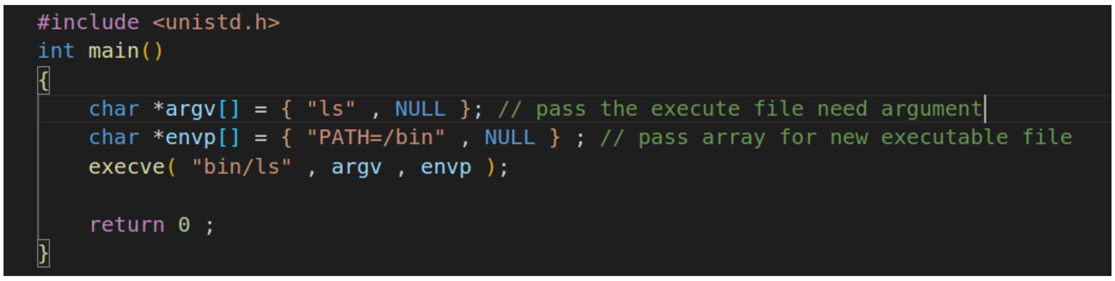
[作業系統概論 hw10]

409410025 邱 x 恩

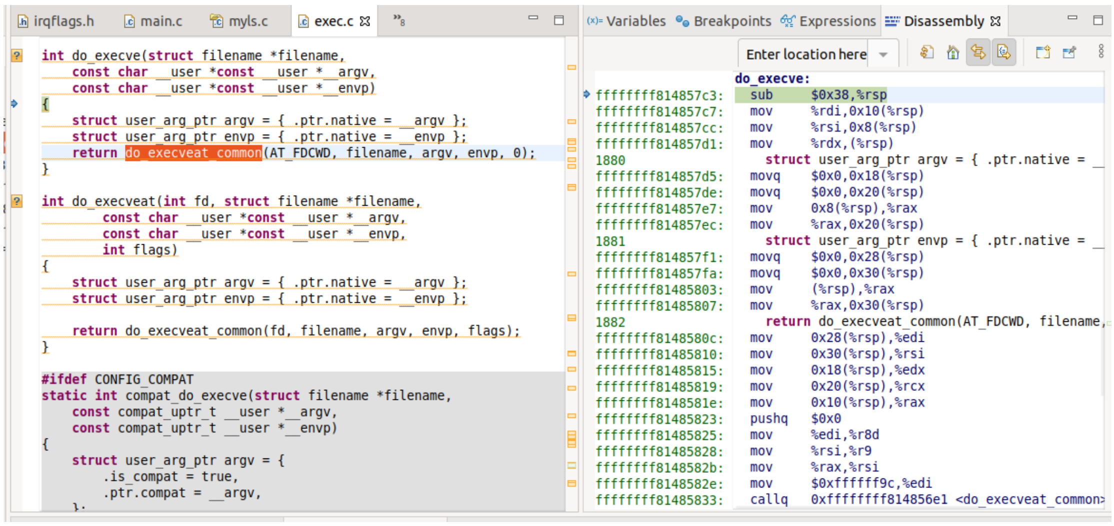
[1]



```c
#include <unistd.h>
int main()
{
    char *argv[] = { "ls" , NULL }; // pass the execute file need argument
    char *envp[] = { "PATH=/bin" , NULL } ; // pass array for new executable file
    execve( "bin/ls" , argv , envp );

    return 0 ;
}
```

[2]

(a) 首先將中斷點設在 do_execve()，發現他呼叫了 do_execveat_common。

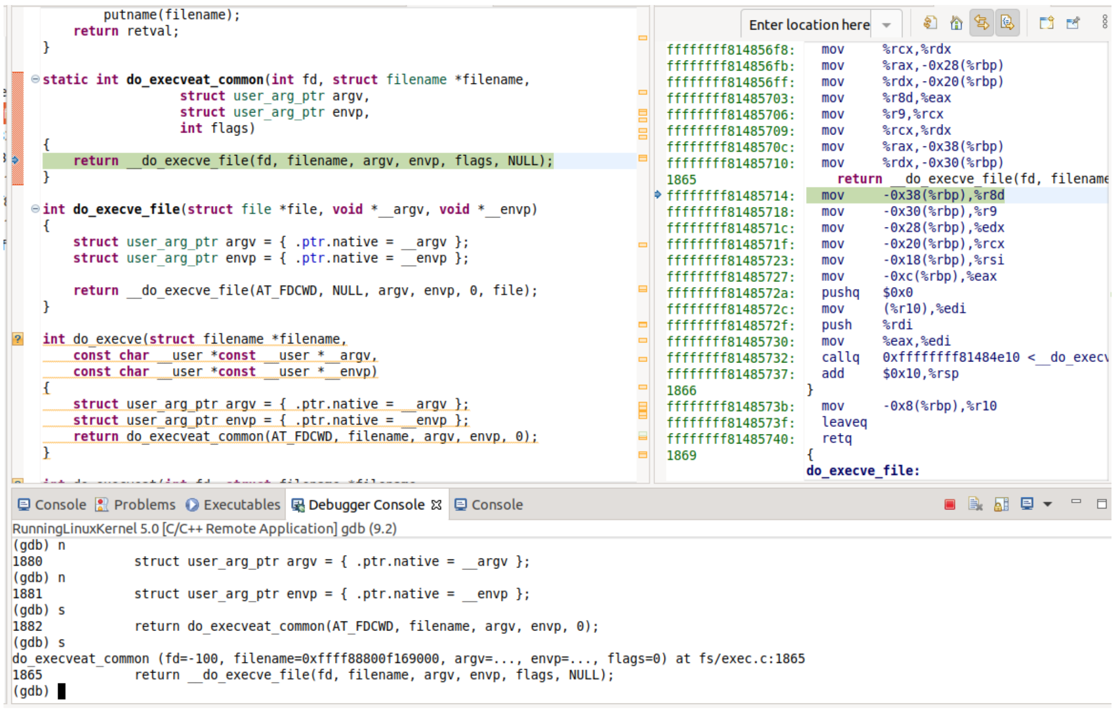


```c
int do_execve(struct filename *filename,
    const char __user *const __user *__argv,
    const char __user *const __user *__envp)
{
    struct user_arg_ptr argv = { .ptr.native = __argv };
    struct user_arg_ptr envp = { .ptr.native = __envp };
    return do_execveat_common(AT_FDCWD, filename,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

