Open RazzmatazZ Laboratory (OrzLab)
http://orzlab.blogspot.com/

深入淺 出 Hello World – Part III

核心 底處三 萬呎

Mar 25, 2007

Jim Huang(黃敬群 /**jserv**)
Email: <**jserv.tw@gmail.com**>
Blog: **http://blog.linux.org.tw/jserv/**

注意

- 本議程針對 x86 硬體架構，至於 ARM 與 MIPS 架構，請另行聯絡以作安排
- 簡報採用創意公用授權條款 (Creative Commons License: **Attribution-ShareAlike**) 發行
- 議程所用之軟體，依據個別授權方式發行
- 系統平台
  - Ubuntu feisty (development branch, **7.04**)
  - Linux kernel 2.6.20
  - gcc 4.1.2
  - glibc 2.5

地底三萬呎

- 地底三萬呎
  - 作者：朱少麟 / 著
  - 出版社：九歌
  - 出版日期： 2005 年 8 月 15 日
- 「閱讀【地底三萬呎】的經驗太奇異了，就像是搭乘自由落體一樣，跟著小說情境，高速跌落到一個深沉的世界裡，在那開滿了航手蘭，金縷馨，野谷仙子葵與倒地銀雪的河谷中，與主角群一起進入生命的深度探索，直到那無人敢逼視的，心靈最底層 ..... 隨著劇情，閱讀者竟然真的有墜毀的錯覺！」
- http://www.books.com.tw/exep/prod/booksfile.php?item=0010304023

核心底下三萬呎

hello.c
./hello

Anything can be hacked.

```
#include <stdio.h>
int main()
{
    printf("Hello World");
    return 42;
}
```

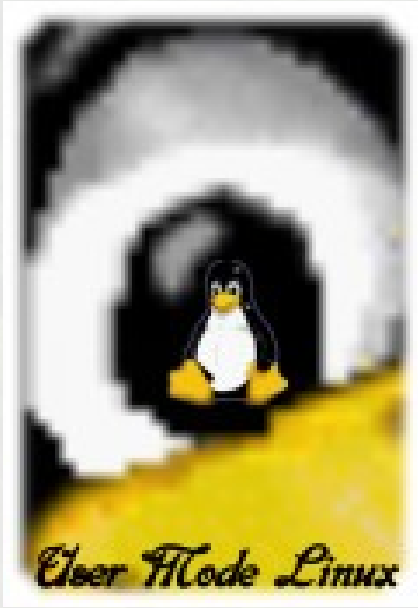
Linux
Kernel !

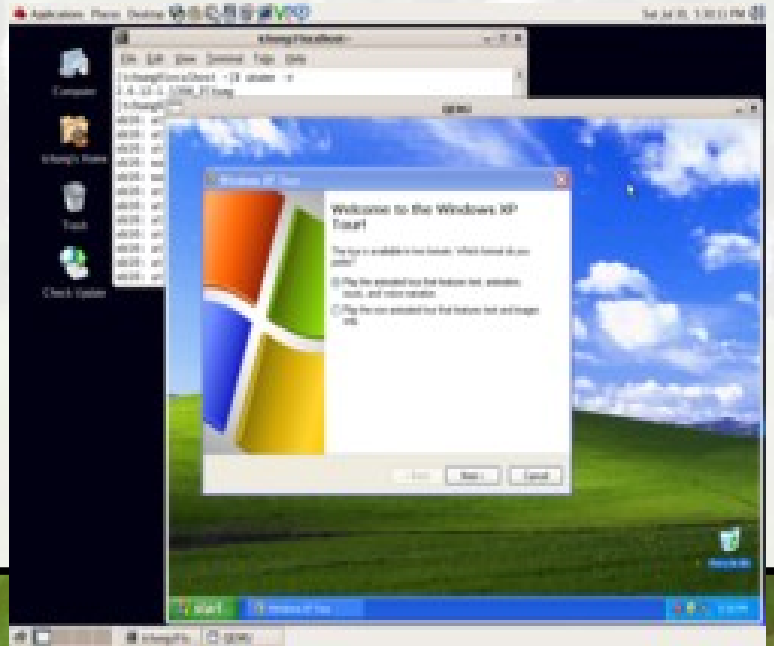
大綱

- 以 User-Mode-Linux 與 qemu 分析系統呼叫
- 探索 Linux 之 Program Loader (基礎篇)
- User/Kernel-space 互動

對抗核心的新武器

- User-Mode-Linux
  - 整合到 Linux 2.6 kernel tree
  - 虛擬架構 (ARCH=um)
- qemu
  - user emulation (instructions)
  - system emulation



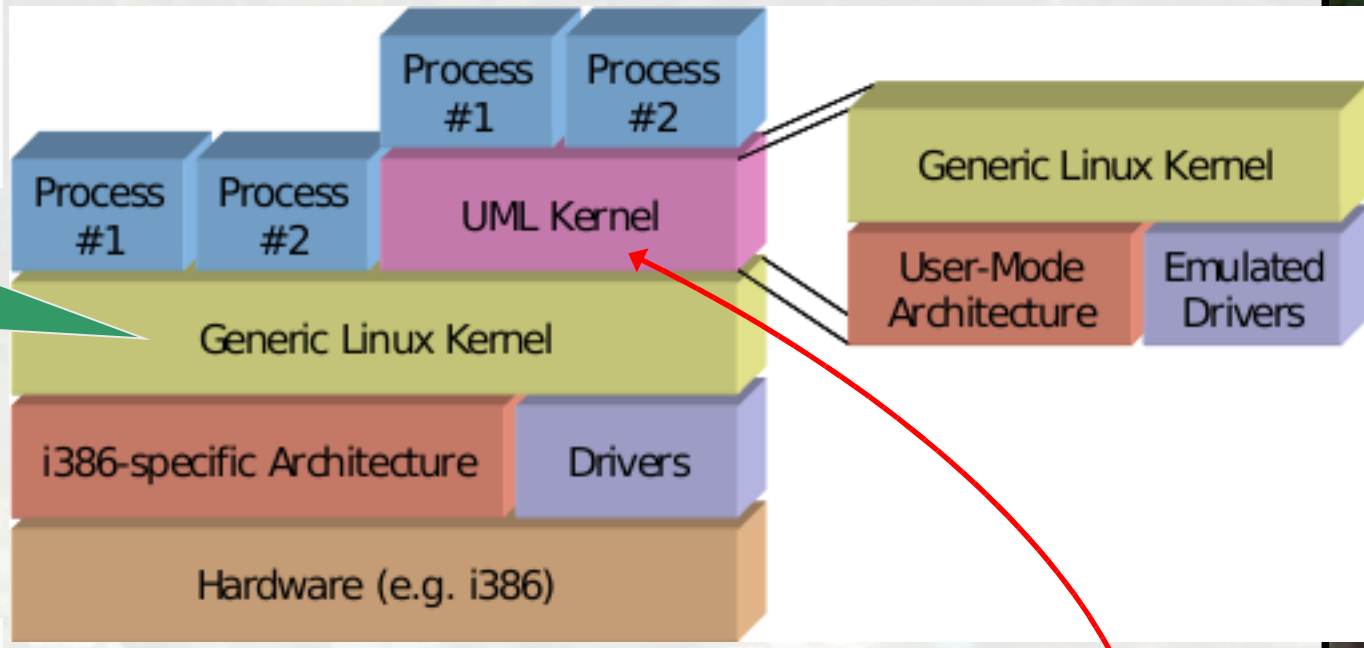


User-Mode Linux (1)

- 將 Linux Kernel 「移植」到 user-space
  - 修改的 "Kernel" 被視為一般的 Linux process 來執行
- 應用
  - 對與硬體無關的的程式作偵錯與安全測試
  - 檢驗 file system 的完整性與正確性
  - 在單機建構虛擬網路環境
  - 追蹤 Linux Kernel 大體流程，允許快速測試新的演算法或改進途徑
  - 完整的 Linux 教學環境

User-Mode Linux (2)

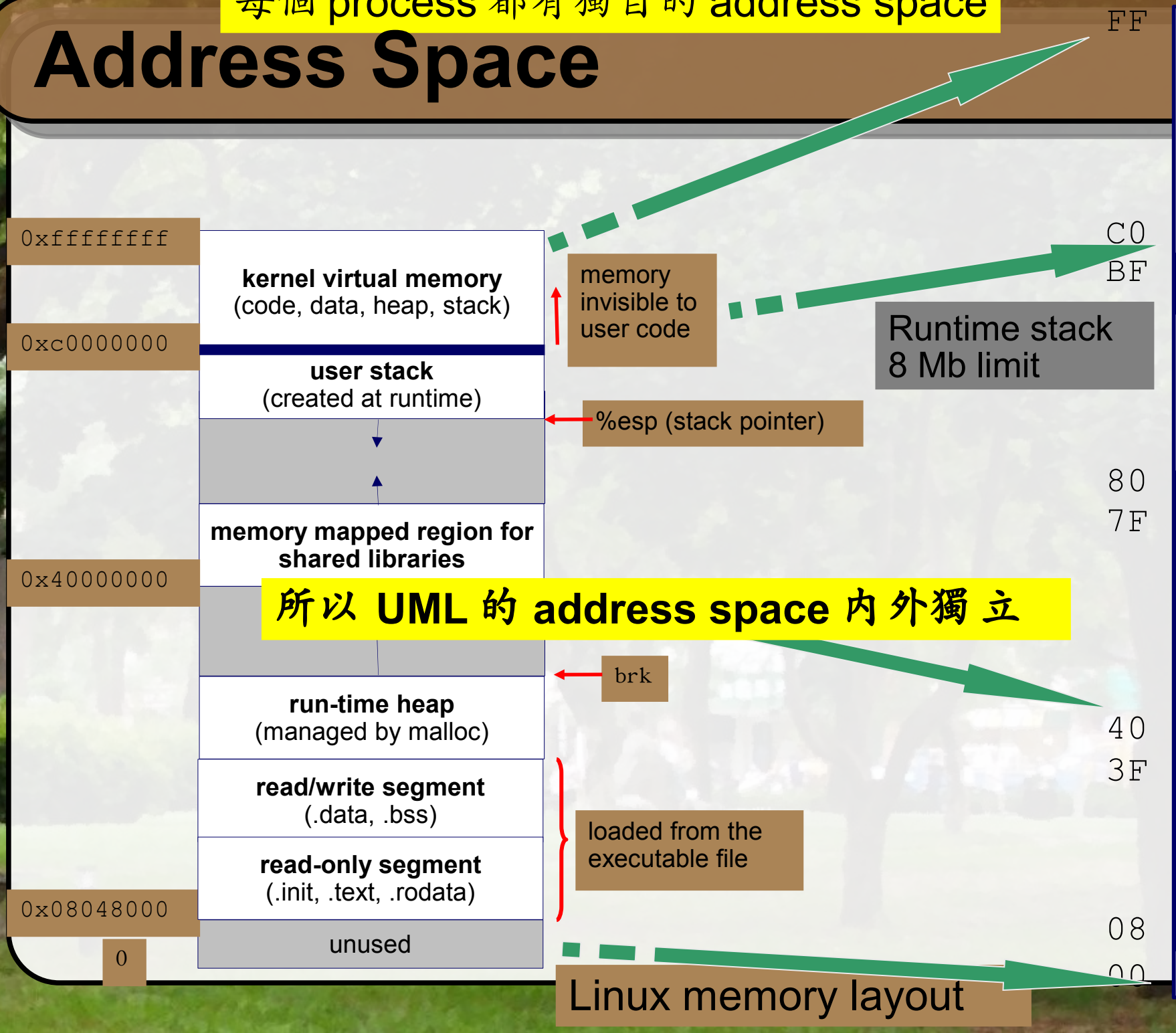
```
+----------+----------+----+
| Process 1 | Process 2 | ...|
+----------+----------+----+
|       Linux Kernel         |
+----------------------------+
|         Hardware           |
+----------------------------+
```




```
            +---------------+
            | Process 2 | ...|
+----------+---------------+
| Process 1 | User-Mode Linux|
+----------------------------+
|       Linux Kernel         |
+----------------------------+
|         Hardware           |
+----------------------------+
```

