

# Linux Debugging

- TRACE32 PowerDebug편 -

**MDS Technology**

# 목 차

---

1. Introduction
2. Linux Platform Overview
3. Linux Debugging 준비사항 및 설정
4. Bootloader
5. Linux Kernel
6. Linux Awareness
7. Daemon(Process)
8. Library
9. Kernel Module
10. Exception Debugging
11. 추가 실습

# 1. Introduction

---

본 과정을 통해 Linux Platform에서의 TRACE32를 이용한 디버깅 방법을 익히도록 합니다

1. 교육 목표 및 교재 설명
2. BSP 및 Hardware 구성
3. Basic Linux Development Environment

# 1-1. 교육 목표 및 교재 설명

---

본 교재는 Linux의 많은 영역에 대한 디버깅 실습으로 구성되어 있습니다

TRACE32를 이용하여 부트로더, 커널, 모듈, 라이브러리 그리고 프로세스 디버깅까지 실습을 통해 학습할 수 있습니다.



# 1-2. BSP 및 Hardware 구성

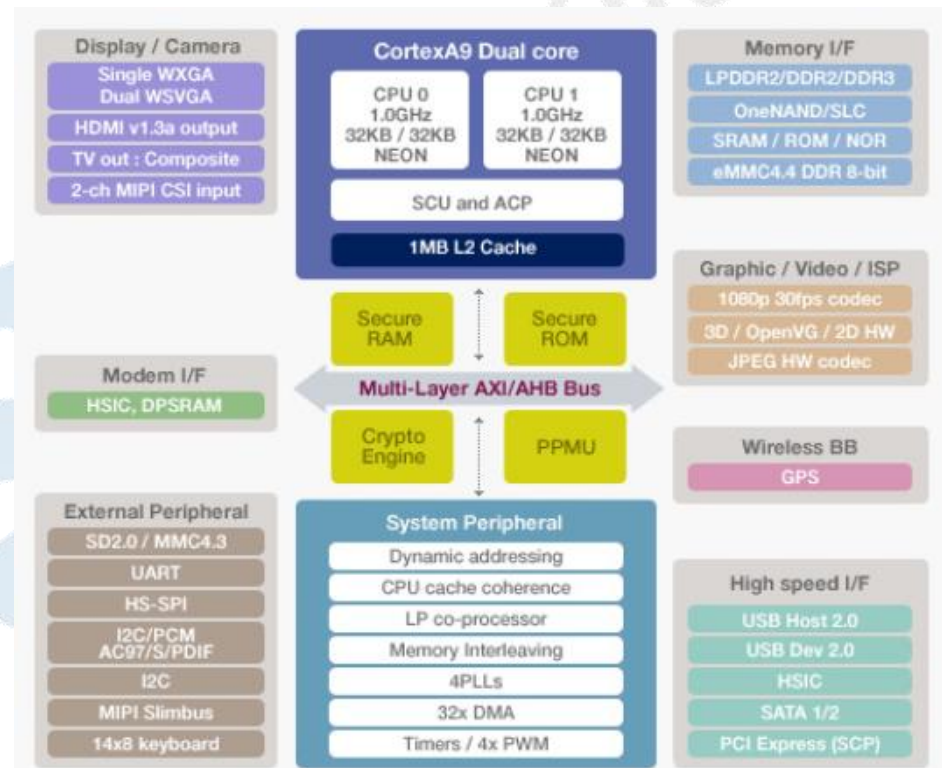
교육에 필요한 BSP 및 Hardware에 대해 확인을 합니다

## Bootloader / Kernel / File System / Application

- U-boot 1.3.4
- Linux kernel 3.0.15
- Ram Disk, YAFFS2 (EXT4)