```
                    do_execve:
ffffffff814857c3:     sub      $0x38,%rsp
ffffffff814857c7:     mov      %rdi,0x10(%rsp)
ffffffff814857cc:     mov      %rsi,0x8(%rsp)
ffffffff814857d1:     mov      %rdx,(%rsp)
1880                    struct user_arg_ptr argv = { .ptr.native = _
ffffffff814857d5:     movq     $0x0,0x18(%rsp)
ffffffff814857de:     movq     $0x0,0x20(%rsp)
ffffffff814857e7:     mov      0x8(%rsp),%rax
ffffffff814857ec:     mov      %rax,0x20(%rsp)
1881                    struct user_arg_ptr envp = { .ptr.native = _
ffffffff814857f1:     movq     $0x0,0x28(%rsp)
ffffffff814857fa:     movq     $0x0,0x30(%rsp)
ffffffff81485803:     mov      (%rsp),%rax
ffffffff81485807:     mov      %rax,0x30(%rsp)
1882                    return do_execveat_common(AT_FDCWD, filename,
ffffffff8148580c:     mov      0x28(%rsp),%edi
ffffffff81485810:     mov      0x30(%rsp),%rsi
ffffffff81485815:     mov      0x18(%rsp),%edx
ffffffff81485819:     mov      0x20(%rsp),%rcx
ffffffff8148581e:     mov      0x10(%rsp),%rax
ffffffff81485823:     pushq    $0x0
ffffffff81485825:     mov      %edi,%r8d
ffffffff81485828:     mov      %rsi,%r9
ffffffff8148582b:     mov      %rax,%rsi
ffffffff8148582e:     mov      $0xffffff9c,%edi
ffffffff81485833:     callq    0xffffffff814856e1 <do_execveat_common>
```

［ 2 ］

（ b ） 進入 do_execveat_common，看到她回傳__do_execve_file()。



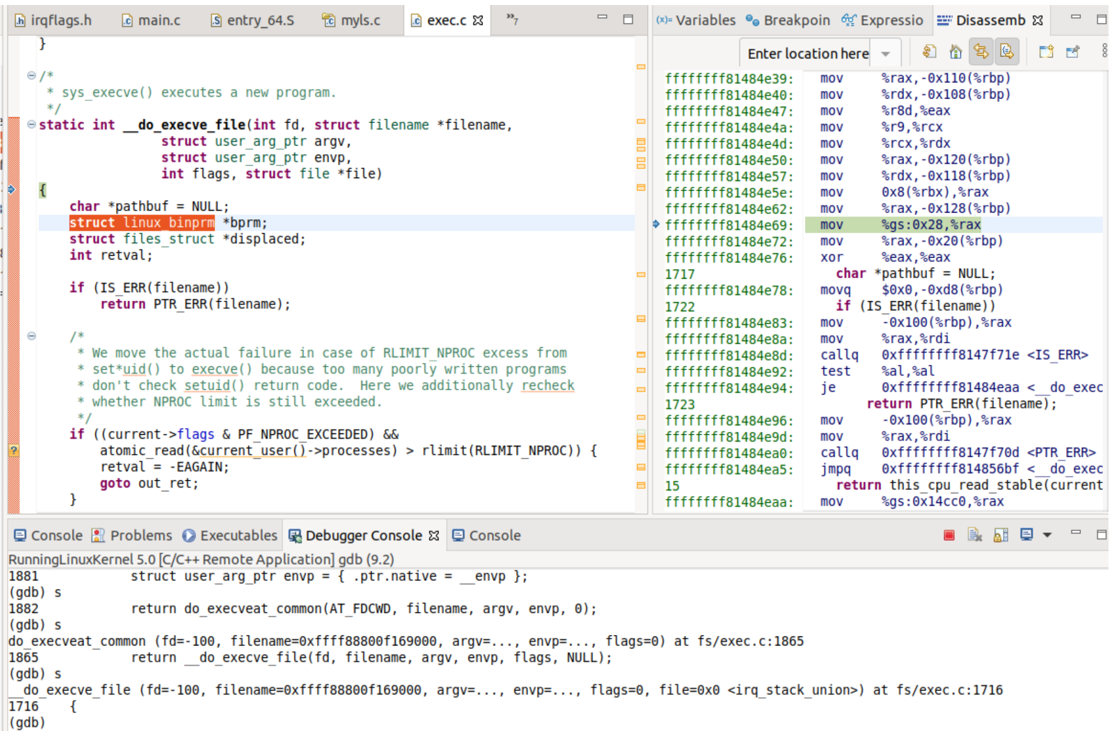
```
        putname(filename);
    return retval;
}

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);
}

int do_execve_file(struct file *file, void *__argv, void *__envp)
{
    struct user_arg_ptr argv = { .ptr.native = __argv };
    struct user_arg_ptr envp = { .ptr.native = __envp };

    return __do_execve_file(AT_FDCWD, NULL, argv, envp, 0, file);
}

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);
}
```

```
ffffffff814856f8:   mov     %rcx,%rdx
ffffffff814856fb:   mov     %rax,-0x28(%rbp)
ffffffff814856ff:   mov     %rdx,-0x20(%rbp)
ffffffff81485703:   mov     %r8d,%eax
ffffffff81485706:   mov     %r9,%rcx
ffffffff81485709:   mov     %rcx,%rdx
ffffffff8148570c:   mov     %rax,-0x38(%rbp)
ffffffff81485710:   mov     %rdx,-0x30(%rbp)
1865                    return __do_execve_file(fd, filename
ffffffff81485714:   mov     -0x38(%rbp),%r8d
ffffffff81485718:   mov     -0x30(%rbp),%r9
ffffffff8148571c:   mov     -0x28(%rbp),%edx
ffffffff8148571f:   mov     -0x20(%rbp),%rcx
ffffffff81485723:   mov     -0x18(%rbp),%rsi
ffffffff81485727:   mov     -0xc(%rbp),%eax
ffffffff8148572a:   pushq   $0x0
ffffffff8148572c:   mov     (%r10),%edi
ffffffff8148572f:   push    %rdi
ffffffff81485730:   mov     %eax,%edi
ffffffff81485732:   callq   0xffffffff81484e10 <__do_execv
ffffffff81485737:   add     $0x10,%rsp
1866                }
ffffffff8148573b:   mov     -0x8(%rbp),%r10
ffffffff8148573f:   leaveq
ffffffff81485740:   retq
1869                {
                    do_execve_file:
```

```
RunningLinuxKernel 5.0 [C/C++ Remote Application] gdb (9.2)
(gdb) n
1880            struct user_arg_ptr argv = { .ptr.native = __argv };
(gdb) n
1881            struct user_arg_ptr envp = { .ptr.native = __envp };
(gdb) s
1882            return do_execveat_common(AT_FDCWD, filename, argv, envp, 0);
(gdb) s
do_execveat_common (fd=-100, filename=0xffff88800f169000, argv=..., envp=..., flags=0) at fs/exec.c:1865
1865            return __do_execve_file(fd, filename, argv, envp, flags, NULL);
(gdb) 
```

[2]

(c) 進入 __do_execve_file()，一個資料結構 struct linux_binprm 用來檢查檔案名稱是否正確。



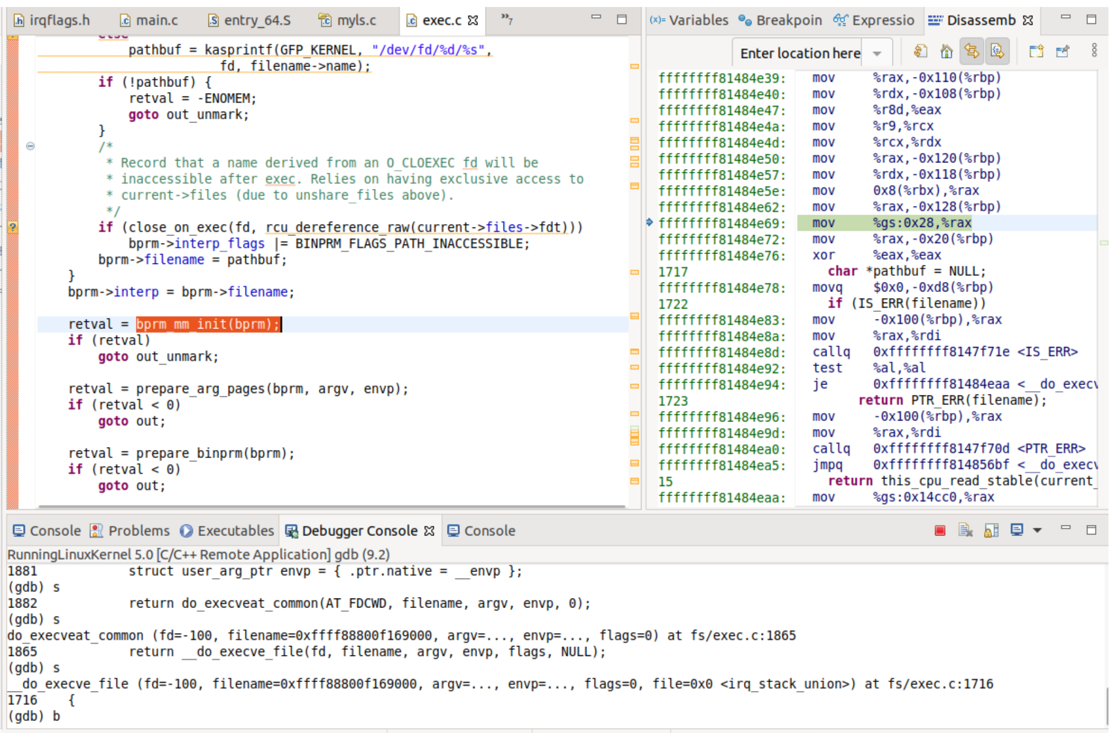
```
}

