

Final-Project-OS

#Add System Call Called RABIX TO Kernel 5.8.1

First I will show the settings of my Virtual Machine :

Number of cores: 1

The capacity of memory is 1G

The kernel version is 5.8.1

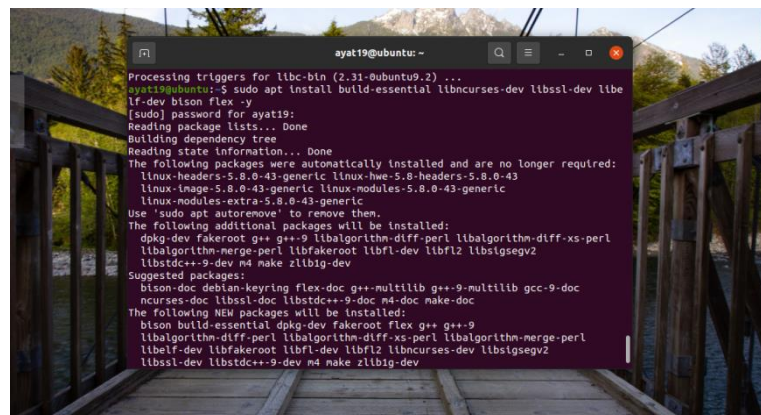
Second Steps to how to add a system call :

1- Make my Linux Ubuntu update:

```
sudo apt update && sudo apt upgrade -y
```

2- Install all packages that i will use to compile Kernal by :

```
sudo apt install build-essential libncurses-dev libssl-dev libelf-dev bison flex -y
```



```
Processing triggers for libc-bin (2.31-0ubuntu9.2) ...
ayat19@ubuntu:~$ sudo apt install build-essential libncurses-dev libssl-dev libelf-dev bison flex -y
[sudo] password for ayat19:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-headers-5.8.0-43-generic linux-hwe-5.8-headers-5.8.0-43
  linux-image-5.8.0-43-generic linux-modules-5.8.0-43-generic
  linux-modules-extra-5.8.0-43-generic
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  dpkg-dev fakeroot g++ g++-9 libalgorithm-diff-perl libalgorithm-diff-xs-perl
  libalgorithm-merge-perl libfakeroot libfl-dev libfl2 libgsgv2
  libstdc++9-dev m4 make zlib1g-dev
Suggested packages:
  bison-doc debian-keyring flex-doc g++-multilib g++-9-multilib gcc-9-doc
  ncurses-doc libssl-doc libstdc++9-doc m4-doc make-doc
The following NEW packages will be installed:
  build-essential dpkg-dev fakeroot flex g++ g++-9
  libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl
  libelf-dev libfakeroot libfl-dev libfl2 libncurses-dev libgsgv2
  libssl-dev libstdc++9-dev m4 make zlib1g-dev
```

3- Clean installed packages:

```
sudo apt clean && sudo apt autoremove -y
```

4- Download the source code of the Linux kernel 5.8.1:

```
wget -P ~/ https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.8.1.tar.xz
```

And unpack it by using

```
tar -xvf ~/linux-5.8.1.tar.xz -C ~/
```

5- Reboot My Computer

6- Change my working directory to the root directory of the recently unpacked source code

```
cd ~/linux-5.8.1/
```

7- Make a directory called RABIX and create file called RABIX.c in this file write a program

```
mkdir RABIX
```

```
nano RABIX/RABIX.c
```

```
#include <linux/kernel.h>
```

```
#include <linux/syscalls.h>
```

```
SYSCALL_DEFINE0(RABIX)
```

```
{
```

```
    printk("Welcome to RABIX.\n");
```

```
    return 0;
```

```
}
```



The screenshot shows a terminal window with the title 'ayat19@ubuntu: ~/linux-5.8.1'. Inside, the nano editor is open, editing the file 'RABIX/RABIX.c'. The editor's status bar at the top indicates 'GNU nano 4.8'. The code visible in the editor is:

```
#include <linux/kernel.h>
#include <linux/syscalls.h>

SYSCALL_DEFINE0(RABIX)
{
    printk("Welcome To RABIX.\n");
    return 0;
}
```

