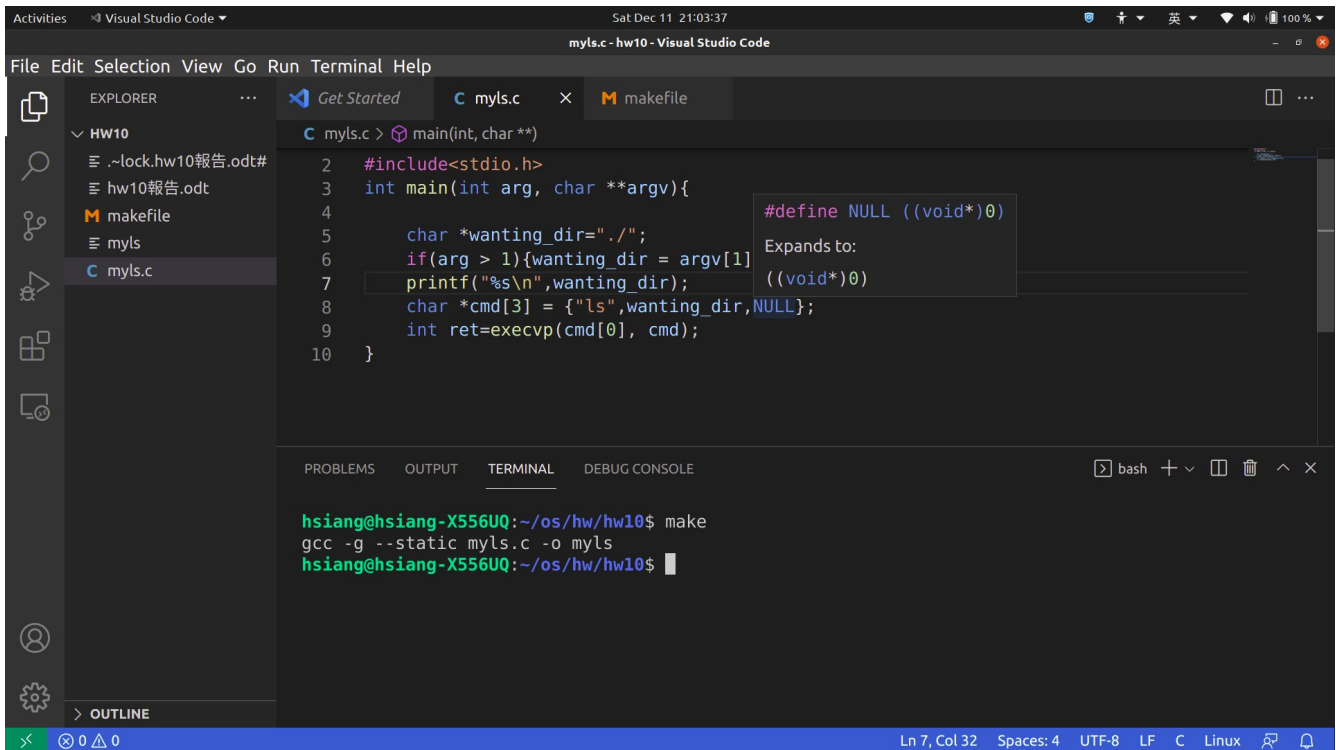


HW10 報告

資工三 408410098 蔡×祥

1.



The screenshot shows the Visual Studio Code editor with a C program named `mysls.c` open. The program is a simple implementation of the `ls` command. It includes `stdio.h` and defines a `main` function that takes command-line arguments. It uses `printf` to print the directory name and `execvp` to execute the `ls` command. A tooltip for `NULL` is visible, showing its expansion to `((void*)0)`.

```
1  #include <stdio.h>
2  int main(int arg, char **argv){
3
4      char *wanting_dir=".";
5      if(arg > 1){wanting_dir = argv[1]}
6      printf("%s\n",wanting_dir);
7      char *cmd[3] = {"ls",wanting_dir,NULL};
8      int ret=execvp(cmd[0], cmd);
9
10 }
```

The terminal output shows the command `make` being executed, which runs `gcc -g --static myls.c -o myls` to compile the program.

