作業系統概論 hw10

學號: 408410113 姓名: 王 X 彥

1. 撰寫程式碼稱之為myls，在程式碼中使用execve系列的任何libc函數，載入新的執行檔案（ls）。

注意：不需要使用fork。

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

int main(int argc, char **argv){
    memset(argv[0], 0, strlen(argv[0]));
    strcpy(argv[0], "ls");
    execvp("ls", argv);
    return 0;
}
```

2. 請問作業系統如何載入執行檔案？附上程式碼的截圖，並約略說明。

execve執行流程: sys_execve() > do_execve() > do_execveat_common ()

> search_binary_handler() > load_elf_binary()

```c
int do_execve(struct filename *filename,
    const char __user *const __user *__argv,
    const char __user *const __user *__envp)
{
    struct user_arg_ptr argv = { .ptr.native = __argv };
    struct user_arg_ptr envp = { .ptr.native = __envp };
    return do_execveat_common(AT_FDCWD, filename, argv, envp, 0);
}
```

```c
static int do_execveat_common(int fd, struct filename *filename,
                              struct user_arg_ptr argv,
                              struct user_arg_ptr envp,
                              int flags)
{
    return __do_execve_file(fd, filename, argv, envp, flags, NULL);
}
```

```c
/*
 * sys_execve() executes a new program.
 */
static int __do_execve_file(int fd, struct filename *filename,
                            struct user_arg_ptr argv,
                            struct user_arg_ptr envp,
                            int flags, struct file *file)
{
    char *pathbuf = NULL;
    struct linux_binprm *bprm;
    struct files_struct *displaced;
    int retval;

    if (IS_ERR(filename))
        return PTR_ERR(filename);

    /*
     * We move the actual failure in case of RLIMIT_NPROC exces
     * set*uid() to execve() because too many poorly written pr
     * don't check setuid() return code.  Here we additionally
     * whether NPROC limit is still exceeded.
     */
    if ((current->flags & PF_NPROC_EXCEEDED) &&
        atomic_read(&current_user()->processes) > rlimit(RLIMIT
        retval = -EAGAIN;
        goto out_ret;
    }
```

A. do_execve()填入參數call do_execveat_common() (左上)

B. do_execveat_common()(上)

C. struct linux_binprm結構描述一個可執行程序檢查檔案名稱是否正確(左)

```
/* We're below the limit (still or again), so we don't want to make
 * further execve() calls fail. */
current->flags &= ~PF_NPROC_EXCEEDED;

retval = unshare_files(&displaced);
if (retval)
    goto out_ret;

retval = -ENOMEM;
bprm = kzalloc(sizeof(*bprm), GFP_KERNEL);
if (!bprm)
    goto out_files;

retval = prepare_bprm_creds(bprm);
if (retval)
    goto out_free;

check_unsafe_exec(bprm);
current->in_execve = 1;
```

D. 用 unshare_files()為行程復制一份檔案表

E. 用 kzalloc()分配一份 linux_binprm 結構

```
retval = prepare_bprm_creds(bprm);
if (retval)
    goto out_free;

check_unsafe_exec(bprm);
current->in_execve = 1;

if (!file)
    file = do_open_execat(fd, filename, flags);
retval = PTR_ERR(file);
if (IS_ERR(file))
    goto out_unmark;

sched_exec();
```

F. 用 open_exec()查找並打開二進制檔案

G. 用 sched_exec()找到最小負載的 CPU，用來執行該二進制檔案

```
bprm->file = file;
if (!filename) {
    bprm->filename = "none";
} else if (fd == AT_FDCWD || filename->name[0] == '/') {
    bprm->filename = filename->name;
} else {
    if (filename->name[0] == '\0')
        pathbuf = kasprintf(GFP_KERNEL, "/dev/fd/%d", fd);
    else
        pathbuf = kasprintf(GFP_KERNEL, "/dev/fd/%d/%s",
                  fd, filename->name);
    if (!pathbuf) {
        retval = -ENOMEM;
        goto out_unmark;
    }
    /*
     * Record that a name derived from an O_CLOEXEC fd will be
     * inaccessible after exec. Relies on having exclusive access to
     * current->files (due to unshare_files above).
     */
    if (close_on_exec(fd, rcu_dereference_raw(current->files->fdt)))
        bprm->interp_flags |= BINPRM_FLAGS_PATH_INACCESSIBLE;
    bprm->filename = pathbuf;
}
bprm->interp = bprm->filename;

retval = bprm_mm_init(bprm);
if (retval)
    goto out_unmark;

retval = prepare_arg_pages(bprm, argv, envp);
if (retval < 0)
    goto out;

retval = prepare_binprm(bprm);
if (retval < 0)
    goto out;
```

H. 填入 linux_binprm 結構中的 file、filename、interp 參數

I. bprm_mm_init()初始化行程的記憶體空間，為新程式初始化記憶體管理

J. 填入 linux_binprm 結構中的 argc、envc 參數

K. 用 prepare_binprm()檢查二進制檔案的執行權限，kernel_read()讀取二進制檔案的前 128 字元（這些字元用於識別二進制檔案的格式及其他訊息，後續會用到）

