[COL331] Assignment 1: Tracker & Limiter

Hemanshu Garg, 2022CS11090

(NOTE: Implemented the latest interpretation of handling cumulative resource tracking and limiting, discussed the design decisions in mechanism sections)

1. Install Linux

Followed the assignment PDF to first install virt-manager, followed by setting up the VM. In setting up the VM, unchecked LVM(to avoid manual partioning) and checked ssh. For cloning the kernel code, used different command as saw on piazza so as to only install the required version code.

```
$ git clone --depth 1 --branch v6.1.6 \
git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git
```

After copying the .config file in ∼/linux folder made two changes in it:-

```
CONFIG_SYSTEM_TRUSTED_KEYS = ""
CONFIG_SYSTEM_REVOCATION_KEYS =""
```

Then compiled the kernel with all default options and switched to this kernel. Verfied correct kernel version and get ssh address with commands:-

```
    $ uname -r
    $ ip -a
```

Then I can just make changes to ∼/linux folder and recompile with same methods that was used to build last kernel and then grub into + version to see the effects. :)

Theory: Implementations of system calls in x86 linux

- To implement system calls the only basic requirement is to write the functionality in ∼/linux/kernel/myfile.c (using appropriate SYSTEM_DEFINE macro) and then add a system call number for it in system_calls_64.tbl.
- In kernel/Makfile, adding obj-y+=myfile.o will make sure our code is compiled (for making it loadable module can explore obj-m)
- If want to give a macro instead of syscall number for our syscall, then in unistd.h can add the macro for our new syscalls.
- For consistency or legacy reasons, can add signature of new functions in syscalls.h.

2. Resource Usage Tracker

2.1 Objective

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.

2.2 Mechanism and High Level Idea

We realize the objective using following design:
We expose three system calls to the user space, namely *sys_register*, *sys_fetch*, and *sys_deregister*. Now a "process" (in isolation) will be tracked as follows :-

- The process registers itself for tracking using *sys_register*. We monitor only the **changes** to its asked parameters **post registration** only.
- The process asks these statistics using *sys_fetch*.
- The process deregisters itself for tracking using *sys_deregister*.

Now, for a group of process (specifically those with same Thread group id(tgid)). The following additional things happen:-

- If the Thread group leader is registered, it will track **changes of all** the threads with same thread group id.
- Indivisual non-leader threads continue to monitor their own changes.
- If user wants to access statistics of the whole group, user should fetch on thread group id (tgid).
- If user wants to access statistics due to only a specific non-leader thread, they can fetch on that pid.

Now here is another design choice - If user is concerned with statistics of whole thread group its their responsibility to register the leader thread first.
- Justification for the above mechanisms is two :-

- There is value in being able to track and control the statistics of indivisual threads as well as of the whole group.
- User space should be responsible in using syscalls, something like fork also can be used in various ways, hence giving user this freedom seems apt.

2.3 Implementation of the syscalls and other changes

linux/arch/x86/entry/syscalls/syscall_32.tbl

```
451 i386  register  sys_register
452 i386  fetch  sys_fetch
453 i386  deregister  sys_deregister
454 i386  resource_cap  sys_resource_cap
455 i386  resource_reset  sys_resource_reset
```

linux/arch/x86/entry/syscalls/syscall_64.tbl

```
451 common  register  sys_register
452 common  fetch  sys_fetch
453 common  deregister  sys_deregister
454 common  resource_cap  sys_resource_cap
455 common  resource_reset  sys_resource_reset
```

linux/kernel/Makfile

```
    obj-y += tracker_limiter.o
```

Defined main implementation (of both tracker and limiter)

linux/kernel/tracker_limiter.c

```
#include <linux/tracker_limiter.h>
static LIST_HEAD(sentinal_head);
static DEFINE_MUTEX(sentinal_lock);
static ll_size = 0 ;
SYSCALL_DEFINE1(register, pid_t, pid){...}
SYSCALL_DEFINE2(fetch,struct per_proc_resource*, stats, pid_t, pid){...}
SYSCALL_DEFINE1(deregister, pid_t, pid){...}
SYSCALL_DEFINE3(resource_cap, pid_t, pid, long, heap_quota, long, file_quota){...}
SYSCALL_DEFINE1(resource_reset, pid_t, pid){...}
```

