

Adding a new System Call to your Kernel

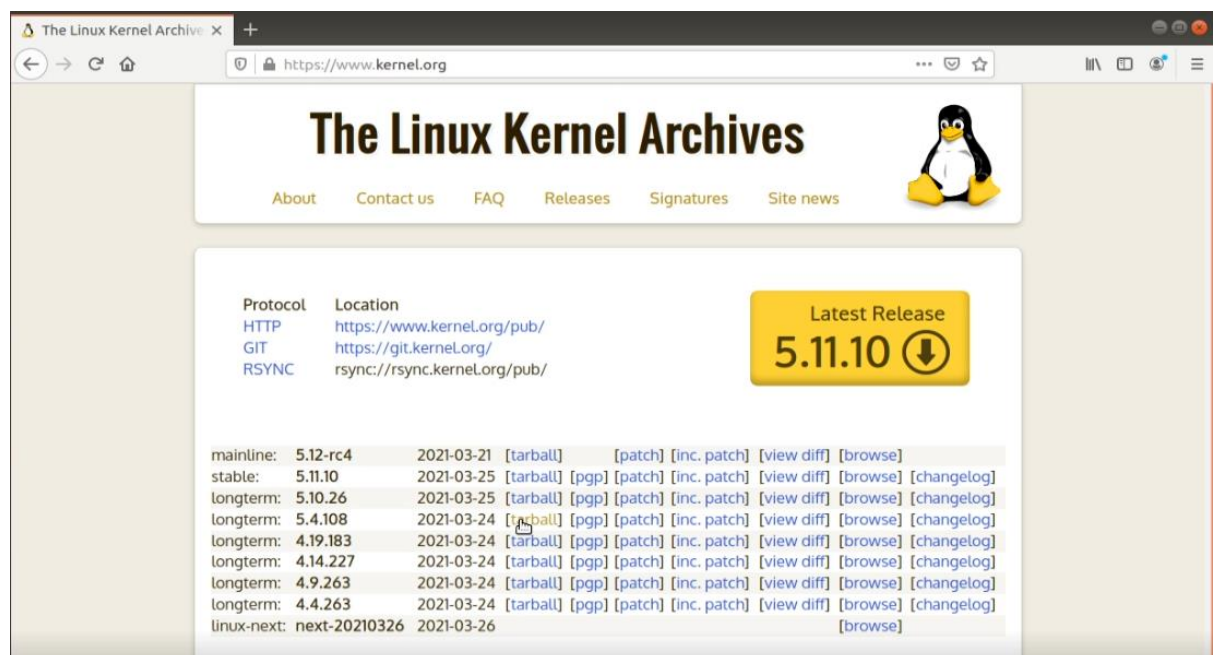
Prerequisites:

- `sudo apt-get install gcc`
- `sudo apt-get install libncurses5-dev`
- `sudo apt-get install bison`
- `sudo apt-get install flex`
- `sudo apt install make`
- `sudo apt-get install libssl-dev`
- `sudo apt-get install libelf-dev`
- `sudo add-apt-repository "deb http://archive.ubuntu.com/ubuntu $(lsb_release -sc) main universe"`
- `sudo apt-get update`
- `sudo apt-get upgrade`

Steps:

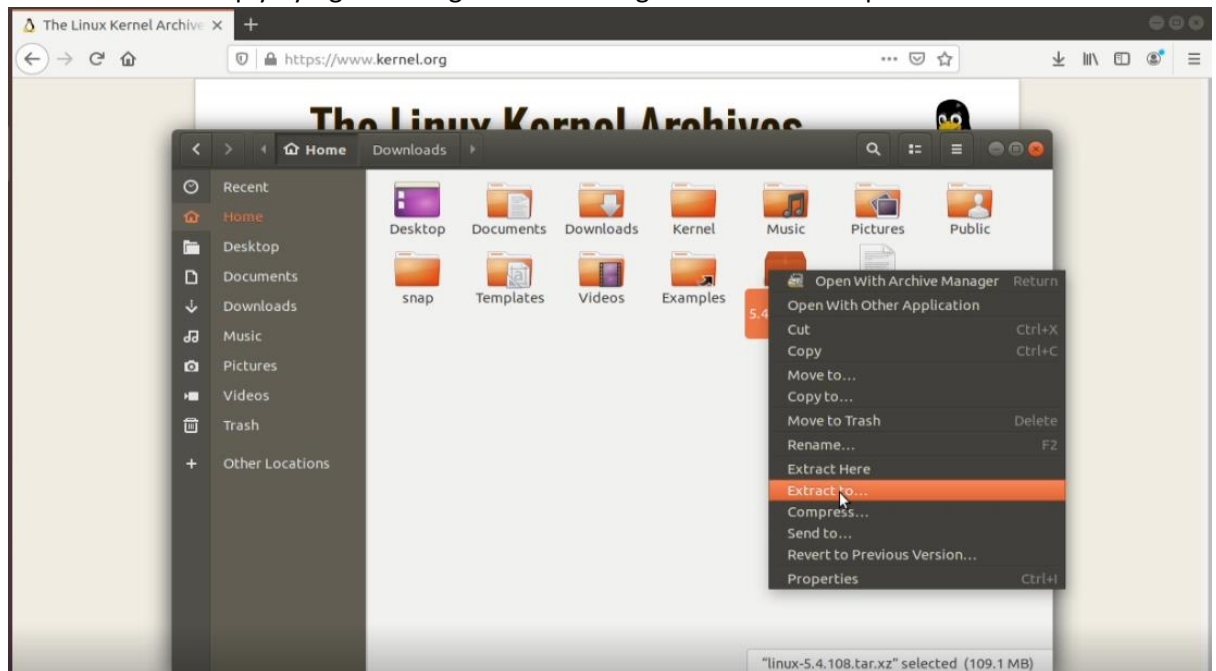
1. Downloading a kernel:

First of all, we have to download a kernel from kernel.org. We can either do this by using the “wget” command or download it manually by clicking the “tarball” option.



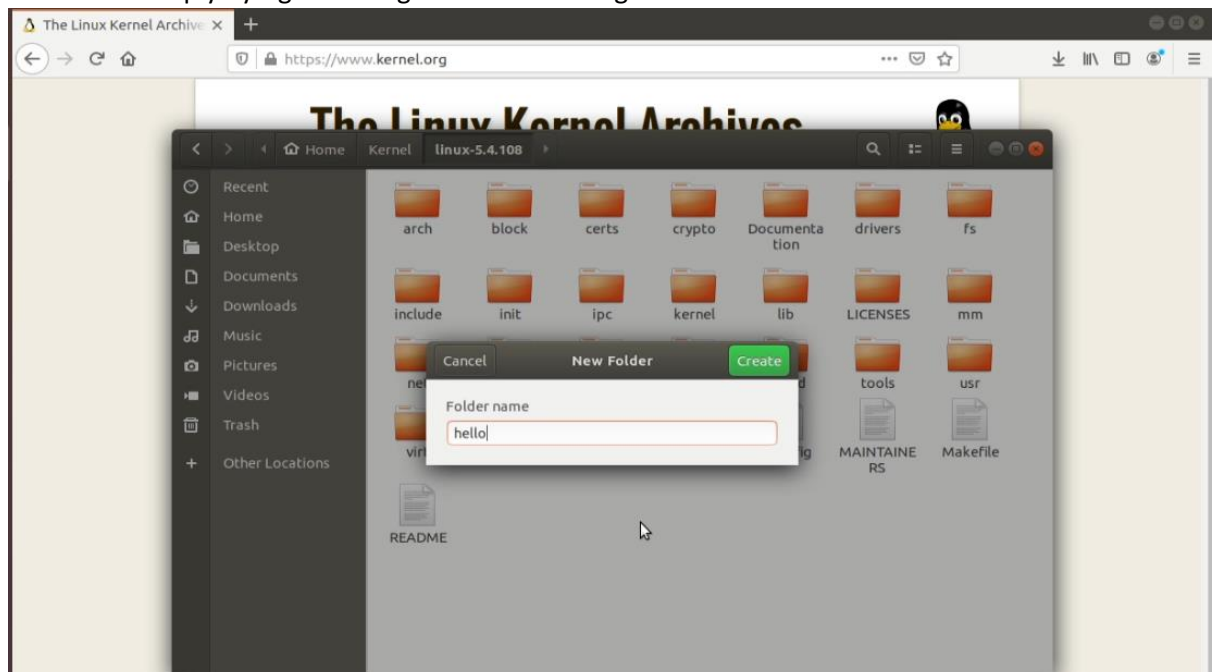
2. Extracting the kernel:

Now, go to the folder where the kernel is downloaded and extract it by typing “tar -xvf *filename*” or simply by right clicking it and selecting the “extract to” option



3. Making a new folder called hello:

Go into the folder where you extracted the kernel and go inside the kernel's folder and create a new directory by either opening the terminal there and typing “mkdir *folder name*” or simply by right clicking there and clicking on new folder



4. Adding a C code for the system call:

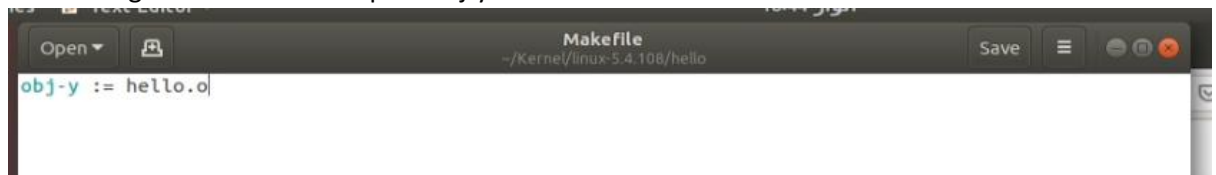
Now, go to the folder which we created just now and open the terminal there and create a new C code file by typing “gedit hello.c” and paste the following code there:

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

asmlinkage long sys_hello(void)
{
    printk("Hello world\n");
    return 0;
}
```

Code explanation:

- a. We used #include <linux/kernel> because we are building a system call for our linux kernel.
 - b. Amslinkage simply means that the arguments for this function will be on the stack instead of the CPU registers.
 - c. Printk is used instead of printf because we are going to print in the kernel's log file.
 - d. If the code is run and it returns 0, then it will mean that our program ran successfully and Hello world is written to out kernel's log file.
5. Creating a Makefile for the C code:
- Now, we have to create a Makefile for our new folder to ensure that the code in the folder is always compiled whenever the kernel is compiled. In order to do this, we type in our terminal "gedit Makefile" and put "obj-y := hello.o"



6. Adding the new code into the system table file:

Since we are creating a 64-bit system call according to our system we have to add the system call entry into the `syscall_64.tbl` file which keeps the name of all the system calls in our system. If our system was a 32-bit system, we would have to add our system call into `syscall_32.tbl` (We can check the type of our system by typing “`uname -m`” in a terminal). This `tbl` file is located inside the kernel folder in `/arch/x86/entry/syscalls/syscall_64.tbl`. We can go into this directory by using `cd` and then edit the file by typing “`gedit syscall_64.tbl`”







```

307 64 sendmmsg __x64_sys_sendmmsg
308 common setns __x64_sys_setns
309 common getcpu __x64_sys_getcpu
310 64 process_vm_readv __x64_sys_process_vm_readv
311 64 process_vm_writev __x64_sys_process_vm_writev
312 common kcmp __x64_sys_kcmp
313 common finit_module __x64_sys_finit_module
314 common sched_setattr __x64_sys_sched_setattr
315 common sched_getattr __x64_sys_sched_getattr
316 common renameat2 __x64_sys_renameat2
317 common seccomp __x64_sys_seccomp
318 common getrandom __x64_sys_getrandom
319 common memfd_create __x64_sys_memfd_create
320 common kexec_file_load __x64_sys_kexec_file_load
321 common bpf __x64_sys_bpf
322 64 execveat __x64_sys_execveat/ptregs
323 common userfaultfd __x64_sys_userfaultfd
324 common membarrier __x64_sys_membarrier
325 common mlock2 __x64_sys_mlock2
326 common copy_file_range __x64_sys_copy_file_range
327 64 preadv2 __x64_sys_preadv2
328 64 pwritev2 __x64_sys_pwritev2
329 common pkey_mprotect __x64_sys_pkey_mprotect
330 common pkey_alloc __x64_sys_pkey_alloc
331 common pkey_free __x64_sys_pkey_free
332 common statx __x64_sys_statx
333 common io_pgetevents __x64_sys_io_pgetevents
334 common rseq __x64_sys_rseq
335 64 hello sys_hello
# don't use numbers 387 through 423, add new calls after the last
# 'common' entry
424 common pidfd_send_signal __x64_sys_pidfd_send_signal
425 common io_uring_setup __x64_sys_io_uring_setup
426 common io_uring_enter __x64_sys_io_uring_enter
427 common io_uring_register __x64_sys_io_uring_register
428 common open_tree __x64_sys_open_tree
429 common move_mount __x64_sys_move_mount

```

Now we have to add the prototype of our system call in the system's header file which is located in the kernel folder then `"/include/linux/syscalls.h"`. We have to add the prototype of our system call function in this file.