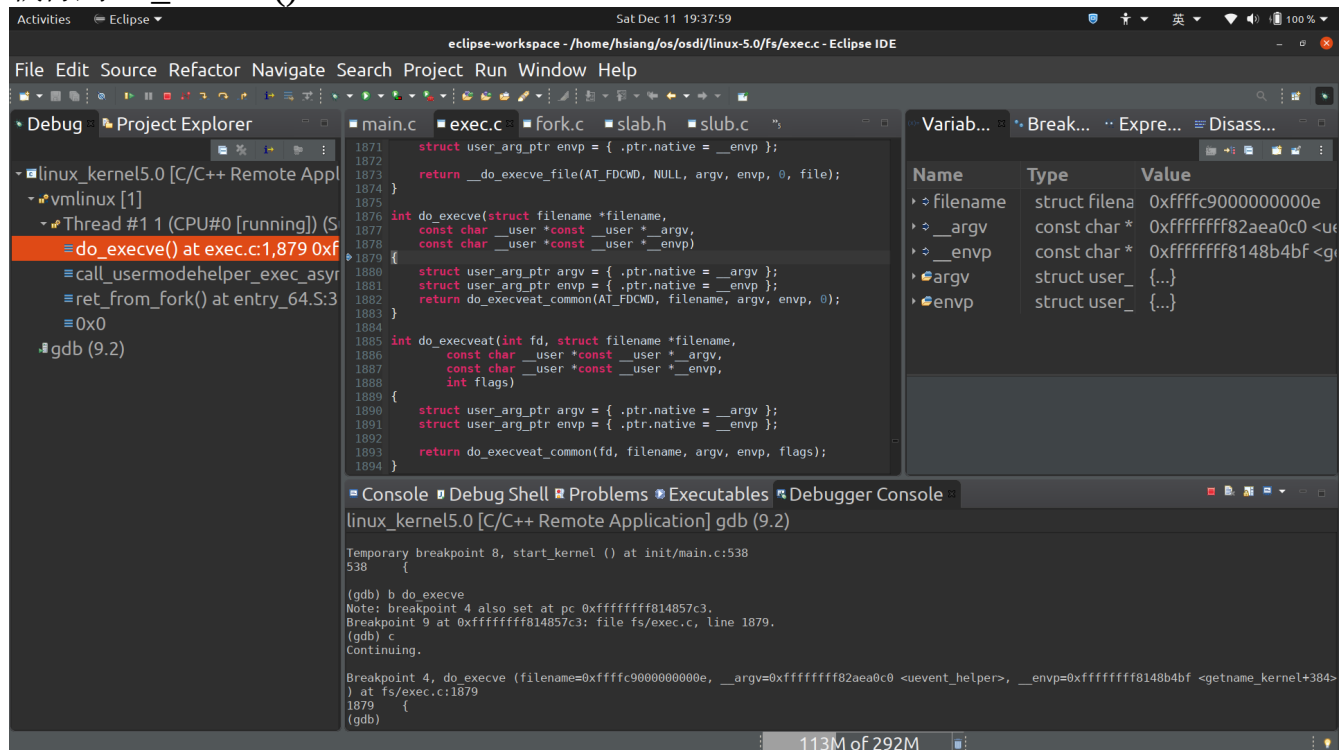
```
hsiang@hsiang-X556UQ:~/os/hw/hw10$ make
gcc -g --static myls.c -o myls
hsiang@hsiang-X556UQ:~/os/hw/hw10$
```