/*
 * 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 excess from
     * set*uid() to execve() because too many poorly written programs
     * don't check setuid() return code.  Here we additionally recheck
     * whether NPROC limit is still exceeded.
     */
    if ((current->flags & PF_NPROC_EXCEEDED) &&
        atomic_read(&current_user()->processes) > rlimit(RLIMIT_NPROC)) {
        retval = -EAGAIN;
        goto out_ret;
    }
```

```
ffffffff81484e39:   mov     %rax,-0x110(%rbp)
ffffffff81484e40:   mov     %rdx,-0x108(%rbp)
ffffffff81484e47:   mov     %r8d,%eax
ffffffff81484e4a:   mov     %r9,%rcx
ffffffff81484e4d:   mov     %rcx,%rdx
ffffffff81484e50:   mov     %rax,-0x120(%rbp)
ffffffff81484e57:   mov     %rdx,-0x118(%rbp)
ffffffff81484e5e:   mov     0x8(%rbx),%rax
ffffffff81484e62:   mov     %rax,-0x128(%rbp)
ffffffff81484e69:   mov     %gs:0x28,%rax
ffffffff81484e72:   mov     %rax,-0x20(%rbp)
ffffffff81484e76:   xor     %eax,%eax
1717                  char *pathbuf = NULL;
ffffffff81484e78:   movq    $0x0,-0xd8(%rbp)
1722                  if (IS_ERR(filename))
ffffffff81484e83:   mov     -0x100(%rbp),%rax
ffffffff81484e8a:   mov     %rax,%rdi
ffffffff81484e8d:   callq   0xffffffff8147f71e <IS_ERR>
ffffffff81484e92:   test    %al,%al
ffffffff81484e94:   je      0xffffffff81484eaa <__do_exec
1723                      return PTR_ERR(filename);
ffffffff81484e96:   mov     -0x100(%rbp),%rax
ffffffff81484e9d:   mov     %rax,%rdi
ffffffff81484ea0:   callq   0xffffffff8147f70d <PTR_ERR>
ffffffff81484ea5:   jmpq    0xffffffff814856bf <__do_exec
15                    return this_cpu_read_stable(current
ffffffff81484eaa:   mov     %gs:0x14cc0,%rax
```

RunningLinuxKernel 5.0 [C/C++ Remote Application] gdb (9.2)

```
1881            struct user_arg_ptr envp = { .ptr.native = __envp };
(gdb) s
1882            return do_execveat_common(AT_FDCWD, filename, argv, envp, 0);
(gdb) s
do_execveat_common (fd=-100, filename=0xffff88800f169000, argv=..., envp=..., flags=0) at fs/exec.c:1865
1865            return __do_execve_file(fd, filename, argv, envp, flags, NULL);
(gdb) s
__do_execve_file (fd=-100, filename=0xffff88800f169000, argv=..., envp=..., flags=0, file=0x0 <irq_stack_union>) at fs/exec.c:1716
1716    {
(gdb)
```

(d) do_execve_file()呼叫 bprm_mm_init()，為了初始化 process 記憶體空間。



```
            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;
```

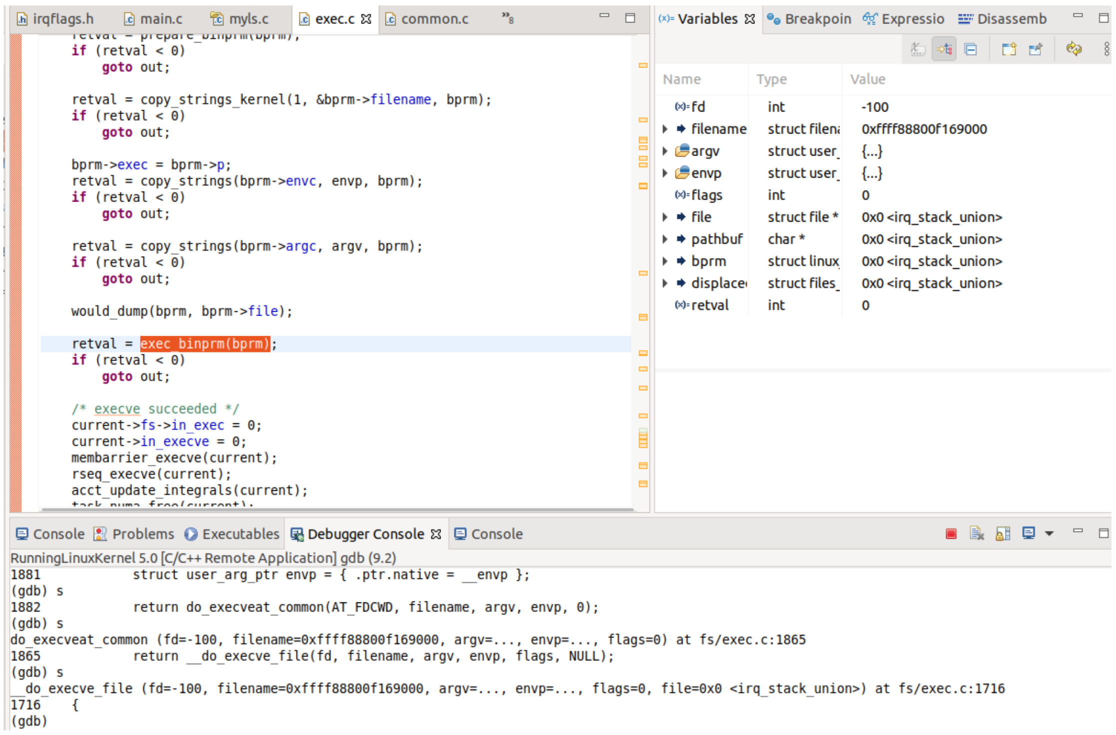
```
ffffffff81484e39:   mov     %rax,-0x110(%rbp)
ffffffff81484e40:   mov     %rdx,-0x108(%rbp)
ffffffff81484e47:   mov     %r8d,%eax
ffffffff81484e4a:   mov     %r9,%rcx
ffffffff81484e4d:   mov     %rcx,%rdx
ffffffff81484e50:   mov     %rax,-0x120(%rbp)
ffffffff81484e57:   mov     %rdx,-0x118(%rbp)
ffffffff81484e5e:   mov     0x8(%rbx),%rax
ffffffff81484e62:   mov     %rax,-0x128(%rbp)
ffffffff81484e69:   mov     %gs:0x28,%rax
ffffffff81484e72:   mov     %rax,-0x20(%rbp)
ffffffff81484e76:   xor     %eax,%eax
1717                  char *pathbuf = NULL;
ffffffff81484e78:   movq    $0x0,-0xd8(%rbp)
1722                  if (IS_ERR(filename))
ffffffff81484e83:   mov     -0x100(%rbp),%rax
ffffffff81484e8a:   mov     %rax,%rdi
ffffffff81484e8d:   callq   0xffffffff8147f71e <IS_ERR>
ffffffff81484e92:   test    %al,%al
ffffffff81484e94:   je      0xffffffff81484eaa <__do_execv
1723                      return PTR_ERR(filename);
ffffffff81484e96:   mov     -0x100(%rbp),%rax
ffffffff81484e9d:   mov     %rax,%rdi
ffffffff81484ea0:   callq   0xffffffff8147f70d <PTR_ERR>
ffffffff81484ea5:   jmpq    0xffffffff814856bf <__do_execv
15                    return this_cpu_read_stable(current_
ffffffff81484eaa:   mov     %gs:0x14cc0,%rax
```

RunningLinuxKernel 5.0 [C/C++ Remote Application] gdb (9.2)

