



Open Razzmatazz Laboratory (OrzLab)

<http://orzlab.blogspot.com/>

快樂樂學

GNU Debugger (gdb)

Part I - 概念與初體驗

July 12, 2008



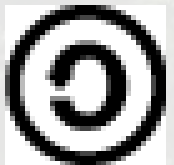
Jim Huang(黃敬群 /jserv)

Email: <jserv.tw@gmail.com>

Blog: <http://blog.linux.org.tw/jserv/>

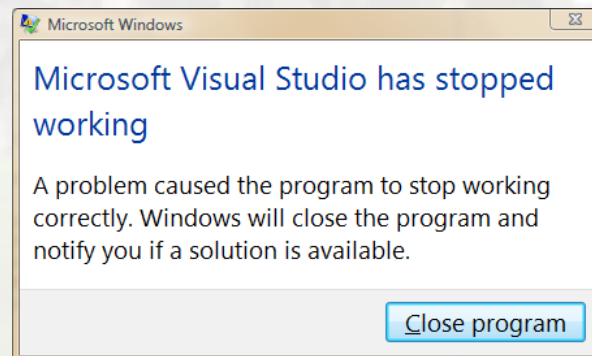
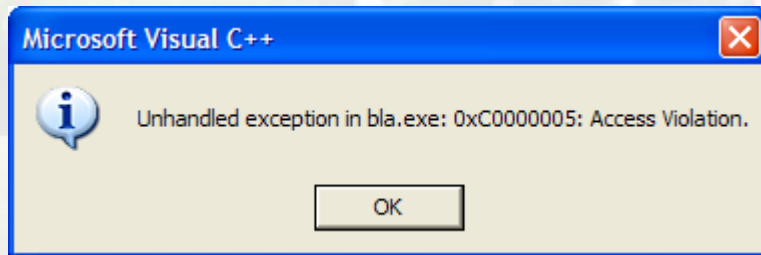
注意

- 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 interpid (development branch, 8.10)
 - Linux kernel 2.6.26
 - gcc 4.3.1
 - glibc 2.8
 - gdb 6.8
- 部份基礎概念請參考「深入淺出 Hello World」系列演講



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™



大綱

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



Debug 的類型



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
 - 不適合強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

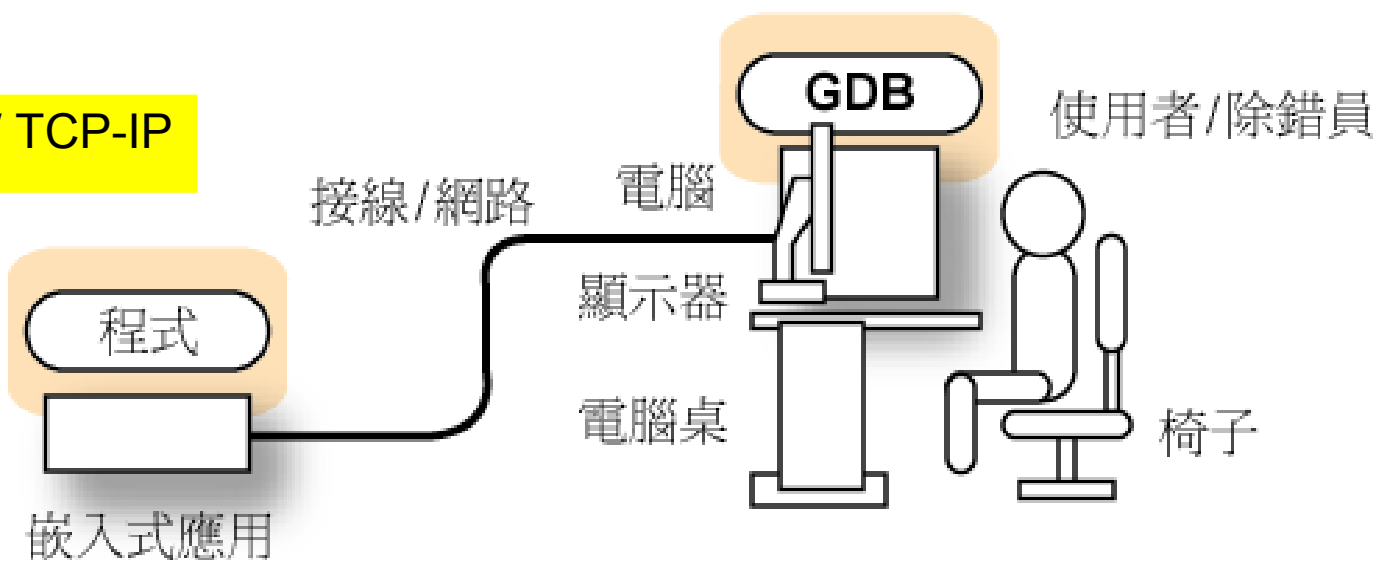


近端/本機型除錯



遠端/遙控型除錯

Serial / JTAG / TCP-IP



Debug 類型 (2)

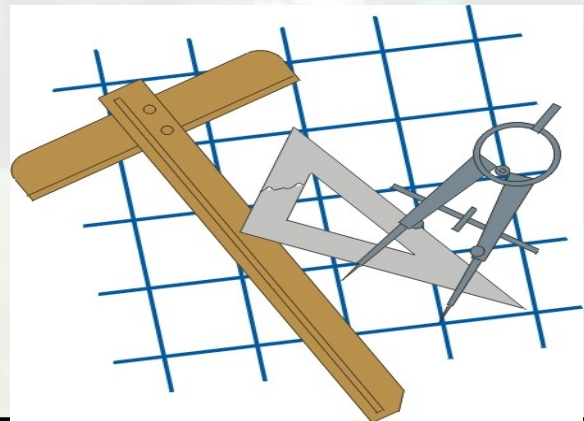
- Kernel-level Debugging
 - Linux - monotlothic code
 - 對象: kernel module 或 builtin code
- User-level Debugging
 - Process
 - 對象: 在 user-space 執行的應用程式
 - 分類:
 - 單一 task/thread 程式
 - Multi-tasking 應用程式
 - Multi-thread 應用程式

Linux 的工具集合



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

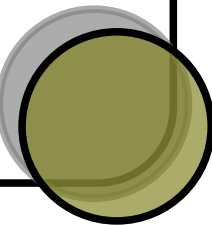
- kdb – assembly-level debugger
- kgdb – 提供核心層面的 gdb 擴充
- Oops – 發生例外情況時產生的訊息
- printk – 列印偵錯訊息
- User-Mode Linux



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

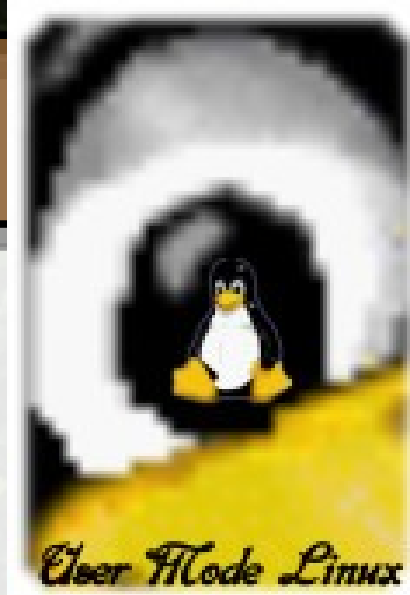
Aug 29 09:51:01 blizard kernel: 00000000 00237810 bfffff00 0010a7fa 00000003 00000001 00000000 bfffff00

Aug 29 09:51:01 blizard kernel: bffffdb3 bffffed4 fffffda 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]

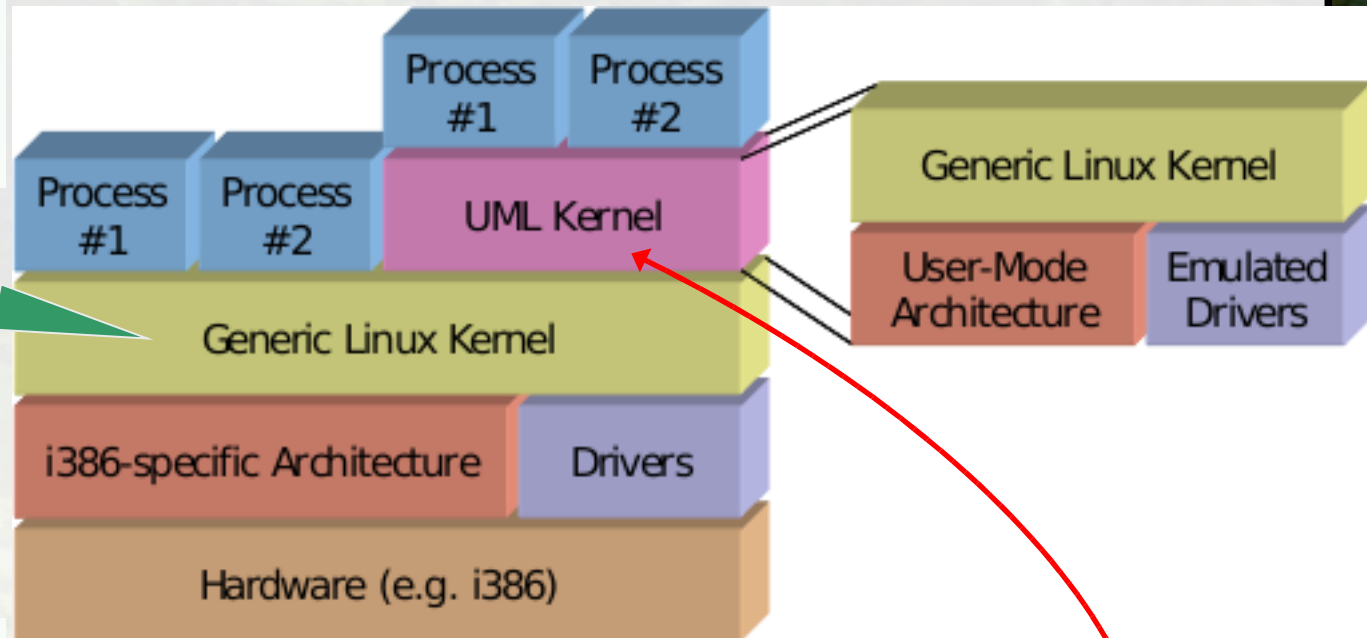
User-Mode Linux (1)

- 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)

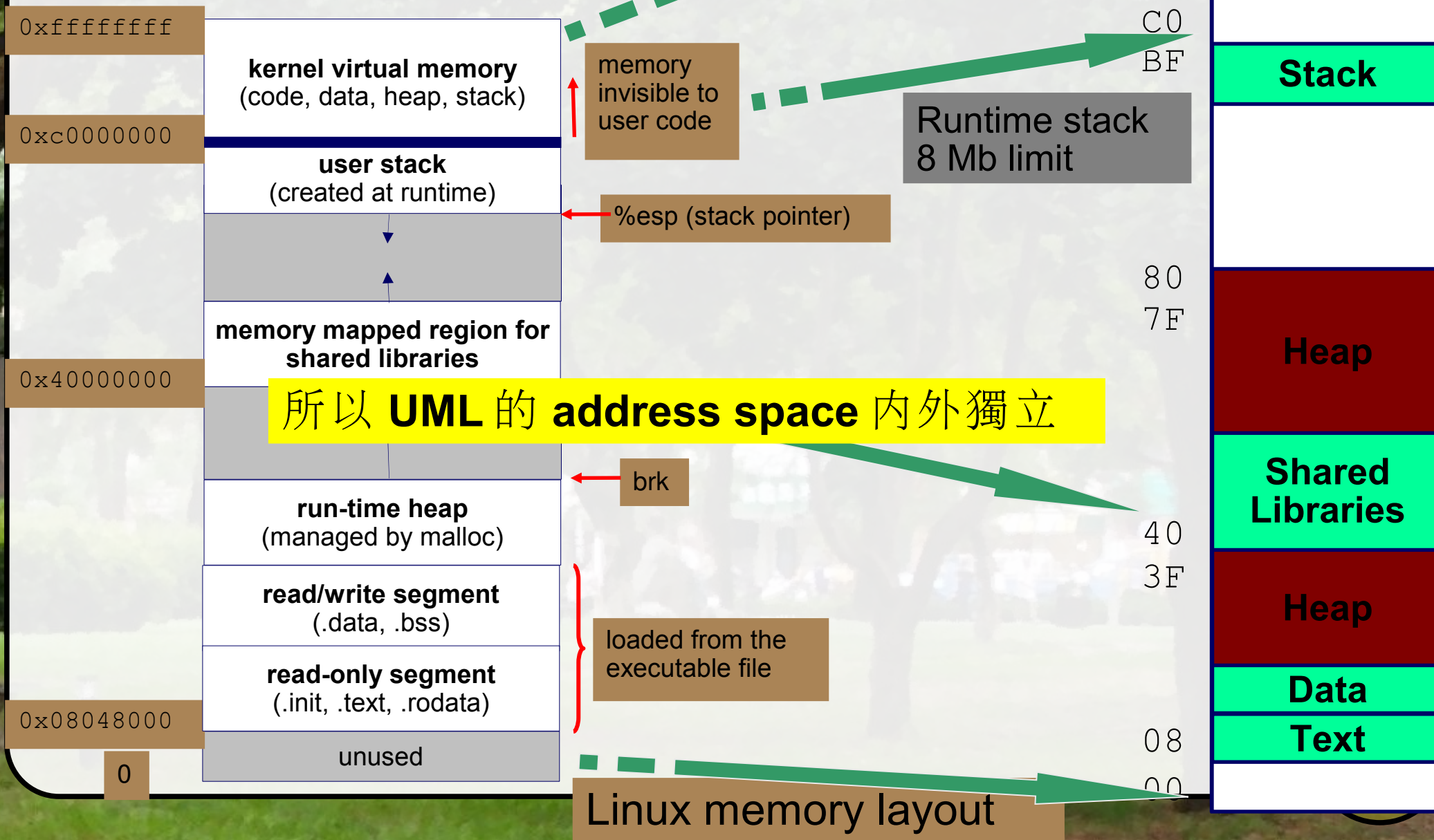
```
+-----+-----+-----+
| Process 1 | Process 2 | ... |
+-----+-----+-----+
|           Linux Kernel           |
+-----+-----+-----+
|           Hardware               |
+-----+-----+-----+
```



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

每個 process 都有獨自的 address space

Address Space



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");
170     }
171
172     for(i = 0; i < argc; i++){
173         new_argv[i] = argv[i];
174     }
175     new_argv[i] = NULL;
176
177     if(!linux_prog){
178         perror("No linux prog");
179         exit(1);
180     }
181
182     if(!linux_argv){
183         perror("No linux argv");
184         exit(1);
185     }
186
187     if(!linux_envp){
188         perror("No linux envp");
189         exit(1);
190     }
191
192     if(!linux_cwd){
193         perror("No linux cwd");
194         exit(1);
195     }
196
197     if(!linux_uid){
198         perror("No linux uid");
199         exit(1);
200     }
201
202     if(!linux_gid){
203         perror("No linux gid");
204         exit(1);
205     }
206
207     if(!linux_iops){
208         perror("No linux iops");
209         exit(1);
210     }
211
212     if(!linux_oops){
213         perror("No linux oops");
214         exit(1);
215     }
216
217     if(!linux_mops){
218         perror("No linux mops");
219         exit(1);
220     }
221
222     if(!linux_sops){
223         perror("No linux sops");
224         exit(1);
225     }
226
227     if(!linux_lops){
228         perror("No linux lops");
229         exit(1);
230     }
231
232     if(!linux_fops){
233         perror("No linux fops");
234         exit(1);
235     }
236
237     if(!linux_cops){
238         perror("No linux cops");
239         exit(1);
240     }
241
242     if(!linux_dops){
243         perror("No linux dops");
244         exit(1);
245     }
246
247     if(!linux_uops){
248         perror("No linux uops");
249         exit(1);
250     }
251
252     if(!linux_sops){
253         perror("No linux sops");
254         exit(1);
255     }
256
257     if(!linux_lops){
258         perror("No linux lops");
259         exit(1);
260     }
261
262     if(!linux_fops){
263         perror("No linux fops");
264         exit(1);
265     }
266
267     if(!linux_cops){
268         perror("No linux cops");
269         exit(1);
270     }
271
272     if(!linux_dops){
273         perror("No linux dops");
274         exit(1);
275     }
276
277     if(!linux_uops){
278         perror("No linux uops");
279         exit(1);
280     }
281
282     if(!linux_sops){
283         perror("No linux sops");
284         exit(1);
285     }
286
287     if(!linux_lops){
288         perror("No linux lops");
289         exit(1);
290     }
291
292     if(!linux_fops){
293         perror("No linux fops");
294         exit(1);
295     }
296
297     if(!linux_cops){
298         perror("No linux cops");
299         exit(1);
300     }
301
302     if(!linux_dops){
303         perror("No linux dops");
304         exit(1);
305     }
306
307     if(!linux_uops){
308         perror("No linux uops");
309         exit(1);
310     }
311
312     if(!linux_sops){
313         perror("No linux sops");
314         exit(1);
315     }
316
317     if(!linux_lops){
318         perror("No linux lops");
319         exit(1);
320     }
321
322     if(!linux_fops){
323         perror("No linux fops");
324         exit(1);
325     }
326
327     if(!linux_cops){
328         perror("No linux cops");
329         exit(1);
330     }
331
332     if(!linux_dops){
333         perror("No linux dops");
334         exit(1);
335     }
336
337     if(!linux_uops){
338         perror("No linux uops");
339         exit(1);
340     }
341
342     if(!linux_sops){
343         perror("No linux sops");
344         exit(1);
345     }
346
347     if(!linux_lops){
348         perror("No linux lops");
349         exit(1);
350     }
351
352     if(!linux_fops){
353         perror("No linux fops");
354         exit(1);
355     }
356
357     if(!linux_cops){
358         perror("No linux cops");
359         exit(1);
360     }
361
362     if(!linux_dops){
363         perror("No linux dops");
364         exit(1);
365     }
366
367     if(!linux_uops){
368         perror("No linux uops");
369         exit(1);
370     }
371
372     if(!linux_sops){
373         perror("No linux sops");
374         exit(1);
375     }
376
377     if(!linux_lops){
378         perror("No linux lops");
379         exit(1);
380     }
381
382     if(!linux_fops){
383         perror("No linux fops");
384         exit(1);
385     }
386
387     if(!linux_cops){
388         perror("No linux cops");
389         exit(1);
390     }
391
392     if(!linux_dops){
393         perror("No linux dops");
394         exit(1);
395     }
396
397     if(!linux_uops){
398         perror("No linux uops");
399         exit(1);
400     }
401
402     if(!linux_sops){
403         perror("No linux sops");
404         exit(1);
405     }
406
407     if(!linux_lops){
408         perror("No linux lops");
409         exit(1);
410     }
411
412     if(!linux_fops){
413         perror("No linux fops");
414         exit(1);
415     }
416
417     if(!linux_cops){
418         perror("No linux cops");
419         exit(1);
420     }
421
422     if(!linux_dops){
423         perror("No linux dops");
424         exit(1);
425     }
426
427     if(!linux_uops){
428         perror("No linux uops");
429         exit(1);
430     }
431
432     if(!linux_sops){
433         perror("No linux sops");
434         exit(1);
435     }
436
437     if(!linux_lops){
438         perror("No linux lops");
439         exit(1);
440     }
441
442     if(!linux_fops){
443         perror("No linux fops");
444         exit(1);
445     }
446
447     if(!linux_cops){
448         perror("No linux cops");
449         exit(1);
450     }
451
452     if(!linux_dops){
453         perror("No linux dops");
454         exit(1);
455     }
456
457     if(!linux_uops){
458         perror("No linux uops");
459         exit(1);
460     }
461
462     if(!linux_sops){
463         perror("No linux sops");
464         exit(1);
465     }
466
467     if(!linux_lops){
468         perror("No linux lops");
469         exit(1);
470     }
471
472     if(!linux_fops){
473         perror("No linux fops");
474         exit(1);
475     }
476
477     if(!linux_cops){
478         perror("No linux cops");
479         exit(1);
480     }
481
482     if(!linux_dops){
483         perror("No linux dops");
484         exit(1);
485     }
486
487     if(!linux_uops){
488         perror("No linux uops");
489         exit(1);
490     }
491
492     if(!linux_sops){
493         perror("No linux sops");
494         exit(1);
495     }
496
497     if(!linux_lops){
498         perror("No linux lops");
499         exit(1);
500     }
501
502     if(!linux_fops){
503         perror("No linux fops");
504         exit(1);
505     }
506
507     if(!linux_cops){
508         perror("No linux cops");
509         exit(1);
510     }
511
512     if(!linux_dops){
513         perror("No linux dops");
514         exit(1);
515     }
516
517     if(!linux_uops){
518         perror("No linux uops");
519         exit(1);
520     }
521
522     if(!linux_sops){
523         perror("No linux sops");
524         exit(1);
525     }
526
527     if(!linux_lops){
528         perror("No linux lops");
529         exit(1);
530     }
531
532     if(!linux_fops){
533         perror("No linux fops");
534         exit(1);
535     }
536
537     if(!linux_cops){
538         perror("No linux cops");
539         exit(1);
540     }
541
542     if(!linux_dops){
543         perror("No linux dops");
544         exit(1);
545     }
546
547     if(!linux_uops){
548         perror("No linux uops");
549         exit(1);
550     }
551
552     if(!linux_sops){
553         perror("No linux sops");
554         exit(1);
555     }
556
557     if(!linux_lops){
558         perror("No linux lops");
559         exit(1);
560     }
561
562     if(!linux_fops){
563         perror("No linux fops");
564         exit(1);
565     }
566
567     if(!linux_cops){
568         perror("No linux cops");
569         exit(1);
570     }
571
572     if(!linux_dops){
573         perror("No linux dops");
574         exit(1);
575     }
576
577     if(!linux_uops){
578         perror("No linux uops");
579         exit(1);
580     }
581
582     if(!linux_sops){
583         perror("No linux sops");
584         exit(1);
585     }
586
587     if(!linux_lops){
588         perror("No linux lops");
589         exit(1);
590     }
591
592     if(!linux_fops){
593         perror("No linux fops");
594         exit(1);
595     }
596
597     if(!linux_cops){
598         perror("No linux cops");
599         exit(1);
600     }
601
602     if(!linux_dops){
603         perror("No linux dops");
604         exit(1);
605     }
606
607     if(!linux_uops){
608         perror("No linux uops");
609         exit(1);
610     }
611
612     if(!linux_sops){
613         perror("No linux sops");
614         exit(1);
615     }
616
617     if(!linux_lops){
618         perror("No linux lops");
619         exit(1);
620     }
621
622     if(!linux_fops){
623         perror("No linux fops");
624         exit(1);
625     }
626
627     if(!linux_cops){
628         perror("No linux cops");
629         exit(1);
630     }
631
632     if(!linux_dops){
633         perror("No linux dops");
634         exit(1);
635     }
636
637     if(!linux_uops){
638         perror("No linux uops");
639         exit(1);
640     }
641
642     if(!linux_sops){
643         perror("No linux sops");
644         exit(1);
645     }
646
647     if(!linux_lops){
648         perror("No linux lops");
649         exit(1);
650     }
651
652     if(!linux_fops){
653         perror("No linux fops");
654         exit(1);
655     }
656
657     if(!linux_cops){
658         perror("No linux cops");
659         exit(1);
660     }
661
662     if(!linux_dops){
663         perror("No linux dops");
664         exit(1);
665     }
666
667     if(!linux_uops){
668         perror("No linux uops");
669         exit(1);
670     }
671
672     if(!linux_sops){
673         perror("No linux sops");
674         exit(1);
675     }
676
677     if(!linux_lops){
678         perror("No linux lops");
679         exit(1);
680     }
681
682     if(!linux_fops){
683         perror("No linux fops");
684         exit(1);
685     }
686
687     if(!linux_cops){
688         perror("No linux cops");
689         exit(1);
690     }
691
692     if(!linux_dops){
693         perror("No linux dops");
694         exit(1);
695     }
696
697     if(!linux_uops){
698         perror("No linux uops");
699         exit(1);
700     }
701
702     if(!linux_sops){
703         perror("No linux sops");
704         exit(1);
705     }
706
707     if(!linux_lops){
708         perror("No linux lops");
709         exit(1);
710     }
711
712     if(!linux_fops){
713         perror("No linux fops");
714         exit(1);
715     }
716
717     if(!linux_cops){
718         perror("No linux cops");
719         exit(1);
720     }
721
722     if(!linux_dops){
723         perror("No linux dops");
724         exit(1);
725     }
726
727     if(!linux_uops){
728         perror("No linux uops");
729         exit(1);
730     }
731
732     if(!linux_sops){
733         perror("No linux sops");
734         exit(1);
735     }
736
737     if(!linux_lops){
738         perror("No linux lops");
739         exit(1);
740     }
741
742     if(!linux_fops){
743         perror("No linux fops");
744         exit(1);
745     }
746
747     if(!linux_cops){
748         perror("No linux cops");
749         exit(1);
750     }
751
752     if(!linux_dops){
753         perror("No linux dops");
754         exit(1);
755     }
756
757     if(!linux_uops){
758         perror("No linux uops");
759         exit(1);
760     }
761
762     if(!linux_sops){
763         perror("No linux sops");
764         exit(1);
765     }
766
767     if(!linux_lops){
768         perror("No linux lops");
769         exit(1);
770     }
771
772     if(!linux_fops){
773         perror("No linux fops");
774         exit(1);
775     }
776
777     if(!linux_cops){
778         perror("No linux cops");
779         exit(1);
780     }
781
782     if(!linux_dops){
783         perror("No linux dops");
784         exit(1);
785     }
786
787     if(!linux_uops){
788         perror("No linux uops");
789         exit(1);
790     }
791
792     if(!linux_sops){
793         perror("No linux sops");
794         exit(1);
795     }
796
797     if(!linux_lops){
798         perror("No linux lops");
799         exit(1);
800     }
801
802     if(!linux_fops){
803         perror("No linux fops");
804         exit(1);
805     }
806
807     if(!linux_cops){
808         perror("No linux cops");
809         exit(1);
810     }
811
812     if(!linux_dops){
813         perror("No linux dops");
814         exit(1);
815     }
816
817     if(!linux_uops){
818         perror("No linux uops");
819         exit(1);
820     }
821
822     if(!linux_sops){
823         perror("No linux sops");
824         exit(1);
825     }
826
827     if(!linux_lops){
828         perror("No linux lops");
829         exit(1);
830     }
831
832     if(!linux_fops){
833         perror("No linux fops");
834         exit(1);
835     }
836
837     if(!linux_cops){
838         perror("No linux cops");
839         exit(1);
840     }
841
842     if(!linux_dops){
843         perror("No linux dops");
844         exit(1);
845     }
846
847     if(!linux_uops){
848         perror("No linux uops");
849         exit(1);
850     }
851
852     if(!linux_sops){
853         perror("No linux sops");
854         exit(1);
855     }
856
857     if(!linux_lops){
858         perror("No linux lops");
859         exit(1);
860     }
861
862     if(!linux_fops){
863         perror("No linux fops");
864         exit(1);
865     }
866
867     if(!linux_cops){
868         perror("No linux cops");
869         exit(1);
870     }
871
872     if(!linux_dops){
873         perror("No linux dops");
874         exit(1);
875     }
876
877     if(!linux_uops){
878         perror("No linux uops");
879         exit(1);
880     }
881
882     if(!linux_sops){
883         perror("No linux sops");
884         exit(1);
885     }
886
887     if(!linux_lops){
888         perror("No linux lops");
889         exit(1);
890     }
891
892     if(!linux_fops){
893         perror("No linux fops");
894         exit(1);
895     }
896
897     if(!linux_cops){
898         perror("No linux cops");
899         exit(1);
900     }
901
902     if(!linux_dops){
903         perror("No linux dops");
904         exit(1);
905     }
906
907     if(!linux_uops){
908         perror("No linux uops");
909         exit(1);
910     }
911
912     if(!linux_sops){
913         perror("No linux sops");
914         exit(1);
915     }
916
917     if(!linux_lops){
918         perror("No linux lops");
919         exit(1);
920     }
921
922     if(!linux_fops){
923         perror("No linux fops");
924         exit(1);
925     }
926
927     if(!linux_cops){
928         perror("No linux cops");
929         exit(1);
930     }
931
932     if(!linux_dops){
933         perror("No linux dops");
934         exit(1);
935     }
936
937     if(!linux_uops){
938         perror("No linux uops");
939         exit(1);
940     }
941
942     if(!linux_sops){
943         perror("No linux sops");
944         exit(1);
945     }
946
947     if(!linux_lops){
948         perror("No linux lops");
949         exit(1);
950     }
951
952     if(!linux_fops){
953         perror("No linux fops");
954         exit(1);
955     }
956
957     if(!linux_cops){
958         perror("No linux cops");
959         exit(1);
960     }
961
962     if(!linux_dops){
963         perror("No linux dops");
964         exit(1);
965     }
966
967     if(!linux_uops){
968         perror("No linux uops");
969         exit(1);
970     }
971
972     if(!linux_sops){
973         perror("No linux sops");
974         exit(1);
975     }
976
977     if(!linux_lops){
978         perror("No linux lops");
979         exit(1);
980     }
981
982     if(!linux_fops){
983         perror("No linux fops");
984         exit(1);
985     }
986
987     if(!linux_cops){
988         perror("No linux cops");
989         exit(1);
990     }
991
992     if(!linux_dops){
993         perror("No linux dops");
994         exit(1);
995     }
996
997     if(!linux_uops){
998         perror("No linux uops");
999         exit(1);
1000    }
```

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

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)

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:~/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
- 以 `xscale PXA27x` 為例 ...

使用 **target** 的 **ld-linux.so.2**

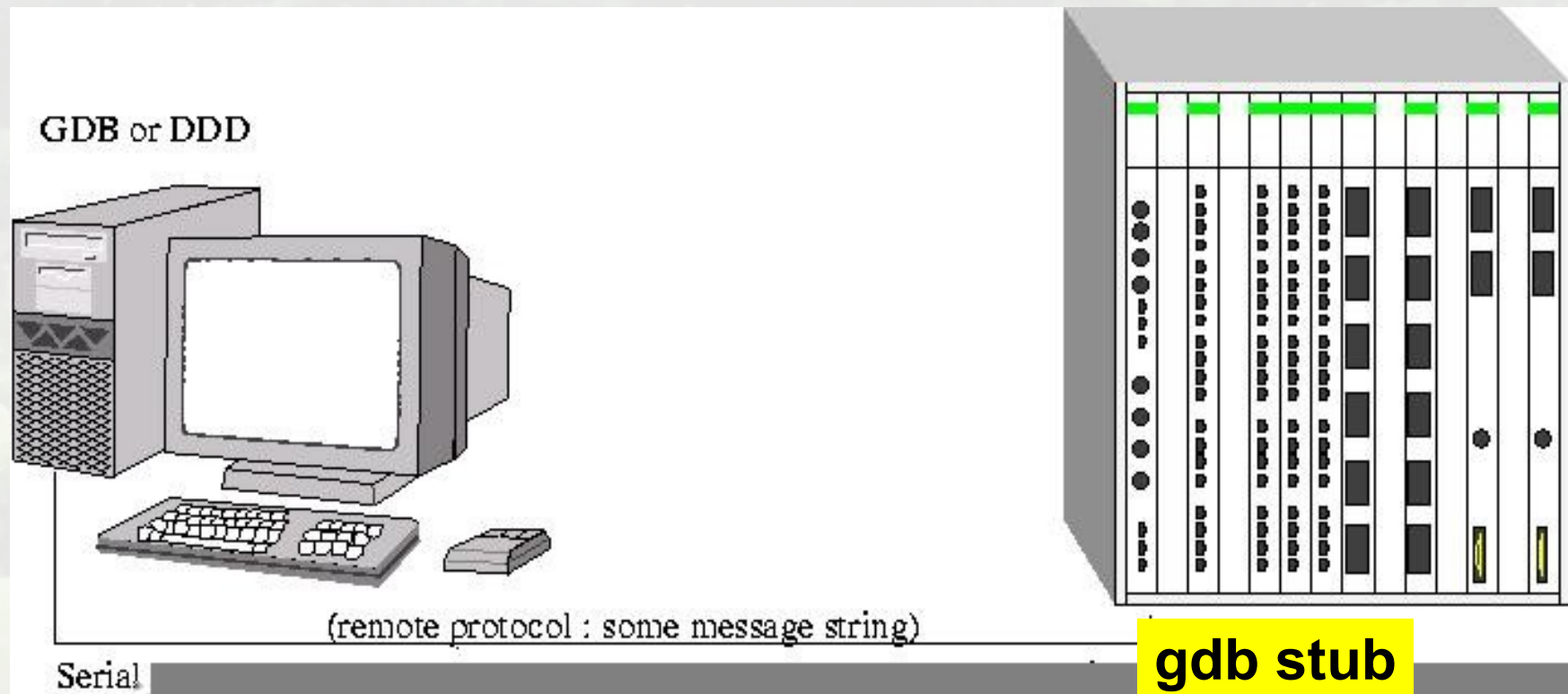
```
~/poky/build/tmp$ file ./rootfs/bin/busybox
./rootfs/bin/busybox: ELF 32-bit LSB executable, ARM, version 1 (ARM), for GNU/Linux 2.4.0, dynamically linked (uses shared libs), for GNU/Linux 2.4.0, stripped
~/poky/build/tmp$ ./qemu-arm ./rootfs/lib/ld-linux.so.2 \
--library-path ./rootfs/lib ./rootfs/bin/busybox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



開發平台 (Host)
運作完整的 GDB

Qemu 所模擬的機器

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(filename

274 if (IS_ERR(filename))

275 goto out;

276 error = do_execve(filename

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

856

857

858

859

860

861

862

863

864

865

866

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

981

982

983

984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

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

模倣 Sharp Zaurus PDA

以 Remote GDB 分析

Breakpoint 1, sys_execve (filename=0e514) at arch/arm/kernel/sys_arm.c:268

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)

系統提供的偵錯 機制



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

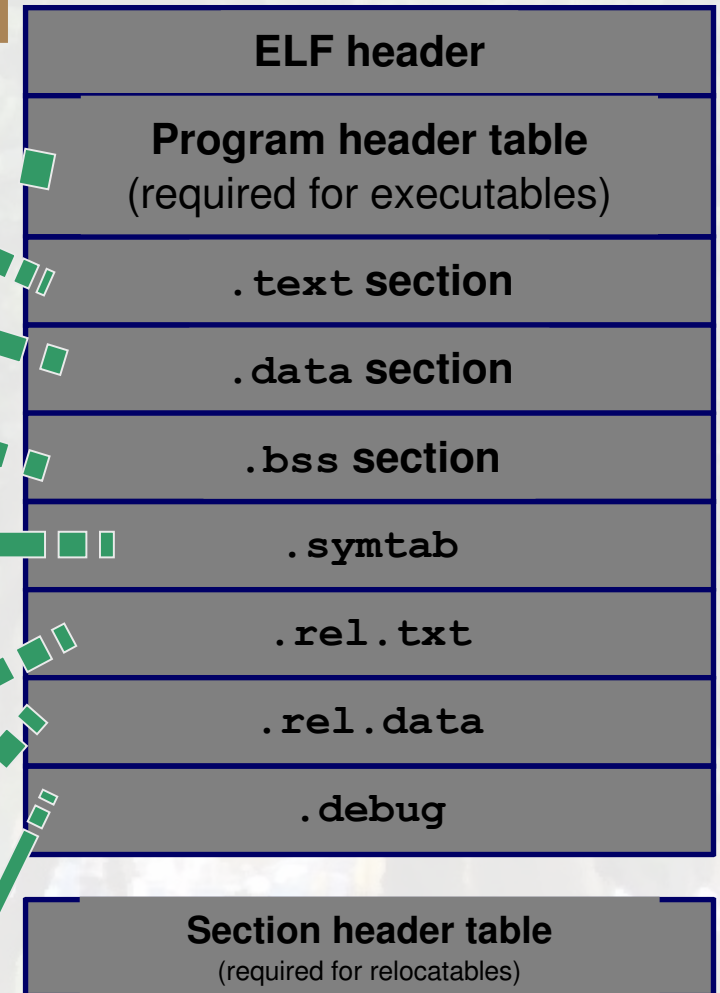
- ◆ Un-initialized (static) data
- ◆ Block started by symbol
- ◆ **Has section header but occupies no space**

- ◆ **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



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

.debug : gcc -g

gdb 初體驗



fibonacci.c

```
12             f2 = fib(n - 2);
13             return f1 + f2;
14         }
15     }
16
17     int main()
18     {
19         printf("%d\n", fib(5));
20         return 0;
21     }
22
23
24
25
26
27
```

B+> 19 printf("%d\n", fib(5));

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)

File Edit View Program Commands Status Source Data Help

0: a.c:26

Lookup Find> Clear Watch Print Display Plot Show Rotate Set Undisp

```
int profile1();
static int profile2();
int profile3();

main()
{
    int number = 5;

    printf("calling b\n");
    b(number);
    printf("returned from b\n");
    profile1();
}

int profile1()
{
    printf("profile1\n");
    profile2();
}

static int profile2()
{
    int bob;
    int fred=3;

    printf("profile2\n");
    profile3();
}

int profile3()
{
    printf("profile3\n");
}

int utility(int a)
{
    printf("in utility\n");
    printf("a is %d\n",a);
}
```

in c, num = 5, newnum = 10
returned from c
returned from b
profile1
profile2

Breakpoint 2, profile2 () at a.c:26
(gdb) print fred
\$1 = 3
(gdb) I

Run
Interrupt
Step StepI
Next NextI
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make

\$1 = 3

ddd : 歷史悠久的 gdb 前端 (Motif)

gdb 初體驗 (1)

- 進入 gdb
 - `gdb filename`
- 列出程式碼
 - `(gdb) list`
 - `(gdb) list 3,9`
 - `(gdb) list ip_vs_in`
- 執行程式
 - `(gdb) run`
- 暫時回到 Linux 提示符號
 - `(gdb) shell`
 - `(gdb) shell ls`

預設一次列出 10 行

欲中斷按下 Ctrl-C

回到 gdb 提示符號: **exit**

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         }
(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          }

```

b = break

r = run

dpm.cc

l = list

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)


```
#include <stdio.h>
```

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

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

hello.c

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.
```

```
(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.
```

```
#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) 1  
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

```
#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);  
    }  
}
```

遞迴

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

fibonacci.c

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) {
```

一旦執行，可再加入中斷條件，
如 `break 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          /* Ending
```

```
6          if (n ==
```

```
7              retur
```

```
8          }
```

(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/qsrt/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}

\$1, \$2, \$3, ... 為歷史變數

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 (gdb) **b quicksort**

31 int last = left, Breakpoint 1 at 0x80484d7: file quicksort.c, line 31.

\$5 = {-2, 3, 1, 3, 0, 4, (gdb) **commands**

Breakpoint 1, quicksort 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
```

十進位

```
$1 = 100
```

```
(gdb) p/x 100
```

十六進位

```
$2 = 0x64
```

```
(gdb) p/o 100
```

八進位

```
$3 = 0144
```

```
(gdb) p/t 100
```

二進位

```
$4 = 1100100
```

```
(gdb)
```

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 -1**

```
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.sourceforge.net>

