CEng 536 Advanced Unix
Fall 2024
HW2 - Linux System Call
Due: 8/1/2024

1 Description

In this project you are going to implement a system call for getting virtual memory area information of a user space pointer. System call `checkaddr(pointer, struct vma_info *v)` will fill the `v` structure if pointer is a valid pointer.

```
struct vma_info {
    long start;          // start of area
    long end;            // end of area
    struct {             // bitmap flags
        unsigned int readable:1;
        unsigned int writeable:1;
        unsigned int executable:1;
        unsigned int shared:1;
        unsigned int filemapped:1;
        unsigned int heap:1;
        unsigned int stack:1;
    } flags;
    struct {             // if area is mapped to a file
        unsigned long inode;    // inode number (ls -i)
        int major, minor;               // device info of fs
        char path[256];                 // path
        unsigned long long offset;      // mapping offset
    } file;
    struct {             // proces vm info
        unsigned long vmpages;  // total vm pages
        unsigned long vmdata;   // total data pages
        unsigned long vmexec;   // total exec pages
        unsigned long vmstack;  // total stack pages
        unsigned long vmresanon;        // resident anonymous pages
        unsigned long vmresfile;        // resident file pages
        int vmref;      // number of references to process memory
    } mm;
};
```

In the project, the new system call `checkaddr(pointer)` will do the following in the kernel:

- Searches the pointer in VM (`find_vma`). If not mapped, returns `-EINVAL`.
- Fill the values in `struct vma_info` structure above
- Use `vm_flags` to fill `flags` sub-structure
- If area is mapped to a file fill `file` sub-structure.
- Go back to `mm_struct` (`current->mm`) and fill the `mm` sub-structure

In proc file system `/proc/processid/maps` give most of this information for all areas. It uses `show_map_vma` function in `fs/proc/task_mmu` to fill report of each virtual memory area. This function contains hints of the code you need. For example stack test, heap test etc.
We will be using kernel source code v6.11:
https://github.com/torvalds/linux/tree/98f7e32f20d28ec452afb208f9cffc08448a2652
You can find deb package in debian backports for bookworm:

http://ftp.tr.debian.org/debian/pool/main/l/linux/linux-source-6.11_6.11.5-1~bpo12+1_all.deb

The number for our system call will be 463. You can create a macro for `check_addr()` in `checkaddr.h` as follows:

```
#define NR_CHECK_ADDR 463
#define check_addr(pointer, vmainfo) syscall(NR_CHECK_ADDR, (pointer), (vmainfo))
```

2 Implementation

You need the recompile the kernel in order to add system call. You need the following steps:

1. Get kernel source 6.11
2. Before compilation edit `arch/x86/entry/syscalls/syscall_64.tbl`. Add following line in the suitable position:
   `463 common check_addr sys_check_addr`
3. Under `mm` directory (chosen as a standard place, actually it can be anywhere with a proper Makefile), create `mm536.c` file with following empty declarations:

```
#include <linux/capability.h>
#include <linux/mm.h>
#include <linux/syscalls.h>
#include <linux/uaccess.h>
#include <linux/printk.h>
#include <linux/fs.h>
#include <linux/file.h>
#include <linux/errno.h>

struct vma_info {
        long start;
        long end;
        struct {
                unsigned int readable:1;
                unsigned int writeable:1;
                unsigned int executable:1;
                unsigned int shared:1;
                unsigned int filemapped:1;
                unsigned int heap:1;
                unsigned int stack:1;
        } flags;
        struct {
                unsigned long inode;
                int major, minor;
                char path[256];
                unsigned long long offset;
        } file;
        struct {
                unsigned long vmpages;
                unsigned long vmdata;
                unsigned long vmexec;
                unsigned long vmstack;
                unsigned long vmresanon;
                unsigned long vmresfile;
                int vmref;
```

```
        } mm;
};

SYSCALL_DEFINE2(check_addr, unsigned long , addr, void __user *, info) {
    printk("check_address: %p %p",addr, info);
    return 0;
}
```

4. Add following line in `mm/Makefile`
   `obj-y += mm536.o`
5. Copy the kernel config provided in the homework page as `.config` in the kernel source code.
6. run `make`
7. Now you can install this kernel and boot Linux system with Qemu or similar VM manager.

Using this template, develop your project. Note that you only need to submit `mm536.c` and an optional header file for kernel. You can use any kernel library and synchronization primitives you like.
You may need the following symbols:
`current`, `struct task_struct`, `struct mm_struct`, `struct file`, `struct inode`
For secure access to mm pointers and other useful kernel utilities:
`mmap_read_lock`, `mmap_read_unlock`, `d_path` (it uses the second parameter buffer bottom up and returns the pointer in the middle), `MAJOR/MINOR()`, `copy_to_user`, `atomic_read/atomic_long_read` (for `atomic_long_t` values),
If you need initializations at startup (probably you won't), use a prototype as:

```
int __init myinitialization(void)
core_initcall(myinitialization)
```

That will register your function `myinitialization` and automatically call when the kernel is loaded.

3 Submission and External libraries

You need to submit a C source code. C++ is not allowed in kernel. Submit `mm536.c` kernel code, and a header file if you like.
Submission details will be announced later. Please ask your questions in ODTUclass forum.