每個 process 都有獨自的 address space

Address Space



User-Mode Linux (3)

- 為了效能與偵錯需求， UML 引入 SKAS 機制
  - SKAS = Separate Kernel Address Space
  - 舊的模式稱為 TT (Trace Thread)： 依賴 ptrace
- SKAS 直接修改 host Linux kernel， 將原本的 TT 模式所需的 context switches 數量從 4 降到 2
- 較新的 kernel 多有支援

```
Checking that ptrace can change system call numbers...OK
Checking syscall emulation patch for ptrace...OK
Checking advanced syscall emulation patch for ptrace...OK
Checking for tmpfs mount on /dev/shm...OK
Checking PROT_EXEC mmap in /dev/shm/...OK
Checking for the skas3 patch in the host:
  - /proc/mm...not found
  - PTRACE_FAULTINFO...not found
  - PTRACE_LDT...not found
UML running in SKAS0 mode
Checking that ptrace can change system call numbers...OK
```

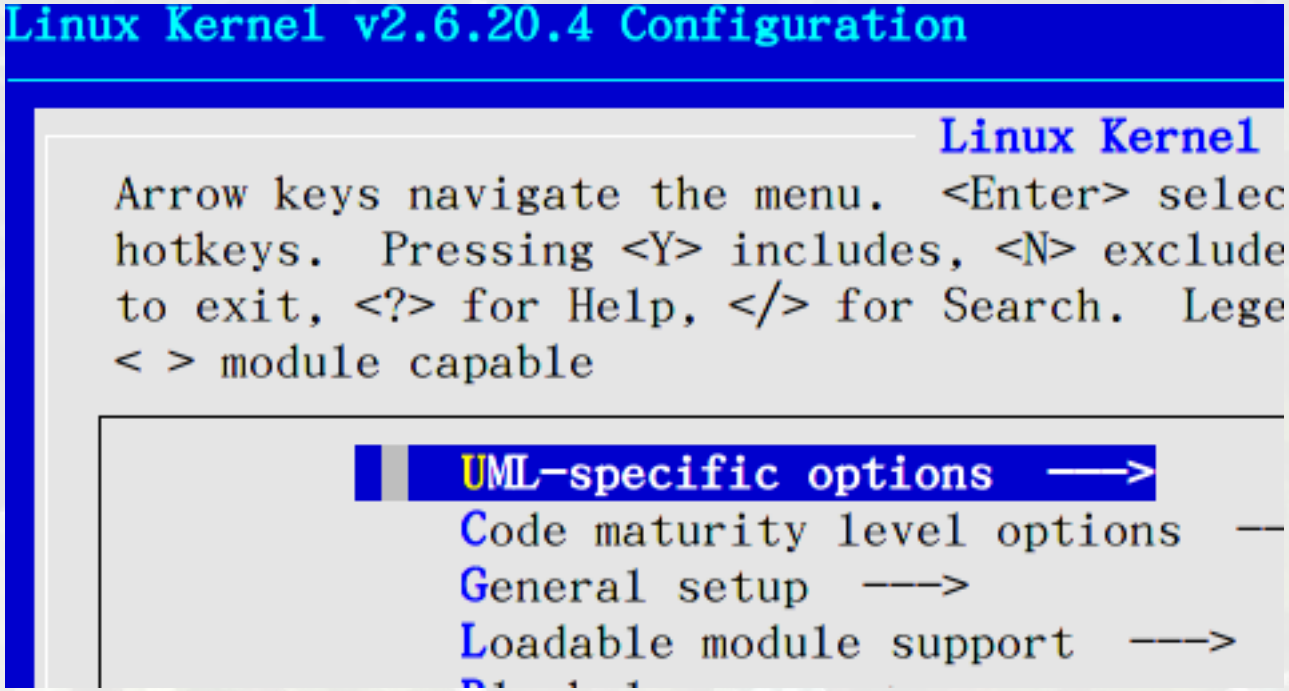
User-Mode Linux (4)

para-
[pærə-;pærə- | pærə-;pærə-]
<<字首>>
1 表示「旁、超、外、誤、不規則」的意思
2『醫學』表示「擬似、副」的意思
paratyphoid（副傷寒）

- 屬於 Para-virtualization 技術
  - 需要對 guest 核心作修改，與 qemu 不同
- 模擬的裝置： arch/um/drivers/*
- 檔案系統
  - 以單一檔案存在
  - UML Block Device => /dev/ubd?
  - 啟動時指定： ./linux ubd0=root_file_system
  - 可對應到實體裝置： ubd0=/dev/fd0
- COW (Copy-On-Write)
- [參考] Jserv's blog: 透過 User-Mode-Linux 來學習核心設計 (1/2)

User-Mode Linux (5)

~/uml/linux-2.6.20.4$ **make menuconfig ARCH=um**



~/uml/linux-2.6.20.4$ **make linux ARCH=um**

- 以 **linux2.6.20.4 (Apr23, 2007)** 爲例
- 不需要核心修改

User-Mode Linux (6)

~/uml/linux-2.6.20.4$ **cgdb ./linux**

```
153                     new_argv[argc + 1] = NULL;
154
155                     execvp(new_argv[0], new_argv);
156                     perror("execing with extended args");
157                     exit(1);
158             }
159     #endif
160
161 ->          linux_prog = argv[0];
162
163             set_stklim();
164
165             setup_env_path();
166
167             new_argv = malloc((argc + 1) * sizeof(char *));
168             if(new_argv == NULL){
169                     perror("Mallocing argv");
/home/jserv/uml/linux-2.6.20.4/arch/um/os-Linux/main.c
```

```
GNU gdb 6.6-debian
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i486-linux-gnu"...
Using host libthread_db library "/lib/tls/i686/cmov/libthread_db.so.1".
(tgdb)
```

CGDB is a curses-based interface to the GNU Debugger (GDB). The goal of CGDB is to be lightweight and responsive; not encumbered with unnecessary features.
http://cgdb.sourceforge.net/
* 內建類似 vim 的程式碼編輯功能

基本分析：
- 設定關鍵 **br eak po int**
- 單步執行
- **Cal l Gra ph**

b **start_kernel**
b **panic**
run **ubd0**=rootfs mem=128M umid=ubuntu

UMID (Unique Machine ID)

```
476         cpu_set(cpu, cpu_possible_map);
477  }
478
479  void __init __attribute__((weak)) smp_setup_proce
480  {
481  }
482
483  asmlinkage void __init start_kernel(void)
484 ->{
485         char * command_line;
486         extern struct kernel_param __start___para
487
488         smp_setup_processor_id();
489
490         /*
491          * Need to run as early as possible, to initialize the
492          * lockdep hash:
```

```
jserv@venux:~/um1/linux-2.6.20.4$ pstree | grep -A10 linux
      |-cmd---rxvt-unicode---bash-+-cgdb---gdb---linux
      |                           `-stardict
      |-cmd---rxvt-unicode---bash
      |-5*[dbus-daemon]
      |-dbus-launch
      |-dd
      |-events/0
      |-gconfd-2
      |-4*[getty]
      |-hald---hald-runner-+-hald-addon-acpi
      |                    |-hald-addon-cpuf
jserv@venux:~/um1/linux-2.6.20.4$
```

pstree

```
/home/jserv/um1/linux-2.6.20.4/init/main.c
Make breakpoint pending on future shared library load? (y or [n]) n
(tgdb) run ubd0=/opt/src/ubuntu-root mem=64M
Starting program: /home/jserv/um1/linux-2.6.20.4/linux ubd0=/opt/src/ubuntu-root mem=64M
Checking that ptrace can change system call numbers...OK
Checking syscall emulation patch for ptrace...OK
Checking advanced syscall emulation patch for ptrace...OK
Checking for tmpfs mount on /dev/shm...OK
Checking PROT_EXEC mmap in /dev/shm/...OK
Checking for the skas3 patch in the host:
  - /proc/mm...not found
  - PTRACE_FAULTINFO...not found
  - PTRACE_LDT...not found
UML running in SKAS0 mode

Breakpoint 1, start_kernel () at init/main.c:484
(tgdb)
```

在 **start_kernel** 之前的前置動作，參考 **arch/um/main.c**

User-Mode Linux (7)

```
~/uml/linux-2.6.20.4$ ./linux ubd0=rootfs mem=128M umid=ubunt
```

