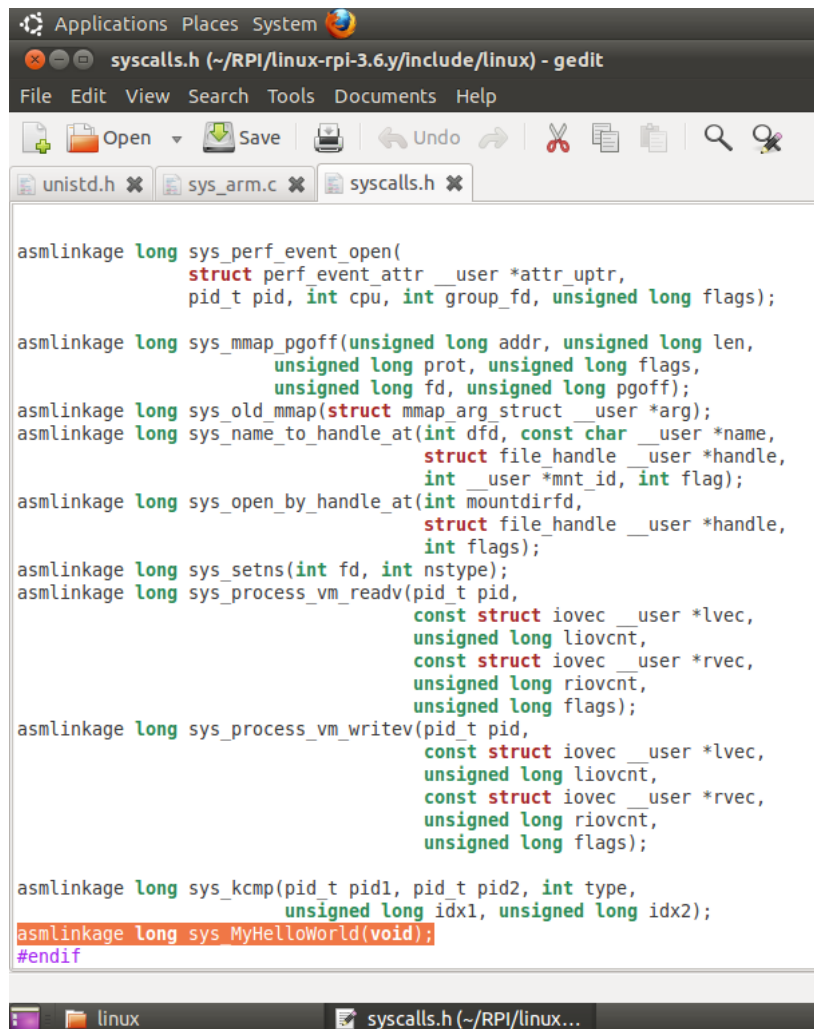


Fig 5. Add the prototype of the System Call handler in 'syscalls.h' file

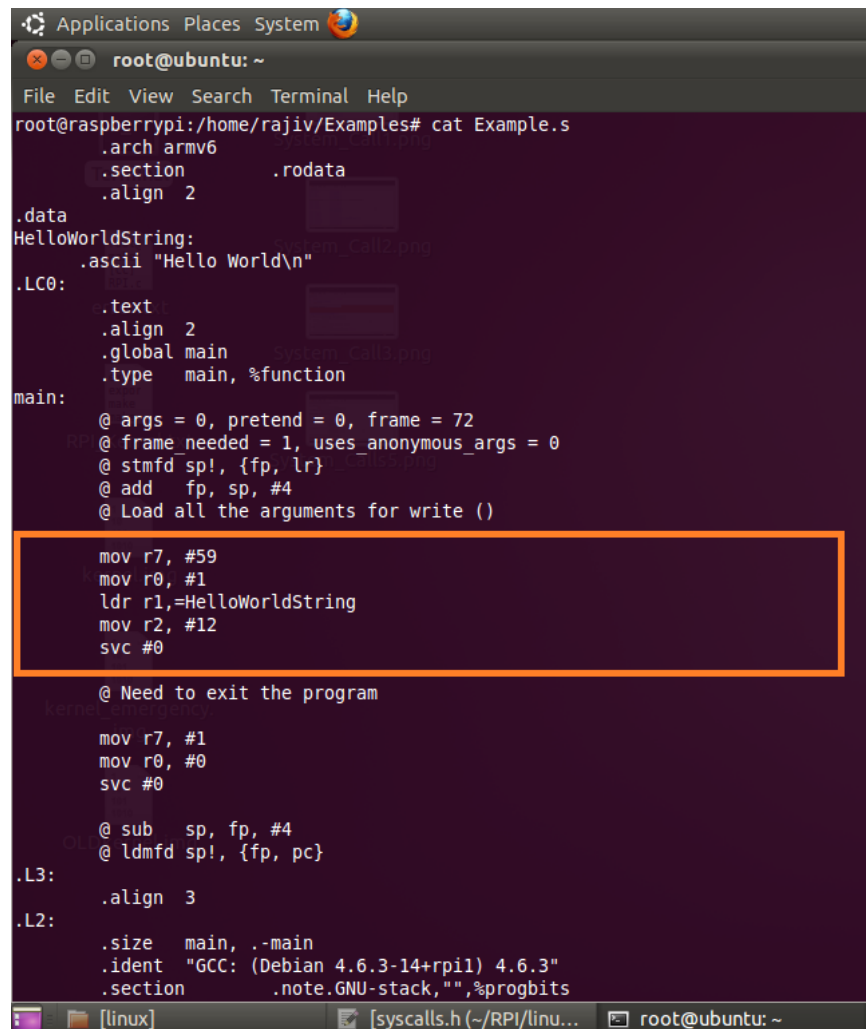
After, working with these four files and adding the System Call Handler, the System Call has to be register Kernel, for which the Kernel has to be cross-compiled, since i am trying to compile it on my x86 machine, installed, and the target is ARM Based, Raspberry Pi.

- Get the tarball, from GIT Hub, <https://github.com/raspberrypi/linux/tree/rpi-3.6.y>, and download it in the ZIP. Make the changes in the above four files, mentioned above.
- Get the tools tarball.
- Extract the Toolchain and the Linux Source, #tar -xvzf rpi-3.6.y.tar.gz and #tar -xvzf master.tar.gz (tools tools are used as the cross compile toolchain.
- Set the environment variable CCPREFIX:
- export CCPREFIX= /home/rajiv/RPI/tools-master/arm-bcm2708-linux-gnueabi/bin/arm-bcm2708-linux-gnueabi-
- Set the environment variable KERNEL_SRC:
- export KERNEL_SRC= /home/rajiv/RPI/linux-rpi-3.6.y

After this, now we begin the Kernel cross-compilation.

- First, clean the source:
- #make ARCH=arm CROSS_COMPILE=\${CCPREFIX} distclean
- Then set the correct config file for the Board, there are lots of config file that one can find '/arch/arm/configs/' folder.
- #make ARCH=arm CROSS_COMPILE=\${CCPREFIX} RPI_defconfig
- Otherwise, place the following config file that i have attached in the form of here(https://drive.google.com/file/d/0B6Cu_2GN5atpMzUxOXJCSmlzOEK/edit?usp=sharing), rename it as RPI_defcof and place it inside '/arch/arm/configs' directory of the Linux Kernel Source.
- After, this start the compilation
- #make ARCH=arm CROSS_COMPILE=\${CCPREFIX}
- #make ARCH=arm CROSS_COMPILE=\${CCPREFIX} modules_install
- #make ARCH=arm CROSS_COMPILE=\${CCPREFIX} install
- Finally, the kernel image will be built in the following path, 'arch/arm/boot/' as 'zimage', rename 'kernel.img'. Place it inside the Boot Partition of the RaspberryPi.

