AOS Assignment 2 Report

Adding Kernel (version: 4.19.210) to the system:

1. Downloading the kernel using the following command:

`wget https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.19.210.tar.xz`

2. Extracting the kernel source code using:

```
sudo tar -xvf linux-4.17.4.tar.xz -C/usr/src/
```

Installing and Loading the new kernel:

1. After extracting, go to the directory : `/usr/src/linux-4.17.4/`

2. Compile the kernel: `sudo make`

3. Installing kernel: `sudo make modules_install install`

4. Reboot the system and during the boot you can find a menu where you need to choose Advanced ubuntu settings, followed by a menu where you can choose the kernel version you want to load i.e `4.19.210`

Adding system call to the kernel:

1. Go to: `/usr/src/linux-4.17.4/`

2. Make a directory and create a new C file where you write your new system call code.

3. Inside the system call add necessary headers to create a system call.

4. To create a system call there are multiple ways to create and one of them is

```
SYSCALL_DEFINE# (<name of the system call>)
```

In the above code # indicates the number of arguments you want to add to the function.

5. After writing the system call, we need to add system call entry in *syscall_64.tbl* which is in: `arch/x86/entry/syscalls/`

6. Every system call function should be declared in a header file called *syscalls.h* at the end of the file.

Ex: `asmlinkage long sys_srikanthello(void);`

7. Each system call's object file statement should be written in a Makefile, and the makefile should be in the same folder as the system call.

Ex: *obj-y := hello.o* should be in the makefile.

8. Make file and the c file which contains system calls should be in the same directory.

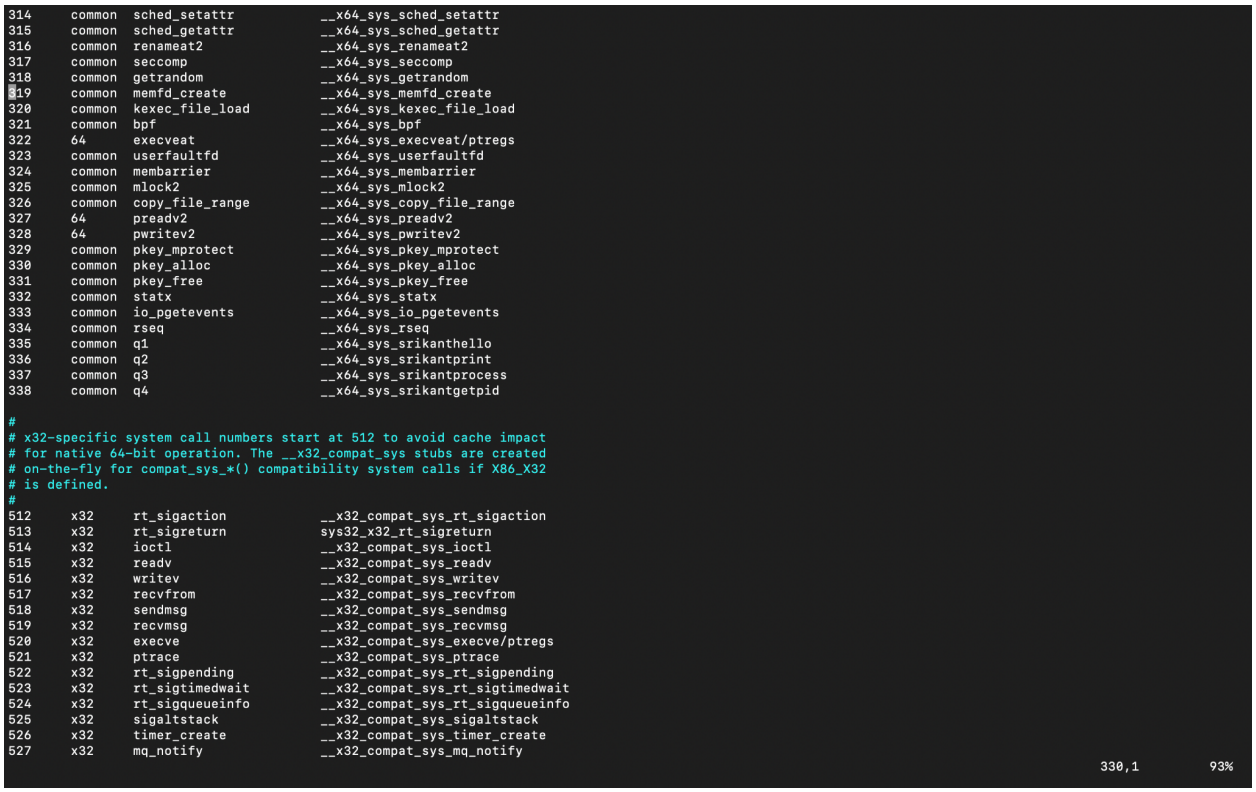
Ex : If *hello.c* contains a system call which is in a directory *sys_dir*, then the *Makefile* should also be in the same directory.