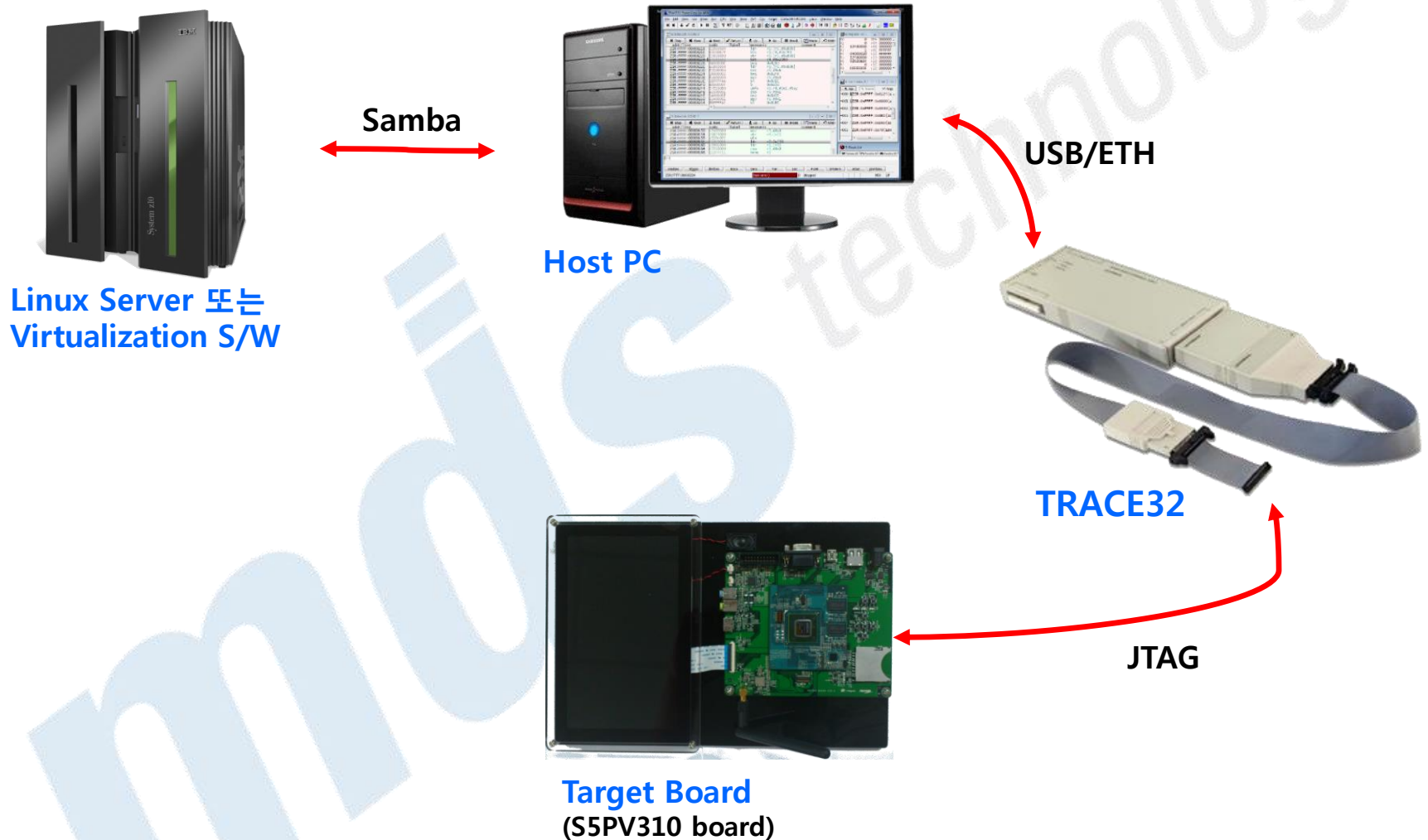
## Hardware Specification

- S5PV310 (Samsung Exynos4, ARMv7, Cortex-A9MPcore Dual)
- 1GB DRAM
- 7" 1024 x 600 LCD
- 1 Port SD/MMC card slot