```
1881            struct user_arg_ptr envp = { .ptr.native = __envp };
(gdb) s
1882            return do_execveat_common(AT_FDCWD, filename, argv, envp, 0);
(gdb) s
do_execveat_common (fd=-100, filename=0xffff88800f169000, argv=..., envp=..., flags=0) at fs/exec.c:1865
1865            return __do_execve_file(fd, filename, argv, envp, flags, NULL);
(gdb) s
__do_execve_file (fd=-100, filename=0xffff88800f169000, argv=..., envp=..., flags=0, file=0x0 <irq_stack_union>) at fs/exec.c:1716
1716    {
(gdb) b
```

[2]

(e) 接著呼叫了 exec_binprm，執行新程式。

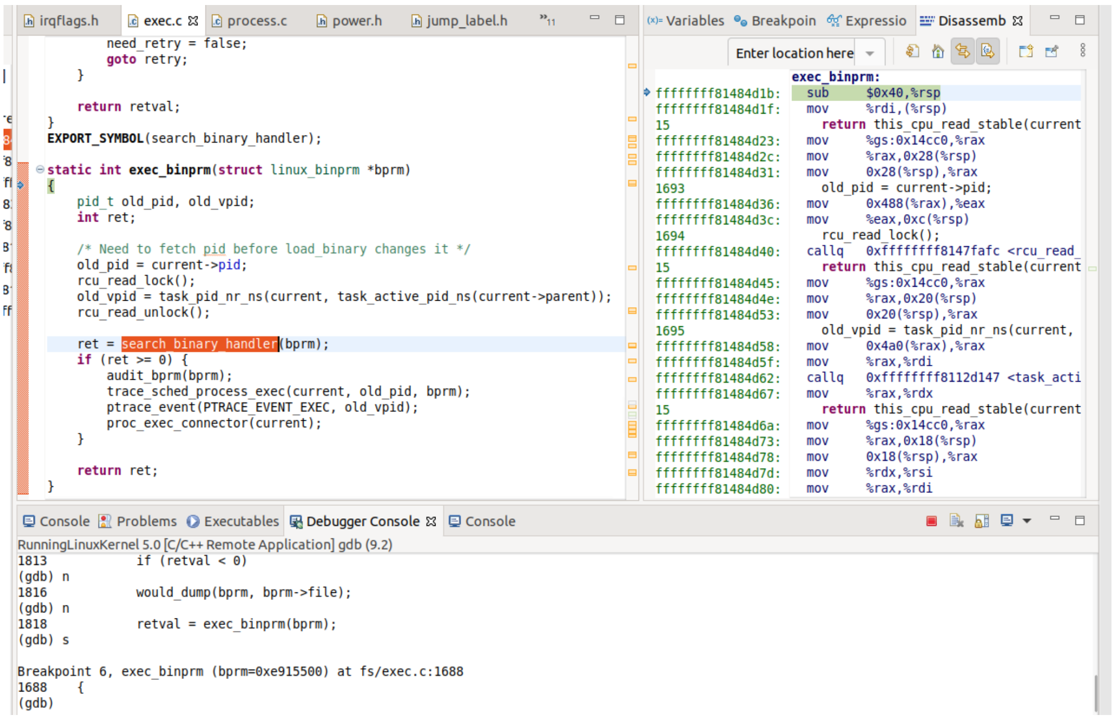


```c
    if (retval < 0)
        goto out;

    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;

    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;

    /* execve succeeded */
    current->fs->in_exec = 0;
    current->in_execve = 0;
    membarrier_execve(current);
    rseq_execve(current);
    acct_update_integrals(current);
```

Name	Type	Value
fd	int	-100
filename	struct filen	0xffff88800f169000
argv	struct user_	{...}
envp	struct user_	{...}
flags	int	0
file	struct file *	0x0 <irq_stack_union>
pathbuf	char *	0x0 <irq_stack_union>
bprm	struct linux	0x0 <irq_stack_union>
displace	struct files_	0x0 <irq_stack_union>
retval	int	0

```
RunningLinuxKernel 5.0 [C/C++ Remote Application] gdb (9.2)
1881            struct user_arg_ptr envp = { .ptr.native = __envp };
(gdb) s
1882            return do_execveat_common(AT_FDCWD, filename, argv, envp, 0);
(gdb) s
do_execveat_common (fd=-100, filename=0xffff88800f169000, argv=..., envp=..., flags=0) at fs/exec.c:1865
1865            return __do_execve_file(fd, filename, argv, envp, flags, NULL);
(gdb) s
__do_execve_file (fd=-100, filename=0xffff88800f169000, argv=..., envp=..., flags=0, file=0x0 <irq_stack_union>) at fs/exec.c:1716
1716    {
(gdb)
```

(f) 進入 exec_binprm，發現他又呼叫了 search_binary_handler()



```c
            need_retry = false;
            goto retry;
        }

    return retval;
}
EXPORT_SYMBOL(search_binary_handler);