So, now our System Call is registered with the Kernel. After this, we write a similar user process as we had section, 'An Example: Invoking a given System Call', however, here we shall pass the index as, #59 to the Regist that is the new System Call Index that we have added.



```
Applications Places System
root@ubuntu: ~
File Edit View Search Terminal Help
root@raspberrypi:/home/rajiv/Examples# cat Example.s
.arch armv6
.section .rodata
.align 2
.data
HelloWorldString:
.ascii "Hello World\n"
.LC0:
.text
.align 2
.global main
.type main, %function
main:
@ args = 0, pretend = 0, frame = 72
@ frame_needed = 1, uses_anonymous_args = 0
@ stmfd sp!, {fp, lr}
@ add fp, sp, #4
@ Load all the arguments for write ()

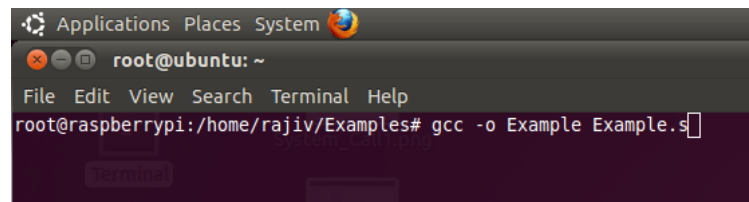
mov r7, #59
mov r0, #1
ldr r1, HelloWorldString
mov r2, #12
svc #0

@ Need to exit the program
mov r7, #1
mov r0, #0
svc #0

@ sub sp, fp, #4
@ ldmfd sp!, {fp, pc}
.L3:
.align 3
.L2:
.size main, .-main
.ident "GCC: (Debian 4.6.3-14+rpil) 4.6.3"
.section .note.GNU-stack,"",%progbits
```

Fig 6. The User Space Program to invoke our added System Call

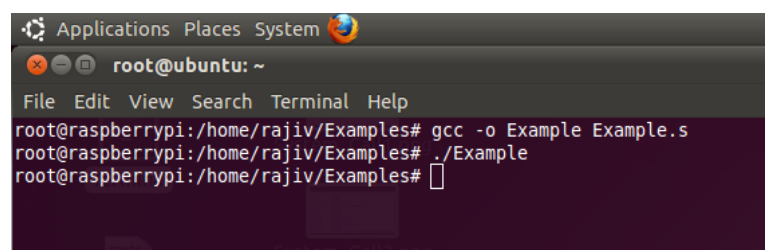
After, this we will compile the above user space program.



```
Applications Places System
root@ubuntu: ~
File Edit View Search Terminal Help
root@raspberrypi:/home/rajiv/Examples# gcc -o Example Example.s
```

Fig 7. Compiling the User Space Program

After, this we execute the Binary executable.



```
Applications Places System
root@ubuntu: ~
File Edit View Search Terminal Help
root@raspberrypi:/home/rajiv/Examples# gcc -o Example Example.s
root@raspberrypi:/home/rajiv/Examples# ./Example
root@raspberrypi:/home/rajiv/Examples#
```

Fig 8. Executing the Binary Executable

After, this we check the 'dmesg' using dmesg | tail command.

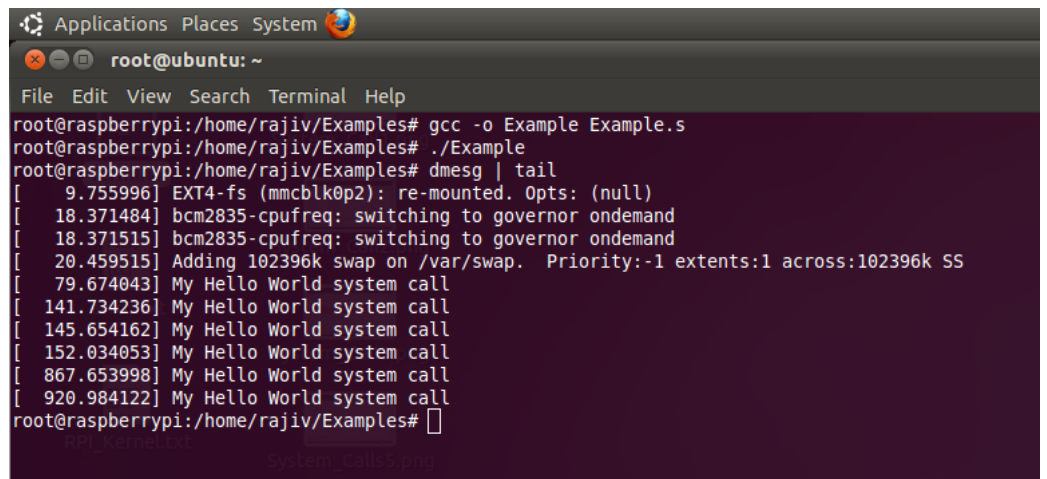

A terminal window titled 'root@ubuntu: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'root@raspberrypi:/home/rajiv/Examples#'. The user has run 'gcc -o Example Example.s', './Example', and 'dmesg | tail'. The output shows various kernel messages, including 'EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)', 'bcm2835-cpufreq: switching to governor ondemand', 'Adding 102396k swap on /var/swap. Priority:-1 extents:1 across:102396k SS', and several 'My Hello World system call' messages with timestamps like '[9.755996]', '[18.371484]', etc. The prompt is now 'root@raspberrypi:/home/rajiv/Examples# '.


Fig 9. The 'dmesg' shows that our added System Call is being executed. The 'printk' statements are present

So, this ends our discussion on Registering a Simple System Call with RaspberryPi.

Comments and thoughts most welcomed.

6 comments:


 **rashmi ramesh** September 25, 2016 at 12:38 PM
I'm seeing an undefined reference error to my newly defined system call in entry-common.S
[Reply](#)


 **rashmi ramesh** September 25, 2016 at 12:38 PM
I'm seeing an undefined reference error to my newly defined system call in entry-common.S
[Reply](#)

 **Unknown** October 10, 2017 at 9:25 AM
Hi. Thank you for your great post. I found a typo error in one of your file path, so I wanted to bring attention. Then second file path is

arch/arm/include/asm/unistd.h

In your post you accidentally typed arch/arm/include/arm/unistd.h
[Reply](#)

 **Rajiv** October 28, 2017 at 10:04 AM
Thanks for the Typo error...yep, i corrected it...Regards, Rajiv.
[Reply](#)

 **Unknown** December 8, 2017 at 4:28 PM
Hi Rajiv, Thank you for the post. I have a doubt, in which file do we set environmental variable?
[Reply](#)

[Replies](#)

 **Rajiv** December 12, 2017 at 3:15 AM
Consider adding this at the end of your \$HOME/.bashrc file to set environment variable
export PATH="\$PATH:/home/PACKAGE/bin"

After this run in the cmdline terminal,
source \$HOME/.bashrc

You can also set environment variables in \$HOME/.bash_aliases
say as an example, run the following in your cmdline terminal,

echo "export SOURCE_FOLDER=\$HOME/sources" >> \$HOME/.bash_aliases
echo "export LOCAL_BUILD=\$HOME/local-builds" >> \$HOME/.bash_aliases
echo "export LD_LIBRARY_PATH=\$HOME/local-builds/lib:\$LD_LIBRARY_PATH" >> \$HOME/.bash_aliases
echo "export PATH=\$HOME/local-builds/bin:\$PATH" >> \$HOME/.bash_aliases
echo "export PKG_CONFIG_PATH=\$HOME/local-builds/lib/pkgconfig:\$PKG_CONFIG_PATH" >> \$HOME/.bash_aliases


```
And after this run,  
source $HOME/.bash_aliases  
which will set your Env variables too.
```

Reply

Enter your comment...

Comment as:

Google Account

Publish

Preview

[Home](#)

Subscribe to: [Posts](#) ([Atom](#))