# 1-3. Basic Linux Development 환경

Linux 개발 환경 구성 시에 기본적으로 갖추게 되는 환경에 대해 알아 봅니다



## 2. Linux Platform Overview

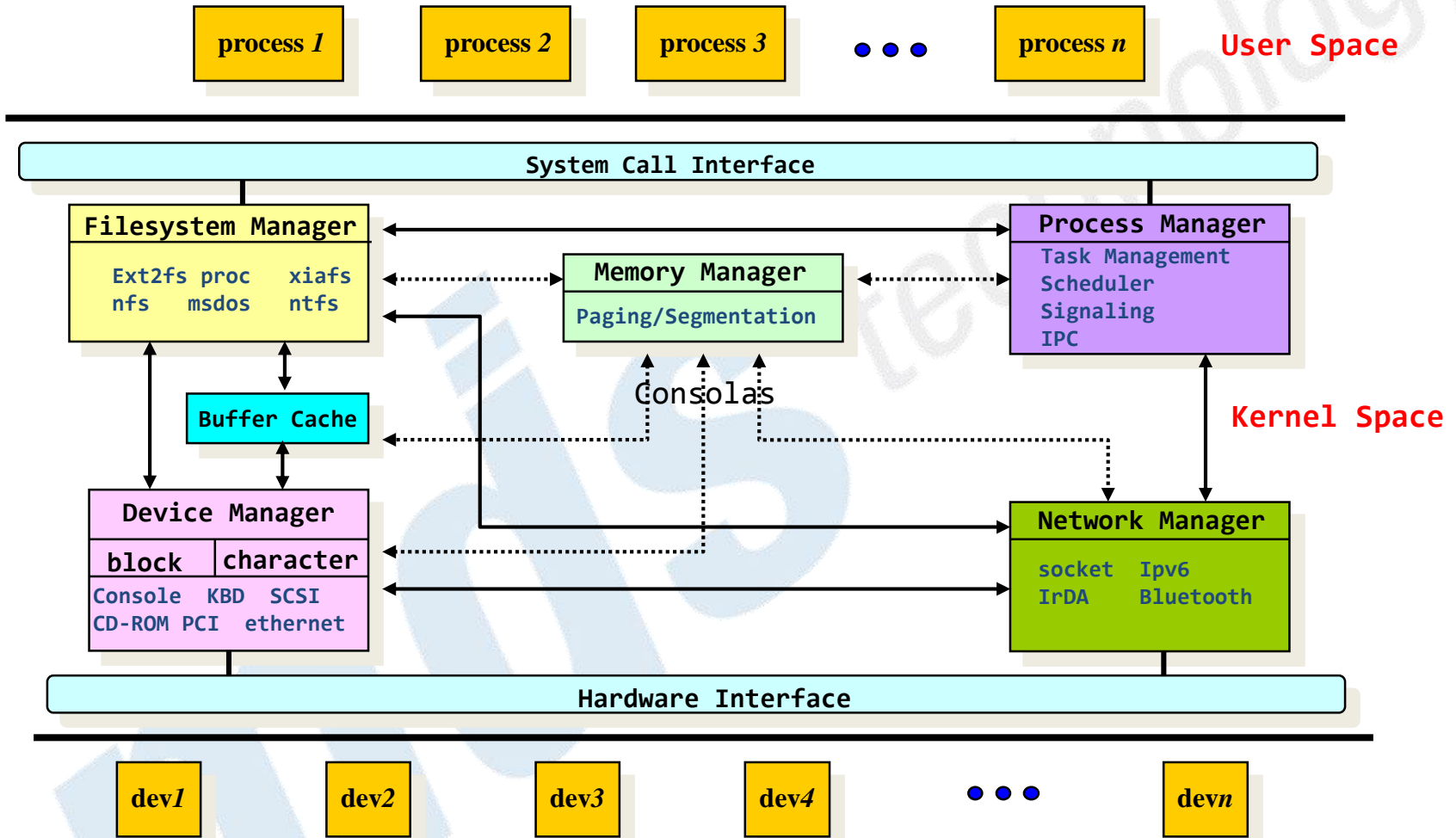
---

Linux System의 원활한 디버깅을 위해 Linux Platform에 대한 이해를 합니다

