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 installing the Linux kernel on a virtual machine (VM). Note that this is one of the methods to install the Linux kernel. Other methods are mentioned on the website of the book.

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. The instructions are relevant for Ubuntu 22.04.1.

```
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 a VM, you should follow the instructions shown 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 Resource Usage Tracker: 15 Marks

In the Linux kernel, the `task_struct` structure maintains information about the state of an individual process (defined in include/linux/sched.h). Our goal is to determine the heap memory usage and the number of open files used by a specific subset of processes and their threads currently running in the system. These processes are 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 resources utilized by monitored processes. Let us create the `pid_node` structure that stores the list of monitored processes and their respective resource utilization 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).

```
struct pid_node {
    struct per_proc_resource* proc_resource   /* Resource
       utilization of a process*/
    struct list_head   next_prev_list; /* contains pointers
       to previous and next elements*/
};

struct per_proc_resource {
    pid_t pid;          /* process id */
    unsigned long   heapsize   /* Total memory allocated by
       a process through the brk system call, and the mmap
       system call when used with the MAP_ANONYMOUS |
       MAP_PRIVATE flags. */
    unsigned long   openfile_count /* Total number of open
       files of a process using the open, openat, and
       openat2 system calls*/
};
```

2.2 Working

This subsection describes how the resource usage 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 creates a `per_proc_resource` structure and initializes its members with appropriate values. Subsequently, it adds the structure 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 inserting an entry 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.
- If the process's pid is already in the monitored list, return -23.

2. Create the *sys_fetch* system call, which allows users to fetch the resource usage statistics of a given process identified using its process id (pid). The system call iterates through the linked list and finds the resource usage of the given process. Subsequently, it sends the result back using the *per_proc_resource* structure. The signature of the system call is as follows:

   ```
   int sys_fetch(struct per_proc_resource *stats, pid_t pid)
   ```

   - *stats:* the heap usage (in MB) and the number of open files.
   - *pid:* pid of the process.

   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 Resource Usage Limiter: 12 Marks

This section addresses imposing quotas on the resource consumption of monitored processes. Users can define quotas, such as the maximum heap size a process may allocate and the maximum number of files it can open. The kernel utilizes the resource usage tracker to continuously monitor processes, ensuring compliance with these quotas. If a process exceeds its defined limits, the kernel promptly issues a SIGKILL signal to terminate the offending process.

3.1 Working

This subsection describes how the resource usage tracker operates.

1. Introduce the *sys_resource_cap* system call, enabling users to specify quotas for heap memory usage (in MB) and the maximum number of open files for a process. This system call sets these quotas as attributes within the `task_struct` structure of the specified process. The system call's signature is as follows:

```
int sys_resource_cap(pid_t pid, long heap_quota, long file_quota)
```

- *pid:* pid of the process.
- *heap_quota:* maximum heap size assigned to the process in MB.
- *file_quota:* maximum count of open files assigned to the process.

Note: The `pid_t` data type is defined in `/usr/include/sys/types.h`. If the values of heap_quota and file_quota are set to -1, then we do not have any limits.

If the system call was successful in setting the attributes of the process, 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 system, return -3.
- If the process's pid does not exist in the monitored processes list, return -22.
- If the quotas are already defined for the process, return -23.

2. Create the *sys_resource_reset* system call, which is used by users to reset the quotas on the resource consumption of monitored processes. The system call iterates through the processes in the monitored process list and resets the quota attributes of the task to -1. The signature of the system call is as follows:

   ```
   int sys_resource_reset(pid_t pid)
   ```

   - *pid:* pid of the process.

   If the system call was successful in resetting the quotas of a process, 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 system, return -3.
   - If the process's pid does not exist in the monitored list, return -22.

4 Report: 10 Marks

Page limit: 10

The report should clearly mention the implementation methodology for all the parts of the assignment. Showing small, representative code snippets in the report is alright, additionally, the pseudocode should also suffice.

- Explain the implementation of the resource usage tracker.
- Explain the implementation of the resource usage limiter.
- Any other details that are relevant to the implementation.
- Submit a pdf file containing the relevant details.
- Say what you have done that is extra.

5 Submission Instructions

1. There will be a demo for assignment 1 in which you must demonstrate how the resource usage tracker and the resource usage limiter work. This will be followed by a viva in which your general theoretical and practical understanding will be tested (in the context of the assignment). Note that regardless of the code that you submit, the viva performance is vitally important.

2. Create a patch file with the resource usage tracker and the resource usage limiter.

```
sudo diff -rupN  linux-change/ linux-base/ > res_usage.patch
# linux-base refers to the vanilla version of the Linux
kernel, and linux-change refers to the modified Linux kernel.
```

3. Create a zip file that contains the report, and res_usage.patch files and then name the zip file as, *assignment1_hard_⟨entryNumber⟩.zip*. Submit this zip file on Moodle. Entry number format: 2020CSZ2445. *Note that all English letters are in capitals.*