```

Open ▾  *syscalls.h -/Kernel/linux-5.4.106/include/linux Save ≡   
extern long do_sys_truncate(const char __user *pathname, loff_t length);

static inline long ksys_truncate(const char __user *pathname, loff_t length)
{
    return do_sys_truncate(pathname, length);
}

static inline unsigned int ksys_personality(unsigned int personality)
{
    unsigned int old = current->personality;

    if (personality != 0xffffffff)
        set_personality(personality);

    return old;
}

/* for __ARCH_WANT_SYS_IPC */
long ksys_semtimeop(int semid, struct sembuf __user *tsops,
                    unsigned int nsops,
                    const struct __kernel_timespec __user *timeout);
long ksys_semget(key_t key, int nsems, int semflg);
long ksys_old_semctl(int semid, int semnum, int cmd, unsigned long arg);
long ksys_msgget(key_t key, int msgflg);
long ksys_old_msgctl(int msqid, int cmd, struct msqid_ds __user *buf);
long ksys_msgrcv(int msqid, struct msgbuf __user *msgp, size_t msgsz,
                 long msgtyp, int msgflg);
long ksys_msgsnd(int msqid, struct msgbuf __user *msgp, size_t msgsz,
                 int msgflg);
long ksys_shmget(key_t key, size_t size, int shmflg);
long ksys_shmdt(char __user *shmaddr);
long ksys_old_shmctl(int shmid, int cmd, struct shmid_ds __user *buf);
long compat_ksys_semtimeop(int semid, struct sembuf __user *tsops,
                           unsigned int nsops,
                           const struct old_timespec32 __user *timeout);
asmlinkage long sys_hello(void);


#endif

```

C/ObjC Header ▾ Tab Width: 8 ▾ Ln 1423, Col 33 ▾ INS

8. Changing version and adding the hello folder in the kernel's Makefile:

Now, we have to add our roll number in the extraversion of the kernel's make file and we have to add the new module that we created into our kernel's make file. For this, we open the Makefile of the kernel and search for "core-y" and go to it's second instance which is under "KBUILD_EXTMOD" and add our new module which is "hello" at the end of it. At the end, our make file will look something like this:



```

export SKIP_STACK_VALIDATION
endif
endif

PHONY += prepare0

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

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

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

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

build-dirs      := $(vmlinux-dirs)
clean-dirs      := $(vmlinux-alldirs)

init-y          := $(patsubst %/, %/built-in.a, $(init-y))
core-y          := $(patsubst %/, %/built-in.a, $(core-y))
drivers-y       := $(patsubst %/, %/built-in.a, $(drivers-y))
net-y           := $(patsubst %/, %/built-in.a, $(net-y))
libs-y1         := $(patsubst %/, %/lib.a, $(libs-y))
libs-y2         := $(patsubst %/, %/built-in.a, $(filter-out %.a, $(libs-y)))
virt-y          := $(patsubst %/, %/built-in.a, $(virt-y))

