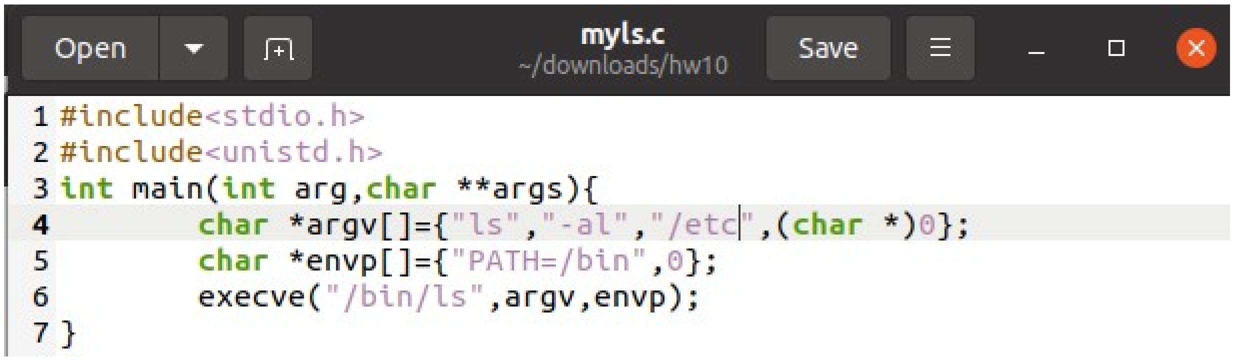
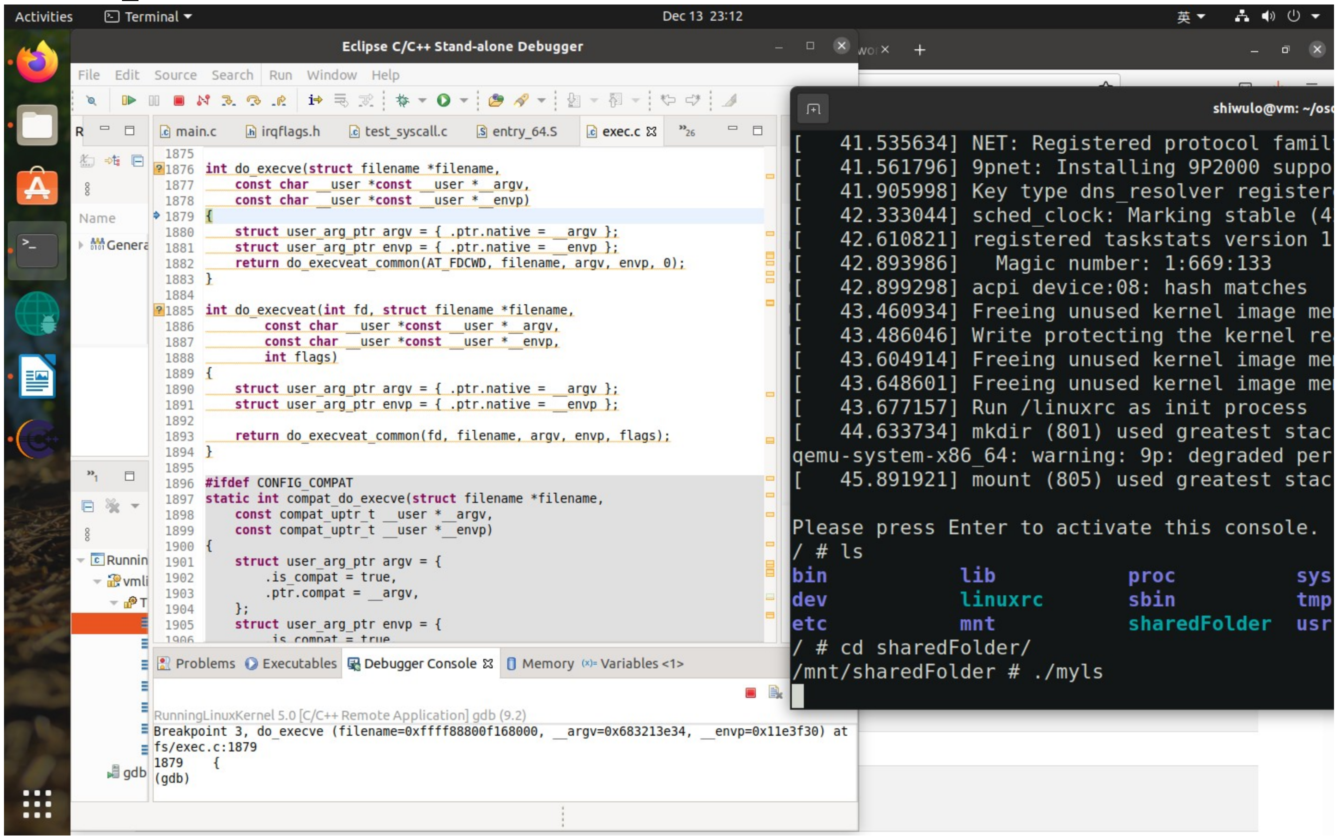
408410007 鄭 O 云