UML

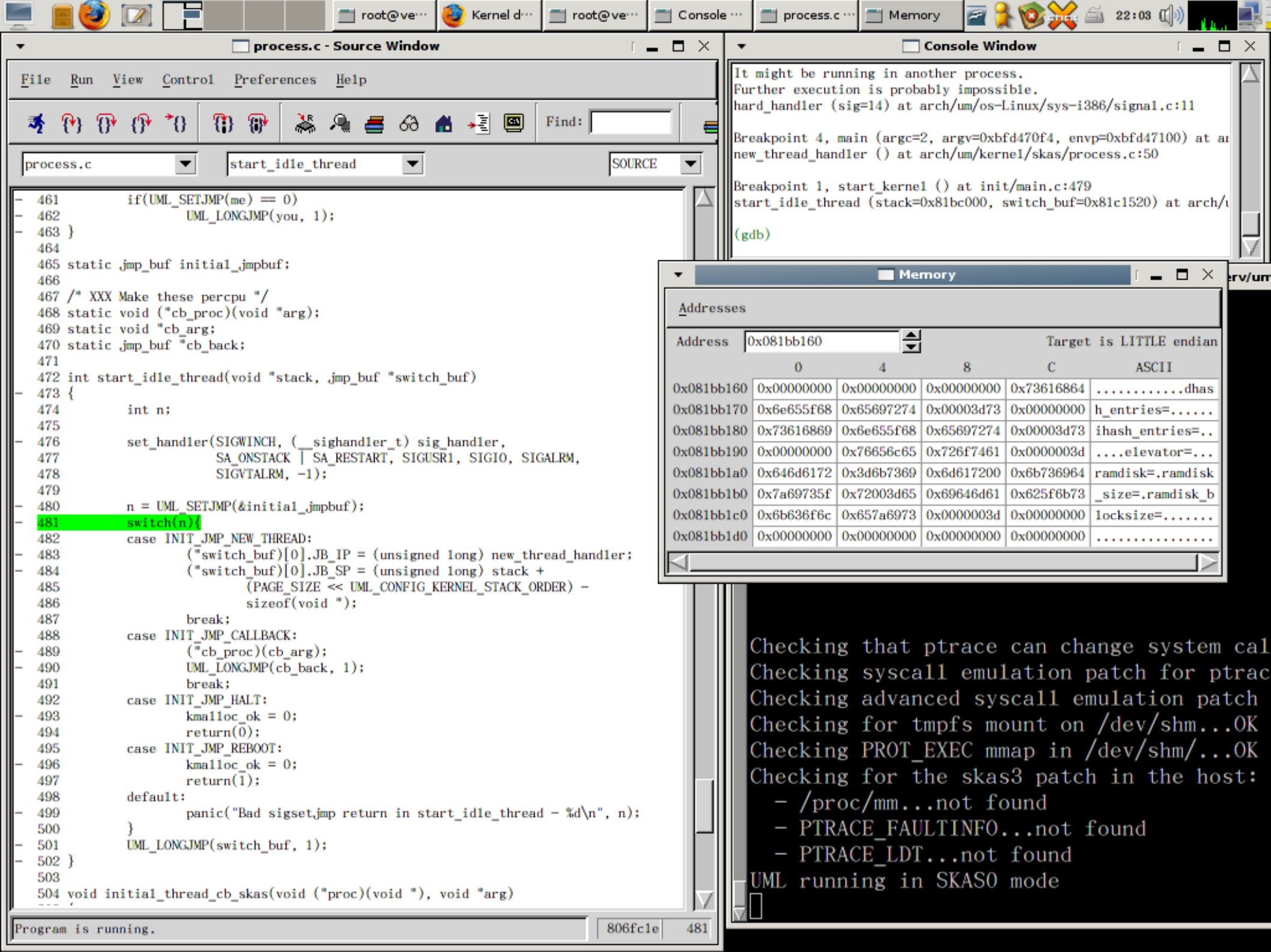
```
~/uml/linux-2.6.20.4$ cat ~/.uml/ubuntu/pid
~/uml/linux-2.6.20.4$ pstree | grep linux
    |-cmd---rxvt-unicode---bash-+-bash---linux---10*[linux]
~/uml/linux-2.6.20.4$ gdb ./linux `cat ~/.uml/ubuntu/pid`
```

Host

- exec syscall ： sys_execve ，將 ELF image 取代原本的 process ，也就是 current process
- do_execve 負責執行 sys_execve 主要工作

```
(gdb) b do_execve
(gdb) c
Continuing

Breakpoint 1, do_execve (filename=0xfc6e000 “/sbib/modprobe”,
argv=0xbf986834, envp=0x8058460, regs=0x820ea38) at
fs/exec.c:1127
(gdb) delete 1
(gdb) c
```

Host


root@ve···
Kernel d···
root@ve···
Console ···
process.c ···
Memory
22:03
process.c - Source Window
File Run View Control Preferences Help
Find:
process.c
start_idle_thread
SOURCE
461 if(UML_SETJMP(me) == 0)
462 UML_LONGJMP(you, 1);
463 }
464
465 static jmp_buf initial_jmpbuf;
466
467 /* XXX Make these percpu */
468 static void (*cb_proc)(void *arg);
469 static void *cb_arg;
470 static jmp_buf *cb_back;
471
472 int start_idle_thread(void *stack, jmp_buf *switch_buf)
473 {
474 int n;
475
476 set_handler(SIGWINCH, (__sighandler_t) sig_handler,
477 SA_ONSTACK | SA_RESTART, SIGUSR1, SIGIO, SIGALRM,
478 SIGVTALRM, -1);
479
480 n = UML_SETJMP(&initial_jmpbuf);
481 switch(n){
482 case INIT_JMP_NEW_THREAD:
483 (*switch_buf)[0].JB_IP = (unsigned long) new_thread_handler;
484 (*switch_buf)[0].JB_SP = (unsigned long) stack +
485 (PAGE_SIZE << UML_CONFIG_KERNEL_STACK_ORDER) -
486 sizeof(void *);
487 break;
488 case INIT_JMP_CALLBACK:
489 (*cb_proc)(cb_arg);
490 UML_LONGJMP(cb_back, 1);
491 break;
492 case INIT_JMP_HALT:
493 kmalloc_ok = 0;
494 return(0);
495 case INIT_JMP_REBOOT:
496 kmalloc_ok = 0;
497 return(1);
498 default:
499 panic("Bad sigsetjmp return in start_idle_thread - %d\n", n);
500 }
501 UML_LONGJMP(switch_buf, 1);
502 }
503
504 void initial_thread_cb_skas(void (*proc)(void *), void *arg)
Program is running.
806fc1e
481
Console Window
It might be running in another process.
Further execution is probably impossible.
hard_handler (sig=14) at arch/um/os-Linux/sys-i386/signal.c:11
Breakpoint 4, main (argc=2, argv=0xbfd470f4, envp=0xbfd47100) at ar
new_thread_handler () at arch/um/kernel/skas/process.c:50
Breakpoint 1, start_kernel () at init/main.c:479
start_idle_thread (stack=0x81bc000, switch_buf=0x81c1520) at arch/u
(gdb)
Memory
Addresses
Address 0x081bb160
Target is LITTLE endian
0 4 8 C ASCII
0x081bb160 0x00000000 0x00000000 0x00000000 0x73616864 .....dhas
0x081bb170 0x6e655f68 0x65697274 0x00003d73 0x00000000 h_entries=.....
0x081bb180 0x73616869 0x6e655f68 0x65697274 0x00003d73 ihash_entries=..
0x081bb190 0x00000000 0x76656c65 0x726f7461 0x0000003d ....elevator=...
0x081bb1a0 0x646d6172 0x3d6b7369 0x6d617200 0x6b736964 ramdisk=.ramdisk
0x081bb1b0 0x7a69735f 0x72003d65 0x69646d61 0x625f6b73 _size=.ramdisk_b
0x081bb1c0 0x6b636f6c 0x657a6973 0x0000003d 0x00000000 locksize=.....
0x081bb1d0 0x00000000 0x00000000 0x00000000 0x00000000 .....
Checking that ptrace can change system cal
Checking syscall emulation patch for ptrac
Checking advanced syscall emulation patch
Checking for tmpfs mount on /dev/shm...OK
Checking PROT_EXEC mmap in /dev/shm/...OK
Checking for the skas3 patch in the host:
- /proc/mm...not found
- PTRACE_FAULTINFO...not found
- PTRACE_LDT...not found
UML running in SKAS0 mode

Qemu (1)

- 快速的模擬器
  - Portable dynamic translator
- 完整系統模擬
  - instruction sets + processor + peripherals

    硬體平台： x86, x86_64, ppc, arm, sparc, mips
  - 指定特定機器

    qemu-system-arm -M ?
  - i386 與 x86_64 系統模擬可透過 kqemu 核心模組加速
- 兩種模擬模式：
  - User
  - System
- 提供 gdb stub ，可配合 gdb 作系統分析

Qemu (2)

兩種執行模式

- User mode emulation： 可執行非原生架構之應用程式
  支援： x86, ppc, arm, sparc, mips
- System emulation
  - qemu linux.img
  - 也可分別指定 kernel image、 initrd， 及相關參數
- 以 xscale PXA27x 為例 ...

使用 **target** 的 **ld-linux.so.2**

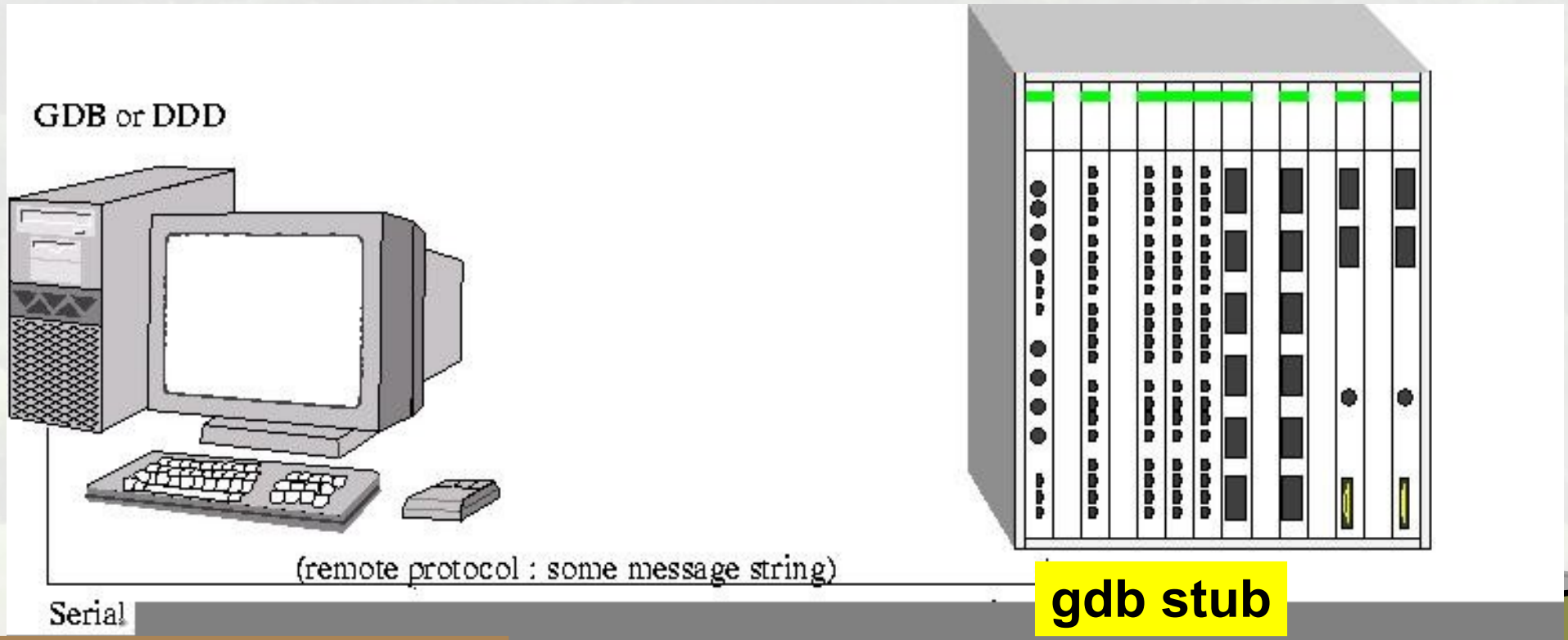
```
~/poky/build/tmp$ file ./rootfs/bin/busybox
./rootfs/bin/busybox: ELF 32-bit LSB executable, ARM, version 1 (ARM),
for GNU/Linux 2.4.0, dynamically linked (uses shared libs), for GNU/Linux
2.4.0, stripped
~/poky/build/tmp$ ./qemu-arm ./rootfs/lib/ld-linux.so.2 \
--library-path ./rootfs/lib ./rootfs/bin/busybox uname -a
Linux venux 2.6.20-12-generic #2 SMP Sun Mar 18 03:07:14 UTC 2007
armv5tel unknown
```