9. In `/usr/src/linux-4.17.4/` directory , there is a *Makefile* and in this make file we need to include the directory of the file which contains our system call.

Ex: If our system call is defined in *hello.c* which is in *sys_dir* then we need to include *sys_dir* in this make file as *core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ sys_dir/*

Describing Makefile, sys_64.tbl, syscalls.h :

sys_64.tbl:



```
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     common  q1                      __x64_sys_srikanthello
336     common  q2                      __x64_sys_srikantprint
337     common  q3                      __x64_sys_srikantprocess
338     common  q4                      __x64_sys_srikantgetpid

#
# 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            __x32_compat_sys_rt_sigaction
513     x32     rt_sigreturn            sys32_x32_rt_sigreturn
514     x32     ioctl                   __x32_compat_sys_ioctl
515     x32     readv                   __x32_compat_sys_readv
516     x32     writev                  __x32_compat_sys_writev
517     x32     recvfrom                __x32_compat_sys_recvfrom
518     x32     sendmsg                 __x32_compat_sys_sendmsg
519     x32     recvmsg                 __x32_compat_sys_recvmsg
520     x32     execve                  __x32_compat_sys_execve/ptregs
521     x32     ptrace                  __x32_compat_sys_ptrace
522     x32     rt_sigpending           __x32_compat_sys_rt_sigpending
523     x32     rt_sigtimedwait         __x32_compat_sys_rt_sigtimedwait
524     x32     rt_sigqueueinfo         __x32_compat_sys_rt_sigqueueinfo
525     x32     sigaltstack             __x32_compat_sys_sigaltstack
526     x32     timer_create            __x32_compat_sys_timer_create
527     x32     mq_notify               __x32_compat_sys_mq_notify
                                                                  330,1         93%
```

Above image is *sys_64.tbl*

In the above image, entries 335 to 338 contains the following system calls:

__x64_sys_srikanthello

__x64_sys_srikantprint

__x64_sys_srikantprocess

__x64_sys_srikantgetpid

- First column is the serial number.
- Third column contains the folder name where our system call is present.

- Fourth column contains the name of the system call.

  Here we have a prefix *__x64_sys_srikanthello*

  *__x64_sys_* : Indicates that this is a system call.

syscalls.h:

```
}

extern int __close_fd(struct files_struct *files, unsigned int fd);

/*
 * In contrast to sys_close(), this stub does not check whether the syscall
 * should or should not be restarted, but returns the raw error codes from
 * __close_fd().
 */
static inline int ksys_close(unsigned int fd)
{
        return __close_fd(current->files, fd);
}

extern long do_sys_open(int dfd, const char __user *filename, int flags,
                        umode_t mode);

static inline long ksys_open(const char __user *filename, int flags,
                             umode_t mode)
{
        if (force_o_largefile())
                flags |= O_LARGEFILE;
        return do_sys_open(AT_FDCWD, filename, flags, mode);
}

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;
}
asmlinkage long sys_hello(void);
asmlinkage long sys_srikanthello(void);
asmlinkage long sys_srikantprint(char*);
asmlinkage long sys_srikantprocess(void);
asmlinkage long sys_srikantgetpid(void);

#endif                                                          1301,1        Bot
```

All the system calls declarations should be added at the end of the file, before the end of last if block i.e ***#endif***

Makefile:

```
ifeq ($(KBUILD_EXTMOD),)
        core-y          += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ hello/ q1/ q2/ q3/ q4/
```

You can find the above line starting with *core-y* in Makefile which is in line: 987

Add the folders where you defined your system call.

Here *q1/, q2/, q3/, q4/* are folders containing *srikanthello*, *srikantprint*, *srikantprocess*, *srikantgetpid* system calls respectively.