我是使用 `execvp()` 函數來調用 `ls` 執行檔

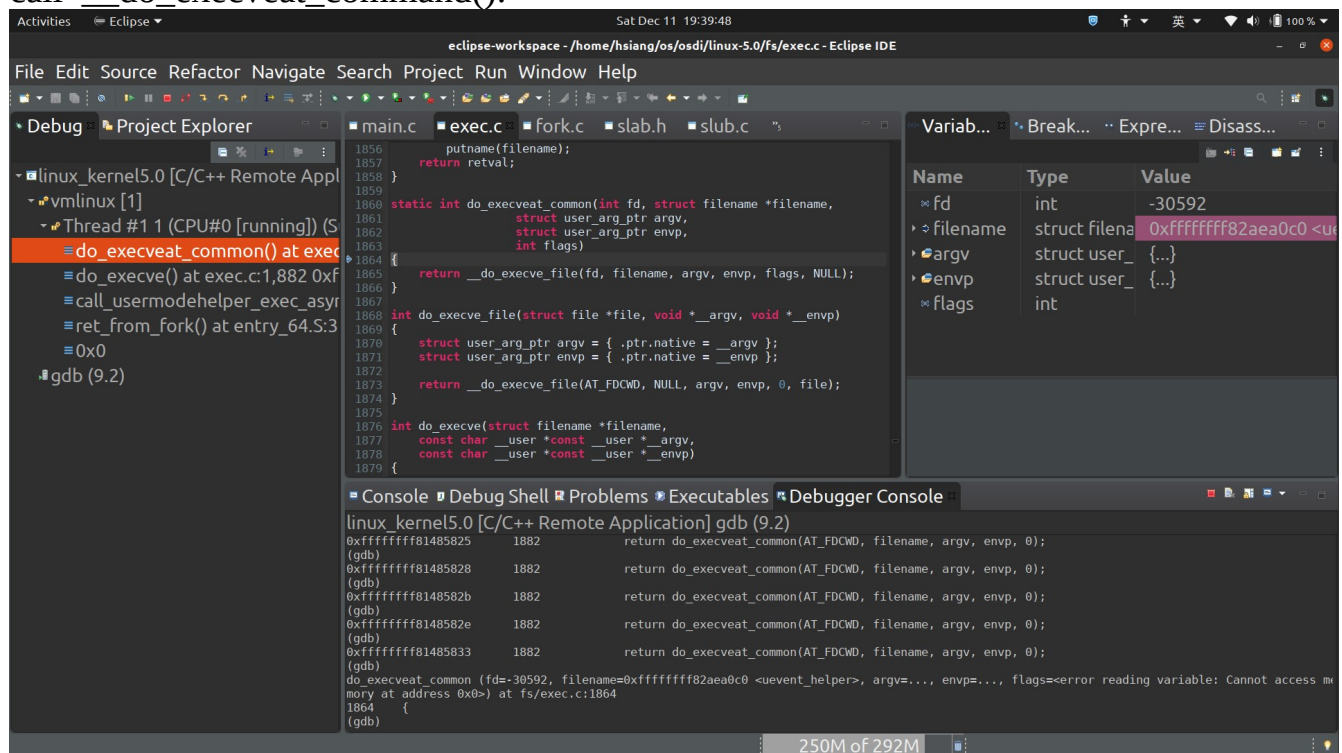
2.

對 linux kernel debug:

執行到 do_execve():



call __do_execveat_command():



The screenshot shows the Eclipse IDE interface. The main editor window displays the source code of 'exec.c' with the function '_do_execve_file' highlighted. The left sidebar shows the 'Project Explorer' with the file 'exec.c' selected. The bottom panel shows the 'Console' output, which includes the program's output and the debugger's internal state. The status bar at the bottom indicates '104M of 292M' memory usage.

The screenshot shows the Eclipse IDE interface with the following components:

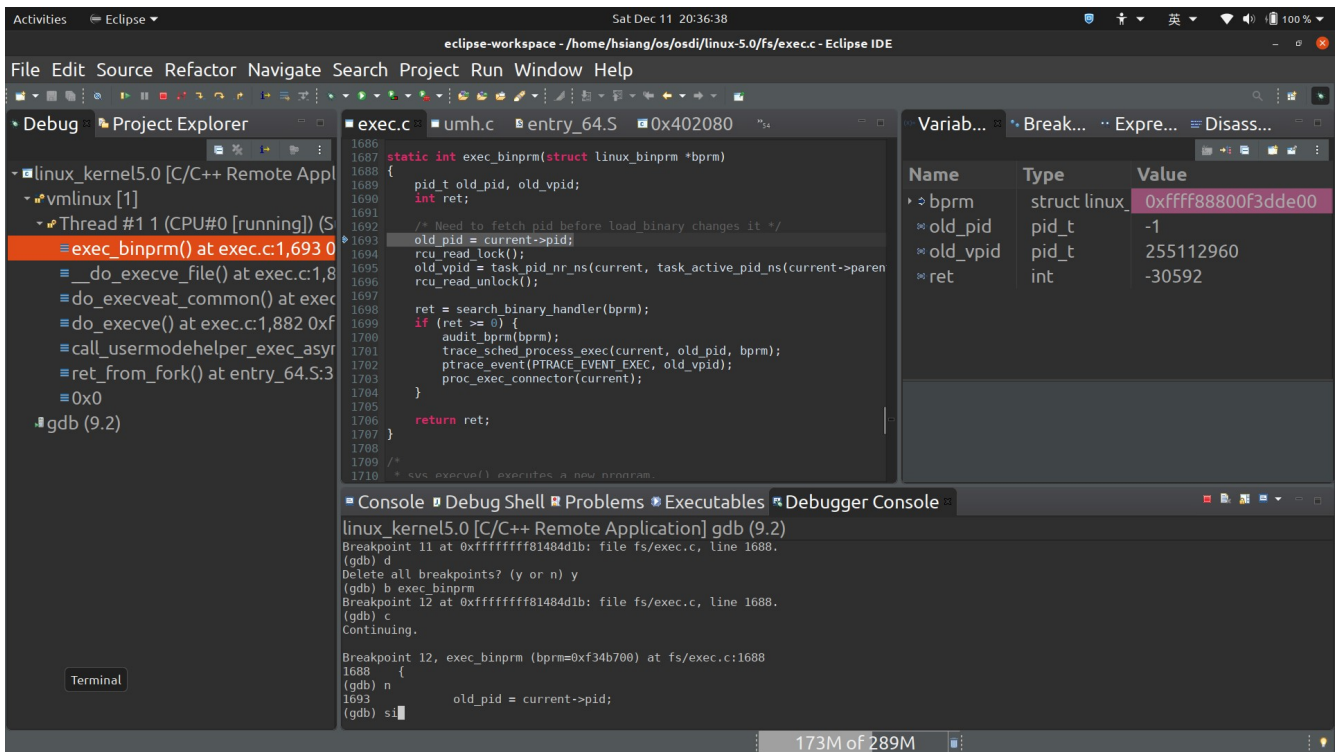
- Top Bar:** Activities, Eclipse, Sat Dec 11, 20:35:15, and system status icons.
- Menu Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Project Explorer:** Shows the project structure:
 - linux_kernel5.0 [C/C++ Remote Appl...
 - vmlinux [1]
 - Thread #1 1 (CPU#0 [running]) (S...)
 - do_execve_file() at exec.c:1,7
 - do_execveat_common() at exec...
 - do_execve() at exec.c:1,882 0xf...
 - call_usermodehelper_exec_asyn...
 - ret_from_fork() at entry_64.S:3
 - 0x0
 - gdb (9.2)
- Source Editor:** Displays the code for 'exec.c'. The function 'do_execveat_common' is selected, showing the logic for preparing bprm, copying strings, and calling 'do_execve'.
- Variables View:**

Name	Type	Value
fd	int	-1
filename	struct filena	0xffff88800f168000
argv	struct user_	{...}
envp	struct user_	{...}
flags	int	0
file	struct file *	0xffff88800f08f100
pathbuf	char *	0xffff88800f08f100 "l
bprm	struct linux	0x1c9a... stack union+
- Debugger Console:**