Processor 變成 **armv5te (Xscale)**

Qemu (3)

gdb stub

- 考慮在 system emulation 模式下，該如何喚起 gdb？
- Remote Debugging： gdb 可透過 serial line 或 TCP/IP 進行遠端除錯



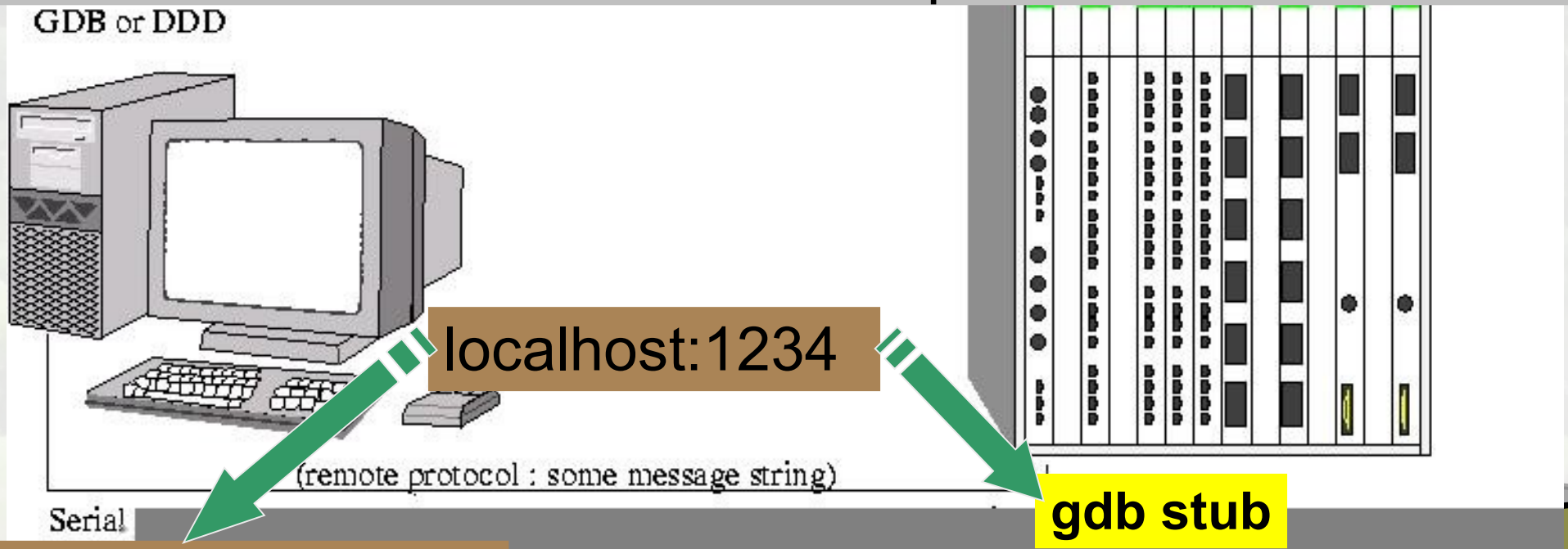
開發平台 (Host)
運作完整的 GDB

Qemu 所模擬的機器

Qemu (4)

gdb stub： 透過 **TCP/IP**

- (gdb) target remote localhost:1234
- qemu 執行選項:
  - **-s** Wait gdb connection to port 1234.
  - **-S** Do not start CPU at startup

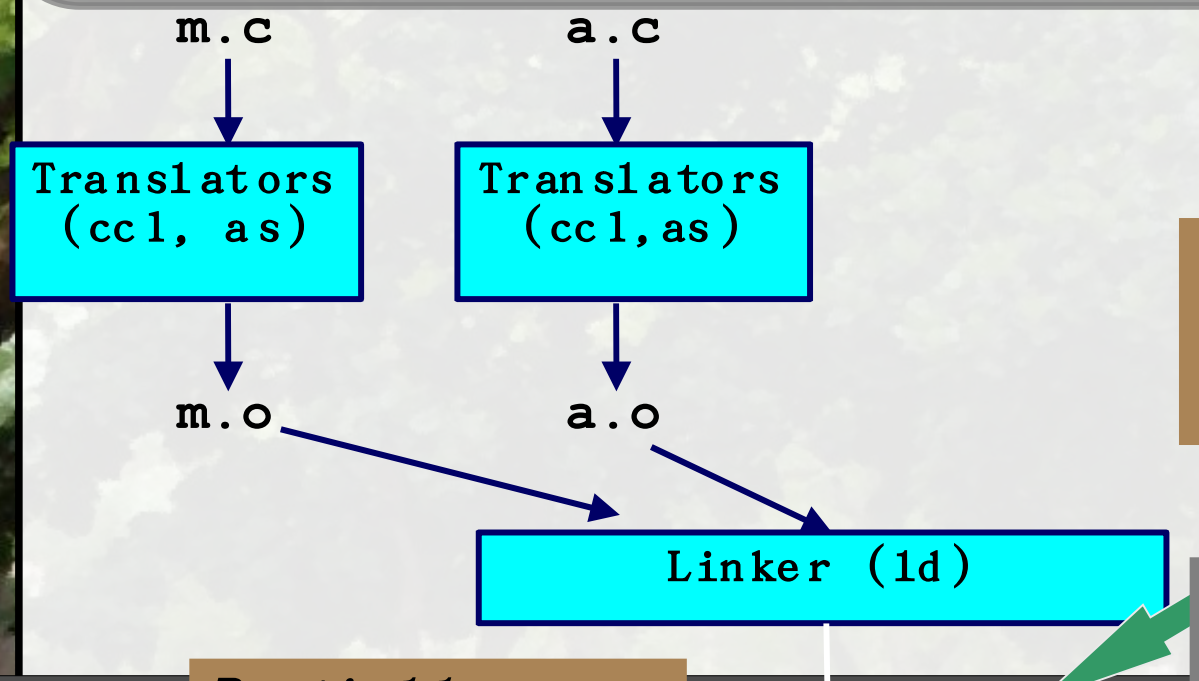



mc - venux:···
Remote pack···
root@venux: /
rxvt-unicode
TightVNC: QE···
10:00
root@venux: /
260 return do_fork(CLONE_VFORK | CLONE_VM | SIGCHLD, regs->ARM_sp, regs, 0, NULL, NULL);
261 }
262
263 /* sys_execve() executes a new program.
264 * This is called indirectly via a small wrapper
265 */
266 asmlinkage int sys_execve(char
267 char
268 ->{
269 int error;
270 char * filename;
271
272 filename = getname(file
273 error = PTR_ERR(filenam
274 if (IS_ERR(filename))
275 goto out;
276 error = do_execve(filen
/home/jserv/virt/linux-rp-2.6.20-r5/
at arch/arm/kernel/sys_arm.c:268
(tgdb) info breakpoints
Num Type Disp Enb Address
1 breakpoint keep y 0xc0027a
breakpoint already hit 17 ti
(tgdb) whatis start_kernel
type = void (void)
(tgdb) call printk("Hello World from
Breakpoint 1, sys_execve (filenamei=
e514) at arch/arm/kernel/sys_arm.c:2
The program being debugged stopped while in a function called from GDB.
When the function (malloc) is done executing, GDB will silently
stop (instead of continuing to evaluate the expression containing
the function call).
(tgdb)
以 Remote GDB 分析
TightVNC: QEMU
OrzLab
0x00140000-0x007f0000 : "Boot PROM Filesystem"
NAND device: Manufacturer ID: 0xec, Chip ID: 0xf1 (Samsung NAND 128MiB 3,3V 8-bi
t)
Scanning device for bad blocks
Creating 3 MTD partitions on "sharpsl-nand":
0x00000000-0x00700000 : "System Area"
0x00700000-0x04100000 : "Root Filesystem"
0x04100000-0x08000000 : "Home Filesystem"
input: Spitz Keyboard as /class/input/input0
power.c: Adding power management to input layer
input: Corgi Touchscreen as /class/input/input1
sa1100-rtc sa1100-rtc: rtc core: registered sa1100-rtc as rtc0
I2C: i2c-0: PXA I2C adapter
Registered led device: spitz:amber
Registered led device: spitz:green
TCP cubic registered
NET: Registered protocol family 1
NET: Registered protocol family 17
XScale iWMMXt coprocessor detected.
sa1100-rtc sa1100-rtc: setting the system clock to 2007-03-15 01:55:22 (11739237
22)
VFS: Mounted root (jffs2 filesystem) readonly.
Freeing init memory: 108K
INIT: version 2.86 booting
模擬 Sharp Zaurus PDA