Describing system calls:

1. srikanthello()

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

SYSCALL_DEFINE0(srikanthello)
{
        printk("Hello world\n");
        return 0;
}
```

- In the above image, the system call is defined using *SYSCALL_DEFINE0* where 0 indicates the number of parameters.
- This system call prints ***Hello world*** inside the log file.
- printk function is used to print our messages in the log file.
- Here k indicates: kernel

2. srikantprint()

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

SYSCALL_DEFINE1(srikantprint, char *, msg)
{
  char buffer[256];
  long chunk = strncpy_from_user(buffer, msg, sizeof(buffer));
  if (chunk < 0 || chunk == sizeof(buffer))
          return -EFAULT;
  printk(KERN_INFO "SYSTEM CALL srikantprint: \"%s\"\n", buffer);
  return 0;
}
```

- In the above image, the system call is defined using *SYSCALL_DEFINE1* where 1 indicates the number of parameters.
- This system call takes one string argument and prints it in the log file.

3. srikantprocess()

```c
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/cred.h>
#include <linux/syscalls.h>

SYSCALL_DEFINE0(srikantprocess)
{
        struct task_struct *parent = current->parent;
        printk("Process ID: %d ; Parent process ID: %d",current->pid,parent->pid);
        return 0;
}
```

- In the above image, the system call is defined using *SYSCALL_DEFINE0* where 0 indicates the number of parameters.
- Here this system call prints the process IDs of parent and the child.
- Predefined struct called *task_struct* is used to get the respective IDs.

4. srikantgetpid()

```c
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/sched.h>
#include <linux/cred.h>


SYSCALL_DEFINE0(srikantgetpid)
{
        return task_tgid_vnr(current);
}
```

- In the above image system call is defined using *SYSCALL_DEFINE0* wher 0 indicates the number of parameters.
- This system call returns the process ID of the process which invoked this system call.
- Here *task_tgid_vnr* is the system call which is invoked to get the process ID of the process.

Testing:

```c
#define _GNU_SOURCE
#include <unistd.h>
#include <sys/syscall.h>
#include <stdio.h>

int main()
{
   long int amma = syscall(335);
   printf("System call srikanthello returned: %ld\n",amma);

   int res1=syscall(337);
   printf("System call srikantprocess returned: %d \n",res1);

   long int res2=syscall(338);
   printf("System call srikantgetpid returned: %ld\n",res2);

   char *arg = "Srikant Konduri";
   printf("Making system call with \"%s\n", arg);
   long res = syscall(336,arg);
   printf("System call srikantprint returned: %ld\n", res);
   return 0;
}
```

Output:

```
azureuser@SkVM:~$ ./a.out
System call srikanthello returned: 0
System call srikantprocess returned: 0
System call srikantgetpid returned: 1276
Making system call with "Srikant Konduri
System call srikantprint returned: 0
azureuser@SkVM:~$
```

After running the above test file named test.c we got the above returned values.