1. Linux OS 구조
2. Linux Software Stack
3. Linux File System
4. Linux Boot Sequence

## 2-1. Linux OS 구조

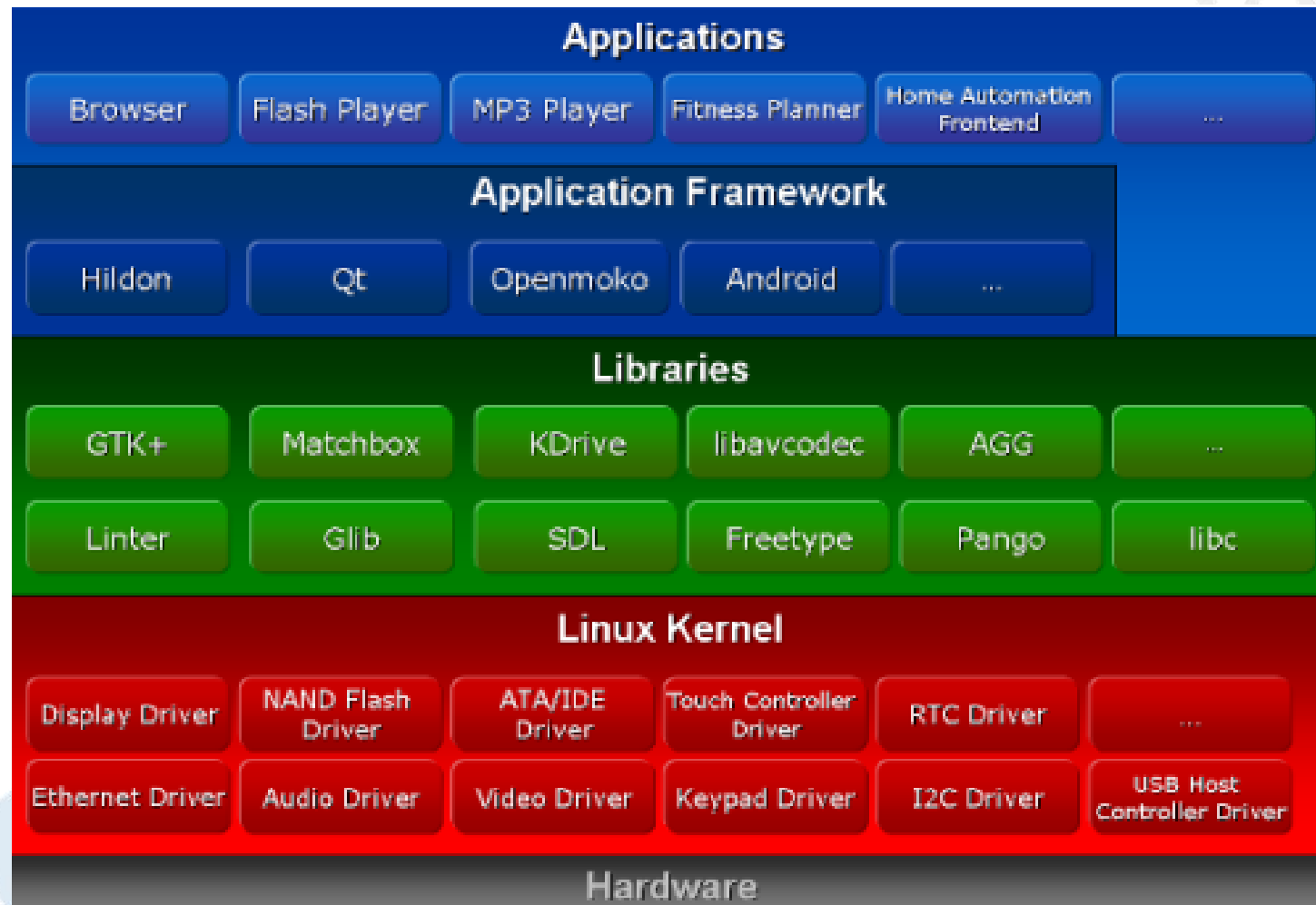
Linux OS는 HW자원 관리함과 동시에 APP서비스를 제공하는 역할을 합니다