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;
}
```

```
                    exec_binprm:
ffffffff81484d1b:   sub     $0x40,%rsp
ffffffff81484d1f:   mov     %rdi,(%rsp)
15                    return this_cpu_read_stable(current
ffffffff81484d23:   mov     %gs:0x14cc0,%rax
ffffffff81484d2c:   mov     %rax,0x28(%rsp)
ffffffff81484d31:   mov     0x28(%rsp),%rax
1693                  old_pid = current->pid;
ffffffff81484d36:   mov     0x488(%rax),%eax
ffffffff81484d3c:   mov     %eax,0xc(%rsp)
1694                  rcu_read_lock();
ffffffff81484d40:   callq   0xffffffff8147fafc <rcu_read_
15                    return this_cpu_read_stable(current
ffffffff81484d45:   mov     %gs:0x14cc0,%rax
ffffffff81484d4e:   mov     %rax,0x20(%rsp)
ffffffff81484d53:   mov     0x20(%rsp),%rax
1695                  old_vpid = task_pid_nr_ns(current,
ffffffff81484d58:   mov     0x4a0(%rax),%rax
ffffffff81484d5f:   mov     %rax,%rdi
ffffffff81484d62:   callq   0xffffffff8112d147 <task_acti
ffffffff81484d67:   mov     %rax,%rdx
15                    return this_cpu_read_stable(current
ffffffff81484d6a:   mov     %gs:0x14cc0,%rax
ffffffff81484d73:   mov     %rax,0x18(%rsp)
ffffffff81484d78:   mov     0x18(%rsp),%rax
ffffffff81484d7d:   mov     %rdx,%rsi
ffffffff81484d80:   mov     %rax,%rdi
```

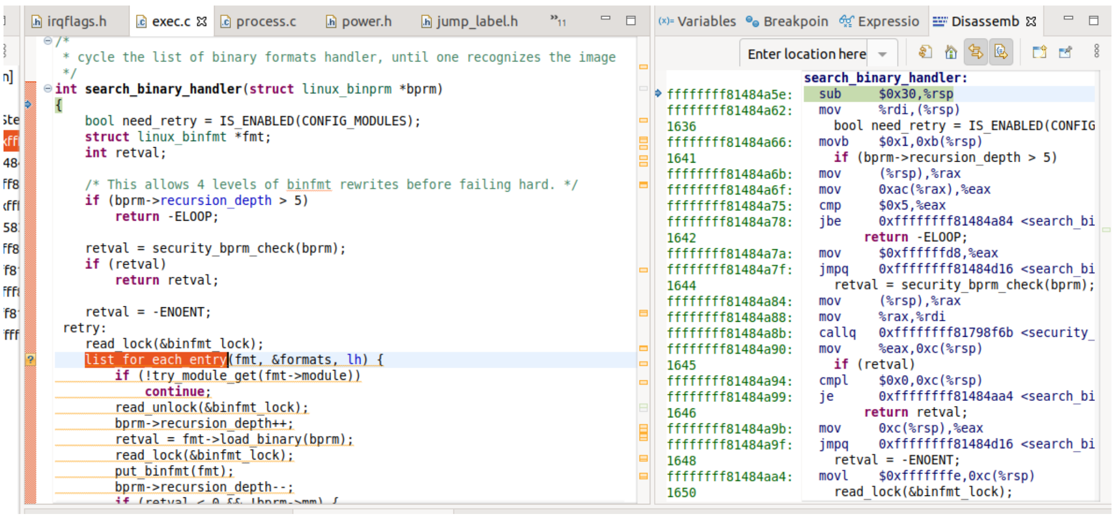
```
RunningLinuxKernel 5.0 [C/C++ Remote Application] gdb (9.2)
1813            if (retval < 0)
(gdb) n
1816            would_dump(bprm, bprm->file);
(gdb) n
1818            retval = exec_binprm(bprm);
(gdb) s

Breakpoint 6, exec_binprm (bprm=0xe915500) at fs/exec.c:1688
1688    {
(gdb)
```

[2]

(g) 進入 search_binary_handler()觀察，

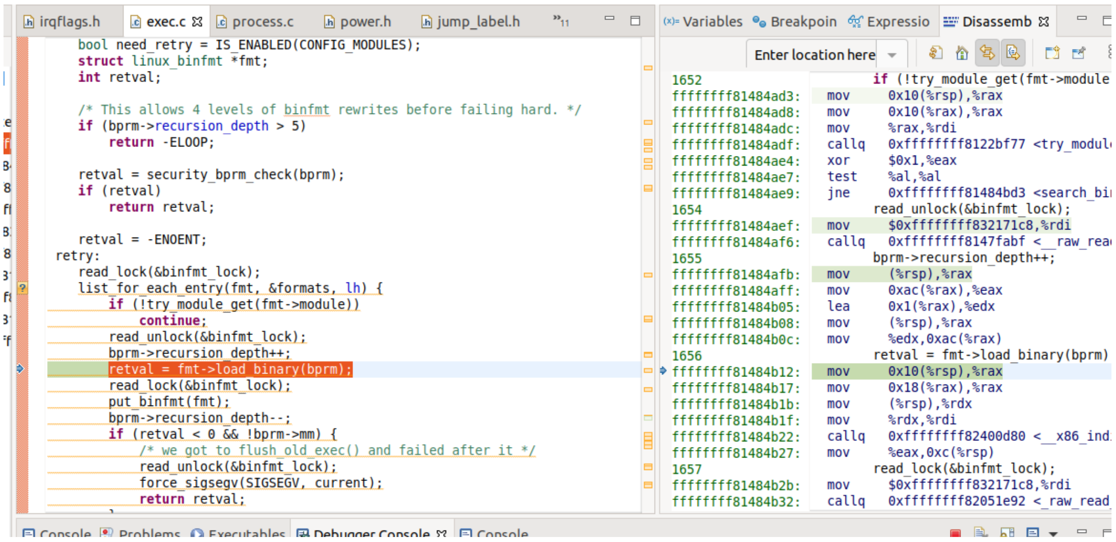
他會尋找可識別的可執行文件，也就是下圖的 list_for_each_entry

找到相對應的文件格式後，呼叫 load_binary。



```
/*
 * 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--;
```

```
                    search_binary_handler:
ffffffff81484a5e:    sub     $0x30,%rsp
ffffffff81484a62:    mov     %rdi,(%rsp)
1636                   bool need_retry = IS_ENABLED(CONFIG
ffffffff81484a66:    movb    $0x1,0xb(%rsp)
1641                   if (bprm->recursion_depth > 5)
ffffffff81484a6b:    mov     (%rsp),%rax
ffffffff81484a6f:    mov     0xac(%rax),%eax
ffffffff81484a75:    cmp     $0x5,%eax
ffffffff81484a78:    jbe     0xffffffff81484a84 <search_bi
1642                       return -ELOOP;
ffffffff81484a7a:    mov     $0xffffffd8,%eax
ffffffff81484a7f:    jmpq    0xffffffff81484d16 <search_bi
1644                   retval = security_bprm_check(bprm);
ffffffff81484a84:    mov     (%rsp),%rax
ffffffff81484a88:    mov     %rax,%rdi
ffffffff81484a8b:    callq   0xffffffff81798f6b <security_
ffffffff81484a90:    mov     %eax,0xc(%rsp)
1645                   if (retval)
ffffffff81484a94:    cmpl    $0x0,0xc(%rsp)
ffffffff81484a99:    je      0xffffffff81484aa4 <search_bi
1646                       return retval;
ffffffff81484a9b:    mov     0xc(%rsp),%eax
ffffffff81484a9f:    jmpq    0xffffffff81484d16 <search_bi
1648                   retval = -ENOENT;
ffffffff81484aa4:    movl    $0xfffffffe,0xc(%rsp)
1650                   read_lock(&binfmt_lock);
```

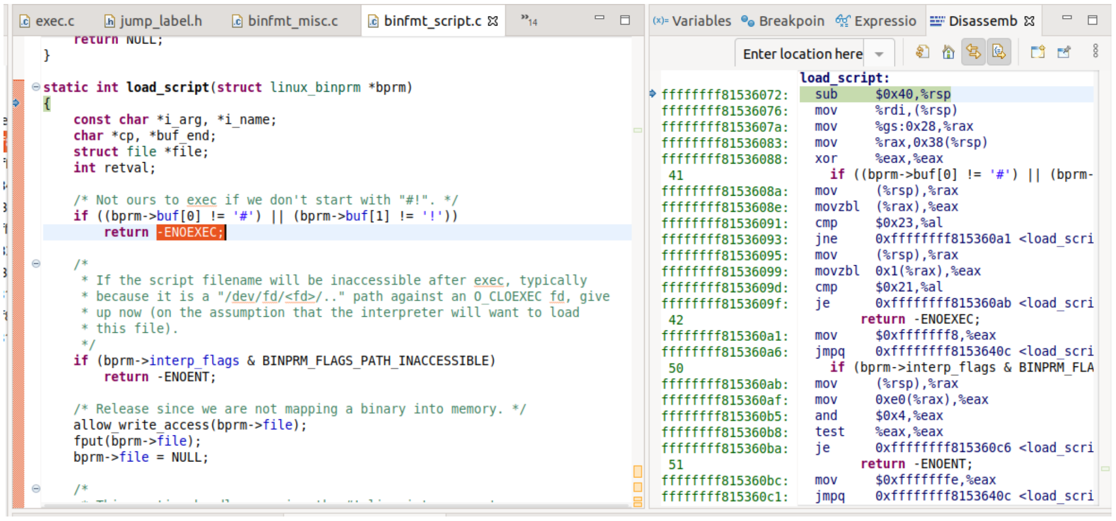


