Open RazzmatazZ Laboratory (OrzLab)

http://orzlab.blogspot.com/

快快樂樂學 GNU Debugger (gdb) Part I - 概念與初體驗

Jan 26, 2008

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

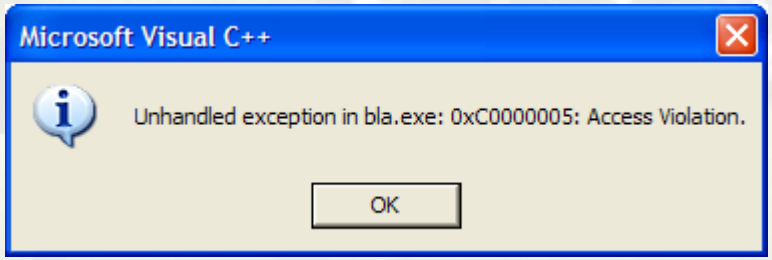
注意

- 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
- 以 x86/IA32 平台為主
- 系統平台
  - Ubuntu hardy (development branch, **8.04**)
  - Linux kernel 2.6.24
  - gcc 4.2.2
  - glibc 2.7
  - gdb 6.7.1
- 部份基礎概念請參考「深入淺出 Hello World」系列演講


DON'T PANIC

不會在此學到 ...

- 如何殺死害蟲
- 如何解決一般的問題
- 寫出無錯誤的 (bug-free) 的程式碼
- 不著頭緒的開發方式
- 使用 MS-Windows 工具或開發環境



會在此玩到 ...

- 除錯偵錯的原理
- 躺在硬碟深處的工具
- 尋幽訪勝靠自己探索
- 開放系統之美
- 快快樂樂 TM

大綱

- Debug 類型
- Linux 的工具集合
- 系統提供之偵錯機制
- GDB 初體驗

概念 (1)

Debugger 能為我們做什麼?

- 知悉程式為何終止或退出
- 執行中程式的具體行為
- 目前資料狀態
- 動態改變執行流程
- 監看或修改某些程式執行時的數值
- 惡搞!

概念 (2)

不要問 Debugger 能為我們做什麼，要問我們能為 Debugger 做什麼

- 青蛙王子的故事
- 就算不是程式開發者，也能用 Debugger 協助開發者釐清問題點
- Programming 2.0 的時代
  - apt-cache search **dbg**

概念 (3)

我就是只用 printf()

(簡單，而且大部分情況可用)，但 ...

- 新增 debug code 會增加風險
  - 加大真實情況與偵錯模式的差異
  - 不適合强調精確時間或資源的環境
- Debugger 則降低許多風險
- Debugger 可快速追蹤程式行為並重新呈現
- Debugger 可更掌握執行時期的表現

有了 Debugger， Coding 是彩色的！

Debug 類型 (1)

- 非互動式偵錯
  - live 或 on-the-fly
  - postmortem
- 互動式偵錯
  - 在相同系統或資源的環境
  - 自遠端系統
    - serial
    - network
    - JTAG
    - ICE

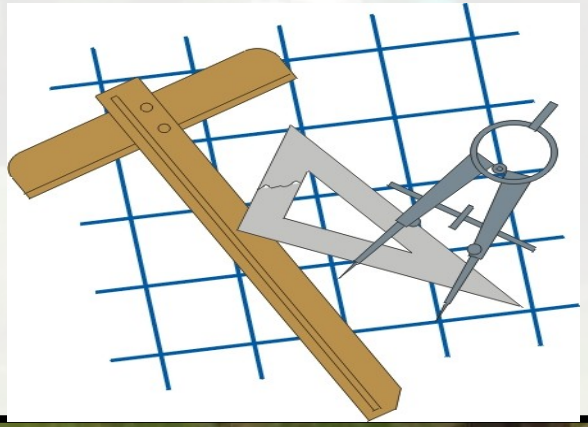
C

Debug 類型 (2)

- Kernel-level Debugging
  - Linux - monotlothic code
  - 對象： kernel module 或 builtin code
- User-level Debugging
  - Process
  - 對象：在 user-space 執行的應用程式
  - 分類:
    - 單一 task/thread 程式
    - Multi-tasking 應用程式
    - Multi-thread 應用程式

Linux 的工具集合

- Trace
  - strace
  - ltrace
- GNU Debugger
- printk
- kdb / kgdb
- User-Mode Linux
- qemu



Trace 工具

- 非互動式偵錯
- strace
  - 追蹤 system call 與 signal
- ltrace
  - 追蹤 library call

GNU Debugger(1)

- GNU Debugger = gdb
  - source-level debugger
  - 支援 thread 、 remote debugging 、硬體架構模擬
- 互動式偵錯
  - set break point 與 watch
  - run
  - step, next, continue
  - 分析資料與系統資訊
  - Thread – threads info

GNU Debugger(2)

- 整合非互動與互動式偵錯
  - core – 當發生例外情況時，所產生的 system/process image，用以表示其狀態
  - gdb **core**=core.XXX
  - **ulimit** - 控制 core 檔案的產生與限制