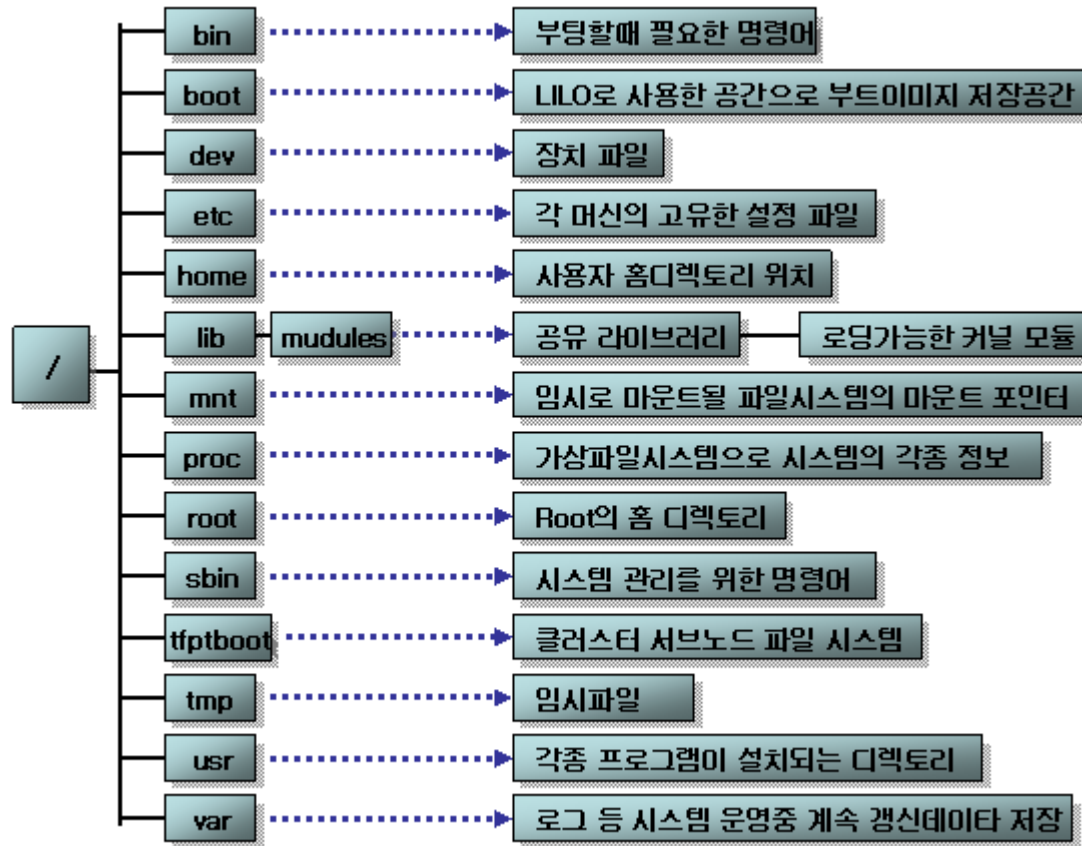
## 2-2. Linux Software Stack

Linux의 경우 3개의 Stack으로 구성(APP / LIB / Kernel )로 구성되어 있습니다



## 2-3. Linux File System

Linux에서 Root file System은 각 폴더 별로 고유의 기능을 담당하고 있습니다

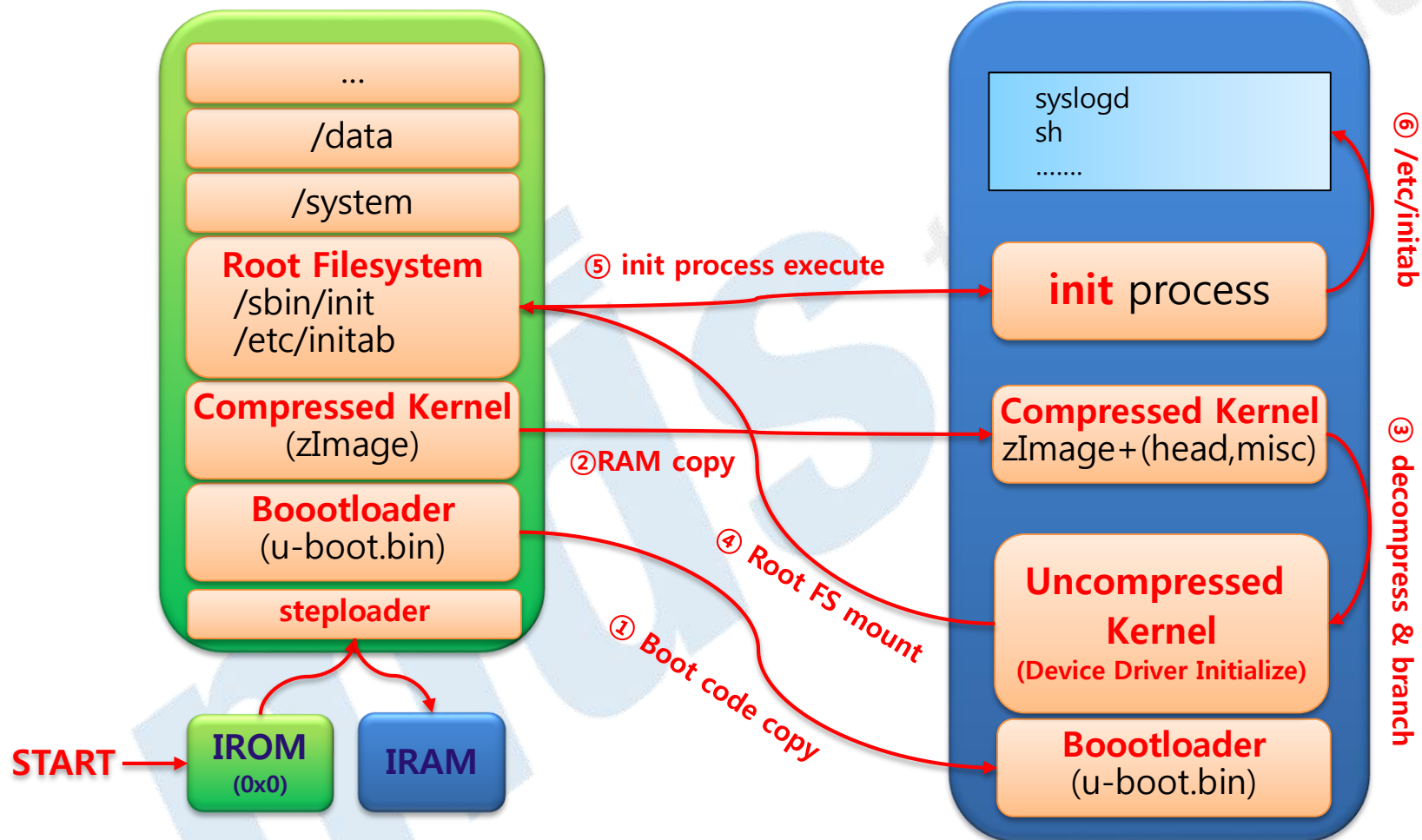


## 2-4. Linux Boot Sequence

Linux의 부트 시퀀스를 이해 할 수 있습니다

### NAND Flash Memory

### DRAM Memory



# 3. Linux Debugging 준비 사항

---

Linux System의 원활한 디버깅을 위해 Linux Platform에 대한 이해를 합니다

## 1. ELF 파일 생성

# 3-1. ELF 파일 생성

Bootloader/Kernel/App/Library/Module 심볼을 포함한 ELF파일을 생성합니다.