# Externally visible symbols (used by link-vmlinux.sh)
export KBUILD_VMLINUX_OBJS := $(head-y) $(init-y) $(core-y) $(libs-y2) \
    $(drivers-y) $(net-y) $(virt-y)
export KBUILD_VMLINUX_LIBS := $(libs-y1)
export KBUILD_LDS           := arch/$(SRCARCH)/kernel/vmlinux.lds
export LDFLAGS_vmlinux
# used by scripts/Makefile.package
export KBUILD_ALLDIRS := $(sort $(filter-out arch/%, $(vmlinux-alldirs)) LICENSES arch include

```

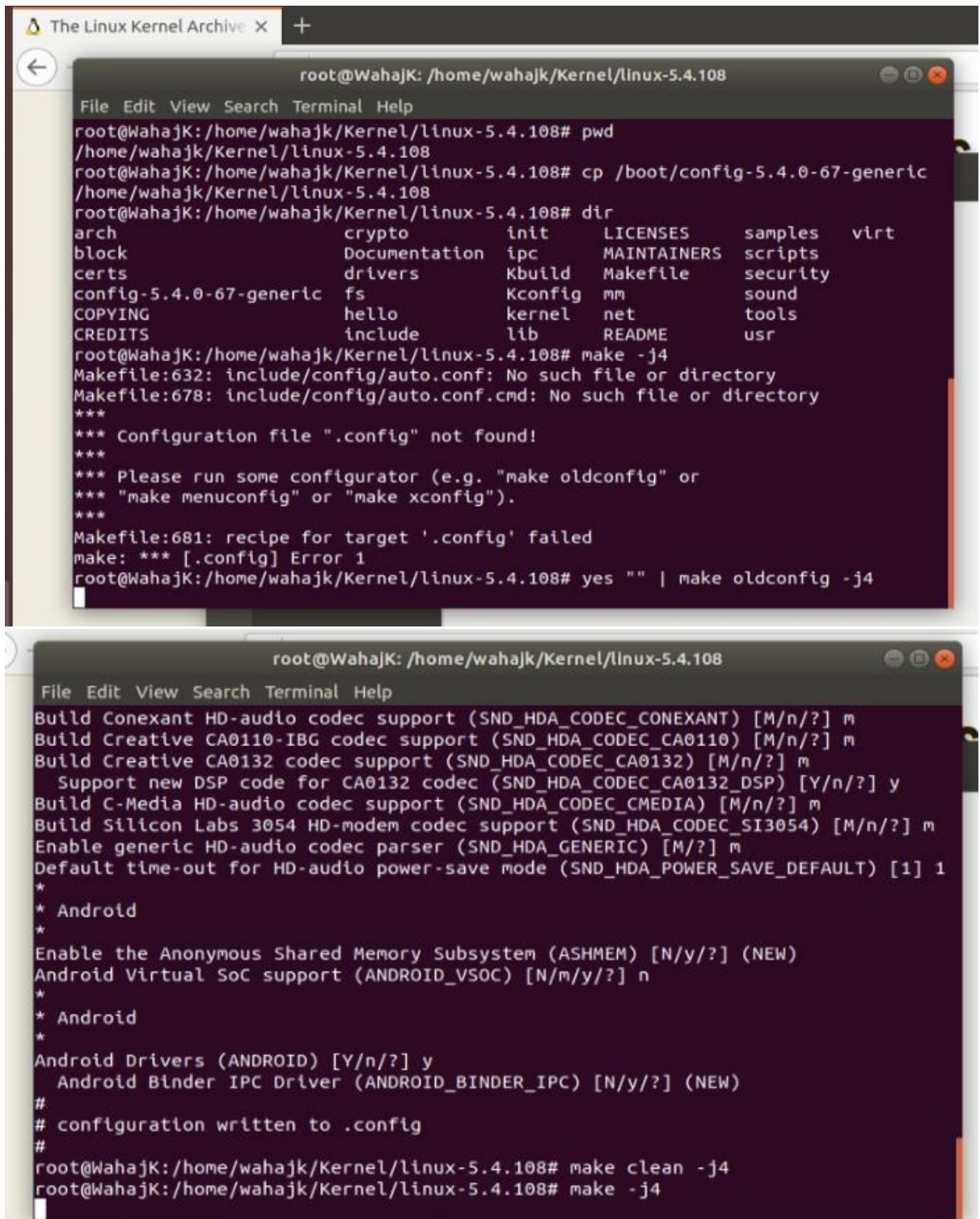
9. Creating a config file:

Now we have to create a config file for our kernel. The order of the steps before this can change but the order of this step and the steps coming right after it can't change. We can either create a Menuconfig or simply copy the oldconfig. I will be copying the oldconfig and using that config for my new kernel. First of all, we search for the config that we currently have by typing "ls /boot | grep config" and then we copy the config that is shown to us by typing "cp /boot/config-4.10.0-28-generic *our linux kernel directory*". Then we create the old config by typing "yes "" | make oldconfig -j4", by doing so, the system will automatically create the new config for us and select the default option for everything.

10. Cleaning and Compiling the kernel:

We have to clean all of our existing object and executable file because compiler sometimes link or compile files incorrectly and to avoid this, we delete all of our old object and executable files by typing "make clean -j4" (It is better to switch to super user mode by typing "sudo su" before running the commands after this) and when this all is done, we type "make -j4" to start building our kernel (-j4 allocates the multiple cores that our system have for compiling. If we don't do this, the system will only use a single core for compiling the

kernel which is rather slow. Adding -j4 will increase the speed of our compiling by almost 4 times. Note that 4 is the number of Cores that I have in my Laptop. You can check your number of cores by typing the command "lscpu" and enter the number accordingly)



```
root@WahajK: /home/wahajk/Kernel/linux-5.4.108
File Edit View Search Terminal Help
root@WahajK: /home/wahajk/Kernel/linux-5.4.108# pwd
/home/wahajk/Kernel/linux-5.4.108
root@WahajK: /home/wahajk/Kernel/linux-5.4.108# cp /boot/config-5.4.0-67-generic
/home/wahajk/Kernel/linux-5.4.108
root@WahajK: /home/wahajk/Kernel/linux-5.4.108# dir
arch          crypto        init          LICENSES      samples      virt
block         Documentation ipc           MAINTAINERS   scripts
certs         drivers      Kbuild       Makefile      security
config-5.4.0-67-generic fs           Kconfig      mm            sound
COPYING       hello        kernel       net           tools
CREDITS       include      lib          README       usr
root@WahajK: /home/wahajk/Kernel/linux-5.4.108# make -j4
Makefile:632: include/config/auto.conf: No such file or directory
Makefile:678: include/config/auto.conf.cmd: No such file or directory
***
*** Configuration file ".config" not found!
***
*** Please run some configurator (e.g. "make oldconfig" or
*** "make menuconfig" or "make xconfig").
***
Makefile:681: recipe for target '.config' failed
make: *** [.config] Error 1
root@WahajK: /home/wahajk/Kernel/linux-5.4.108# yes "" | make oldconfig -j4

Build Conexant HD-audio codec support (SND_HDA_CODEC_CONEXANT) [M/n/?] m
Build Creative CA0110-IBG codec support (SND_HDA_CODEC_CA0110) [M/n/?] m
Build Creative CA0132 codec support (SND_HDA_CODEC_CA0132) [M/n/?] m
  Support new DSP code for CA0132 codec (SND_HDA_CODEC_CA0132_DSP) [Y/n/?] y
Build C-Media HD-audio codec support (SND_HDA_CODEC_CMEDIA) [M/n/?] m
Build Silicon Labs 3054 HD-modem codec support (SND_HDA_CODEC_SI3054) [M/n/?] m
Enable generic HD-audio codec parser (SND_HDA_GENERIC) [M/?] m
Default time-out for HD-audio power-save mode (SND_HDA_POWER_SAVE_DEFAULT) [1] 1
*
* Android
*
Enable the Anonymous Shared Memory Subsystem (ASHMEM) [N/y/?] (NEW)
Android Virtual SoC support (ANDROID_VSOC) [N/m/y/?] n
*
* Android
*
Android Drivers (ANDROID) [Y/n/?] y
  Android Binder IPC Driver (ANDROID_BINDER_IPC) [N/y/?] (NEW)
#
# configuration written to .config
#
root@WahajK: /home/wahajk/Kernel/linux-5.4.108# make clean -j4
root@WahajK: /home/wahajk/Kernel/linux-5.4.108# make -j4
```

Now we have to wait until our Kernel image is built and ready. If we see "Kernel image is ready" when the command is done executing, that means that our kernel image is ready to be installed. (I forgot to take screenshots of this point)

11. Installing modules:

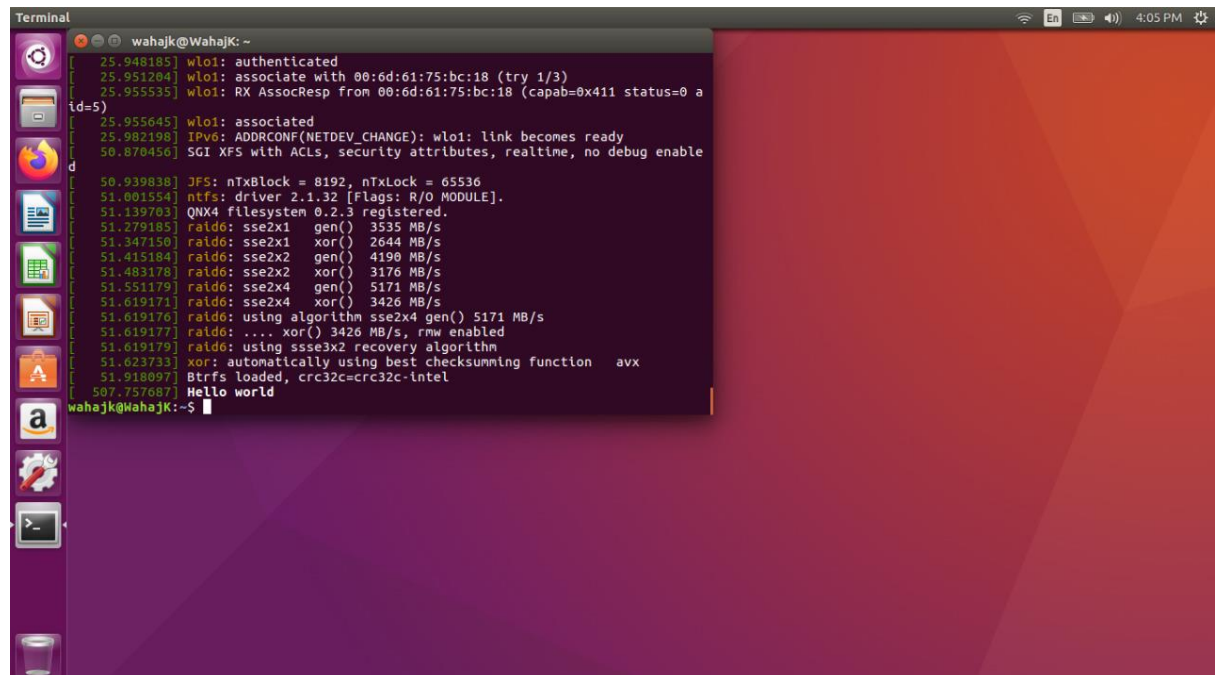
Now we have to install the kernel that we built by typing “make modules_install install” which will install the kernel and update our grub as well. When this all is done and the terminal says “done”, then we can restart our laptop either manually or by typing “shutdown -r now” and hold the “Shift” key while it is restarting to open up the grub menu and switch to the new kernel which we just installed. (I forgot to take screenshot of this point as well.)

12. Checking if the System call is Working Properly:

After logging into the newly compiled kernel, we check the system call by making a C code named “userspace.c” and putting the following code in it:

```
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>
int main()
{
    long int i = syscall(335);
    printf("System call sys_hello returned %ld\n", i);
    return 0;
}
```

Now we compile the code by typing “gcc userspace.c” and executing it by typing “./a.out”. If it returns 0, this means that our code has compiled successfully and the system call is working fine (Note that in calling syscall(335), 335 is the number where we added our system call in the table) and finally, we run “dmesg” to see the kernel messages and we will find “Hello World” written at the end of it.



```
Terminal
wahajk@WahajK: ~
25.948185] wlo1: authenticated
25.951204] wlo1: associate with 00:6d:61:75:bc:18 (try 1/3)
25.955535] wlo1: RX AssocResp from 00:6d:61:75:bc:18 (capab=0x411 status=0 a
id=5)
25.955645] wlo1: associated
25.982198] IPv6: ADDRCONF(NETDEV_CHANGE): wlo1: link becomes ready
50.870456] SGI XFS with ACLs, security attributes, realtime, no debug enable
50.939838] JFS: nTxBlock = 8192, nTxLock = 65536
51.001554] ntfs: driver 2.1.32 [Flags: R/O MODULE].
51.139703] QNX4 filesystem 0.2.3 registered.
51.279185] raid6: sse2x1 gen() 3535 MB/s
51.347150] raid6: sse2x1 xor() 2644 MB/s
51.415184] raid6: sse2x2 gen() 4190 MB/s
51.483178] raid6: sse2x2 xor() 3176 MB/s
51.551179] raid6: sse2x4 gen() 5171 MB/s
51.619171] raid6: sse2x4 xor() 3426 MB/s
51.619176] raid6: using algorithm sse2x4 gen() 5171 MB/s
51.619177] raid6: .... xor() 3426 MB/s, rmw enabled
51.619179] raid6: using ssse3x2 recovery algorithm
51.623733] xor: automatically using best checksumming function avx
51.918097] Btrfs loaded, crc32c=crc32c-intel
507.757687] Hello world
wahajk@WahajK:~$
```