探索 Linux 之 Program Loader

- ELF Image 的變化
- 執行時期的觀念 （基礎篇）
  - User-mode 與 Kernel-mode 的交錯

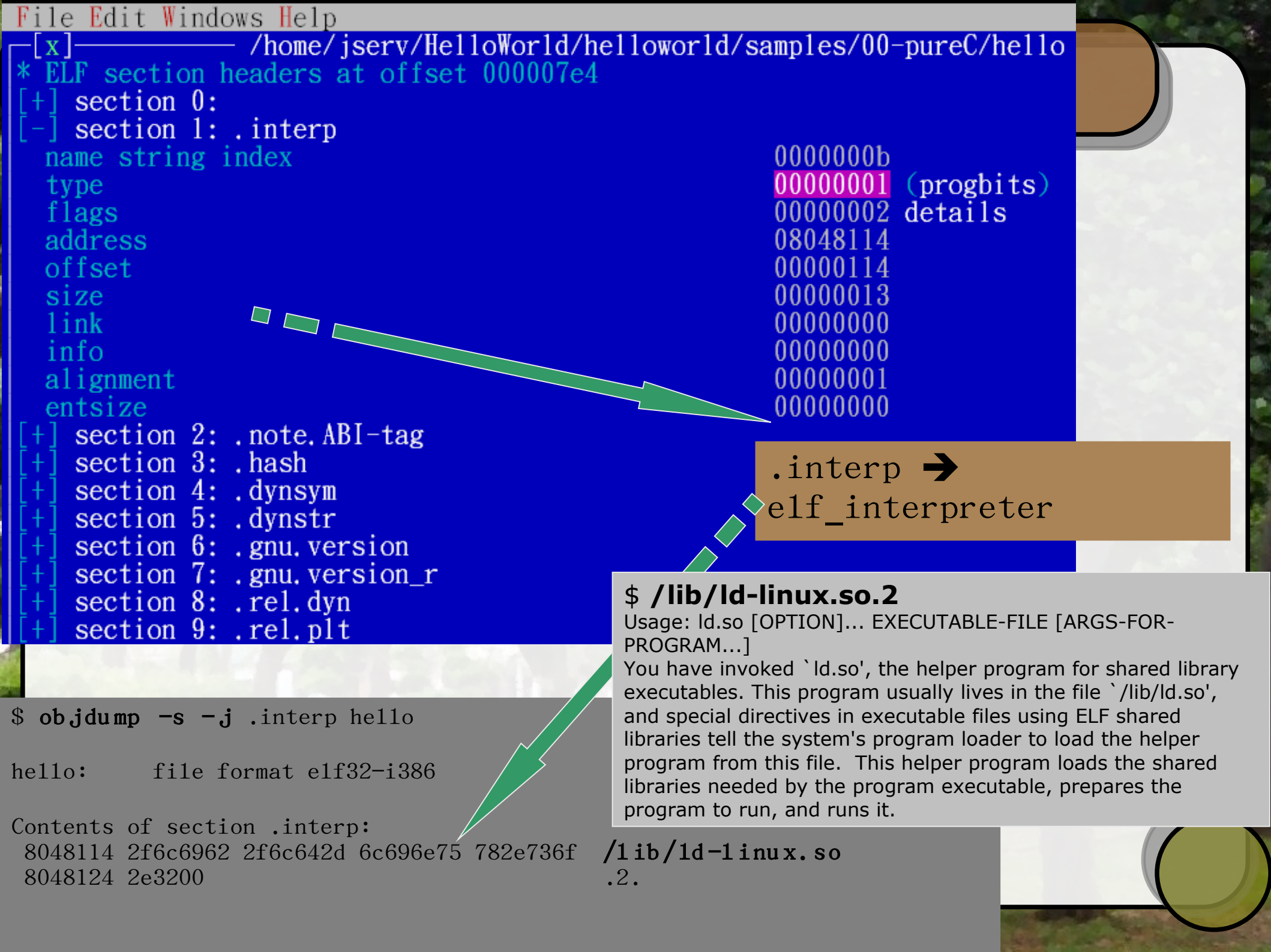
ELF Image 行爲



Shared Library
Dynamically relocatable object files

$ **ldd** hello
linux-gate.so.1 => (0xffffe000)
libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0xb7e62000)
/lib/ld-linux.so.2 (0xb7fad000)

libc.so functions called by m.c and a.c are loaded, linked, and (potentially) shared among processes.


File Edit Windows Help
[x] /home/jserv/HelloWorld/helloworld/samples/00-pureC/hello
* ELF section headers at offset 000007e4
[+] section 0:
[-] section 1: .interp
name string index 0000000b
type 00000001 (progbits)
flags 00000002 details
address 08048114
offset 00000114
size 00000013
link 00000000
info 00000000
alignment 00000001
entsize 00000000
[+] section 2: .note.ABI-tag
[+] section 3: .hash
[+] section 4: .dynsym
[+] section 5: .dynstr
[+] section 6: .gnu.version
[+] section 7: .gnu.version_r
[+] section 8: .rel.dyn
[+] section 9: .rel.plt
.interp ➔ elf_interpreter
$ /lib/ld-linux.so.2
Usage: ld.so [OPTION]... EXECUTABLE-FILE [ARGS-FOR-PROGRAM...]
You have invoked `ld.so', the helper program for shared library executables. This program usually lives in the file `/lib/ld.so', and special directives in executable files using ELF shared libraries tell the system's program loader to load the helper program from this file. This helper program loads the shared libraries needed by the program executable, prepares the program to run, and runs it.
$ objdump -s -j .interp hello
hello: file format elf32-i386
Contents of section .interp:
8048114 2f6c6962 2f6c642d 6c696e75 782e736f /lib/ld-linux.so
8048124 2e3200 .2.

Linux Program Loader (1)

- (User-Mode)
  - 使用者在 shell 執行外部程式
  - shell 以 fork+exec syscall 方式執行外部程式
  - 透過 int 0x80 軟體中斷觸發 kernel 的 exec syscall 服務 (sys_execvp)

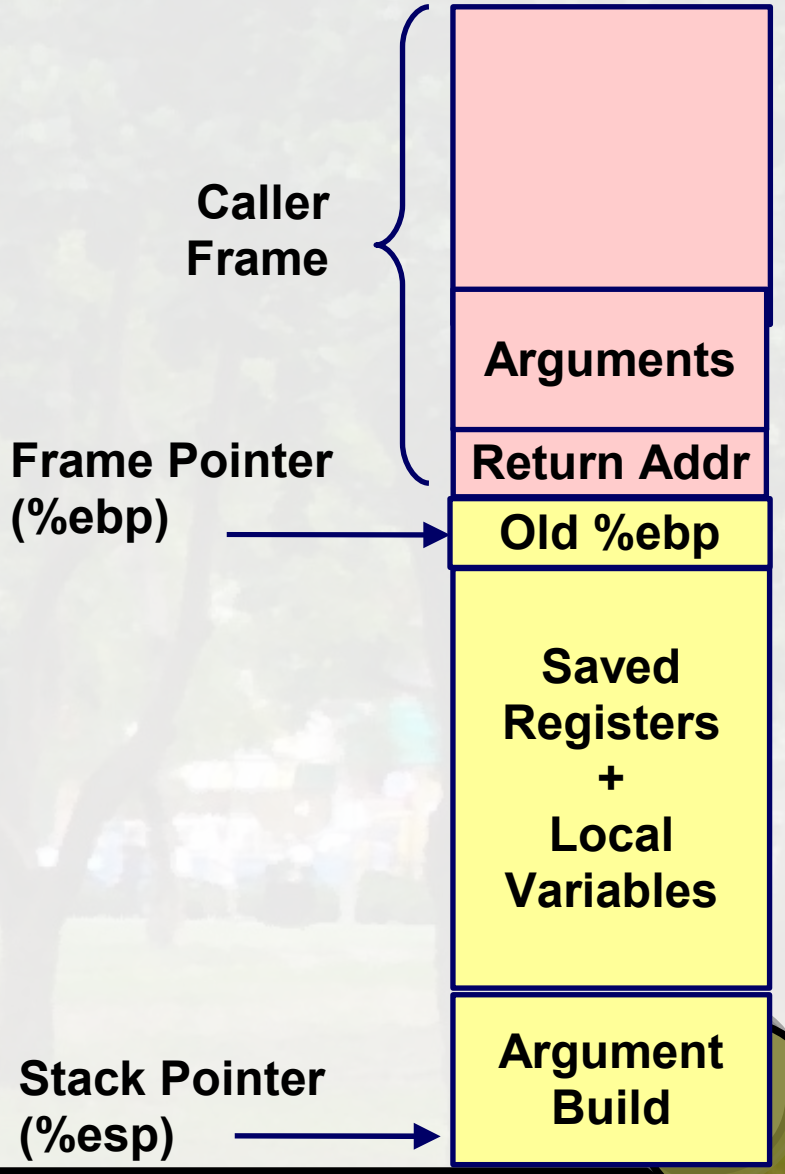
Linux Program Loader (2)

- (Kernel-Mode)
  - exec syscall 執行 ELF Loader， 載入與建立 process image (ELF image)
  - Program loader 找到 PT_INTERP segment。
  - Program loader 將 PT_LOAD segment mapping 為新的 text/data segment
    - text segment 由虛擬位址 0x0804_8000 開始， data segment 緊接其後
  - Program loader 呼叫 interpreter loader 將 program interpreter (ld.so) 載入，並 mapping 到 process memory
    - interpreter 的 text segment 由虛擬位址 0x4000_0000 開始， interpreter 的 data segment 緊接其後

ELF / .interp ：
elf_interpreter

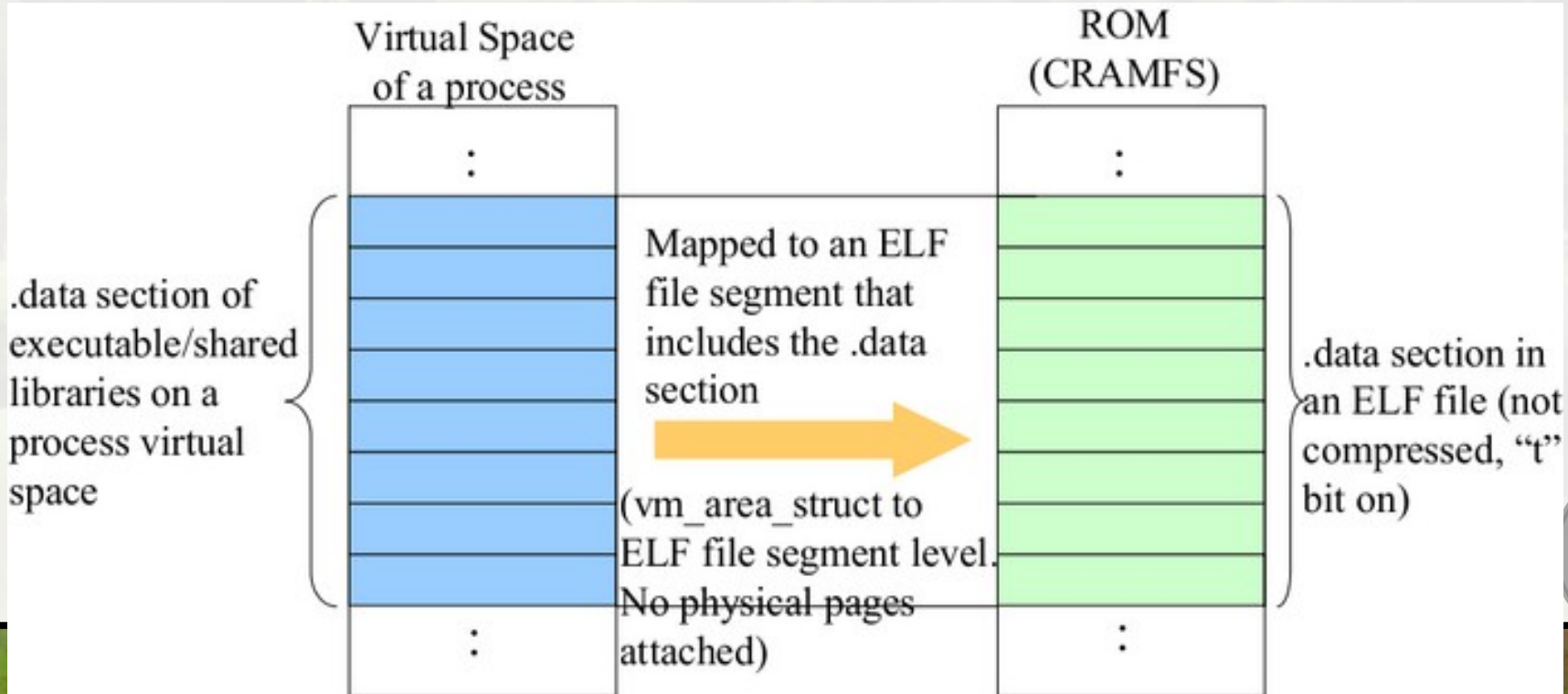
Linux Program Loader (3)