ELF(Executable and Linkable Format) 파일 Format

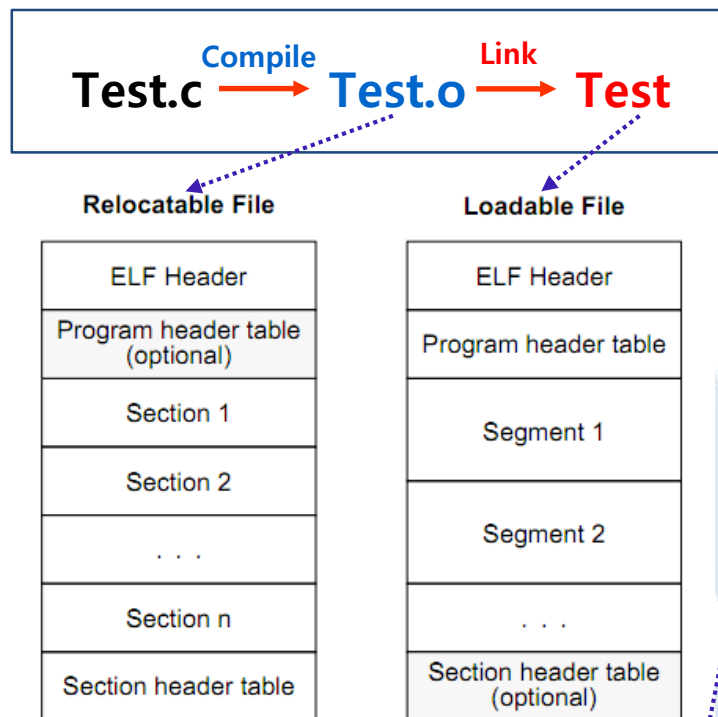


Figure 1. Structure of an ELF File

Table 12. Standard Sections for Code and Data

Section Name	Section Type	Flags	Use
.bss	SHT_NOBITS	A, W	Uninitialized data
.data	SHT_PROGBITS	A, W	Initialized data
.interp	SHT_PROGBITS	[A]	Program interpreter path name
.rodata	SHT_PROGBITS	A	Read-only data (constants and literals)
.text	SHT_PROGBITS	A, X	Executable code

Table 13. Other Standard Sections

Section Name	Section Type	Flags	Use
.comment	SHT_PROGBITS	none	Version control information
.dynamic	SHT_DYNAMIC	A, W	Dynamic linking tables
.dynstr	SHT_STRTAB	A	String table for .dynamic section
.dynsym	SHT_DYNSYM	A	Symbol table for dynamic linking
.got	SHT_PROGBITS	mach. dep.	Global offset table
.hash	SHT_HASH	A	Symbol hash table
.note	SHT_NOTE	none	Note section
.plt	SHT_PROGBITS	mach. dep.	Procedure linkage table
.relname	SHT_REL	[A]	Relocations for section name
.relname	SHT_RELA	[A]	Relocations for section name
.shstrtab	SHT_STRTAB	none	Section name string table
.strtab	SHT_STRTAB	none	String table
.symtab	SHT_SYMTAB	[A]	Linker symbol table