- System call *`srikanthello()`* returns 0 on successful call and logs *`hello world`* in the log file.
- System call *`srikantprocess()`* returns 0 on successful call and logs process ID of parent and current process in the log file.
- System call *`srikantgetpid()`* return the process ID of a.out (i.e test.c) which is the current process.
- System call *`srikantprint()`* prints the argument in the log file which we have sent. Here “Srikant Konduri” is sent as an argument.

```
[    5.478086] audit: type=1400 audit(1663346158.980:13): apparmor="STATUS" operation="profile_load" profile="unconfined" name="man_groff" pid=518 comm="appar
mor_parser"
[    5.483258] audit: type=1400 audit(1663346158.984:14): apparmor="STATUS" operation="profile_load" profile="unconfined" name="lsb_release" pid=520 comm="app
armor_parser"
[    5.486477] audit: type=1400 audit(1663346158.988:15): apparmor="STATUS" operation="profile_load" profile="unconfined" name="/usr/sbin/tcpdump" pid=521 com
m="apparmor_parser"
[    5.531992] audit: type=1400 audit(1663346159.032:16): apparmor="STATUS" operation="profile_load" profile="unconfined" name="/snap/snapd/16292/usr/lib/snap
d/snap-confine" pid=534 comm="apparmor_parser"
[    5.531995] audit: type=1400 audit(1663346159.032:17): apparmor="STATUS" operation="profile_load" profile="unconfined" name="/snap/snapd/16292/usr/lib/snap
d/snap-confine//mount-namespace-capture-helper" pid=534 comm="apparmor_parser"
[    5.534266] audit: type=1400 audit(1663346159.036:18): apparmor="STATUS" operation="profile_load" profile="unconfined" name="/snap/snapd/16778/usr/lib/snap
d/snap-confine" pid=535 comm="apparmor_parser"
[    6.174932] mlx5_core 971e:00:02.0 enP38686s1: Link up
[    6.175657] hv_netvsc 6045bdae-92bb-6045-bdae-92bb6045bdae eth0: Data path switched to VF: enP38686s1
[    6.176834] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
[    6.176851] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[    8.792715] hv_netvsc 6045bdae-92bb-6045-bdae-92bb6045bdae eth0: Data path switched from VF: enP38686s1
[    8.959447] mlx5_core 971e:00:02.0 enP38686s1: Link up
[    8.960369] hv_netvsc 6045bdae-92bb-6045-bdae-92bb6045bdae eth0: Data path switched to VF: enP38686s1
[    9.136876] hv_netvsc 6045bdae-92bb-6045-bdae-92bb6045bdae eth0: Data path switched from VF: enP38686s1
[    9.498569] mlx5_core 971e:00:02.0 enP38686s1: Link up
[    9.499333] hv_netvsc 6045bdae-92bb-6045-bdae-92bb6045bdae eth0: Data path switched to VF: enP38686s1
[   15.289759] EXT4-fs (sdb1): mounted filesystem with ordered data mode. Opts: (null)
[   18.412837] kauditd_printk_skb: 14 callbacks suppressed
[   18.412839] audit: type=1400 audit(1663346171.912:33): apparmor="STATUS" operation="profile_replace" info="same as current profile, skipping" profile="unco
nfined" name="/snap/snapd/16778/usr/lib/snapd/snap-confine" pid=978 comm="apparmor_parser"
[   18.412843] audit: type=1400 audit(1663346171.912:34): apparmor="STATUS" operation="profile_replace" info="same as current profile, skipping" profile="unco
nfined" name="/snap/snapd/16778/usr/lib/snapd/snap-confine//mount-namespace-capture-helper" pid=978 comm="apparmor_parser"
[   18.415828] audit: type=1400 audit(1663346171.916:35): apparmor="STATUS" operation="profile_replace" info="same as current profile, skipping" profile="unco
nfined" name="snap-update-ns.lxd" pid=980 comm="apparmor_parser"
[   18.418481] audit: type=1400 audit(1663346171.920:36): apparmor="STATUS" operation="profile_replace" info="same as current profile, skipping" profile="unco
nfined" name="snap.lxd.activate" pid=981 comm="apparmor_parser"
[   18.418703] audit: type=1400 audit(1663346171.920:37): apparmor="STATUS" operation="profile_replace" info="same as current profile, skipping" profile="unco
nfined" name="snap.lxd.benchmark" pid=982 comm="apparmor_parser"
[   18.421278] audit: type=1400 audit(1663346171.920:38): apparmor="STATUS" operation="profile_replace" info="same as current profile, skipping" profile="unco
nfined" name="snap.lxd.buginfo" pid=983 comm="apparmor_parser"
[   18.421356] audit: type=1400 audit(1663346171.920:39): apparmor="STATUS" operation="profile_replace" info="same as current profile, skipping" profile="unco
nfined" name="snap.lxd.check-kernel" pid=984 comm="apparmor_parser"
[   18.424922] audit: type=1400 audit(1663346171.924:40): apparmor="STATUS" operation="profile_replace" info="same as current profile, skipping" profile="unco
nfined" name="snap.lxd.hook.configure" pid=986 comm="apparmor_parser"
[   18.425162] audit: type=1400 audit(1663346171.924:41): apparmor="STATUS" operation="profile_replace" info="same as current profile, skipping" profile="unco
nfined" name="snap.lxd.daemon" pid=985 comm="apparmor_parser"
[   18.427457] audit: type=1400 audit(1663346171.928:42): apparmor="STATUS" operation="profile_replace" info="same as current profile, skipping" profile="unco
nfined" name="snap.lxd.hook.install" pid=987 comm="apparmor_parser"
[   49.104437] hv_balloon: Max. dynamic memory size: 8192 MB
[   50.213304] Hello world
[   50.213388] Process ID: 1276 ; Parent process ID: 1211
[   50.213415] SYSTEM CALL srikantprint: "Srikant Konduri"
azureuser@SkVM:~$
```

Answer to Question 3:

- Here current process ID and parent process ID are different.
- Parent process ID is the process ID of the process which is invoking.
- Here Parent process is 1211 which is the ID of bash which is invoking a.out (test.c)
- Current process ID is 1276 which is the process ID of a.out