- (Kernel-Mode)
  - Program loader 將 BSS segment 準備好
  - Program loader 將 process 的 register %eip (user-mode) 修改為 program interpreter 的進入點，並將 %esp 設定為 user mode 的 stack



Linux Program Loader (4)

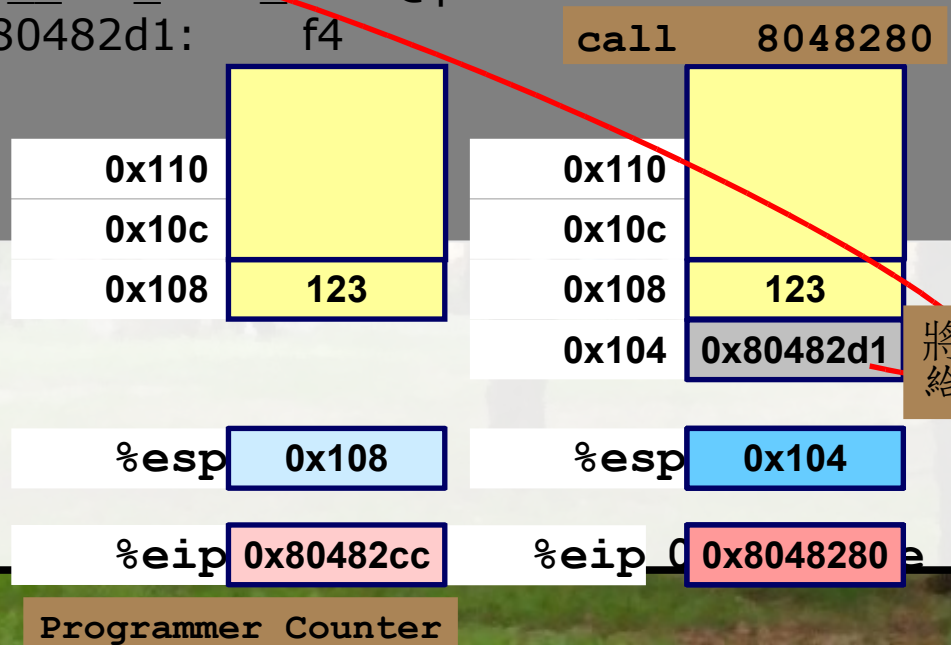
- (User-Mode)
  - Program interpreter 會找到 process 所需的 shared library(名稱與路徑)
  - Program interpreter 透過 mmap(2)， 將 shared library 予以 mapping 到 process memory， 以完成整個 Process Image 的建立



Linux Program Loader (5)

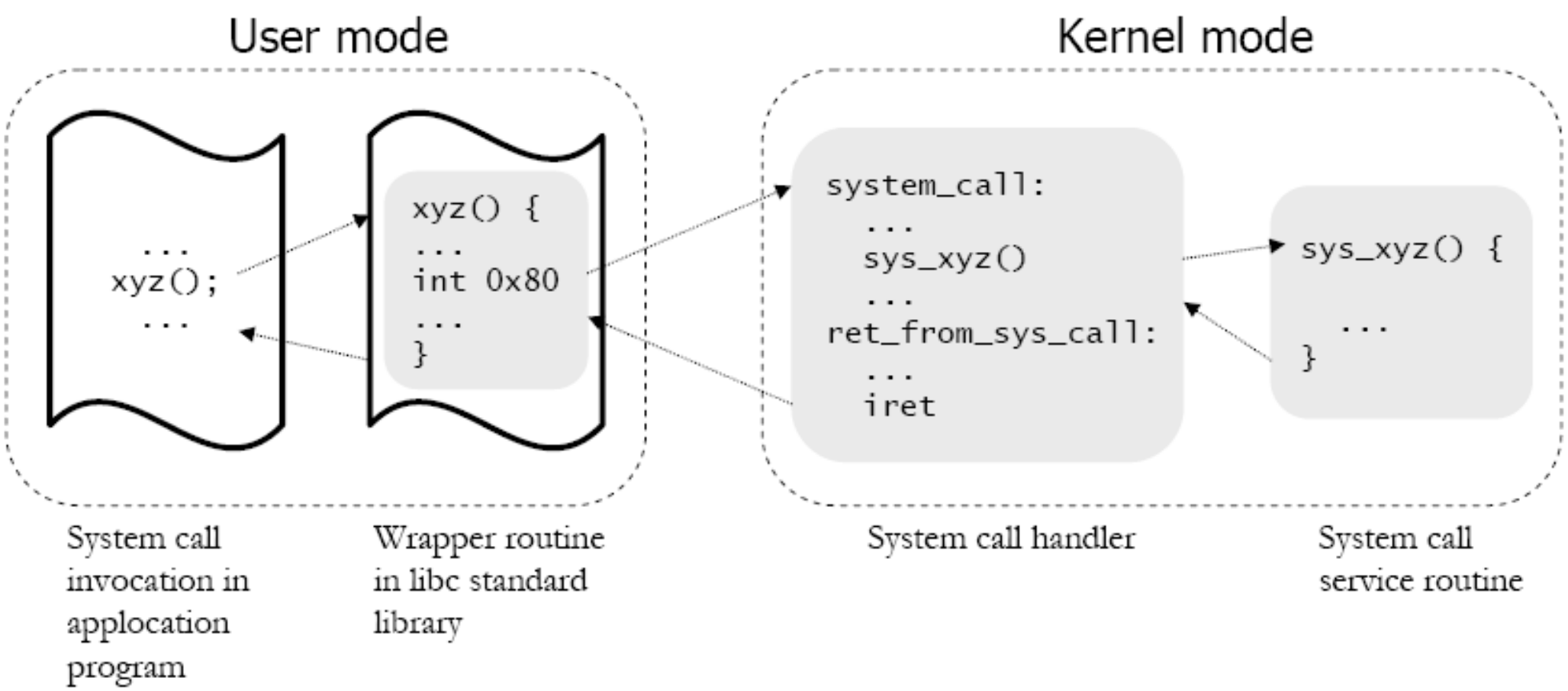
- 更新 Shared library 的符號表
  - Program interpreter 執行 x86 jump 動作，到 process 的進入點
    - 紀錄於 ELF header 的 entry point
  - 程式正式執行

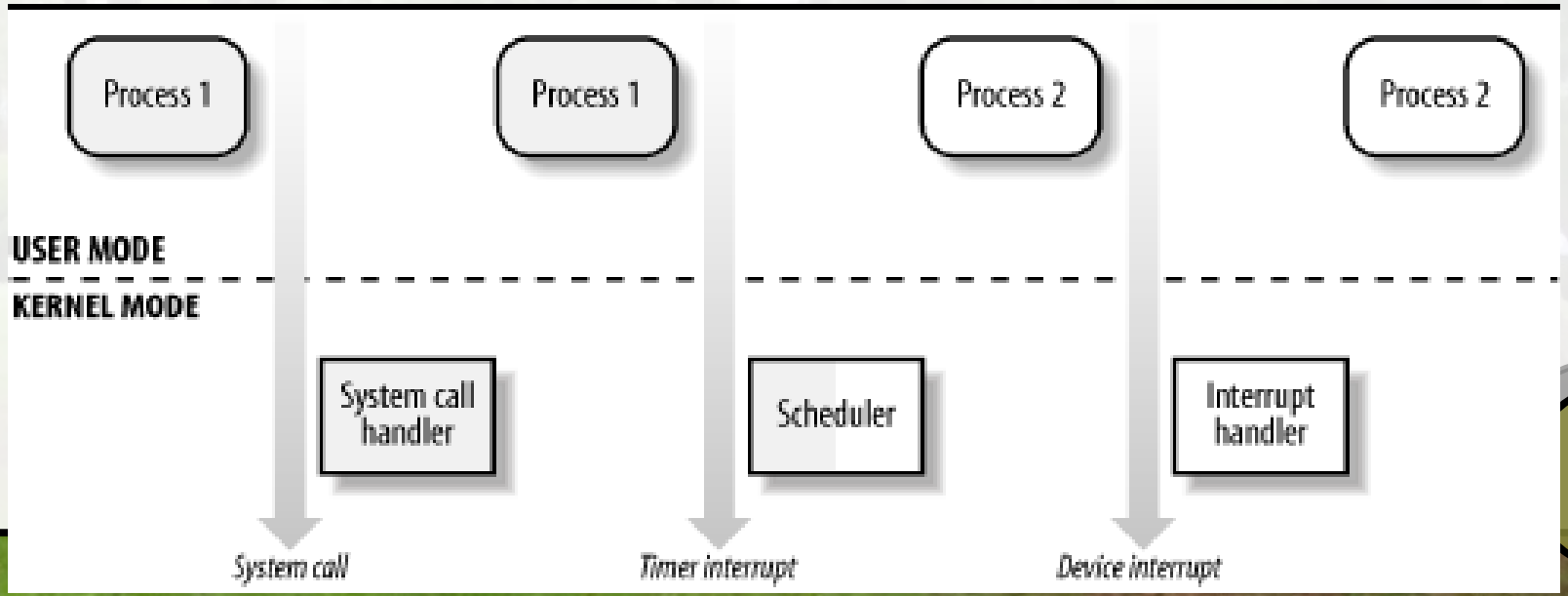
```
Disassembly of section .text:

080482b0 <_start>:
...
80482cc:      e8 af ff ff ff      call   8048280
<__libc_start_main@plt>
 80482d1:      f4
```



User/Kernel Mode 互動

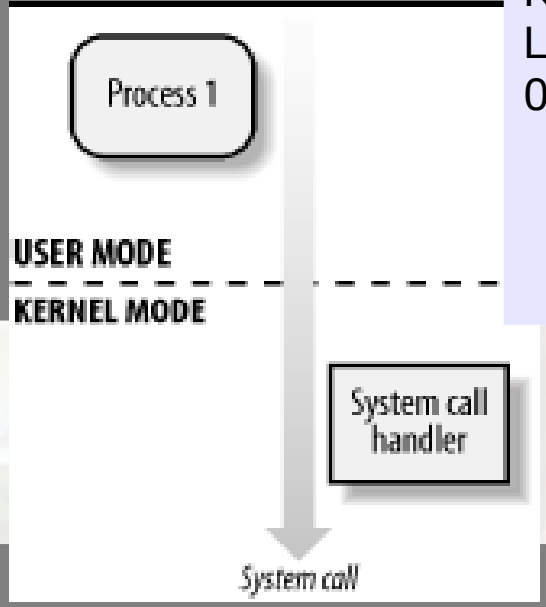
- 回顧系統呼叫
  - SCI (System Call Interface)
- 執行時期的系統呼叫


User mode
Kernel mode
...
xyz();
...
xyz() {
...
int 0x80
...
}
system_call:
...
sys_xyz()
...
ret_from_sys_call:
...
iret
sys_xyz() {
...
}
System call invocation in applocation program
Wrapper routine in libc standard library
System call handler
System call service routine


Process 1
Process 1
Process 2
Process 2
USER MODE
KERNEL MODE
System call handler
Scheduler
Interrupt handler
System call
Timer interrupt
Device interrupt

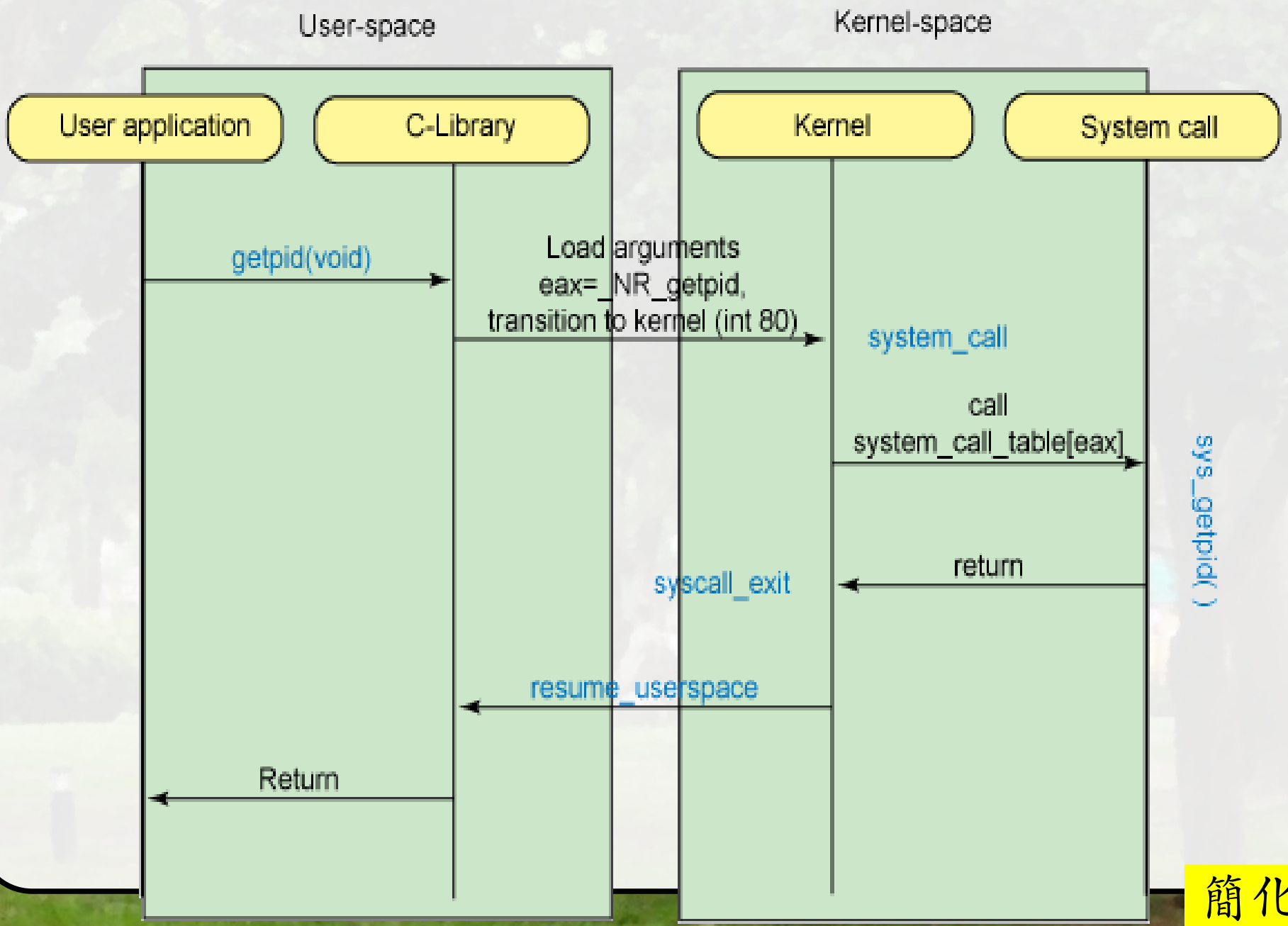
```
$ cat hello-loop.c
#include <stdio.h>
#include <unistd.h>
int main(int argc, char **argv)
{
        printf("Hello World!\n");
        while (1) {
                usleep(10000);
        }
        return 0;
}
$ ./hello-loop
Hello World!
```

```
$ pidof hello-loop
6987
$ gdb
(gdb) attach 6987
Attaching to process 6987
Reading symbols from
/home/jserv/HelloWorld/samples/hello-loop...done.
Using host libthread_db library
"/lib/tls/i686/cmov/libthread_db.so.1".
Reading symbols from
/lib/tls/i686/cmov/libc.so.6...done.
Loaded symbols for /lib/tls/i686/cmov/libc.so.6
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2
0xffffe410 in __kernel_vsyscall ()
```



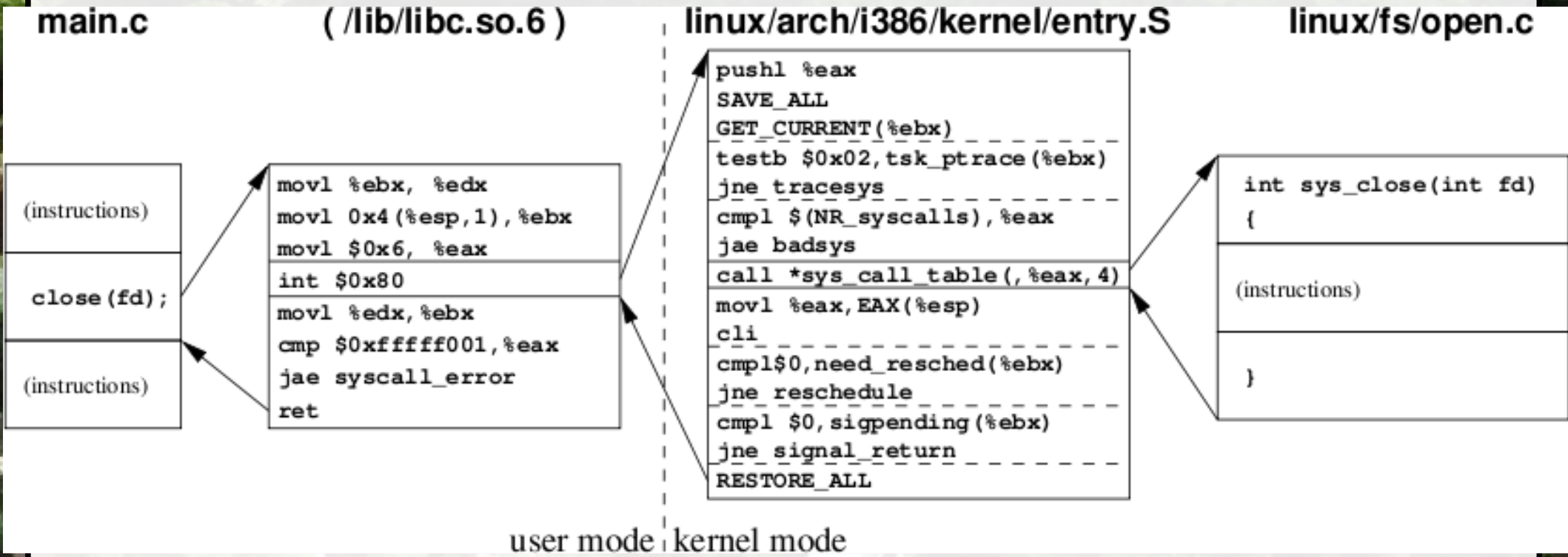

```
(gdb) bt
#0  0xffffe410 in __kernel_vsyscall ()
#1  0xb7e37ef0 in nanosleep () from /lib/tls/i686/cmov/libc.so.6
#2  0xb7e6f93a in usleep () from /lib/tls/i686/cmov/libc.so.6
#3  0x080483ad in main () at hello-loop.c:7
```

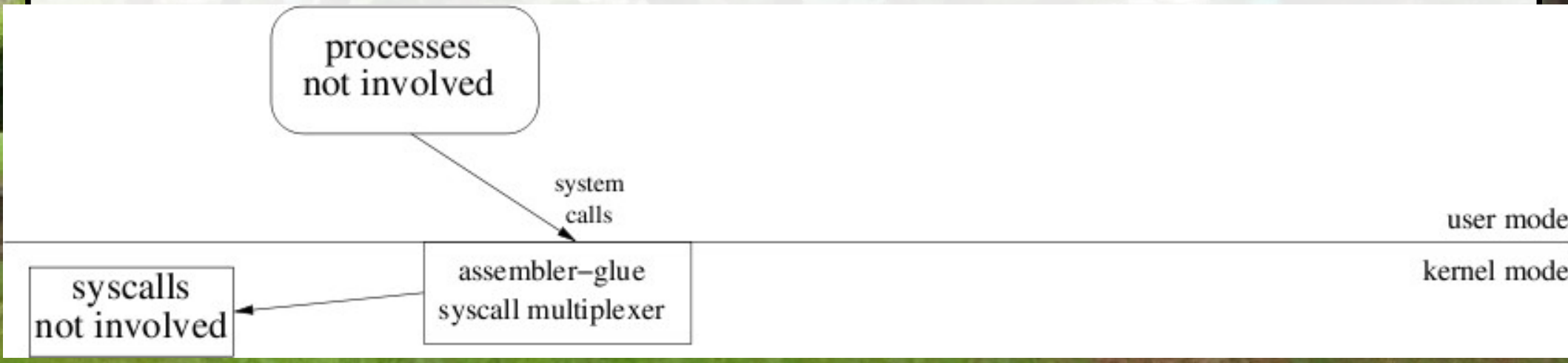
System Call Interface



簡化版

系統呼叫 (1)


main.c
(/lib/libc.so.6)
linux/arch/i386/kernel/entry.S
linux/fs/open.c
(instructions)
close(fd);
(instructions)
movl %ebx, %edx
movl 0x4(%esp,1),%ebx
movl $0x6, %eax
int $0x80
movl %edx,%ebx
cmp $0xfffff001,%eax
jae syscall_error
ret
pushl %eax
SAVE_ALL
GET_CURRENT(%ebx)
testb $0x02,tsk_ptrace(%ebx)
jne tracesys
cmpl $(NR_syscalls),%eax
jae badsys
call *sys_call_table(,%eax,4)
movl %eax,EAX(%esp)
cli
cmpl$0,need_resched(%ebx)
jne reschedule
cmpl $0,sigpending(%ebx)
jne signal_return
RESTORE_ALL
int sys_close(int fd)
{
(instructions)
}
user mode
kernel mode

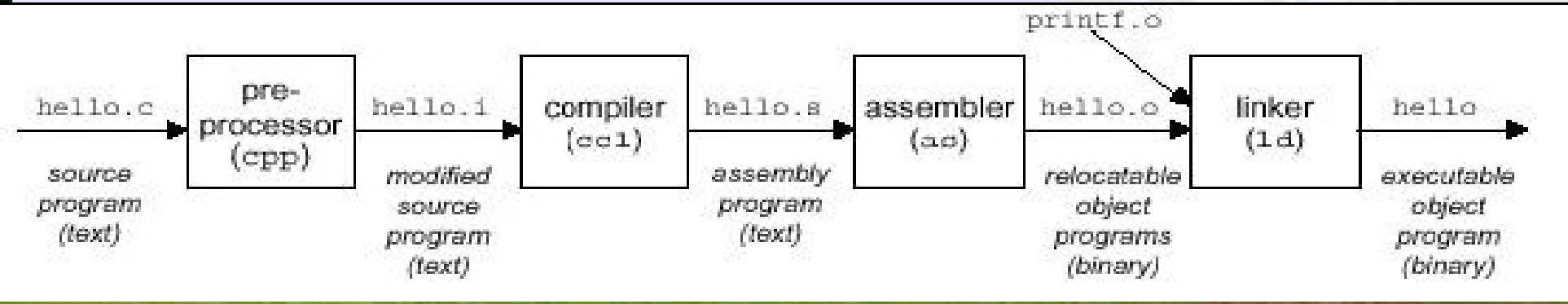

processes
not involved
system
calls
user mode
kernel mode
syscalls
not involved
assembler–glue
syscall multiplexer

回頭看 hello.c 的編譯過程

```
$ gcc -v -o hello hello.c
Using built-in specs.
Target: i486-linux-gnu
...
 /usr/lib/gcc/i486-linux-gnu/4.1.2/collect2 --eh-frame-hdr
-m elf_i386 -dynamic-linker /lib/ld-linux.so.2 -o hello
/usr/lib/gcc/i486-linux-gnu/4.1.2/../../../../lib/crt1.o
/usr/lib/gcc/i486-linux-gnu/4.1.2/../../../../lib/crti.o
/usr/lib/gcc/i486-linux-gnu/4.1.2/crtbegin.o -L
/usr/lib/gcc/i486-linux-gnu/4.1.2 -L/usr/lib/gcc/i486-
linux-gnu/4.1.2 -L/usr/lib/gcc/i486-linux-
gnu/4.1.2/../../../../lib -L/lib/../lib -L/usr/lib/../lib
/tmp/ccyj1YoV.o -lgcc --as-needed -lgcc_s --no-as-
needed -lc -lgcc --as-needed -lgcc_s --no-as-needed
/usr/lib/gcc/i486-linux-gnu/4.1.2/crtend.o
/usr/lib/gcc/i486-linux-gnu/4.1.2/../../../../lib/crtn.o
$ wc -c hello
6749 hello
```

crt*.o
C Runtime object files

```
$ cat hello.c
int main() {
   printf("Hello World\n");
   return 0;
}
$ strace ./hello
...

write(1, "Hello World\n", 12Hello World.) = 12
exit_group(0)                           = ?
Process 14211 detached
```



系統呼叫 (2)

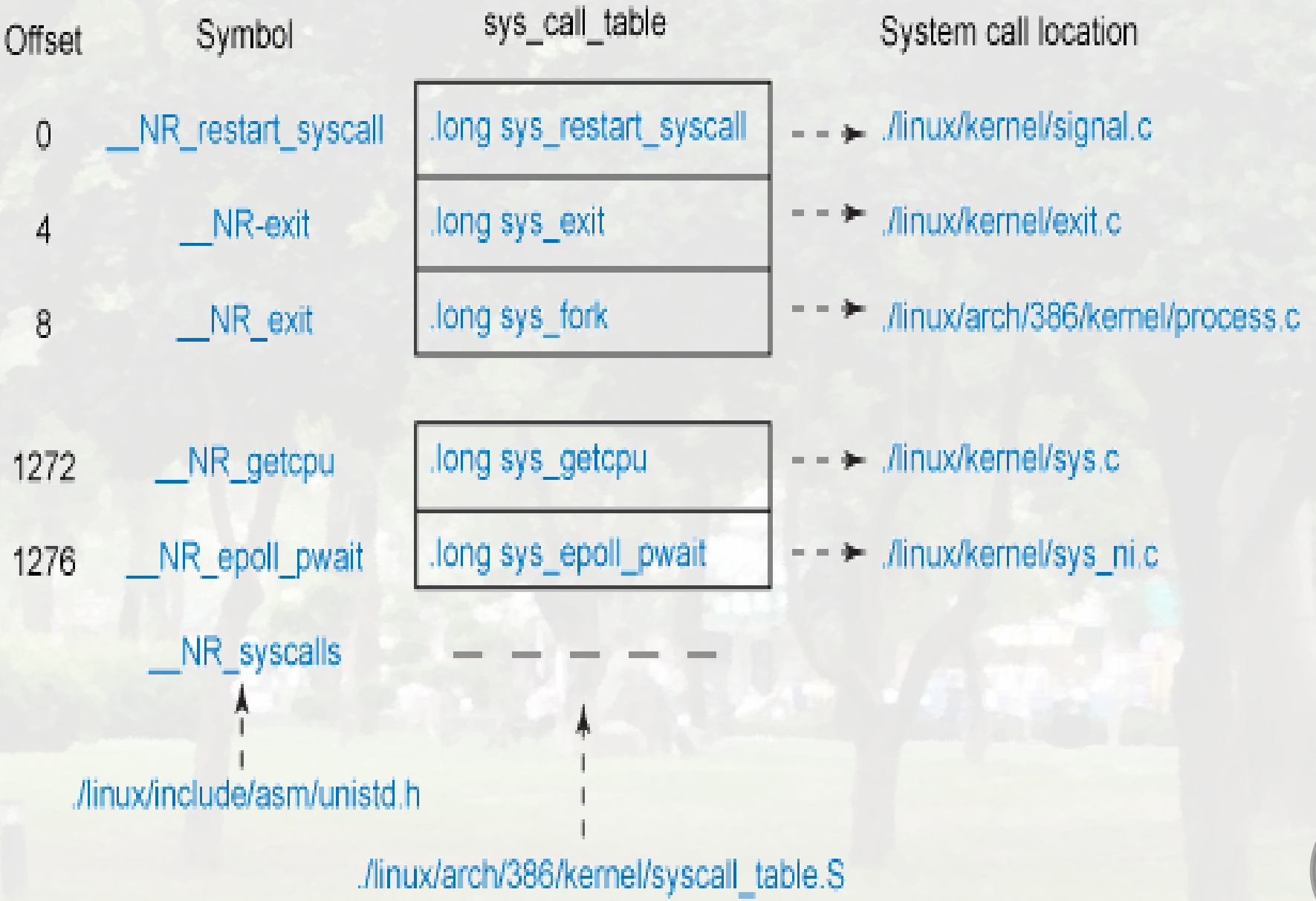
```
$ cat hello-syscall3.c
#include <stdio.h>
#include <sys/syscall.h>
#include <unistd.h>

int main()
{
        int ret;
        ret = syscall(__NR_write, 1, "Hello World\n", 12);
        return 0;
}
$ ./hello-syscall3
Hello World
```

```
#include <stdio.h>
char message[] = "Hello,
world!\n";
int main(void) {
  long _res;
  __asm__ volatile (
    "int $0x80"
    : "=a" (_res)
    : "a" ((long) 4),
      "b" ((long) 1),
      "c" ((long) message),
      "d" ((long) sizeof(message)));
    return 0;
}
```

```
$ head /usr/include/bits/syscall.h
/* Generated at libc build time from kernel syscall list.  */

#ifndef _SYSCALL_H
# error "Never use <bits/syscall.h> directly; include <sys/syscall.h>
instead."
#endif

#define SYS__llseek __NR__llseek
#define SYS__newselect __NR__newselect
#define SYS__sysctl __NR__sysctl
```

```
$ head -n 12 /usr/include/asm-
i386/unistd.h
#ifndef _ASM_I386_UNISTD_H_
#define _ASM_I386_UNISTD_H_

/*
 * This file contains the system call
numbers.
 */

#define __NR_restart_syscall      0
#define __NR_exit                 1
#define __NR_fork                 2
#define __NR_read                 3
#define __NR_write                4
```

系統呼叫 (3)


Offset
Symbol
sys_call_table
System call location
0
__NR_restart_syscall
.long sys_restart_syscall
./linux/kernel/signal.c
4
__NR-exit
.long sys_exit
./linux/kernel/exit.c
8
__NR_exit
.long sys_fork
./linux/arch/386/kernel/process.c
1272
__NR_getcpu
.long sys_getcpu
./linux/kernel/sys.c
1276
__NR_epoll_pwait
.long sys_epoll_pwait
./linux/kernel/sys_ni.c
__NR_syscalls
./linux/include/asm/unistd.h
./linux/arch/386/kernel/syscall_table.S

執行時期的系統呼叫 (1)

- glibc 的實做 sysdeps/unix/sysv/linux/i386/sysdep.h

```
# define INTERNAL_SYSCALL(name, err, nr, args...) \
({    \
   register unsigned int resultvar; \
   EXTRAVAR_##nr     \
   asm volatile ( \
      LOADARGS_##nr \
      "movl %1, %%eax\n\t"     \
      "int $0x80\n\t"     \
      RESTOREARGS_##nr     \
      : "=a" (resultvar) \
      : "i" (__NR_##name) \
            ASMFMT_##nr(args)
      : "memory", "cc"); \
   ( int) resultvar; })
```

```
#define INLINE_SYSCALL(name, nr, args…) \
({   \
    unsigned int resultvar = INTERNAL_SYSCALL
   (name, , nr, args); \
    if (__builtin_expect
   (INTERNAL_SYSCALL_ERROR_P (resultvar, ), 0))
    { \
        __set_errno (INTERNAL_SYSCALL_ERRNO
   (resultvar, )); \
        resultvar = 0xffffffff; \
    } \
} \
(int) resultvar; })
```

執行時期的系統呼叫 (2)

- 程式載入器或 shell 會有類似的操作
  - execve syscall
- Linux Kernel
  - arch/i386/kernel/traps.c

```
mov1 <envp>, %edx
mov1 <argv>, %ecx
mov1 <file>, %ebx
mov1 $11, %eax                ; execve
int $0x80
```

```
void __init trap_init(void)
{
    ...
  set_system_gate(SYSCALL_VECTOR,
                  &system_call);
}
```

Incoming Part IV

- 持續追蹤系統呼叫與 Program Loader 行為
- 透過 qemu 追蹤系統呼叫
- 即時分析： Kernel & User Process
  - 兩棲「Hello World」

參考資料

- Kernel Hacking with UML
  - http://user-mode-linux.sourceforge.net/new/hacking.html
- 用 Open Source 工具開發軟體：新軟體開發觀念
  - http://www.study-area.org/cyril/opentools/
- Kernel command using Linux system calls
  - http://www-128.ibm.com/developerworks/linux/library/l-system-calls/
- Jollen 的 Blog
  - http://www.jollen.org/blog/
- QEMU::Documentation
  - http://fabrice.bellard.free.fr/qemu/user-doc.html