Helper functions and include files in :-

linux/include/linux/tracker_limiter.h

```
struct per_proc_resource {...}
struct pid_node {...}
int __deregister(pid_t pid);
int check_if_monitored(struct pid_node** node,pid_t pid);
int monitor_brk(unsigned long oldbrk,unsigned long newbrk);
int monitor_mmap(unsigned long len);
int monitor_files(int open);
int quota_check(struct pid_node *iter, struct per_proc_resource *proc_resource, pid_t pid);
int kill_and_dereg(struct pid_node *iter,struct task_struct *cur_task);
```

Find Pseudo code below:-

sys_register

```
int sys_register(pid_t pid){
    DO_SANITY_CHECKS{
    pid < 1: return -22 ;
    pid does not exists: return -3 ;
    pid already exists: return -23;
    }
    // do other initializations like heapquota and filequota <= -1
    new_per_proc_resource = kmalloc(sizeof(*new_per_proc_resource),GFP_KERNEL);
    new_pid_node = kmalloc(sizeof(*new_pid_node),GFP_KERNEL);
    new_pid_node->proc_resource = new_per_proc_resource;
    list_add_tail(&new_pid_node->next_prev_list,&sentinal_head);
    return 0;
}
```

sys_fetch

```
int sys_fetch(struct per_proc_resource* stats,pid_t pid){
    DO_SANITY_CHECKS{
    pid < 1: return -22 ;
    pid does not exist in LL: return -22;
    }
    if(copy_to_user(stats,iter->proc_resource,sizeof(struct per_proc_resource))==0)
```

```
        {
            return 0 ;
        }

        return -22 ;
    }
```

sys_deregister

```
    int sys_register(pid_t pid){
        DO_SANITY_CHECKS{
        pid < 1: return -22 ;
        pid does not exist in LL: return -3 ;
        if ll_size ==0 : return -3;
        }
        list_del(&iter->next_prev_list);
        kfree(iter->proc_resource);
        kfree(iter);
        ll_size-= 1;
     return 0;
    }
```

Another set of important functions are monitor_brk, monitor_mmap, monitor_files. These are actually responsible for updating the stats. These functions are called in mmap, brk, do_sys_openat2, __x64_close functions of the kernel (syscalls and helper functions of the statistics to monitor). Here is psuedo code of monitor functions:-

monitor

```
    int monitor_CONCERNEDCALLER(args = oldbrk,newbrk|len|open)
    {
        pid = current->pid ;
        tgid = current->tgid ;
        if (tgid in LL )
        {
            if(update_stats(tgid,args)==0 && tgid == pid)
            return 0 ;
        }
        if(pid in LL)
        {
            if(update_stats(tgid,args)==0) ;
            return 0;
        }
        return -22;
    }
```

So here is what happens in summary step by step :-

- everytime the concerned system calls of do_exit (discussed later), mmap,brk and other file opening closing are called, we call the monitor functions update the global LL.
- If no process is registered, monitor functions return with negative values and nothing happens.
- If some process are registered, then each monitor executes and updates concerned nodes (tgid and pid).

- At any point fetch can be done or processes be deregistered.

3. Resource Usage Limiter

3.1 Objective

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.

3.2 Mechanism and High Level Idea

We realize our objective using mechanism defined earlier with following additional details :-

- Whenever quota of any process exceeds it is sent SIGKILL.
- In a group of processes with same tgid, whenever any one of the processes exceed their quota that process is sent SIGKILL and and by linux standard all process of that tgid also are killed.
- for a process that is killed it is automatically deregisterd, this is achieved by calling our deregister function in do_exit function which every process must call for terminating.