```
	bool need_retry = IS_ENABLED(CONFIG_MODULES);
	struct linux_binfmt *fmt;
	int retval;

	/* 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;
```

```
1652                       if (!try_module_get(fmt->module
ffffffff81484ad3:    mov     0x10(%rsp),%rax
ffffffff81484ad8:    mov     0x10(%rax),%rax
ffffffff81484adc:    mov     %rax,%rdi
ffffffff81484adf:    callq   0xffffffff8122bf77 <try_modul
ffffffff81484ae4:    xor     $0x1,%eax
ffffffff81484ae7:    test    %al,%al
ffffffff81484ae9:    jne     0xffffffff81484bd3 <search_bi
1654                       read_unlock(&binfmt_lock);
ffffffff81484aef:    mov     $0xffffffff832171c8,%rdi
ffffffff81484af6:    callq   0xffffffff8147fabf <__raw_rea
1655                       bprm->recursion_depth++;
ffffffff81484afb:    mov     (%rsp),%rax
ffffffff81484aff:    mov     0xac(%rax),%eax
ffffffff81484b05:    lea     0x1(%rax),%edx
ffffffff81484b08:    mov     (%rsp),%rax
ffffffff81484b0c:    mov     %edx,0xac(%rax)
1656                       retval = fmt->load_binary(bprm)
ffffffff81484b12:    mov     0x10(%rsp),%rax
ffffffff81484b17:    mov     0x18(%rax),%rax
ffffffff81484b1b:    mov     (%rsp),%rdx
ffffffff81484b1f:    mov     %rdx,%rdi
ffffffff81484b22:    callq   0xffffffff82400d80 <__x86_ind
ffffffff81484b27:    mov     %eax,0xc(%rsp)
1657                       read_lock(&binfmt_lock);
ffffffff81484b2b:    mov     $0xffffffff832171c8,%rdi
ffffffff81484b32:    callq   0xffffffff82051e92 <_raw_read
```

[2]

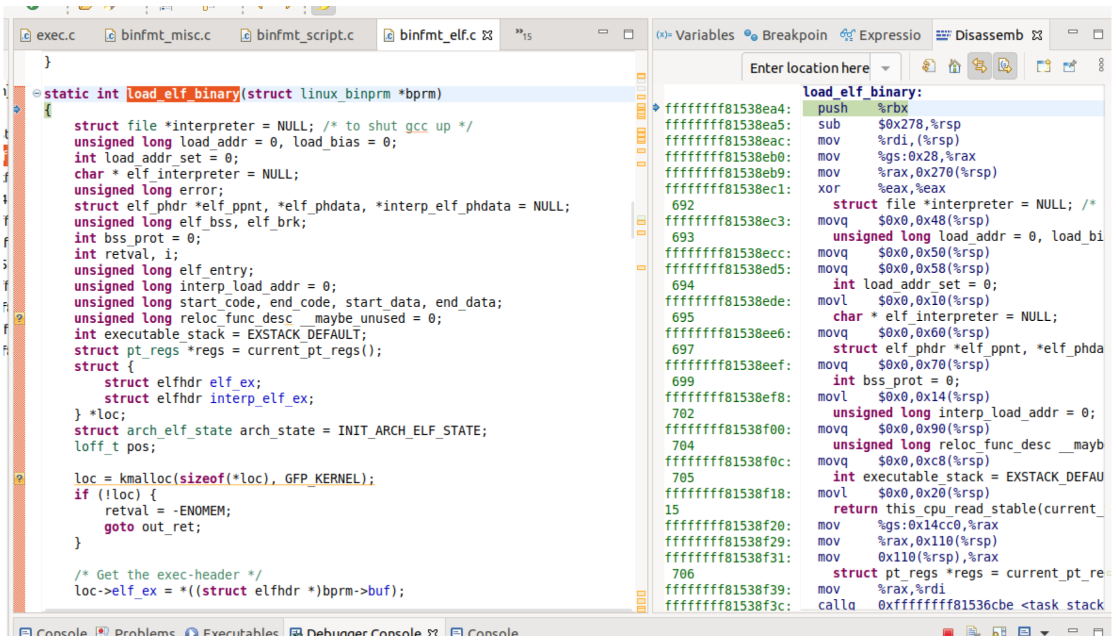
(h)進入 load_binary 後，發現會進入 load_script，他是在檢查檔案格式。
   若是格式不同，他就會回報 enoexec 這個錯誤



```
    return NULL;
}

static int load_script(struct linux_binprm *bprm)
{
    const char *i_arg, *i_name;
    char *cp, *buf_end;
    struct file *file;
    int retval;

    /* Not ours to exec if we don't start with "#!". */
    if ((bprm->buf[0] != '#') || (bprm->buf[1] != '!'))
        return -ENOEXEC;

    /*
     * If the script filename will be inaccessible after exec, typically
     * because it is a "/dev/fd/<fd>/.." path against an O_CLOEXEC fd, give
     * up now (on the assumption that the interpreter will want to load
     * this file).
     */
    if (bprm->interp_flags & BINPRM_FLAGS_PATH_INACCESSIBLE)
        return -ENOENT;

    /* Release since we are not mapping a binary into memory. */
    allow_write_access(bprm->file);
    fput(bprm->file);
    bprm->file = NULL;

    /*
```

```
                   load_script:
ffffffff81536072:   sub     $0x40,%rsp
ffffffff81536076:   mov     %rdi,(%rsp)
ffffffff8153607a:   mov     %gs:0x28,%rax
ffffffff81536083:   mov     %rax,0x38(%rsp)
ffffffff81536088:   xor     %eax,%eax
 41                   if ((bprm->buf[0] != '#') || (bprm-
ffffffff8153608a:   mov     (%rsp),%rax
ffffffff8153608e:   movzbl  (%rax),%eax
ffffffff81536091:   cmp     $0x23,%al
ffffffff81536093:   jne     0xffffffff815360a1 <load_scri
ffffffff81536095:   mov     (%rsp),%rax
ffffffff81536099:   movzbl  0x1(%rax),%eax
ffffffff8153609d:   cmp     $0x21,%al
ffffffff8153609f:   je      0xffffffff815360ab <load_scri
 42                       return -ENOEXEC;
ffffffff815360a1:   mov     $0xfffffff8,%eax
ffffffff815360a6:   jmpq    0xffffffff8153640c <load_scri
 50                   if (bprm->interp_flags & BINPRM_FLA
ffffffff815360ab:   mov     (%rsp),%rax
ffffffff815360af:   mov     0xe0(%rax),%eax
ffffffff815360b5:   and     $0x4,%eax
ffffffff815360b8:   test    %eax,%eax
ffffffff815360ba:   je      0xffffffff815360c6 <load_scri
 51                       return -ENOENT;
ffffffff815360bc:   mov     $0xfffffffe,%eax
ffffffff815360c1:   jmpq    0xffffffff8153640c <load_scri
```

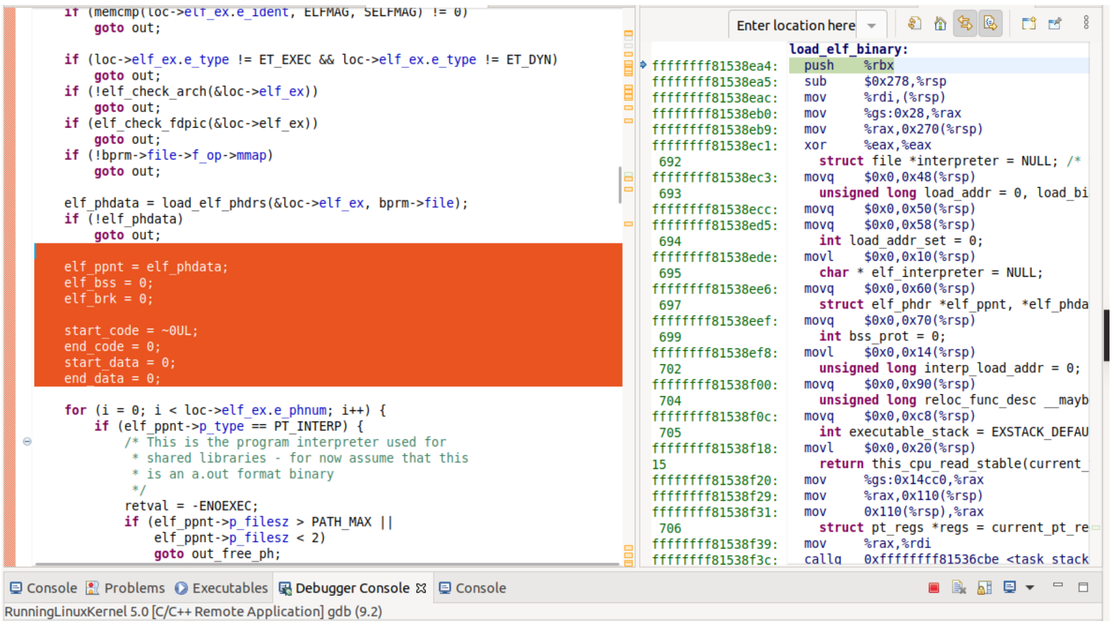
(i) 接著會跳回 load_binary，在進入一次 load_binary。
     會發現進入了 load_elf_binary
     elf 是一個 linux 環境下可執行文件格式。