8- Now i will create a makefile

```
nano RABIX/Makefile
```

And write

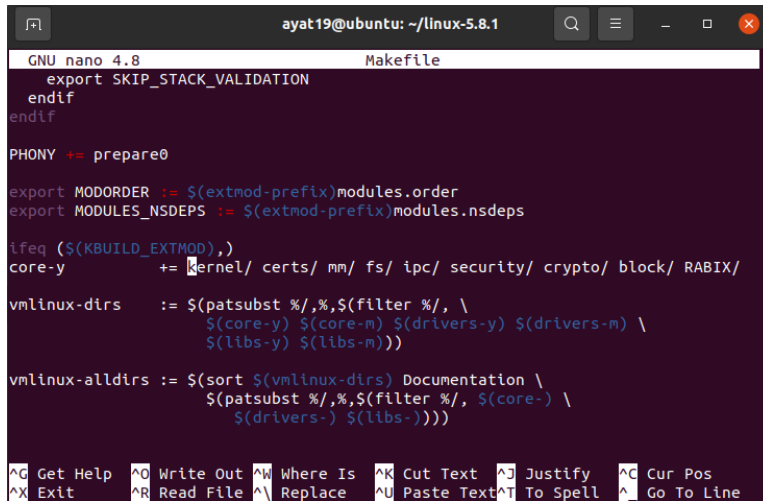
```
obj-y := RABIX.c
```

9- And i will open the Makefile to add the home directory to my system call to the main Makefile of the kernel.

Open the Makefile with the following command.

```
nano Makefile
```

and i will search for `core-y` it will apper in the second time of searching . We did the search to see this `kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/` I will add my home directory called `RABIX` .



```
GNU nano 4.8 Makefile
export SKIP_STACK_VALIDATION
endif
endif

PHONY += prepare0

export MODORDER := $(extmod-prefix)modules.order
export MODULES_NSDEPS := $(extmod-prefix)modules.nsdeps

ifeq ($(KBUILD_EXTMOD),)
core-y      += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ RABIX/

vmlinux-dirs := $(patsubst %/,,$(filter %/, \
    $(core-y) $(core-m) $(drivers-y) $(drivers-m) \
    $(libs-y) $(libs-m)))

vmlinux-alldirs := $(sort $(vmlinux-dirs) Documentation \
    $(patsubst %/,,$(filter %/, $(core-) \
    $(drivers-) $(libs-))))

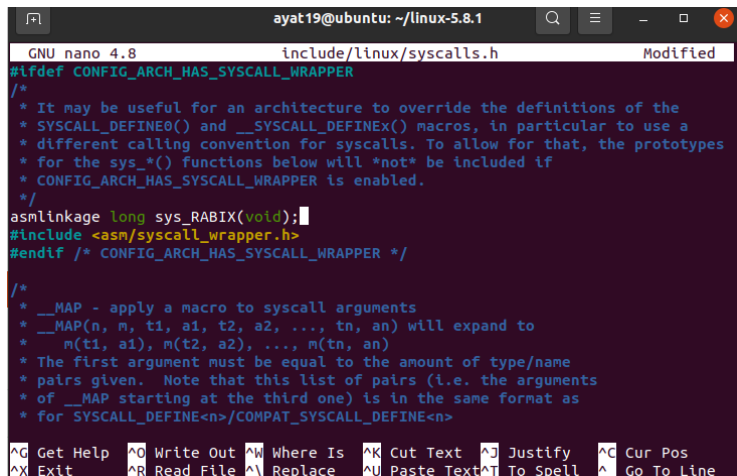
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos
^X Exit      ^R Read File ^\ Replace  ^U Paste Text ^T To Spell  ^_ Go To Line
```

10- And I will open the header file with the following command.

```
nano include/linux/syscalls.h
```

to add a corresponding function prototype for my system call to the header file of system calls.

Search for `endif` and put `asm linkage long sys_RABIX(void);` above it .



```
GNU nano 4.8 include/linux/syscalls.h Modified
#ifdef CONFIG_ARCH_HAS_SYSCALL_WRAPPER
/*
 * It may be useful for an architecture to override the definitions of the
 * SYSCALL_DEFINE() and __SYSCALL_DEFINE() macros, in particular to use a
 * different calling convention for syscalls. To allow for that, the prototypes
 * for the sys_*() functions below will *not* be included if
 * CONFIG_ARCH_HAS_SYSCALL_WRAPPER is enabled.
 */
asm linkage long sys_RABIX(void);
#include <asm/syscall_wrapper.h>
#endif /* CONFIG_ARCH_HAS_SYSCALL_WRAPPER */

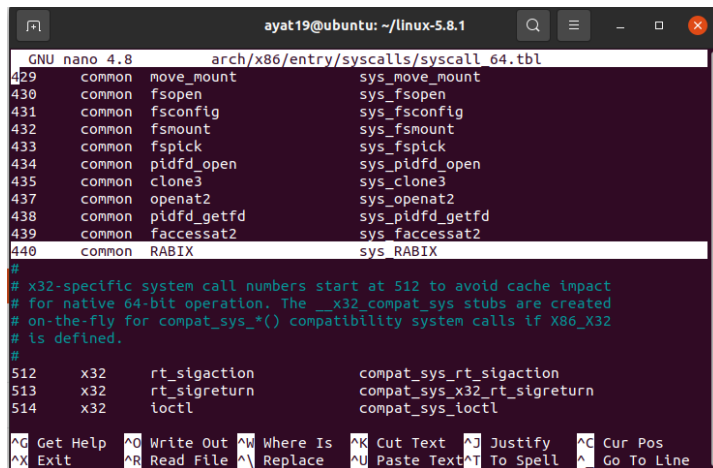
/*
 * __MAP - apply a macro to syscall arguments
 * __MAP(n, m, t1, a1, t2, a2, ..., tn, an) will expand to
 * m(t1, a1), m(t2, a2), ..., m(tn, an)
 * The first argument must be equal to the amount of type/name
 * pairs given. Note that this list of pairs (i.e. the arguments
 * of __MAP starting at the third one) is in the same format as
 * for SYSCALL_DEFINE<n>/COMPAT_SYSCALL_DEFINE<n>
 */
```