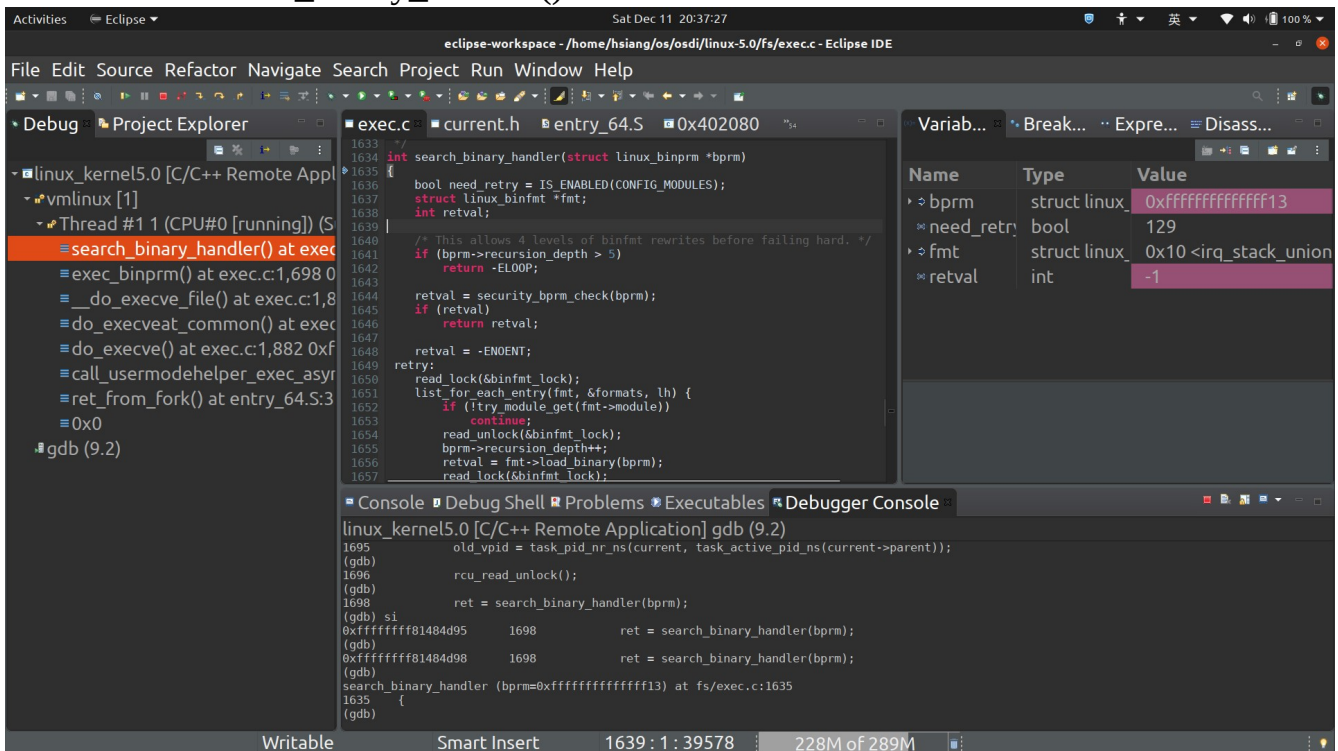
```

linux_kernel5.0 [C/C++ Remote Application] gdb (9.2)
0xffffffffff8148572a 1865 return __do_execve_file(fd, filename, argv, envp, flags, NULL);
(gdb)
0xffffffffff8148572c 1865 return __do_execve_file(fd, filename, argv, envp, flags, NULL);
(gdb)
0xffffffffff8148572f 1865 return __do_execve_file(fd, filename, argv, envp, flags, NULL);
(gdb)
0xffffffffff81485730 1865 return __do_execve_file(fd, filename, argv, envp, flags, NULL);
(gdb)
0xffffffffff81485732 1865 return __do_execve_file(fd, filename, argv, envp, flags, NULL);
(gdb)
do_execve_file (fd=1, filename=0xffff88800f168000, argv=..., envp=..., flags=0, file=0xffff88800f08f100) at fs/exec.c:1716
1716 {
(gdb)

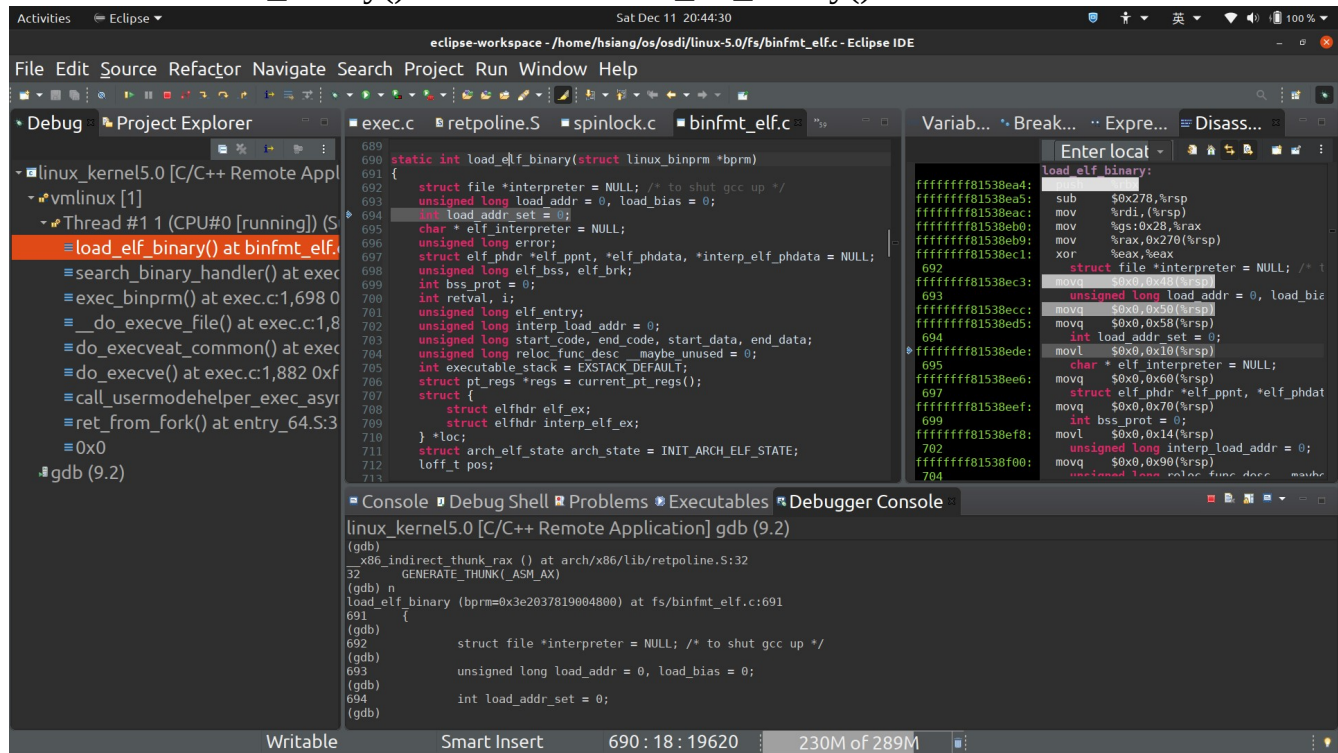
```
- Bottom Bar:** Writable, Smart Insert, 1818: 27: 43865, 194M of 289M.