1.



```c
#include<stdio.h>
#include<unistd.h>
int main(int arg,char **args){
        char *argv[]={"ls","-al","/etc",(char *)0};
        char *envp[]={"PATH=/bin",0};
        execve("/bin/ls",argv,envp);
}
```

2.進來 do_execve

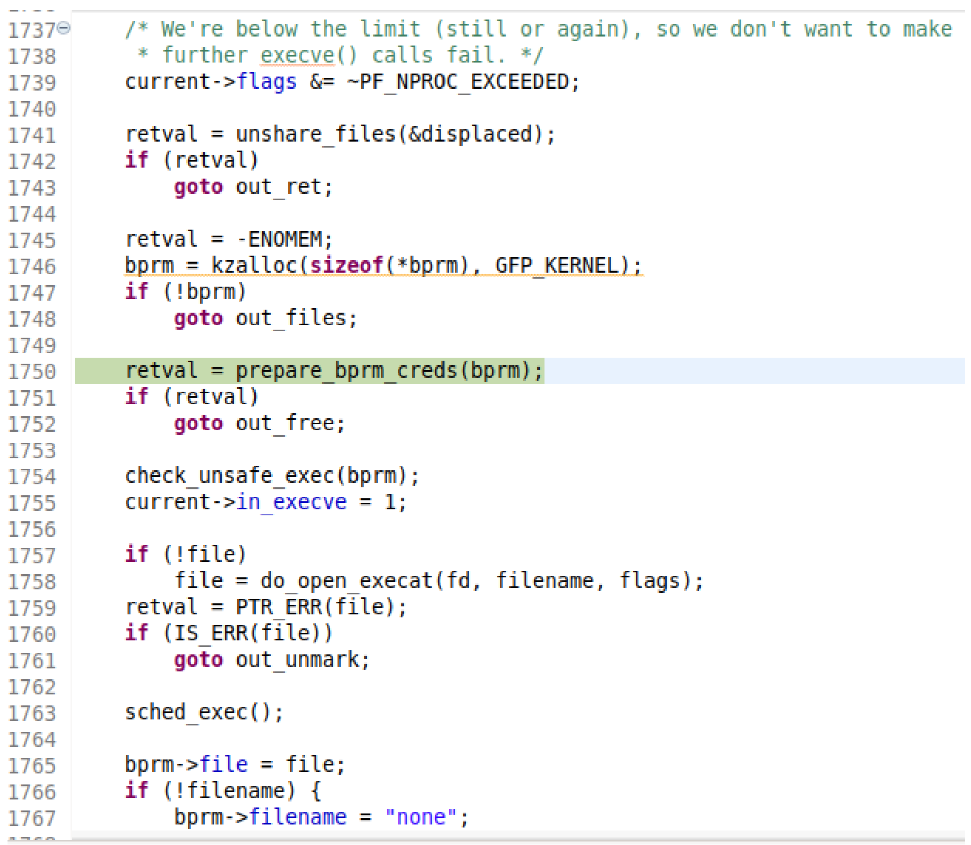


```c
int do_execve(struct filename *filename,
	const char __user *const __user *__argv,
	const char __user *const __user *__envp)
{
	struct user_arg_ptr argv = { .ptr.native = __argv };
	struct user_arg_ptr envp = { .ptr.native = __envp };
	return do_execveat_common(AT_FDCWD, filename, argv, envp, 0);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

int do_execveat(int fd, struct filename *filename,
		const char __user *const __user *__argv,
		const char __user *const __user *__envp,
		int flags)
{
	struct user_arg_ptr argv = { .ptr.native = __argv };
	struct user_arg_ptr envp = { .ptr.native = __envp };

	return do_execveat_common(fd, filename, argv, envp, flags);
}

#ifdef CONFIG_COMPAT
static int compat_do_execve(struct filename *filename,
	const compat_uptr_t __user *__argv,
	const compat_uptr_t __user *__envp)
{
	struct user_arg_ptr argv = {
		.is_compat = true,
		.ptr.compat = __argv,
	};
	struct user_arg_ptr envp = {
		.is_compat = true
```

```
RunningLinuxKernel 5.0 [C/C++ Remote Application] gdb (9.2)
Breakpoint 3, do_execve (filename=0xffff88800f168000, __argv=0x683213e34, __envp=0x11e3f30) at
fs/exec.c:1879
1879	{
(gdb)
```

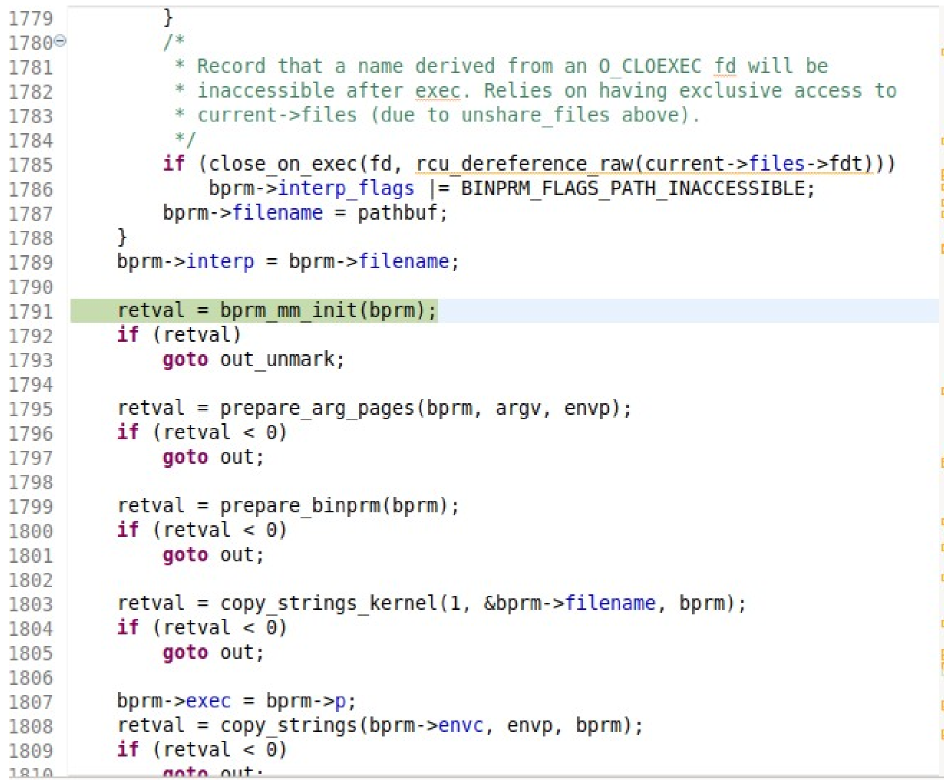
```
[   41.535634] NET: Registered protocol famil
[   41.561796] 9pnet: Installing 9P2000 suppo
[   41.905998] Key type dns_resolver register
[   42.333044] sched_clock: Marking stable (4
[   42.610821] registered taskstats version 1
[   42.893986]   Magic number: 1:669:133
[   42.899298] acpi device:08: hash matches
[   43.460934] Freeing unused kernel image me
[   43.486046] Write protecting the kernel re
[   43.604914] Freeing unused kernel image me
[   43.648601] Freeing unused kernel image me
[   43.677157] Run /linuxrc as init process
[   44.633734] mkdir (801) used greatest stac
qemu-system-x86_64: warning: 9p: degraded per
[   45.891921] mount (805) used greatest stac

Please press Enter to activate this console.
/ # ls
bin          lib          proc         sys
dev          linuxrc      sbin         tmp
etc          mnt          sharedFolder usr
/ # cd sharedFolder/
/mnt/sharedFolder # ./myls
```

追蹤: do_execve -> do_execveat_common -> __do_execve_file



```c
	/* 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;

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

	sched_exec();

	bprm->file = file;
	if (!filename) {
		bprm->filename = "none";
```

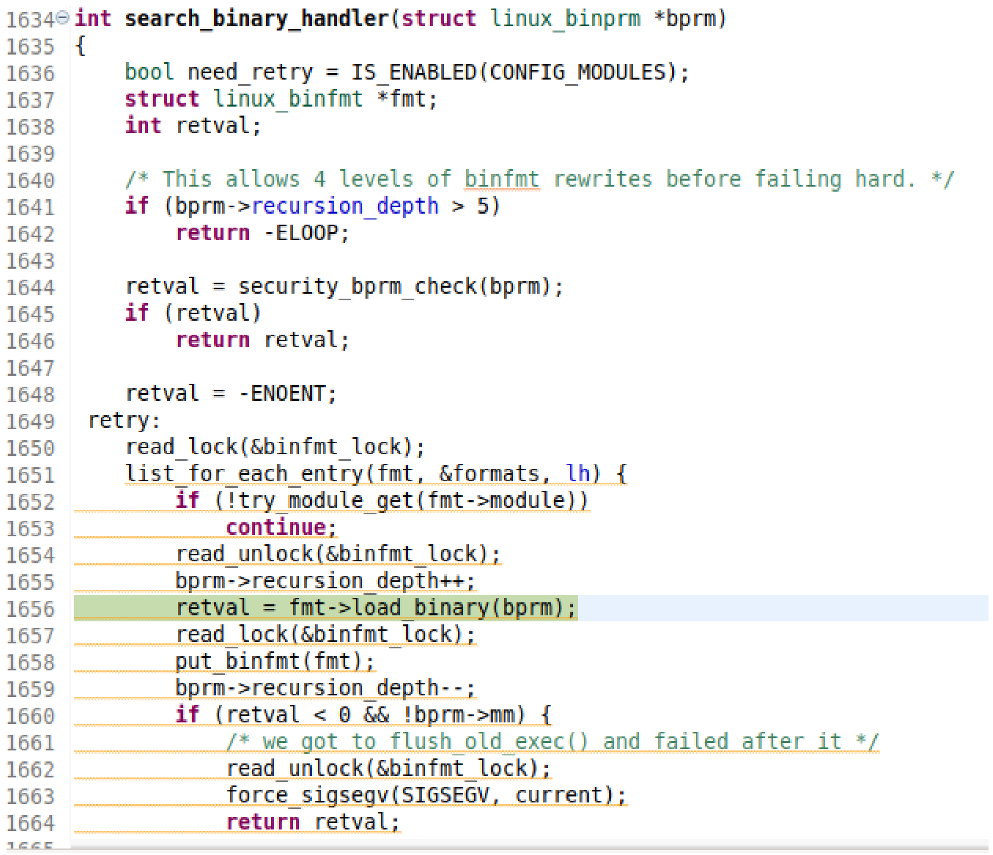
進入 prepare_bprm_creds(bprm);
//新建一個 cred 结構



```c
		}
		/*
		 * 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;

	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;
```