11- Add my system call to the kernel's system call table.

```
nano arch/x86/entry/syscalls/syscall_64.tbl
```

And I will navigate to the bottom of it even find a series of x32 system calls. I will put

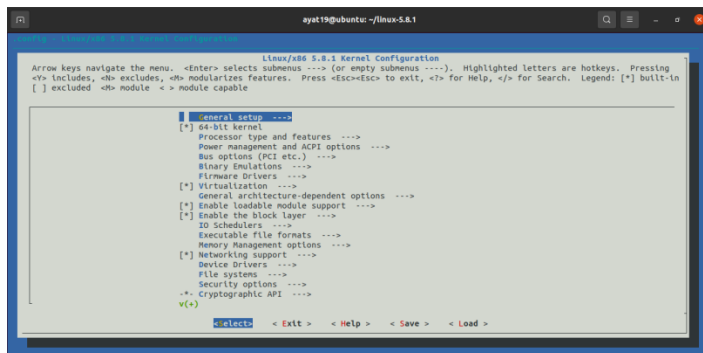
```
440 common RABIX sys_RABIX
```



```
GNU nano 4.8 arch/x86/entry/syscalls/syscall_64.tbl
429 common move_mount sys_move_mount
430 common fsopen sys_fsopen
431 common fsconfig sys_fsconfig
432 common fsmount sys_fsmount
433 common fspick sys_fspick
434 common pidfd_open sys_pidfd_open
435 common clone3 sys_clone3
437 common openat2 sys_openat2
438 common pidfd_getfd sys_pidfd_getfd
439 common faccessat2 sys_faccessat2
440 common RABIX sys_RABIX
#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation. The __x32_compat_sys stubs are created
# on-the-fly for compat_sys*() compatibility system calls if X86_X32
# is defined.
#
512 x32 rt_sigaction compat_sys_rt_sigaction
513 x32 rt_sigreturn compat_sys_x32_rt_sigreturn
514 x32 ioctl compat_sys_ioctl
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^M Replace ^U Paste Text ^T To Spell ^_ Go To Line
```

12- Configure the kernel.

```
make menuconfig
```



```
Linux/x86 5.8.1 Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus --- (or empty submenu ---). Highlighted letters are hotkeys. Pressing
<Y> includes, <N> excludes, <M> modularizes features. Press <Esc>+<Esc> to exit, <?> for help, </> for search. Legend: [*] built-in
[ ] excluded <M> module <+> module capable

[*] General setup ---
[*] 64-bit kernel
Processor type and features ---
Power management and ACPI options ---
Bus options (PCI etc.) ---
Binary Emulations ---
Firmware Drivers ---
[*] Virtualization ---
General architecture-dependent options ---
[*] Enable loadable module support ---
[*] Enable the block layer ---
IO Schedulers ---
Executable file formats ---
Memory Management options ---
[*] Networking support ---
Device Drivers ---
File systems ---
Security options ---
+-. Cryptographic API ---
V(*)

select < Exit > < Help > < Save > < Load >
```

13- Find out how many logical cores you have.

```
Nproc
```

14- Compile the kernel's source code.

```
make -j1
```



```

#include <linux/kernel.h>
#include <sys/syscall.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>

#define __NR_identity 440

long identity_syscall(void)
{
    return syscall(__NR_identity);
}

int main(int argc, char *argv[])
{
    long activity;
    activity = identity_syscall();

    if(activity < 0)
    {
        perror("Sorry Try again .");
    }

    else
    {
        printf("Congrats, And Weclome to RABIX\n");
    }

    return 0;
}

```

20- Compile the C file just created, and run C file

```
gcc -o rabix rabix.c
```

```
./rabix
```

Will display Congrats, And Weclome to RABIX

21- Check the last line of the dmesg output

The print function:

```
Welcome to RABIX
```