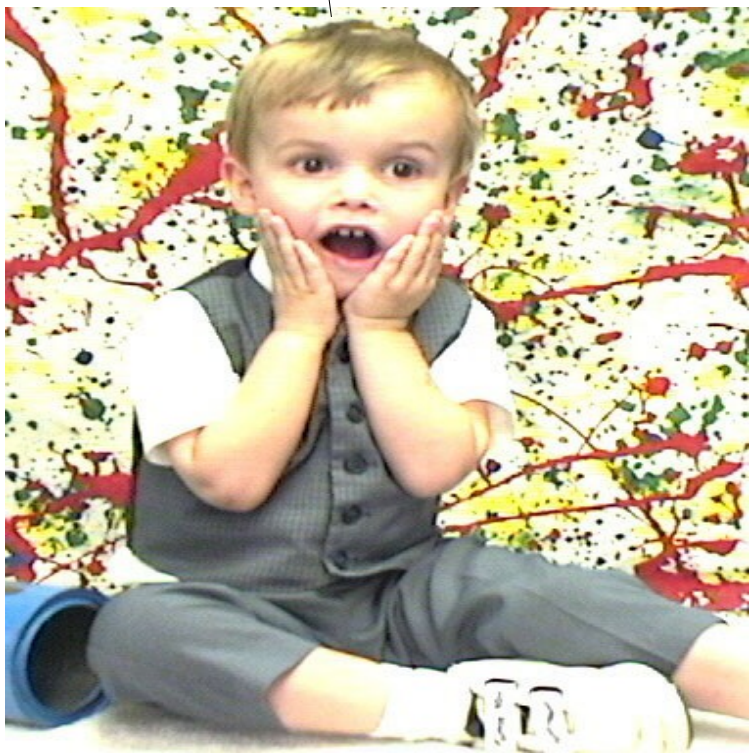
Linux 核心機制

- kdb – assembly-level debugger
- kgdb – 提供核心層面的 gdb 擴充
- Oops – 發生例外情況時產生的訊息
- printk – 列印偵錯訊息
- User-Mode Linux

Kgdb

- 使用 gdb 來對 Kernel 偵錯
- 介面： serial line 或 Ethernet
- 需要兩台機器
  - host（操作 gdb)
  - target
    - append="gdb gdbttyS=0 gdbbaud=115200"
- 以 kernel patch 形式存在
- host 機器
  - stty ispeed 115200 ospeed 115200 < /dev/ttyS0
  - gdb vmlinux
  - (gdb) remote target /dev/ttys0

Oops!!

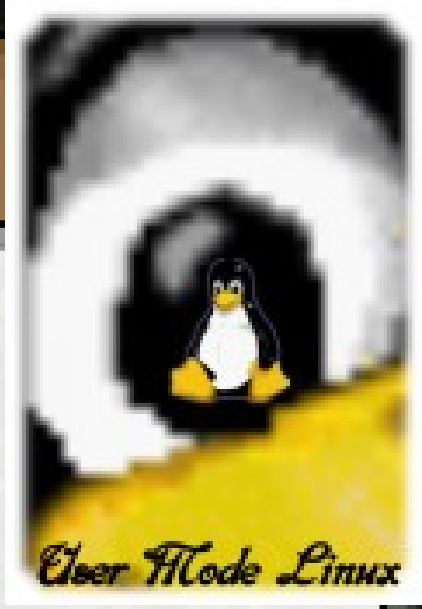


Oops

- 當核心產生例外情況時，訊息經由 klogd 輸出到 kernel ring buffer
  - dmesg 或 /var/log/*
- Kernel 2.4 前 (含) 可以 ksymoops 解讀， 2.6 系列對 Oops 輸出做了改進
  - Documentation/oops-tracing.txt

```
Aug 29 09:51:01 blizard kernel: Unable to handle kernel paging request at virtual address f15e97cc
Aug 29 09:51:01 blizard kernel: current->tss.cr3 = 0062d000, %cr3 = 0062d000
Aug 29 09:51:01 blizard kernel: *pde = 00000000
Aug 29 09:51:01 blizard kernel: Oops: 0002
Aug 29 09:51:01 blizard kernel: CPU:    0
Aug 29 09:51:01 blizard kernel: EIP:    0010:[oops:_oops+16/3868]
...
Aug 29 09:51:01 blizard kernel: Process oops_test (pid: 3374, process nr: 21, stackpage=00589000)
Aug 29 09:51:01 blizard kernel: Stack: 315e97cc 00589f98 0100b0b4 bffffed4 0012e38e 00240c64 003a6f80 00000001
Aug 29 09:51:01 blizard kernel:        00000000 00237810 bfffff00 0010a7fa 00000003 00000001 00000000 bfffff00
Aug 29 09:51:01 blizard kernel:        bffffdb3 bffffed4 ffffffda 0000002b 0007002b 0000002b 0000002b 00000036
Aug 29 09:51:01 blizard kernel: Call Trace: [oops:_oops_ioctl+48/80] [_sys_ioctl+254/272] [_system_call+82/128]
```

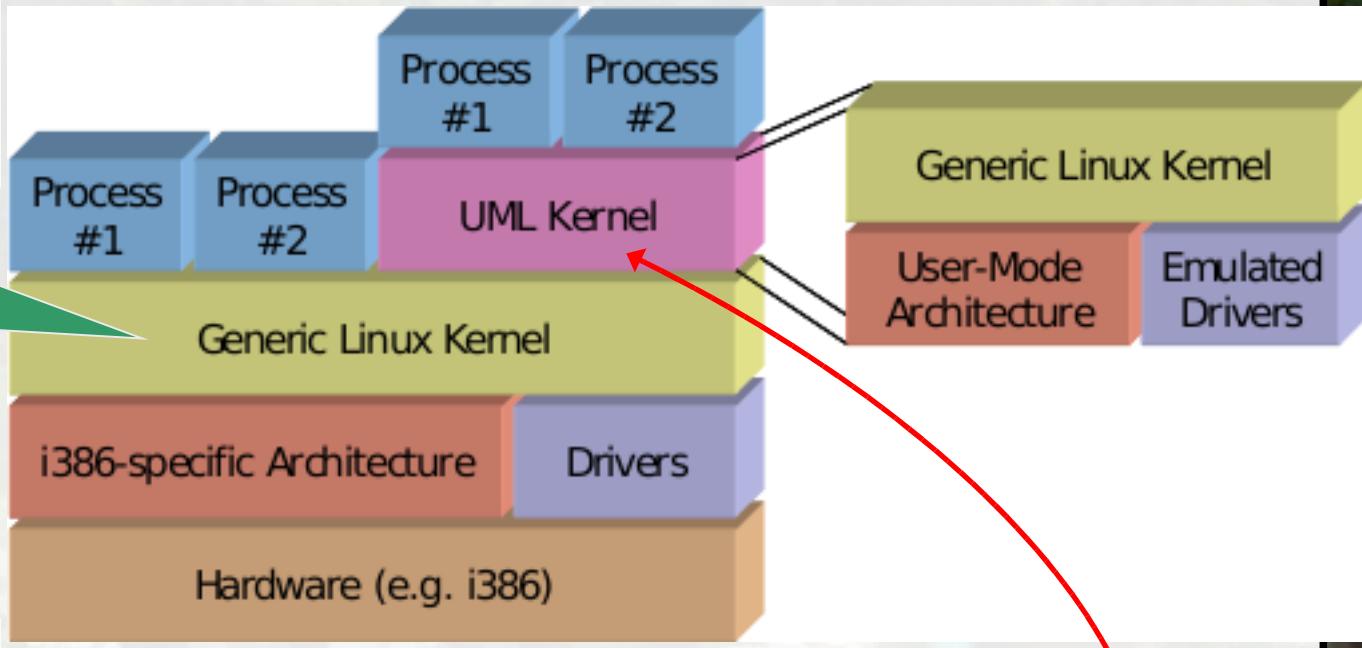
User-Mode Linux (1)