```
}

static int load_elf_binary(struct linux_binprm *bprm)
{
    struct file *interpreter = NULL; /* to shut gcc up */
    unsigned long load_addr = 0, load_bias = 0;
    int load_addr_set = 0;
    char * elf_interpreter = NULL;
    unsigned long error;
    struct elf_phdr *elf_ppnt, *elf_phdata, *interp_elf_phdata = NULL;
    unsigned long elf_bss, elf_brk;
    int bss_prot = 0;
    int retval, i;
    unsigned long elf_entry;
    unsigned long interp_load_addr = 0;
    unsigned long start_code, end_code, start_data, end_data;
    unsigned long reloc_func_desc __maybe_unused = 0;
    int executable_stack = EXSTACK_DEFAULT;
    struct pt_regs *regs = current_pt_regs();
    struct {
        struct elfhdr elf_ex;
        struct elfhdr interp_elf_ex;
    } *loc;
    struct arch_elf_state arch_state = INIT_ARCH_ELF_STATE;
    loff_t pos;

    loc = kmalloc(sizeof(*loc), GFP_KERNEL);
    if (!loc) {
        retval = -ENOMEM;
        goto out_ret;
    }

    /* Get the exec-header */
    loc->elf_ex = *((struct elfhdr *)bprm->buf);
```

```
                   load_elf_binary:
ffffffff81538ea4:   push    %rbx
ffffffff81538ea5:   sub     $0x278,%rsp
ffffffff81538eac:   mov     %rdi,(%rsp)
ffffffff81538eb0:   mov     %gs:0x28,%rax
ffffffff81538eb9:   mov     %rax,0x270(%rsp)
ffffffff81538ec1:   xor     %eax,%eax
 692                  struct file *interpreter = NULL; /*
ffffffff81538ec3:   movq    $0x0,0x48(%rsp)
 693                  unsigned long load_addr = 0, load_bi
ffffffff81538ecc:   movq    $0x0,0x50(%rsp)
ffffffff81538ed5:   movq    $0x0,0x58(%rsp)
 694                  int load_addr_set = 0;
ffffffff81538ede:   movl    $0x0,0x10(%rsp)
 695                  char * elf_interpreter = NULL;
ffffffff81538ee6:   movq    $0x0,0x60(%rsp)
 697                  struct elf_phdr *elf_ppnt, *elf_phda
ffffffff81538eef:   movq    $0x0,0x70(%rsp)
 699                  int bss_prot = 0;
ffffffff81538ef8:   movl    $0x0,0x14(%rsp)
 702                  unsigned long interp_load_addr = 0;
ffffffff81538f00:   movq    $0x0,0x90(%rsp)
 704                  unsigned long reloc_func_desc __mayb
ffffffff81538f0c:   movq    $0x0,0xc8(%rsp)
 705                  int executable_stack = EXSTACK_DEFAU
ffffffff81538f18:   movl    $0x0,0x20(%rsp)
15                    return this_cpu_read_stable(current_
ffffffff81538f20:   mov     %gs:0x14cc0,%rax
ffffffff81538f29:   mov     %rax,0x110(%rsp)
ffffffff81538f31:   mov     0x110(%rsp),%rax
 706                  struct pt_regs *regs = current_pt_re
ffffffff81538f39:   mov     %rax,%rdi
ffffffff81538f3c:   callq   0xffffffff81536cbe <task_stack
```

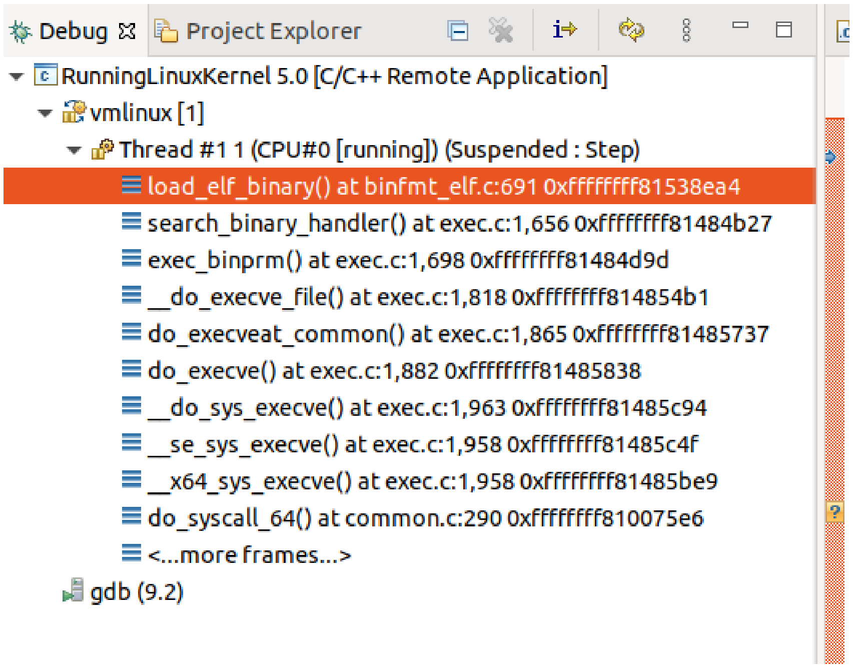
（i）這邊接著就會針對檔案做載入的動作，就完成了載入檔案主要的工作。