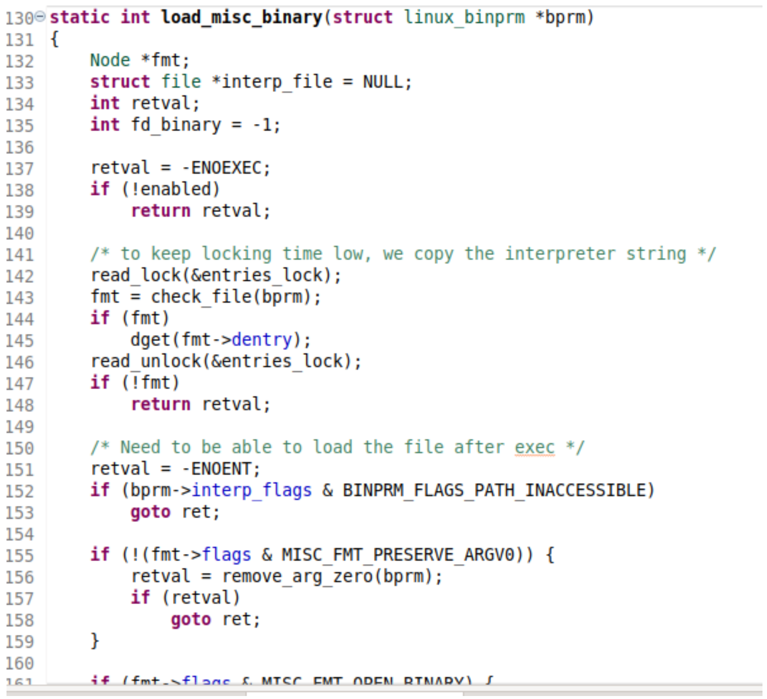
建立內存的地址空間，用 mm_struct

search_binary_handler 識別二進程程序



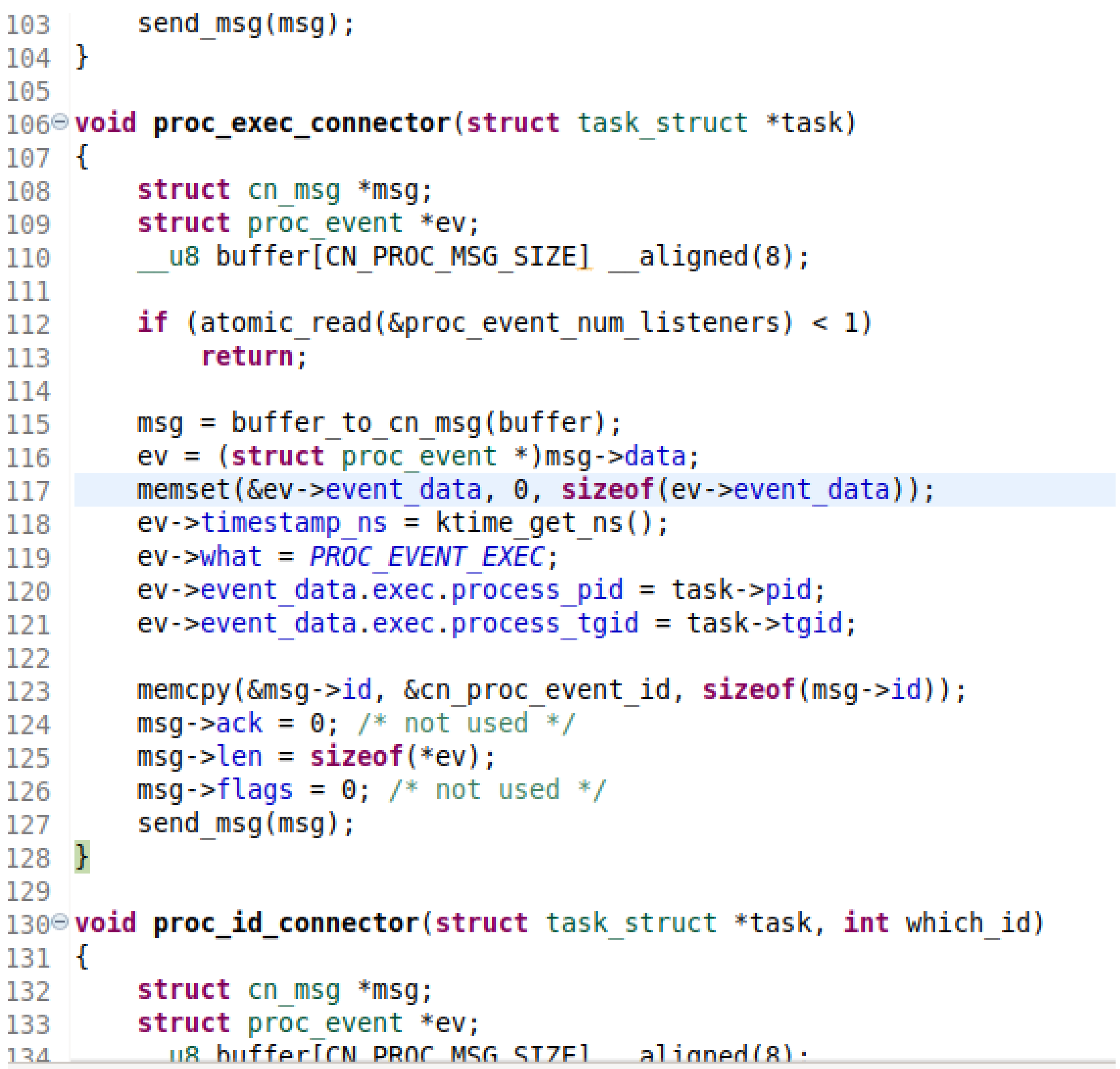
```c
int search_binary_handler(struct linux_binprm *bprm)
{
    bool need_retry = IS_ENABLED(CONFIG_MODULES);
    struct linux_binfmt *fmt;
    int retval;

    /* 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;
```

