Operating Systems
Assignment 1 – *Hard*

Total: 40 Marks

Instructions:

1. The assignment has to be done individually.
2. You can use Piazza for any queries related to the assignment and avoid asking queries on the last day.

The assignment has 4 parts.

1 Install Linux: 3 Marks

In this section, we discuss the procedure of installation of the Linux kernel on a virtual machine (VM).

1.1 Virtual Machine Setup

In this subsection, we discuss the procedure to spawn a virtual machine (VM) on the host machine. Before spawning a VM, you must use the following instructions to determine whether your system supports KVM virtualization. We use the Ubuntu 20.04.3 version on the system.

```
sudo apt-get install cpu-checker
sudo kvm-ok  # Check if your system supports KVM virtualization
```

After confirming that your system supports KVM virtualization, you must install the required tools to spawn a VM using the *libvirt* library. Additionally, you must download the Ubuntu server ISO from this page. The following instructions must be executed to install *libvirt*:

```
sudo apt install -y qemu qemu-kvm libvirt-daemon \
libvirt-clients bridge-utils virt-manager
sudo usermod -G libvirt -a $USER # Add the user to the libvirt \
                                   group
sudo systemctl status libvirtd  # Check if libvirtd-service is \
                                  running
```

Execute the command below to generate a graphical user interface for creating a new virtual machine on your system.

```
virt-manager # Command to start the virt-manager
```

To create an VM, you should follow the instructions below. :

1. Navigate to the *File* tab and choose *New Virtual Machine*.
2. Select the local install media option and navigate to the ISO file on your local machine using the Browse option.
3. Minimum requirements are 4 CPU cores, 4 GB of RAM, and a 50 GB disk image. Select the appropriate resource choices for your computer.
4. Install the Ubuntu server ISO on the virtual machine, you can refer to the following guide.
5. While configuring the storage for Ubuntu, make sure you unmount the /boot mount partition and adjust the size of the partitions accordingly.

1.2 Build the Linux Kernel

In this section, we discuss the build process, and installation of the Linux kernel in the VM. First, you need to install the necessary tools for installing the Linux kernel using the following command:

```
sudo apt-get install git fakeroot build-essential ncurses-dev \
xz-utils libssl-dev bc flex libelf-dev bison
```

You need to download the source code of the latest stable Linux kernel using the following commands.

```
git clone \
git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git
cd linux
git checkout v6.1.6
```

To compile and run Linux on the virtual machine you can run the commands:

```
cp -v /boot/config-$(uname -r) .config
# Open .config file and set the CONFIG_SYSTEM_TRUSTED_KEYS\
option to ""
make -j <num_cores>
sudo make modules_install -j <num_cores>
sudo make install  -j <num_cores>
# Open the grub configuartion file at /etc/default/grub \
set the GRUB_TIMEOUT to 60 and comment GRUB_TIMEOUT_STYLE option.
sudo update-grub
```

Restart the virtual machine and select the latest kernel from the grub menu.

2 Context Switch Tracker: 15 Marks

In the Linux kernel, the *task_struct* structure stores the state of a single process (refer to the *include/linux/sched.h* file). We want to find the total number of voluntary and involuntary context switch events suffered by a subset of processes and their threads currently running on the system (referred to as *monitored* processes).

2.1 Data Structure

In this subsection, we discuss the data structure that you need to create in the Linux kernel to keep track of the monitored processes. Let us create the *pid_node* structure that stores the list of monitored processes in a doubly linked list. To perform any operation on the *pid_node* structure use the **list mechanism** provided by the Linux kernel (see *include/linux/list.h* file). Create the *pid_ctxt_switch* structure, to store the cumulative number of context switch events sufferred by the monitored processes.

```
struct pid_node {
    pid_t pid;          /* process id */
    struct list_head   next_prev_list; /* contains pointers
        to previous and next elements*/
};

struct pid_ctxt_switch {
    unsigned long   ninvctxt /* Count of involuntary context
         switches */
    unsigned long   nvctxt /* Count of voluntary context
        switches */
};
```

2.2 Working

This subsection describes how the context switch tracker operates.

1. Create the *sys_register* system call, which is used by users to add a process to the list of monitored processes. The system call adds the process's pid to the end of the linked list. The signature of the system call is as follows:

   ```
   int sys_register(pid_t pid)
   ```

   - *pid:* pid of the process.

   Note: The pid_t data type is defined in /usr/include/sys/types.h
   If the system call was successful in adding the process's pid to the linked list, it should return 0. The system call should detect and respond to the following conditions:

   - If the process's pid does not exist, return -3.
   - If the process's pid is less than 1, return -22.

2. Create the *sys_fetch* system call, which allows users to fetch the cumulative count of the context switch events (*#ncse*) for the monitored processes. The system call iterates through the linked list and saves the *#ncse* values. Subsequently, it sends the result in the *pid_ctxt_switch* structure. The signature of the system call is as follows:

   ```
   int sys_fetch(struct pid_ctxt_switch *stats)
   ```

   - *stats:* #ncse.

   If the system call was successful, it should return 0; otherwise, it should return -22.

3. Create the *sys_deregister* system call, which is used by users to remove a process from the list of monitored processes. The system call iterates through the linked list and removes the process from the list. The signature of the system call is as follows:

   ```
   int sys_deregister(pid_t pid)
   ```

   - *pid:* pid of the process.

   If the system call was successful in removing the process's pid from the linked list, it should return 0. The system call should detect and respond to the following conditions:

   - If the process's pid does not exist in the linked list, return -3.
   - If the process's pid is less than 1, return -22.

3 On Demand Signal Generator using a Kernel Module: 12 Marks

This part aims to create a Linux kernel module that generates a signal for the target process. Any process that wants to fire a signal to another process (target) uses the *proc* filesystem to insert the **PID** of the target process and the signal number in the */proc/sig_target* file. Refer link for more details on the Linux kernel module.

The created kernel module must check this file at a regular interval of 1 second and fire the appropriate signal to the target process. The working of the signal generator is as follows:

1. When the kernel module initializes, it creates a */proc/sig_target* file.
2. Any process that wants to generate a signal must create an entry in the */proc/sig_target* file in the format specified below:

   ```
   PID1, SIGNAL
   PID2, SIGNAL
   ```

3. The kernel module regularly reads the file (with a 1 second delay) and sends an appropriate signal to the target process.

4 Report: 10 Marks

Page limit: 10
The report should clearly mention the implementation methodology for all the parts of the assignment. Small code snippets are alright, additionally, the pseudocode should also suffice.

- Explain the implementation of the context switch tracker.
- How exactly does your Linux module work?
- Any other details that are relevant to the implementation.
- Submit a pdf containing the relevant details.
- Say what you have done that is extra.