```
	if (memcmp(loc->elf_ex.e_ident, ELFMAG, SELFMAG) != 0)
		goto out;

	if (loc->elf_ex.e_type != ET_EXEC && loc->elf_ex.e_type != ET_DYN)
		goto out;
	if (!elf_check_arch(&loc->elf_ex))
		goto out;
	if (elf_check_fdpic(&loc->elf_ex))
		goto out;
	if (!bprm->file->f_op->mmap)
		goto out;

	elf_phdata = load_elf_phdrs(&loc->elf_ex, bprm->file);
	if (!elf_phdata)
		goto out;

	elf_ppnt = elf_phdata;
	elf_bss = 0;
	elf_brk = 0;

	start_code = ~0UL;
	end_code = 0;
	start_data = 0;
	end_data = 0;

	for (i = 0; i < loc->elf_ex.e_phnum; i++) {
		if (elf_ppnt->p_type == PT_INTERP) {
			/* This is the program interpreter used for
			 * shared libraries - for now assume that this
			 * is an a.out format binary
			 */
			retval = -ENOEXEC;
			if (elf_ppnt->p_filesz > PATH_MAX ||
			    elf_ppnt->p_filesz < 2)
				goto out_free_ph;
```

```
                   load_elf_binary:
ffffffff81538ea4:  push    %rbx
ffffffff81538ea5:  sub     $0x278,%rsp
ffffffff81538eac:  mov     %rdi,(%rsp)
ffffffff81538eb0:  mov     %gs:0x28,%rax
ffffffff81538eb9:  mov     %rax,0x270(%rsp)
ffffffff81538ec1:  xor     %eax,%eax
 692                 struct file *interpreter = NULL; /*
ffffffff81538ec3:  movq    $0x0,0x48(%rsp)
 693                 unsigned long load_addr = 0, load_bi
ffffffff81538ecc:  movq    $0x0,0x50(%rsp)
ffffffff81538ed5:  movq    $0x0,0x58(%rsp)
 694                 int load_addr_set = 0;
ffffffff81538ede:  movl    $0x0,0x10(%rsp)
 695                 char * elf_interpreter = NULL;
ffffffff81538ee6:  movq    $0x0,0x60(%rsp)
 697                 struct elf_phdr *elf_ppnt, *elf_phda
ffffffff81538eef:  movq    $0x0,0x70(%rsp)
 699                 int bss_prot = 0;
ffffffff81538ef8:  movl    $0x0,0x14(%rsp)
 702                 unsigned long interp_load_addr = 0;
ffffffff81538f00:  movq    $0x0,0x90(%rsp)
 704                 unsigned long reloc_func_desc __mayb
ffffffff81538f0c:  movq    $0x0,0xc8(%rsp)
 705                 int executable_stack = EXSTACK_DEFAU
ffffffff81538f18:  movl    $0x0,0x20(%rsp)
15                   return this_cpu_read_stable(current_
ffffffff81538f20:  mov     %gs:0x14cc0,%rax
ffffffff81538f29:  mov     %rax,0x110(%rsp)
ffffffff81538f31:  mov     0x110(%rsp),%rax
 706                 struct pt_regs *regs = current_pt_re
ffffffff81538f39:  mov     %rax,%rdi
ffffffff81538f3c:  callq   0xffffffff81536cbe <task_stack
```

RunningLinuxKernel 5.0 [C/C++ Remote Application] gdb (9.2)

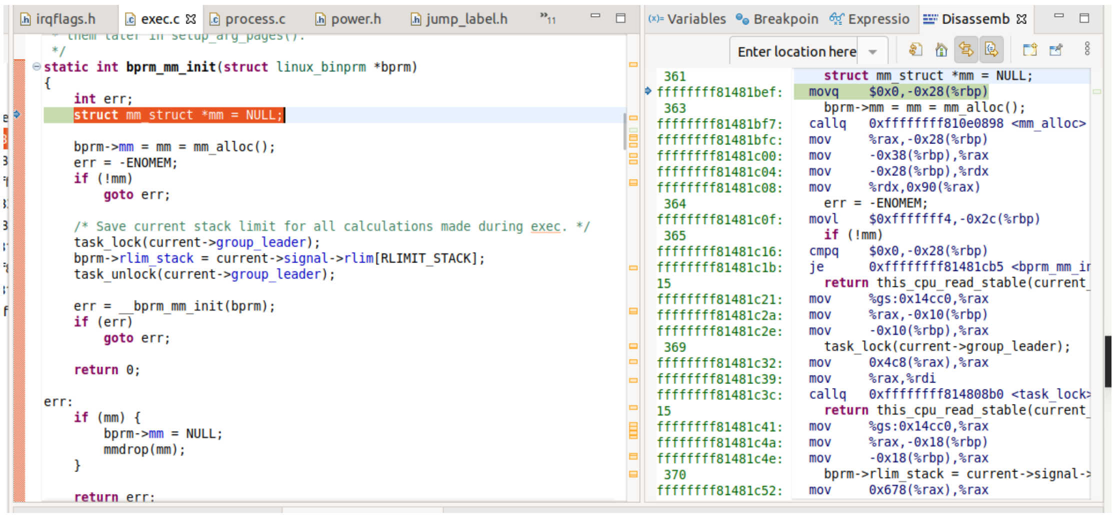


```
RunningLinuxKernel 5.0 [C/C++ Remote Application]
  vmlinux [1]
    Thread #1 1 (CPU#0 [running]) (Suspended : Step)
      load_elf_binary() at binfmt_elf.c:691 0xffffffff81538ea4
      search_binary_handler() at exec.c:1,656 0xffffffff81484b27
      exec_binprm() at exec.c:1,698 0xffffffff81484d9d
      __do_execve_file() at exec.c:1,818 0xffffffff814854b1
      do_execveat_common() at exec.c:1,865 0xffffffff81485737
      do_execve() at exec.c:1,882 0xffffffff81485838
      __do_sys_execve() at exec.c:1,963 0xffffffff81485c94
      __se_sys_execve() at exec.c:1,958 0xffffffff81485c4f
      __x64_sys_execve() at exec.c:1,958 0xffffffff81485be9
      do_syscall_64() at common.c:290 0xffffffff810075e6
      <...more frames...>
  gdb (9.2)
```

[3]

不會立即載入。

在下圖當中可以觀察到，
bprm_mm_init 傳入*bprm 後，
接著做初始化。
OS 只會先幫忙修改 task_struct 中的 mm_struct，**不會**立即載入。



```
 * 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;
```

```
 361                   struct mm_struct *mm = NULL;
ffffffff81481bef:   movq    $0x0,-0x28(%rbp)
 363                   bprm->mm = mm = mm_alloc();
ffffffff81481bf7:   callq   0xffffffff810e0898 <mm_alloc>
ffffffff81481bfc:   mov     %rax,-0x28(%rbp)
ffffffff81481c00:   mov     -0x38(%rbp),%rax
ffffffff81481c04:   mov     -0x28(%rbp),%rdx
ffffffff81481c08:   mov     %rdx,0x90(%rax)
 364                   err = -ENOMEM;
ffffffff81481c0f:   movl    $0xfffffff4,-0x2c(%rbp)
 365                   if (!mm)
ffffffff81481c16:   cmpq    $0x0,-0x28(%rbp)
ffffffff81481c1b:   je      0xffffffff81481cb5 <bprm_mm_in
15                     return this_cpu_read_stable(current_
ffffffff81481c21:   mov     %gs:0x14cc0,%rax
ffffffff81481c2a:   mov     %rax,-0x10(%rbp)
ffffffff81481c2e:   mov     -0x10(%rbp),%rax
 369                   task_lock(current->group_leader);
ffffffff81481c32:   mov     0x4c8(%rax),%rax
ffffffff81481c39:   mov     %rax,%rdi
ffffffff81481c3c:   callq   0xffffffff814808b0 <task_lock>
15                     return this_cpu_read_stable(current_
ffffffff81481c41:   mov     %gs:0x14cc0,%rax
ffffffff81481c4a:   mov     %rax,-0x18(%rbp)
ffffffff81481c4e:   mov     -0x18(%rbp),%rax
 370                   bprm->rlim_stack = current->signal->
ffffffff81481c52:   mov     0x678(%rax),%rax
```