```
int prepare_binprm(struct linux_binprm *bprm)
{
    int retval;
    loff_t pos = 0;

    bprm_fill_uid(bprm);

    /* fill in binprm security blob */
    retval = security_bprm_set_creds(bprm);
    if (retval)
        return retval;
    bprm->called_set_creds = 1;

    memset(bprm->buf, 0, BINPRM_BUF_SIZE);
    return kernel_read(bprm->file, bprm->buf, BINPRM_BUF_SIZE, &pos);
}
```

```
retval = copy_strings_kernel(1, &bprm->filename, bprm);
if (retval < 0)
    goto out;

bprm->exec = bprm->p;
retval = copy_strings(bprm->envc, envp, bprm);
if (retval < 0)
    goto out;

retval = copy_strings(bprm->argc, argv, bprm);
if (retval < 0)
    goto out;

would_dump(bprm, bprm->file);

retval = exec_binprm(bprm);
if (retval < 0)
    goto out;
```

L. 用 copy_strings_kernel()從 kernel spcae 獲取二進制檔案的路徑

M. 用 copy_string()從 user space 複製環境變量和參數

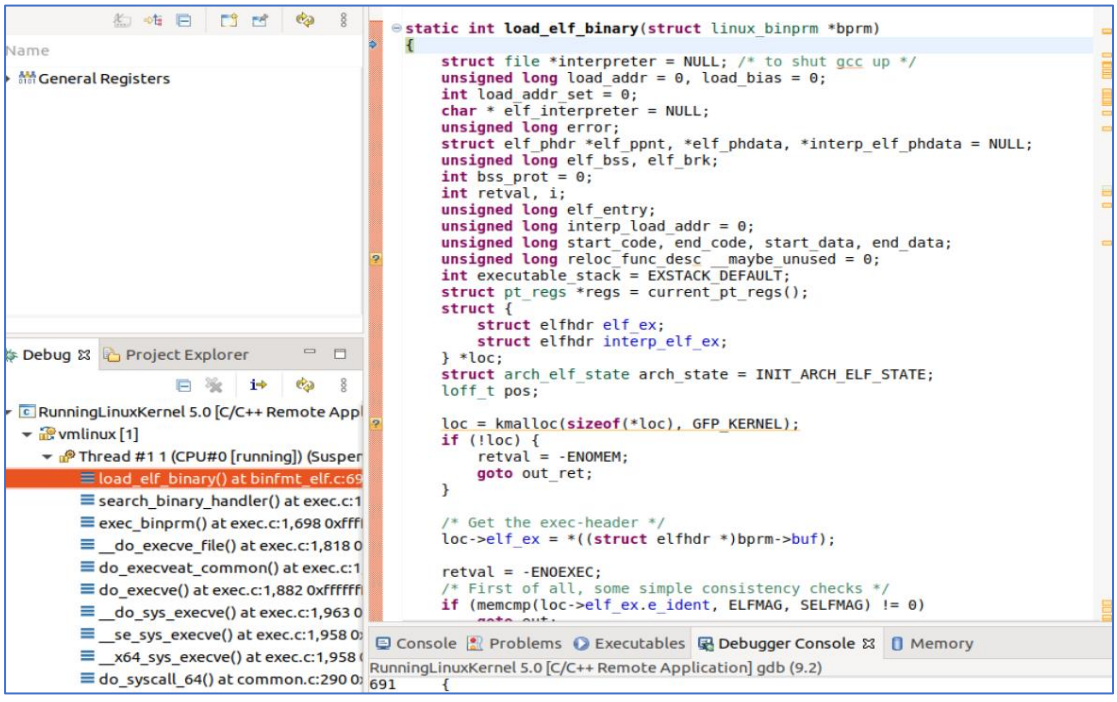
```
static int exec_binprm(struct linux_binprm *bprm)
{
    pid_t old_pid, old_vpid;
    int ret;

    /* Need to fetch pid before load_binary changes it */
    old_pid = current->pid;
    rcu_read_lock();
    old_vpid = task_pid_nr_ns(current, task_active_pid_ns(current->parent));
    rcu_read_unlock();

    ret = search_binary_handler(bprm);
    if (ret >= 0) {
        audit_bprm(bprm);
        trace_sched_process_exec(current, old_pid, bprm);
        ptrace_event(PTRACE_EVENT_EXEC, old_vpid);
        proc_exec_connector(current);
    }

    return ret;
}
```

N. 到這邊, 二進制檔案已經被打開, linux_binprm 結構中也記錄了重要訊息, kernel 使用 exec_binprm 執行可執行程式

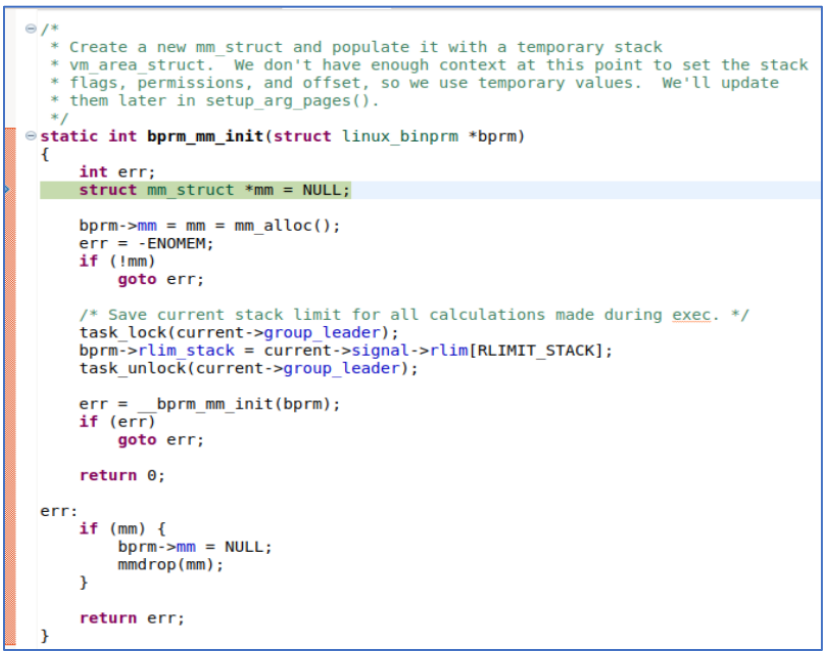
```
/*
 * cycle the list of binary formats handler, until one recognizes the image
 */
int search_binary_handler(struct linux_binprm *bprm)
{
    bool need_retry = IS_ENABLED(CONFIG_MODULES);
    struct linux_binfmt *fmt;
    int retval;

    /* This allows 4 levels of binfmt rewrites before failing hard. */
    if (bprm->recursion_depth > 5)
        return -ELOOP;

    retval = security_bprm_check(bprm);
    if (retval)
        return retval;

    retval = -ENOENT;
 retry:
    read_lock(&binfmt_lock);
    list_for_each_entry(fmt, &formats, lh) {
        if (!try_module_get(fmt->module))
            continue;
        read_unlock(&binfmt_lock);
        bprm->recursion_depth++;
        retval = fmt->load_binary(bprm);
        read_lock(&binfmt_lock);
        put_binfmt(fmt);
        bprm->recursion_depth--;
        if (retval < 0 && !bprm->mm) {
            /* we got to flush_old_exec() and failed after it */
            read_unlock(&binfmt_lock);
            force_sigsegv(SIGSEGV, current);
            return retval;
        }
```

O. 用 search_binary_handler()函數對 linux_binprm 的 formats list 進行掃描, 並嘗試每個 load_binary 函數, 直到成功加載了文件的執行格式

P. 最後分別對不同format使用不同的載入函數



3. 請問作業系統是否立即載入執行檔案到記憶體中？附上程式碼的截圖並約略說明



```c
/*
 * Create a new mm_struct and populate it with a temporary stack
 * vm_area_struct.  We don't have enough context at this point to set the stack
 * flags, permissions, and offset, so we use temporary values.  We'll update
 * them later in setup_arg_pages().
 */
static int bprm_mm_init(struct linux_binprm *bprm)
{
    int err;
    struct mm_struct *mm = NULL;

    bprm->mm = mm = mm_alloc();
    err = -ENOMEM;
    if (!mm)
        goto err;

    /* Save current stack limit for all calculations made during exec. */
    task_lock(current->group_leader);
    bprm->rlim_stack = current->signal->rlim[RLIMIT_STACK];
    task_unlock(current->group_leader);

    err = __bprm_mm_init(bprm);
    if (err)
        goto err;

    return 0;

err:
    if (mm) {
        bprm->mm = NULL;
        mmdrop(mm);
    }

    return err;
}
```

否，如左圖所示，在初始化時，

OS只會修改task_struct中的mm_struct