load_binary -> load_script -> load_misc_binary
load_binary 是在加載可執行程序。linux 內核支持多種可執行程序格式, 每種格式都被註冊為一個 linux_binfmt 結構, 其中存儲了對應可執行程序格式加載函數。



```c
static int load_misc_binary(struct linux_binprm *bprm)
{
    Node *fmt;
    struct file *interp_file = NULL;
    int retval;
    int fd_binary = -1;

    retval = -ENOEXEC;
    if (!enabled)
        return retval;

    /* to keep locking time low, we copy the interpreter string */
    read_lock(&entries_lock);
    fmt = check_file(bprm);
    if (fmt)
        dget(fmt->dentry);
    read_unlock(&entries_lock);
    if (!fmt)
        return retval;

    /* Need to be able to load the file after exec */
    retval = -ENOENT;
    if (bprm->interp_flags & BINPRM_FLAGS_PATH_INACCESSIBLE)
        goto ret;

    if (!(fmt->flags & MISC_FMT_PRESERVE_ARGV0)) {
        retval = remove_arg_zero(bprm);
        if (retval)
            goto ret;
    }

    if (fmt->flags & MISC_FMT_OPEN_BINARY) {
```

3.



```
    send_msg(msg);
}

void proc_exec_connector(struct task_struct *task)
{
    struct cn_msg *msg;
    struct proc_event *ev;
    __u8 buffer[CN_PROC_MSG_SIZE] __aligned(8);

    if (atomic_read(&proc_event_num_listeners) < 1)
        return;

    msg = buffer_to_cn_msg(buffer);
    ev = (struct proc_event *)msg->data;
    memset(&ev->event_data, 0, sizeof(ev->event_data));
    ev->timestamp_ns = ktime_get_ns();
    ev->what = PROC_EVENT_EXEC;
    ev->event_data.exec.process_pid = task->pid;
    ev->event_data.exec.process_tgid = task->tgid;

    memcpy(&msg->id, &cn_proc_event_id, sizeof(msg->id));
    msg->ack = 0; /* not used */
    msg->len = sizeof(*ev);
    msg->flags = 0; /* not used */
    send_msg(msg);
}

void proc_id_connector(struct task_struct *task, int which_id)
{
    struct cn_msg *msg;
    struct proc_event *ev;
    __u8 buffer[CN_PROC_MSG_SIZE] __aligned(8);
```

請問作業系統是否立即載入執行檔案到記憶體中?

我覺得是不會立即載入，memset 這個是追到 exec_binprm 裡面才有的，而這個函式是在註解 execve succeeded 以前一點點執行的，前面還跑了很多初始化還有建結構的動作。