3.3 Additional syscalls and other changes

In task_struct in linux/include/linux/sched.h

```
    long heapquota ;
    long filequota ;
```

In do_exit in linux/kernel/exit.c

```
    __deregister(current->pid);
```

Find Pseudo code below :-

sys_resource_cap

```
int sys_resource_cap(pid_t pid, long heap_quota, long file_quota){
    DO_SANITY_CHECKS{
    pid does not exists: return -3 ;
    pid not in monitored list: return -22;
    pid quotas already set: return -23;
    }
    cur_task->heap_quota = heap_quota;
    cur_task->file_quota = file_quota;
    check_quota_exceed(pid);
    return 0;
}
```

sys_resource_reset

```
int sys_resource_reset(pid_t pid){
    DO_SANITY_CHECKS{
    pid does not exists: return -3 ;
    pid not in monitored list: return -22;
    pid quotas already set: return -23;
    }
    cur_task->heap_quota = -1 ;
    cur_task->file_quota = -1 ;
    return 0;
}
```

4. Salient Features and Extra things considered

- Handled automatic de-registration of process from linked list even if its killed at any point.
- Zombie pids will not conflict with Linked list.
- Handled fact that task_struct == NULL is not sufficient and process should also not be a zombie before registering.
- Used mutexes for locking mechanism to make our system calls thread safe. Used mutexes instead of spinlocks because by default mmap and brk use sleeping locks, therefore did not make sense to use busy wait locks for sub function call in them.
- The File tracking is done accuratelly where if files is closed that change is recoreded as well.

5. Submission and Testing

Got patch file and can apply it by commands :-

```
#Stage your changes first
$ git diff --staged --output res_usage.patch

$ git apply res_usage.patch
```

Tested various scenarios :-

- Correct edge case handling (i.e correct return errno for pids)
- Correct heap_usgae and file count.
- Correct killing of the process if quotas exceeds.
- Correct thread group leader and non-leader thread interactions.

6. What could have been further done

- Namespace handling, one idea to implement this would involve keeping track of both namespace and pid pair in our pid_node and use this pair to distinguish between processes.
- handling of the fact if memory is freed then our statistics should be updated as well. (implemented this at first but had to rollback as we are only tracking MAP_ANONYMOUS and MAP_PRIVATE allocations, so found difficult to track its frees).

References

[1] Kernel Data Structures: Linked List. https://medium.com/@414apache/kernel-data-structures-linkedlist-b13e4f8de4bf.

[2] Adding a System Call in Linux [Video]. https://www.youtube.com/watch?v=Kn6D7sH7Fts.

[3] Adding a System Call to the Linux Kernel 5.8.1 in Ubuntu 20.04 LTS. https://dev.to/jasper/adding-a-system-call-to-the-linux-kernel-5-8-1-in-ubuntu-20-04-lts-2ga8.

[4] Kernel Development Episode 3. https://brennan.io/2016/11/14/kernel-dev-ep3/#:~:text=Add%20Your%20System%20Call,if%20you%20compile%20right%20now..

[5] Adding System Calls in the Linux Kernel. https://docs.kernel.org/process/adding-syscalls.html.

[6] Adding a System Call to the Linux Kernel 5.8.1 in Ubuntu 20.04 LTS (Duplicate). https://dev.to/jasper/adding-a-system-call-to-the-linux-kernel-5-8-1-in-ubuntu-20-04-lts-2ga8.

[7] Errno Documentation. https://kdave.github.io/errno.h/.

[8] Elixir: Linux Kernel Source Code Browser. https://elixir.bootlin.com/linux/v6.1.107/source.

[9] Checking if a Process is Still Running in the Linux Kernel. https://stackoverflow.com/questions/17795739/linux-kernel-check-if-process-is-still-running.

[10] Kernel Hacking Guidelines. https://docs.kernel.org/kernel-hacking/hacking.html.