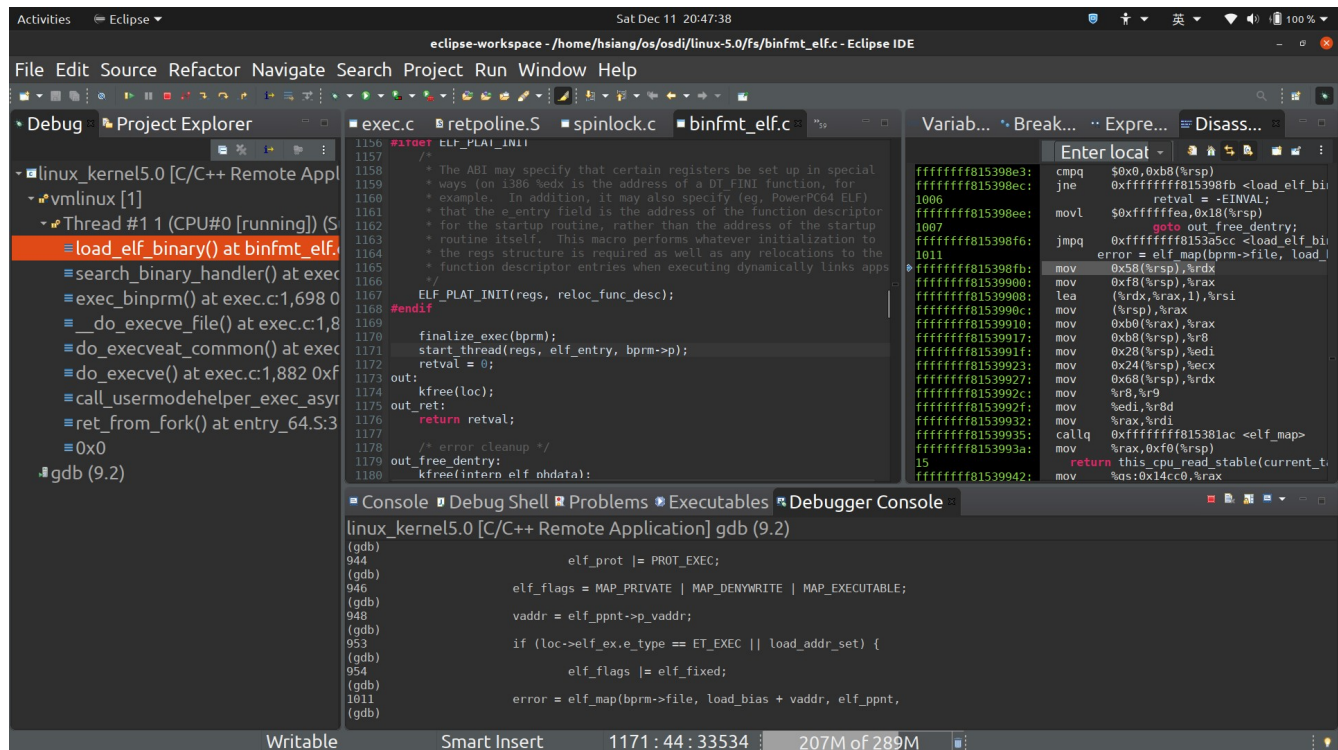
裡面去調用 `search_binary_handler()`:



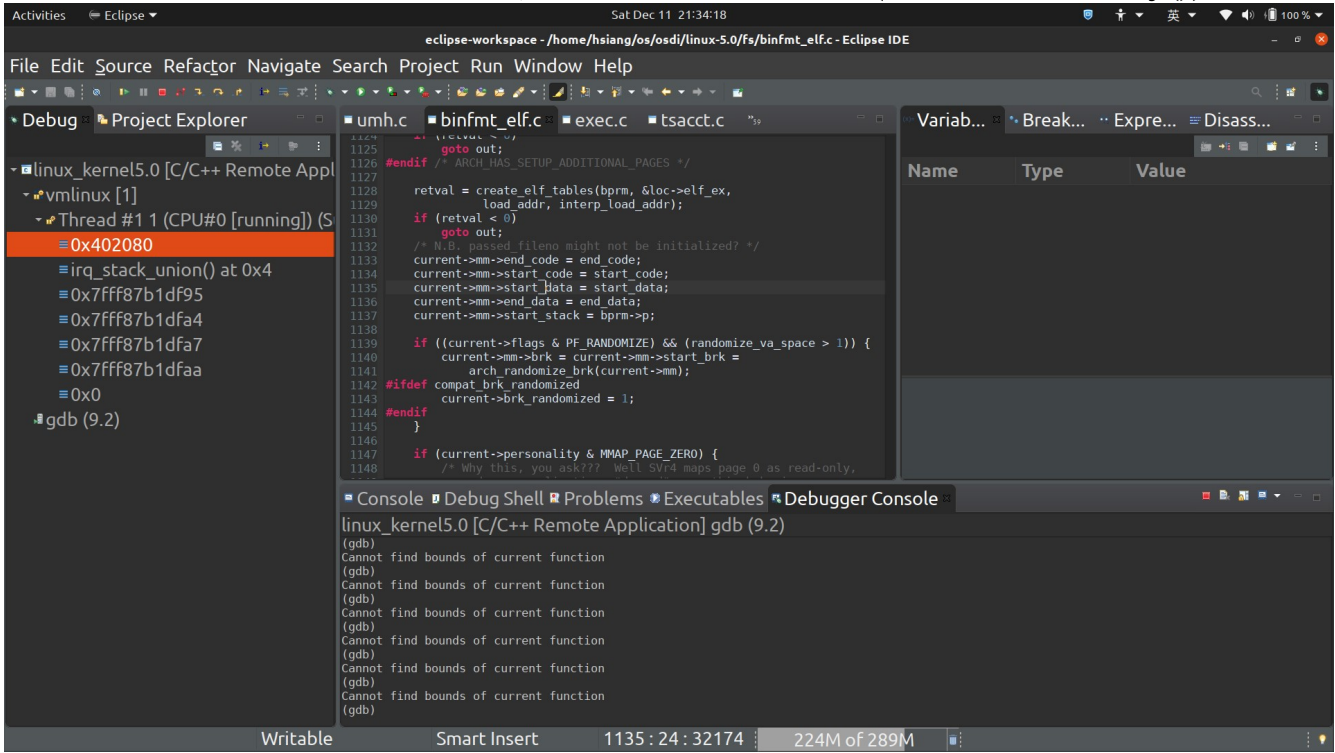
裡面有一個 load_binary():會去調用到 load_elf_binary():



execve 系列 lib 就是在這裡載入的



3. 不會立即載入執行擋到記憶體，只修改 mm_struct: (在 load_elf_binary()):



然後 `start_thread()`:

