Embedded Systems

Ch 10
Debugging Techniques

Byung Kook Kim
Dept of EECS
Korea Advanced Institute of Science and Technology

Overview

- *1. Debugging Techniques*
- *2. Gdb in EZ-X5*

1. Debugging Techniques

- Kernel debugging
  - 일반적인 debugger로는 불가능하다.
    - Kernel code는 특정 process에 종속적이지 않은 기능의 집합이기 때문.

- Kernel debugging techniques
  - A. Printk
  - B. Queue debugging
  - C. Strace
  - D. System fault debugging
  - E. Debugger

A. Printk

- **일반적인** debugging
  - Application program: printf. **결과** monitoring
  - Kernel: printk.

- printk
  - **각각 다른** loglevel (log **출력의 우선 순위) 또는** message **우선순위를 적용하여**, **그들의** message **중요성에 따른** message **계층을 나눌 수 있다**.
    - printk(KERN_DDEBUG “Here I am: line %i₩n”, __LINE__);
    - printk(KERN_CRIT “I’m trashed: giving up on %p₩n”, ptr);
  - **만약** console_loglevel**보다 우선순위가 낮다면 이** message**는 출력되지 않는다**.
    - Console_loglevel **변수는** DEFAULT_CONSOLE_LOGLEVEL**로 초기화되며** sys_syslog system call**로 값을 바꿀 수 있다**.

Printk (II)

- Message 기록 방법
  - Printk 함수는 LOG_BUF_LEN 길이의 circular buffer에 message를 쓴다.
    - Circular buffer가 완전히 채워지면 printk는 앞의 message를 덮어씌우며 계속 진행한다.
  - Printk를 어디에서나 수행시킬 수 있다.
    - Interrupt service routine에서도 가능.
  - 사용하지 않는 xterm 하나를 열어 놓고 “cat /proc/kmesg” 명령을 실행한다.

B. Queue Debugging

- Printk의 단점
  - 많이 사용하면 system이 현저히 느려질 수 있다.
    - Syslogd가 출력을 file에 계속 보내려고 하고, 이로 인해 한줄 쓰기 마다 disk 동작이 일어나기 때문.

- 정보가 필요할 때 system에 질의하는 방법
  - /proc file system에 file을 만드는 것
  - ioctl driver method를 사용하는 것

Queue Debugging (II)

- **B1. /proc file system 이용**
  - /proc file system
    - 어떤 device에도 종속적이지 않다.
    - /proc file들은 읽혀질 때 kernel에 의해 생성된다.
    - Text file이므로 이해 가능하다.
    - /proc의 구현은 노드의 동적생성을 제공하는데, 사용자 모듈이 쉽게 정보를 검색할 수 있는 진입점을 생성하도록 허용한다.
    - /proc 안에 모든 특성을 가진 file node(read, write, seek, etc)를 생성시키기 위해서는 file_operations structure와 inode_operations structure를 모두 정의해야 한다.
  - 생성
    - proc_register_dynamic(&proc_root, &scull_proc_entry);
  - 소멸
    - proc_unregister(&proc_root, scull_proc_entry.low_ino);

Queue Debugging (III)

- **B2. ioctl method**
  - ioctl
    - File descriptor에 대하여 작용하는 system call
    - Driver를 시험할 시점에서 driver로부터 사용자 공간으로 관심있는 data structure를 복사해 준다.
    - Ioctl을 수행하고 그 결과를 보여주는 program을 추가로 만들어야 한다.
  - 장점
    - /proc을 읽는 것 보다 훨씬 빠르게 수행된다.
    - 전달하려는 data 양이 한 page(4 Kbyte)로 제한하지 않는다.
    - Debugging mode를 없앴을 때에도 driver에 ioctl debug 명령이 남아 있을 수 있다.
  - 단점
    - Module의 크기가 약간 커진다.

C. Strace

- strace **명령**
  - User space program**이 사용하는 모든** system call**을 보여주는 강력한 도구**.
  - **전달하는 인자와** return value**를** symbol **형태로 나타내 준다**.
  - System call**이 실패했을 때** error**의** symbol **값과 그 문자열을 출력한다**.
    - Ex: ENOMEM - Out of memory
  - **정보를** kernel **자체에서 받는다**.

- Strace command options
  - -t: Show time instant for each system call
  - -T: Show execution time for each system call
  - -o: Output redirection

Strace (II)

- *Example of strace*
  - % strace ls /dev > /dev/scull0
  - ...
  - readdir(3, {d_ino=894, d_name="scull0"}) = 1
  - ...
  - close(3) = 0
  - brk(0x8035000) = 0x8035000
  - ...
  - ioctl(1, TCGETS, 0xbffffac4) = -1 EINVAL (Invalid argument)
  - Write(1, "MAKEDEV₩nXOR₩narp₩natibm₩naudio₩n"..., 4096) = 4000
  - Write(1, "3₩nttyr4₩nttyr5₩nttyr6₩nttyr7₩nt"..., 96) = 96
  - ...
  - _exit(0)

D. System Fault Debugging

- Fault
  - Linux code는 대부분의 error에 대해 유연하게 대처할 수 있도록 견고하다.
  - Fault는 현 process를 파괴하지만 system은 계속 수행될 수 있다.
- Oops message
  - Shows registers, stack, code in hexadecimal
- Ksymoops
  - Oops message에 있는 주소값을 kernel symbol로 변환, code를 disassemble해 주는 utility.
- System halt
  - Infinite loop: scheduler cannot be called.
  - Special keyboard processing:
    - RightAlt-PrScr-m (Show memory): memory usage, buffer cache.
    - RightAlt-PrSCr-t (Show state): Show process state.
    - RightAlt-PrScr-p (Show registers): Show registers.

E. Debugger

- **Debugger**
  - Code를 한 줄씩 추적하며 변수값과 register를 조사한다.
  - 시간이 많이 소요되지만 code 전체에 대해 훤히 투시할 수 있다.

- **E1. gdb**
  - Kernel을 하나의 응용 program으로 간주하고 실행해야 한다.
  - > gdb /usr/src/linux/vmlinux /proc/kcore
    - 1st arg: 압축하지 않은 kernel execution file의 이름
    - 2nd arg: core file name (읽혀질 때 생성)
  - Kernel data의 변경은 불가.
  - Usage:
    - p jiffies: Show clock count (from boot-up)
  - p *module_list, p *module_list->next, p *chrdev[4]->fops: Show structures
  - x /20i: Disassemble 20 lines.

Debugger (II)

- **E2. kdebug**
  - 실행중인 kernel과 교신할 수 있는 gdb의 remote debugging interface를 사용하는 작은 도구
  - /dev/kdebug: Kernel 공간에 접근하는 통신 channel
    - Module 그 자체는 kernel space에서 동작하기 때문에 일반 debugger로 접근할 수 없는 kernel address space를 조사할 수 있다.
  - Kdebug에서도 kernel code 안으로 code의 단계적 수행과 정지점을 지정하지 못한다.
    - System이 계속 정상적으로 동작하도록 하기 위해서 kernel이 항상 실행되고 있어야 하기 때문에 불가피.
  - Debug하고 있는 kernel의 data 항목을 바꾸는 것, 가변인자를 전달하여 함수를 호출하는 것, module이 점유하고 있는 주소영역을 읽기 쓰기 형태로 접근하는 것을 허용한다.

Debugger (III)

- **E3. Remote debugging**
  - Two computers connected with serial line:
    - 1st: Executing gdb
    - 2nd: Ececuting kernel to be debugged.
  - Gdb **제어는 제어하려는** kernel**의** binary format**을 이해할 수 있어야 한다**.
  - Debugger**는** target platform **지원이 가능하도록** compile**된 것이어야 한다**.
  - Kernel**의 초기화 함수**
    - **자신의 중단점들을 처리하고**, **중단점으로** jump**하도록** setup**하는** routine.
    - Kernel**의 표준 수행을 중단하고 중단점** service routine**으로 제어를 전달한다**.
    - Serial line**을 통해서** gdb**로부터의 명령을 기다렸다가 그것을 수행한다**.

2. Gdb in EZ-X5

- **프로그램 개발**
  - Understand the problem
  - Design your own algorithm
  - Edit the source program
  - Compile & link
    - Compile error: Edit & recompile
  - Test
    - printk**나** printf**문을 대량 살포 하여 꼭꼭 숨은 버그가 걸리기 만을 기다림: 내 거미줄에만 걸려 주면 좋겠는데... ㅠㅠㅠ**
    - **Gdb**
      - gcc**로 프로그램한 코드를 추적**
      - **어디서 결정적인 버그가 발생 하는지 적극적으로 찾아 나설 수 있다.**
      - gdb**서버를 이용하여 타겟보드에서 발생하는 오류까지도 찾아 낼 수 있다!**
  - Improve the algorithm: **실행 속도 향상**.

Gdb in EZ-X5 (II)

Installing gdb in EZ-X5

- **1. 파일 찾기**
  - gdb 5.2.1: gdb **개발 호스트와** gdbserver **타겟보드 패키지**
  - ftp://ftp.gnu.org./pub/gnu/gdb/gdb-5.2.1.tar.gz
  - Mirror: ftp://linux.sarang.net/pub/mirror/gnu/gnu/gdb/gdb-5.2.1.tar.gz

- **2. 컴파일 하기**
  - gcc**는 크로스 컴파일이 가능 하기 때문에 각 플랫폼별로 컴파일러의 버전이 다르므로,** gdb**도 각각의 용도에 맞춰 컴파일을 다시 수행해야 한다.**
  - EZ-X5**는** ARM **기반의** MCU**를 쓰고 있기 때문에** ARM**용** gdb**컴파일 수행.**

Gdb in EZ-X5 (III)

- ARM **용** gdb **컴파일** commands
  - $ cd /home/embedded/arm/
  - $ tar xvzf gdb-5.2.1.tar.gz
  - # cd gdb-5.2.1
  - $./configure --target=arm-linux --prefix=/usr/local/arm-dev –v
    - **어떤 컴파일러로 할 지 지정**: arm-linux
    - **결과물이 어디로 들어 갈 것인가 지정**: /usr/local/arm-dev
  - $make
  - $su - root
  - Password: *****
  - #make install
    - gdb**에서 생성된 라이브러리가 지정장소에 설치**
  - **이렇게 하여 gdb호스트를 컴파일 완료.**

Gdb in EZ-X5 (IV)

- 3. gdb server 설치
  - PATH에 컴파일 하여 생성한 들어간 gdb관련 파일이 있는 위치를 등록.
    - $cd /home/embedded/arm/gdb-5.2.1
    - $export PATH='echo $PATH':/usr/local/arm-dev/bin
  - 다시 configure를 실행
    - $./configure --target=arm-linux --host=arm-linux -v
  - sed 부분은 편집기를 열어 HAVE_SYS_REG_H를 선언한 부분을 주석처리
    - $cd gdb/gdbserver
    - $cp config.h config.h.org
    - $sed "s/#define HAVE_SYS_REG_H 1/₩/*#define HAVE_SYS_REG_H 1*₩//1" config.h.org > config.h
  - gdb전체를 다시 컴파일 할 필요 없이 /gdb/gdbserver 디렉터리에서 명시적으로 gdbserver를 컴파일. make 하는데 옵션은 cross gcc로 설정.
    - $make CC=/usr/armv5l-linux/bin/arm-linux-gcc LD=/usr/armv5l-linux/bin/ld

Gdb in EZ-X5 (V)

- **4. Gdb 사용**
  - **개발 호스트와 타겟보드 사이에는 TCP/IP통신이 되어야 한다.** gdb는 시리얼 통신에 의한 디버깅도 가능하다.
    - **양쪽에 모두 IP주소가 설정되어 있어야 한다.**
  - **실행 파일을 컴파일 할 때 -g 옵션을 주어 컴파일 해야 한다.**
    - gdb**가 실행 파일로부터 정보를 얻어 올 수 있기 때문**
  - **컴파일이 끝나면** gdbserver**와 실행 파일을 타겟보드에 카피한다.**
  - **타겟 보드 설정**
    - [root@EZ-X5 /gdbserver]$ ./gdbserver 192.168.10.150:6161 [**실행파일이름**]
    - Process debug create; pid = 994
      - IP **뒤의** port **번호는** 1024 **이상을 지정해 주는 것이 좋다.**
      - pid**는 수행 할 때 마다 달라질 수 있다.**

Gdb in EZ-X5 (VI)

- **Gdb 사용 (II)**
  - **개발 호스트 디버깅 본격적인 디버깅 작업은 개발 호스트에서 한다. 타겟보드에서 실행할 실행 파일이** gdb**와 같이 있어야 한다.**
    - [root@jdt gdb]$ gdb [**아까 만든 타겟보드에서 실행할 실행 파일의 이름**]
    - GNU gdb 5.2
    - Copyright 2002 Free Software Foundation, Inc.
    - GDB is free ...
    - (gdb)
  - Target board**를 설정해 준다**.
    - (gdb) target remote 192.168.10.51:6161
    - 192.168.10.51:6161: Success.
    - **본격적으로** gdb**가 타겟보드와 함께 수행**!
  - gdb**명령어를 사용해 디버깅 작업을 수행한다**.
    - Start with “C[ontinue]”.

Gdb in EZ-X5 (VII)

- ***Gdb commands***
  - # gdb filename [pid] Start gdb
  - Cmd options:
    - -help, -h Help
    - -symbols file Read symbol tables
    - -x file Execute gdb commands from file
    - -d directory Add directory for searching source files
    - -batch Run in batch mode
    - -cd directory Change working directory
    - -b bps Set serial line speed for remote debugging
    - -tty device (/dev/ttyb) Run using device for stdio

Gdb in EZ-X5 (VIII)

- ***Gdb Commands (II)***
  - > help name — Help command name
  - > list n1, n2 — List program lines
  - > break [file:]function — Set breakpoint
  - > run arglist — Run with argument list
  - > bt — Backtrace stack
  - > print expr — Print the value of expression
  - > continue — Continue execution
  - > next — Skip over function
  - > step — Step into function
  - > x addr — Examine memory
  - > q — Quit gdb

References

- Debugging Techinques
  - Alessandro Lubini, “Linux Device Drivers”, O’Reilly, 1998.

- Gdb in EZ-X5
  - http://www.falinux.com