- 將 Linux Kernel 「移植」到 user-space
  - 修改的 "Kernel" 被視為一般的 Linux process 來執行
- 屬於 Para-virtualization 技術
  - 需要對 guest 核心作修改
- 應用
  - 對與硬體無關的的程式作偵錯與安全測試
  - 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)

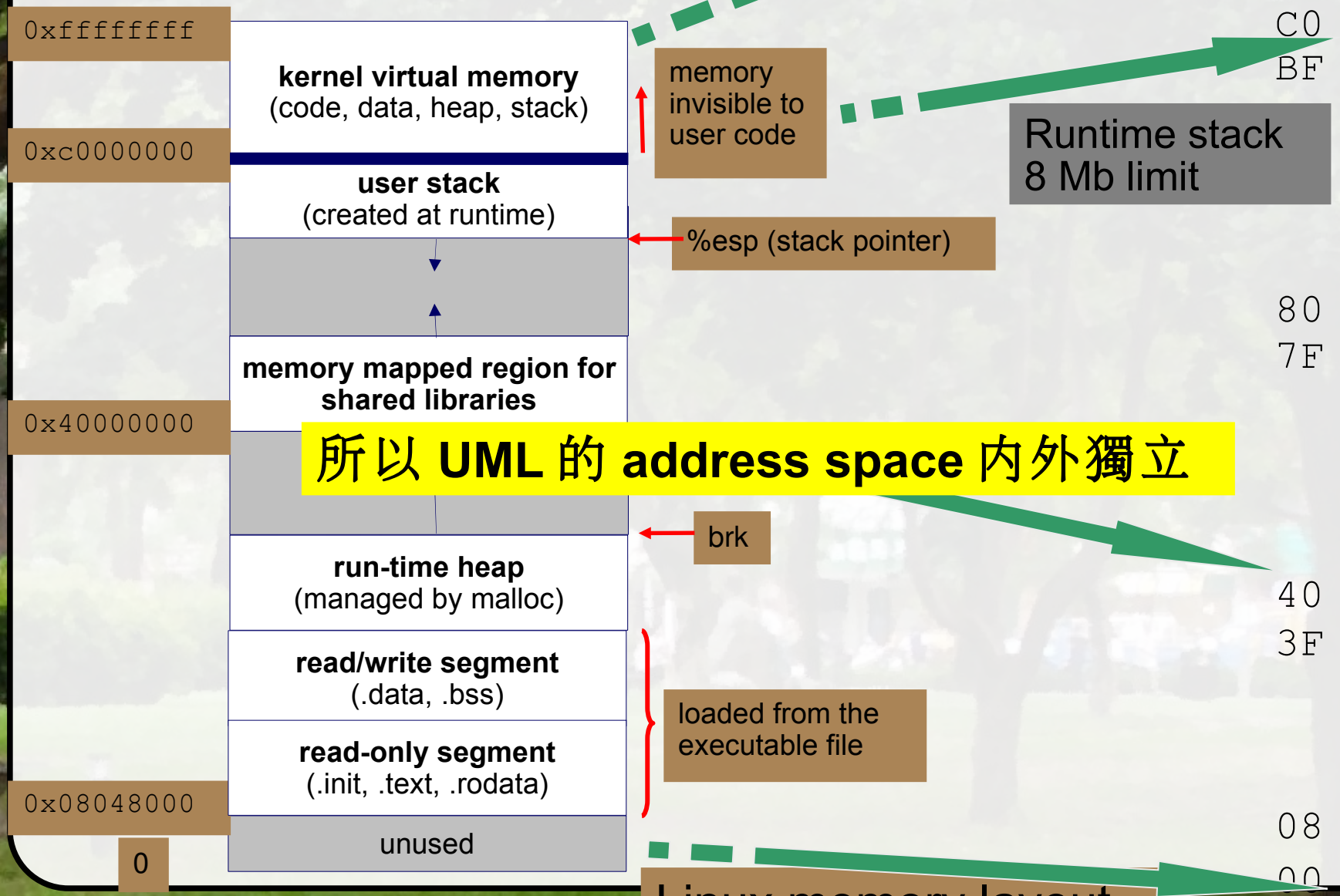
```
+-----------+-----------+----+
| Process 1 | Process 2 | ...|
+-----------+-----------+----+
|        Linux Kernel         |
+-----------------------------+
|          Hardware           |
+-----------------------------+
```



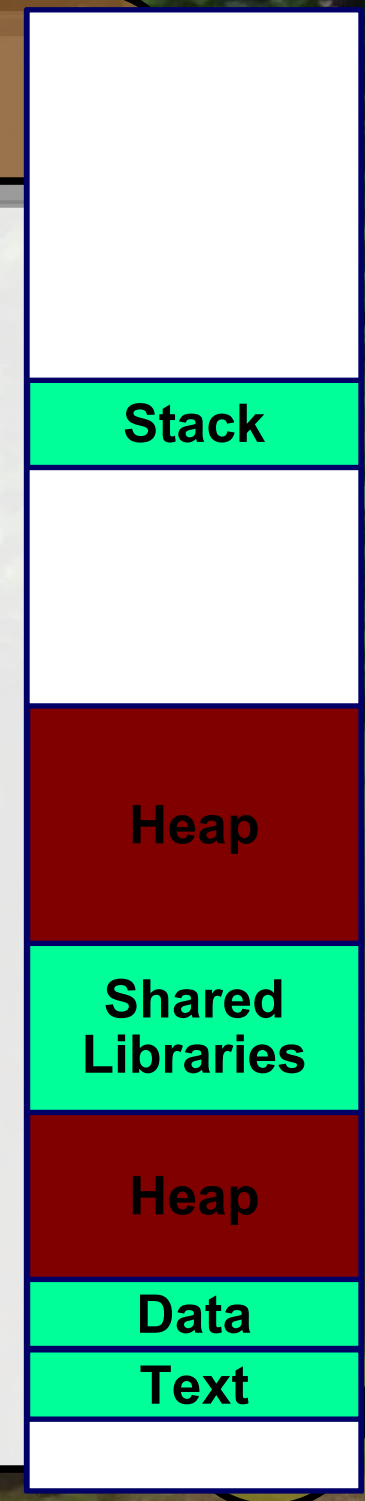

```
            +----------------+
            | Process 2 | ...|
+-----------+----------------+
| Process 1 | User-Mode Linux|
+-----------------------------+
|        Linux Kernel         |
+-----------------------------+
|          Hardware           |
+-----------------------------+
```

每個 process 都有獨自的 address space

Address Space



所以 **UML** 的 **address space** 內外獨立



Linux memory layout

User-Mode Linux (3)

~/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
```

基本分析:

- 設定關鍵 breakpoint
- 單步執行
- Call Graph

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

UMID (Unique Machine ID)

```
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)
```

```
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:~/uml/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:~/uml/linux-2.6.20.4$
```

pstree

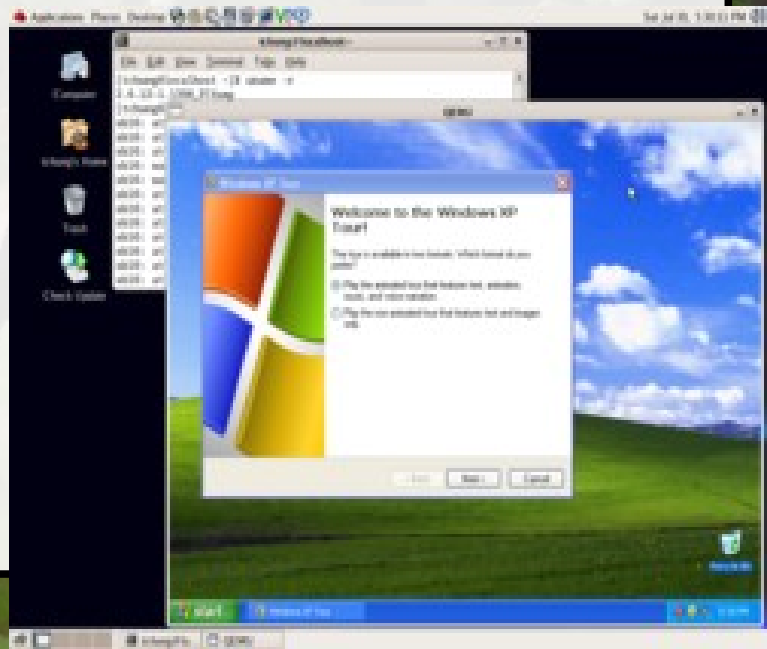
```
/home/jserv/uml/linux-2.6.20.4/init/main.c
Make breakpoint pending on future shared library load? (y or [n]) n
(tgdb) run ubd0=/opt/src/ubuntu-root mem=64M
Starting program: /home/jserv/uml/linux-2.6.20.4/linux ubd0=/opt/src/ubuntu-root mem=64M
Checking that ptrace can change system call numbers...OK
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**

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 ?`
- 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

使用 **target** 的 **ld-linux.so.2**

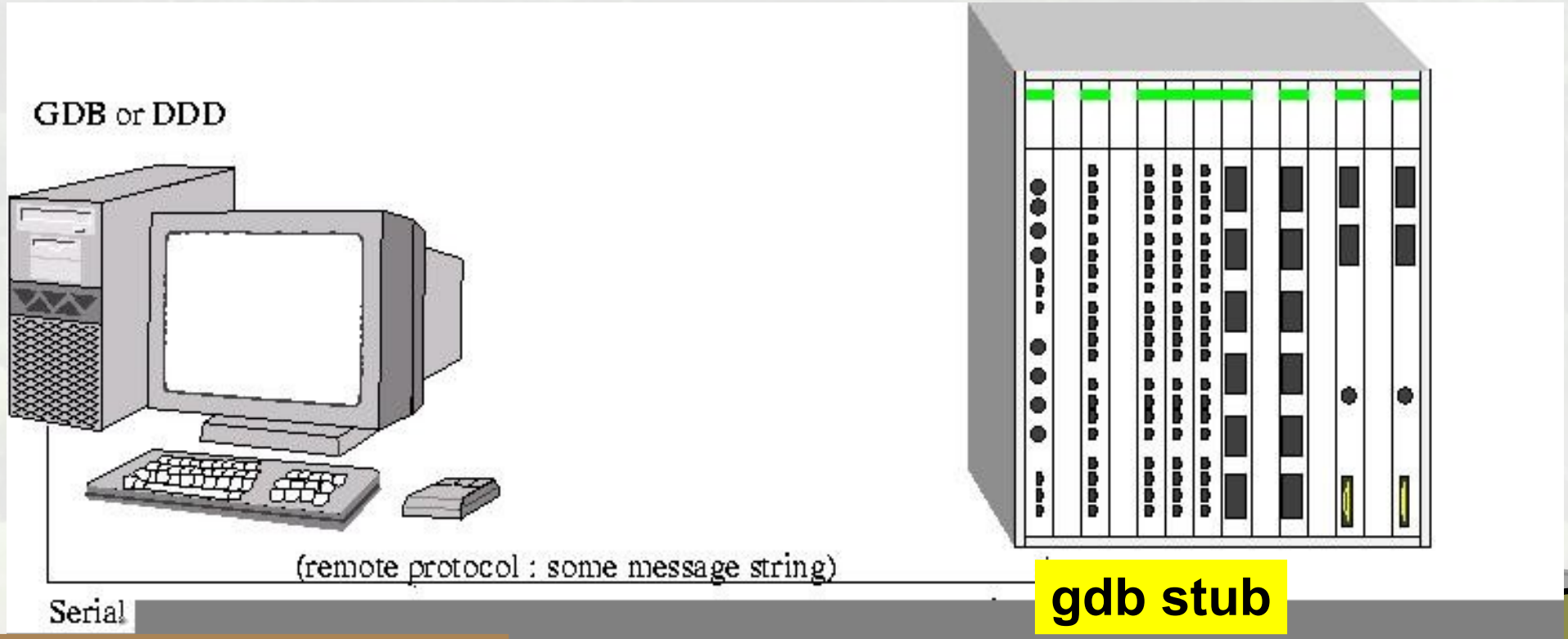
```
~/poky/build/tmp$ file ./rootfs/bin/busybox
./rootfs/bin/busybox: ELF 32-bit LSB executable, ARM, version 1 (ARM), for
GNU/Linux 2.4.0, dynamically linked (uses shared libs), for GNU/Linux 2.4.0,
stripped
~/poky/build/tmp$ ./qemu-arm ./rootfs/lib/ld-linux.so.2 \
--library-path ./rootfs/lib ./rootfs/bin/busybox 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



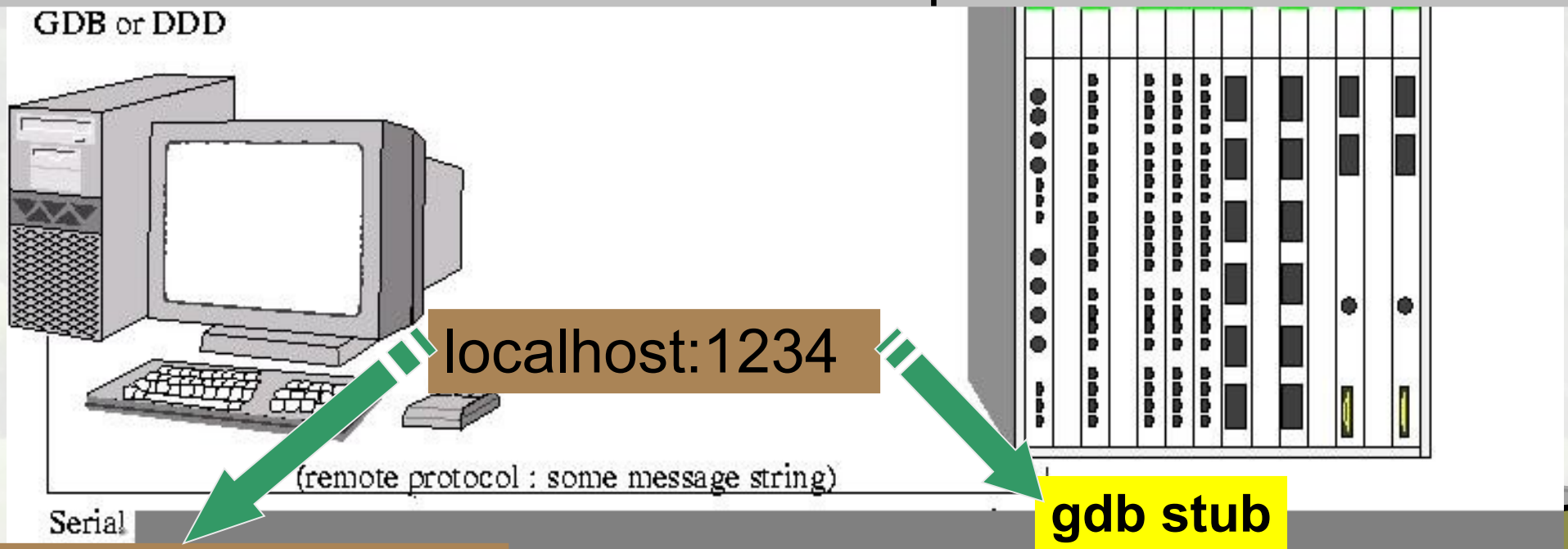
開發平台 (Host)
運作完整的 GDB

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



開發平台 (Host)
運作完整的 GDB

Qemu 所模擬的機器

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(filer
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)
```

TightVNC: QEMU

```
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**

以 **Remote GDB** 分析

系統提供之偵錯機制

- printk()
  - 定義在 <linux/kernel.h> 所宣告的八個巨集中
  - 展開分別成為 <0><1><2>…<7> 之類的字串，數字越低，等級越高
  - 如果沒在 printk() 註明分類代碼，則訊息的預設分類為 DEFAULT_MESSAGE_LOGLEVEL
    - kernel/printk.c
- /proc, kernel magic, ...
- ptrace

```
long ptrace(enum __ptrace_request request,
            pid_t pid, void *addr, void *data);
```

PTRACE_TRACEME
PTRACE_PEEKTEXT, PTRACE_PEEKDATA
PTRACE_PEEKUSR
PTRACE_POKETEXT, PTRACE_POKEDATA
PTRACE_POKEUSR
PTRACE_GETREGS, PTRACE_GETFPREGS
PTRACE_SETREGS, PTRACE_SETFPREGS
PTRACE_CONT
PTRACE_SYSCALL, PTRACE_SINGLESTEP
PTRACE_KILL
PTRACE_ATTACH
PTRACE_DETACH

追蹤系統呼叫

- 觀察 user-space 應用程式的行為
  - 透過 debugger 單步執行
  - 適當處印出訊息
  - 將程式交給 strace 來執行
- strace 提供的除錯資訊，直接取自核心本身
  - 顯示由 user-space 程式所發出的所有系統呼叫，輸入輸出資料是否一致

#strace ls /dev 2> log

- strace 最有用之處，在於可從系統呼叫中發現執行期的錯誤，一般應用程式中的 perror() 往往不夠詳細

ELF(1)

- ELF (Executable and Linkable Format)
  - 最初由 UNIX System Laboratories 發展，為 AT&T System V Unix 所使用，稍後成為 BSD 家族與 GNU/Linux 上 object file 的標準二進位格式
- COFF (Common Object File Format)
  - System V Release 3 使用的二進位格式
- DWARF-1/2 (Debug Information Format)
  - 通常搭配 ELF 或 COFF 等格式

$ **man gcc**
...
GCC has various special options that are used for debugging either your program or GCC:

-g Produce debugging information in the operating system's native format (stabs, COFF, XCOFF, or **DWARF** 2). GDB can work with this debugging information

- 只要符合 DWARF 規範的 object file，即可使用 **gdb** 一類 source-level debugger
- 格式上， **Machine-Independent**

ELF(2)

- Page size
- Virtual address memory segment (sections)
- Segment size

- Magic number
- type (.o / .so / exec)
- Machine
- byte order
- …

- Initialized (static) data

- code

- Un-initialized (static) data
- Block started by symbol
- **Has section header but occupies no space**

- **Symbol table**
- Procedure and static variable names
- Section name

- Relocation info for .text section
- Addresses of instructions that need to be modified in the executable instructions for modifying.

- Relocation info for .data section
- Address pointer data will need to be modified in the merged executable

- Info for symbolic debugging

ELF header
Program header table (required for executables)
`.text` section
`.data` section
`.bss` section
`.symtab`
`.rel.txt`
`.rel.data`
`.debug`
Section header table (required for relocatables)

注意：忽略部份細節

ELF(3)

- Page size
- Virtual address memory segment (sections)
- Segment size

- Magic number
- type (.o / .so / exec)
- Machine
- byte order
- …

- Initialized (static) data

- code

- Un-initialized (static) data
- Block started by symbol
- **Has section header but occupies no space**

注意： .dynsym 還保留

Runtime 只需要左邊欄位
可透過 “ **strip”** 指令去除不
需要的 section

ELF header
Program header table (required for executables)
.text section
.data section
.bss section
.symtab
.rel.txt
.rel.data
.debug
Section header table (required for relocatables)

ELF header
Program header table (required for executables)
.text section
.data section
.bss section

```
fibonacci.c
12                      f2 = fib(n - 2);
13                      return f1 + f2;
14              }
15      }
16
17      int main()
18      {
B+> 19              printf("%d\n", fib(5));
20              return 0;
21      }
22
23
24
25
26
27
child process 15737 In: main                    Line: 19   PC: 0x80483db
(gdb) b main
Breakpoint 1 at 0x80483db: file fibonacci.c, line 19.
(gdb) r
Starting program: /home/jserv/debugging/gdb-samples/fibonacci

Breakpoint 1, main () at fibonacci.c:19
(gdb) win
Usage: winheight <win_name> [+ | -] <#lines>
(gdb)
```

gdbtui： gdb 的文字 curses 介面前端

```
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
```

CGDB 是另一個 curses 為基礎的 GDB 前端程式，內建類似 vim 的程式碼編輯功能

http://cgdb.sourceforge.net/

```
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)
```

gdb 初體驗 (1)

- 進入 gdb
  - gdb filename
- 列出程式碼
  - (gdb) **list** 預設一次列出 10 行
  - (gdb) **list** 3,9
  - (gdb) **list** ip_vs_in
- 執行程式
  - (gdb) **run** 欲中斷按下 Ctrl-C
- 暫時回到 Linux 提示符號
  - (gdb) **shell** 回到 gdb 提示符號： **exit**
  - (gdb) **shell** ls

gdb 初體驗 (2)

- 線上說明
  - (gdb) **help**
  - (gdb) **help** all 列出 gdb 所有操作命令
- 設定中斷點
  - (gdb) **break** 7
  - (gdb) **break** ip_vs_in
  - (gdb) **break** 9 **if** result > 50　考慮中斷點是否合理
  - (gdb) **watch** result > 50 執行過變數宣告後才能用
- 檢視變數值
  - (gdb) **print** result
- 檢視變數資料型態
  - (gdb) **whatis** result

gdb 初體驗 (3)

- 程式流程控制
  - (gdb) **run**
  - (gdb) **continue**
  - (gdb) **step**
  - (gdb) **next**
- 檢視所有中斷點的狀態
  - (gdb) **info** breakpoints
- 使中斷點失效
  - (gdb) **disable**
- 使中斷點生效
  - (gdb) **enable**

command	format	effect
`run`	run arg1 arg2 ... < stdin	run program as if invoked by the shell
`CTL-C`	(control-C keystroke)	interrupt the running program
`where`	where	show the stack with line numbers
`list`	list	list the source code around the point of current interest
`print`	print expression	evaluate the expression in the current context and display the result
`up`	up	move the current context one frame up the stack
`down`	down	move the current context one frame back down the stack
`break`	break [function | n]	set a breakpoint at entry to the named function at line n in the current context
`watch`	watch expression	when begin running, keep evaluating the expression and stop if it becomes true
`continue`	continue	continue execution (after stopping at a breakpoint or watchpoint)
`step`	step	continue, but stop after executing just one source line
`clear`	clear [function | n]	clear the breakpoint set at the function or at line n
`info`	info break	show information about all breakpoints
`help`	help	display help information
`quit`	quit	quit GDB

```
# gdb dpm
GNU gdb 6.7.1-debian
Copyright (C) 2007 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu".
(gdb) b main
Breakpoint 1 at 0x8048626: file dpm.cc line 6.
(gdb) r
Starting program: /home/jserv/dpm

Breakpoint 1, main () at dpm.cc:6
6         for (i = 0; i < 2; i++)
(gdb) l
1       #include <iostream>
2
3       int main()
4       {
5         int i;
6         for (i = 0; i < 2; i++)
7           {
8             std::cout << i << s
9           }
10      }
```

b = break

r = run

dpm.cc

l = list

```
(gdb) b 8
Breakpoint 2 at 0x8048638: file dpm.cc, line 8.
(gdb) c
Continuing.

Breakpoint 2, main () at dpm.cc:8
8             std::cout << i << std::endl;
(gdb) p i
$1 = 0
(gdb) n
0
9         }
```

c = continue

p = print

```
(gdb) b 8
Breakpoint 2 at 0x8048638: file dpm.cc, line 8.
(gdb) c
Continuing.

Breakpoint 2, main () at dpm.C:8
8               std::cout << i << std::endl;
(gdb) p i
$1 = 0
(gdb) n
0
9       }
```

```
(gdb) b 8
Breakpoint 1 at 0x8048638: file dpm.C, line 8.
(gdb) cond 1 (i>0)
(gdb) r
Starting program: /home/jserv/dpm
0

Breakpoint 1, main () at dpm.cc:8
8               std::cout << i << std::endl;
(gdb) p i
$1 = 1
(gdb)
```

hellot.c

```
#include <stdio.h>

static char *my_str = "Hello World!";
int main()
{
        puts(my_str);
        return 0;
}
```

```
$ ./hello
Hello World!
```

```
(gdb) b main
Breakpoint 1 at 0x8048385: file hello.c, line 6.
(gdb) l
1       #include <stdio.h>
2
3       static char *my_str = "Hello World!";
4       int main()
5       {
6               puts(my_str);
7               return 0;
8       }
(gdb) r
Starting program: gdb-samples/hello

Breakpoint 1, main () at hello.c:6
6               puts(my_str);
(gdb) jump 7
Continuing at 0x8048392.

Program exited normally.
```

動態改變記憶內容

```
(gdb) b main
Breakpoint 1 at 0x8048385: file hello.c,
(gdb) r
Starting program: gdb-samples/hello

Breakpoint 1, main () at hello.c:6
6               puts(my_str);
(gdb) p my_str="I need my space."
$1 = 0x804a008 "I need my space."
(gdb) c
Continuing.
I need my space.

Program exited normally.
```

```c
#include <stdio.h>

static char buff [256];
static char* string;

int main ()
{
    printf ("Please input a string: ");
    gets (string);
    printf ("\nYour string is: %s\n", string);
    return 0;
}
```

input.c

```
$ ./input
Please input a string: jserv
Segmentation fault
```

```
(gdb) l
1   #include <stdio.h>
2
3   static char buff [256];
4   static char* string;
5
6   int main ()
7   {
8       printf ("Please input a string: ");
9       gets (string);
10      printf ("\nYour string is: %s\n", string);
(gdb) b 8
Breakpoint 1 at 0x80483b5: file input.c, line 8.
(gdb) r
```

顯然記憶體操作有問題，
不需修改程式，直接檢驗

```
Starting program: /home/jserv/debugging/gdb-samples/input

Breakpoint 1, main () at input.c:8
8       printf ("Please input a string: ");
(gdb) set variable string="jserv_longer"
(gdb) c
Continuing.
Please input a string: jserv

Your string is: jserv

Program exited normally.
```

```
$ ./input
Please input a string: jserv
Segmentation fault
```

set variable 可協助釐清執行時期的行為表現

fibonacci.c

```c
#include <stdio.h>

int fib(int n)
{
    /* Ending condition of recursive */
    if (n == 0) {
        return 0;
    }
    /* Ending condition of recursive */
    else if (n == 1) {
        return 1;
    }
    else {
        return fib(n - 1) + fib(n - 2);
    }
}

int main()
{
    printf("%d\n", fib(5));
    return 0;
}
```

遞迴

break 後接 commands，
可指定中斷點發生的對應指令

```
(gdb) b fib
Breakpoint 1 at 0x804837b: file fibonacci.c, line 6.
(gdb) commands
Type commands for when breakpoint 1 is hit, one per line.
End with a line saying just "end".
>info args
>list
>continue
>end
(gdb) r
```

```
Starting program: /home/jserv/debugging/gdb-samples/fibonacci

Breakpoint 1, fib (n=5) at fibonacci.c:6
6       if (n == 0) {
n = 5
1   #include <stdio.h>
2
3   int fib(int n)
4   {
5       /* Ending condition of recursive */
6       if (n == 0) {
7           return 0;
8       }
9       /* Ending condition of recursive */
10      else if (n == 1) {

Breakpoint 1, fib (n=4) at fibonacci.c:6
6       if (n == 0) {
n = 4
1   #include <st
2
3   int fib(int
4   {
5       /* Endin
6       if (n ==
7           retur
8       }
```

一旦執行，可再加入中斷條件，如 break if n==1

```
(gdb) b fib
Breakpoint 1 at 0x804837b: file fibonacci.c, line 6.
(gdb) commands
Type commands for when breakpoint 1 is hit, one per line.
End with a line saying just "end".
>info args
>list
>continue
>end
(gdb) r
```

```
void quicksort(int a[], int left, int right) {
    int last = left, i;

    if (left < right) {
        swap(a, left, Random(left, right));
        for (i = left + 1; i <= right; i++)
            if (a[i] < a[left])
                swap(a, ++last, i);
        swap(a,left,last);
        quicksort(a,left, last - 1);
        quicksort(a, last + 1, right);
    }
}
```

qsort.c

print **a[0] @ 10**
a[0] 與其後 10 個元素的值

Break 後接 commands，
可指定中斷點發生的對應指令

```
(gdb) b quicksort
Breakpoint 1 at 0x80484d7: file quicksort.c, line 31.
(gdb) commands
Type commands for when breakpoint 1 is hit, one per line.
End with a line saying just "end".
>print a[0] @ 10
>continue
>end
(gdb) r
```

```
Starting program: /home/jserv/debugging/qsort/quicksort
    7    3    1    9   -2    0   16    3   11    4

Breakpoint 1, quicksort (a=0xbfacda98, left=0, right=9) at quicksort.c:31
31      int last = left, i;
$1 = {7, 3, 1, 9, -2, 0, 16, 3, 11, 4}

Breakpoint 1, quicksort (a=0xbfacda98, left=0, right=6) at quicksort.c:31
31      int last = left, i;
$2 = {4, 3, 1, 7, -2, 0, 3, 9, 11, 16}

Breakpoint 1, quicksort (a=0xbfacda98, left=0, right=-1) at quicksort.c:31
31      int last = left, i;
$3 = {-2, 3, 1, 7, 4, 0, 3, 9, 11, 16}

Breakpoint 1, quicksort (a=0xbfacda98, left=1, right=6) at quicksort.c:31
31      int last = left, i;
$4 = {-2, 3, 1, 7, 4, 0, 3, 9, 11, 16}

Breakpoint 1, quicksort
31      int last = left,
$5 = {-2, 3, 1, 3, 0, 4,

Breakpoint 1, quicksort
}
```

$1, $2, $3, ... 為歷史變數

```
(gdb) b quicksort
Breakpoint 1 at 0x80484d7: file quicksort.c, line 31.
(gdb) commands
Type commands for when breakpoint 1 is hit, one per line.
End with a line saying just "end".
>print a[0] @ 10
>continue
>end
(gdb) r
```

```
(gdb) p 100
$1 = 100
(gdb) p/x 100
$2 = 0x64
(gdb) p/o 100
$3 = 0144
(gdb) p/t 100
$4 = 1100100
(gdb)
```

十進位

十六進位

八進位

二進位

拿來當 C 語言直譯器

```
$ gdb `which gdb`
GNU gdb 6.7.1-debian
(gdb) start
Breakpoint 1 at 0x807f55e
0x0807f55e in main ()
(gdb) set debug target 1
(gdb) set debug infrun 1
(gdb) set debug lin-lwp 1
(gdb) p chdir("/tmp")
...
(gdb) p free(0xb7f42200)
```

```
$ gdb `which gdb`
GNU gdb 6.7.1-debian
(gdb) start
Breakpoint 1 at 0x807f55e
Starting program: /usr/bin/gdb
0x0807f55e in main ()
(gdb) p 1+2
$1 = 3
(gdb) p abs(-50)
$2 = 50
(gdb) p puts("Hello World")
Hello World
$3 = 12
(gdb)
```

```
(gdb) p getenv("HOME")
$4 = -1074340217
(gdb) x/s $4
0xbff6de87: "/home/jserv"
(gdb) x/s $
0xbff6de87: "/home/jserv"
(gdb) p (char)*$4
$5 = 47 '/'
(gdb)
```

```
(gdb) shell kill -l
 1) SIGHUP  2) SIGINT  3) SIGQUIT 4) SIGILL
 5) SIGTRAP 6) SIGABRT 7) SIGBUS  8) SIGFPE
 9) SIGKILL10) SIGUSR111) SIGSEGV12) SIGUSR2
13) SIGPIPE14) SIGALRM15) SIGTERM16) SIGSTKFLT
17) SIGCHLD18) SIGCONT19) SIGSTOP20) SIGTSTP
21) SIGTTIN22) SIGTTOU23) SIGURG 24) SIGXCPU
25) SIGXFSZ26) SIGVTALRM  27) SIGPROF28) SIGWINCH
29) SIGIO  30) SIGPWR 31) SIGSYS 34) SIGRTMIN
35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3 38) SIGRTMIN+4
39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10    45) SIGRTMIN+11    46) SIGRTMIN+12
47) SIGRTMIN+13    48) SIGRTMIN+14    49) SIGRTMIN+15    50) SIGRTMAX-14
51) SIGRTMAX-13    52) SIGRTMAX-12    53) SIGRTMAX-11    54) SIGRTMAX-10
55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7 58) SIGRTMAX-6
59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX
```

Signal handler 是 UNIX process 的重要設計

```
(gdb) signal 11
Continuing with signal SIGSEGV.

Program terminated with signal SIGSEGV, Segmentation fault.
The program no longer exists
```

```
(gdb) handle SIGPIPE stop print
Signal        Stop Print   Pass to program    Description
SIGPIPE       Yes  Yes Yes     Broken pipe
(gdb) b main
Breakpoint 1 at 0x80483b5: file input.c, line 8.
(gdb) r
Starting program: /home/jserv/debugging/gdb-samples/input

Breakpoint 1, main () at input.c:8
8       printf ("Please input a string: ");
(gdb) signal SIGPIPE
Continuing with signal SIGPIPE.
```

handle 命令可控制 signal 的處理

- nostop： 接到 signal， 不發送給 proc， 也不結束 proc
- stop： 接到 signal 時停止 proc 的執行
- print： 接到 signal 時顯示訊息
- noprint： 接到 signal 不顯示訊息
- pass： 將 signal 發送給 proc， 並允許 proc 處理
- nopass： 停止 proc 運行， 並且不將 signal 發給 proc

參考資料

- 「深入淺出 Hello World 」系列演講
  - http://wiki.debian.org.tw/HackingHelloWorld
- 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/
- Kgdb
  - http://kgdb.sourcefourge.net