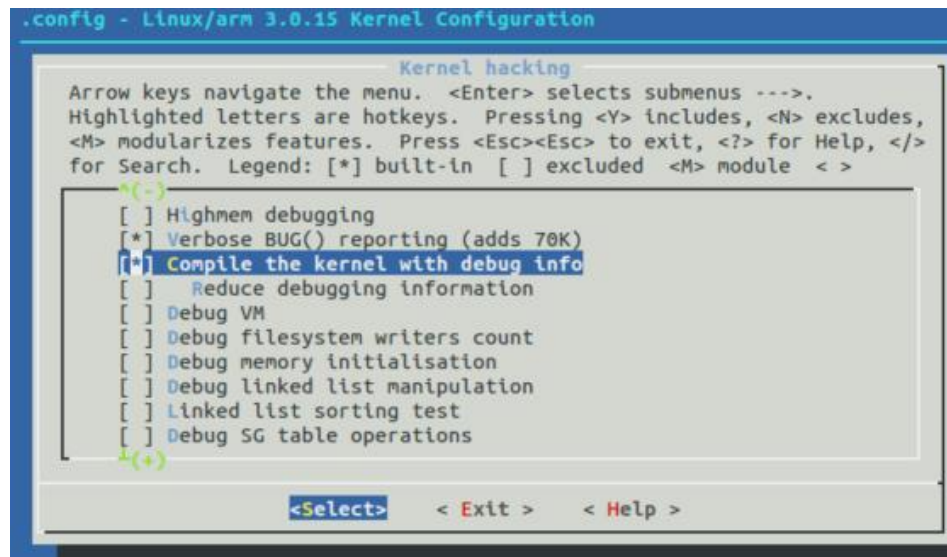
# 3-1. ELF 파일 생성

Bootloader/Kernel/App/Library/Module 심볼을 포함한 ELF파일을 생성합니다.

ELF(Executable and Linkable Format) 파일에 **디버그 심볼을 포함하여 빌드**

- Makefile 내에 **"-g"** 옵션을 추가하여 디버그 정보 생성
- **vmlinux** (심볼이 포함된 ELF) 파일을 생성

kernel hacking >> Compile the kernel with debug info 옵션 적용



# 4. Bootloader

---

부트로더 역할 및 종류에 대한 이해하고 디버깅을 합니다

1. Bootloader의 역할 및 종류
2. U-boot 특징
3. 디버깅 환경 cmm 작성
4. Bootloader 디버깅 실습

# 4-1. Bootloader 역할 및 종류

Bootloader가 하는 일과 동작의 흐름을 파악하여 디버깅에 이용합니다

Bootloader는 System Hardware 초기화하고, OS의 Kernel을 Memory에 Load하고 실행시키는 프로그램

- Bootloader의 위치 및 기능
  - ROM, Flash ROM, SRAM 등 정적인 메모리에 위치
  - 초기화 코드는 대부분 Assembler로 작성
  - 메모리 초기화 / 하드웨어 초기화  
( 직렬포트, 네트워크, 프로세서 속도, 인터럽트 등)
  - Kernel과 Ramdisk를 RAM에 Load 및 실행

Linux System에서 다양한 Bootloader가 존재하나 디버깅 방법은 동일

- U-Boot : 유니버설 플랫폼을 위한 Open Source Bootloader
- LK : 퀄컴, TI, nVidia 등 대부분의 칩 벤더에서 Release한 Bootloader
- SBL: Samsung Bootloader
  - BOOTP/TFTP (RARP/TFTP)를 이용한 네트워크 부팅
  - Serial을 이용한 다운로드



## 4-2. U-Boot 특징

---

실습 환경에서 제공되는 U-boot의 특징을 이해합니다.

### U-boot 개요

- U-boot 특징
  1. Universal Bootloader for Linux kernel
  2. ARM, PowerPC, MIPS 등 다양한 processor 환경 지원
  3. 필수적인 flash memory 지원 동작 및 다양한 유형의 flash memory 지원 가능
  4. Tftp, dhcp 등 기본적인 네트워크 프로토콜 지원
  5. Tftp 기반의 remote booting 지원 등
- U-boot 주요 기능
  1. POST(Power-On Self-Test)
  2. Linux kernel loading
  3. Linux kernel execution

## 4-3. U-boot 디버깅 cmm 작성

bootloader 디버깅을 위한 cmm을 작성합니다.

기본 부트로더 디버깅 스크립트 작성

- Target Interface

```
SYStem.CPU.S5PV310  
SYStem.Option.ResBreak OFF  
SYStem.Option.WaitReset 100.ms  
SYStem.Up
```

- 부트로더 심볼 로드

```
Data.Load.ELF C:\T32\T32_Linux_Edu\bsp_src\u-boot-2010.03\u-boot /nocode
```

- 소스패스 맞추기

```
sYmbol.SourcePATH.Translate "\Exynos4210_Linux" "C:\T32\T32_Linux_Edu"
```

Format:           sYmbol.SourcePATH.Translate <original\_string> <new\_string>

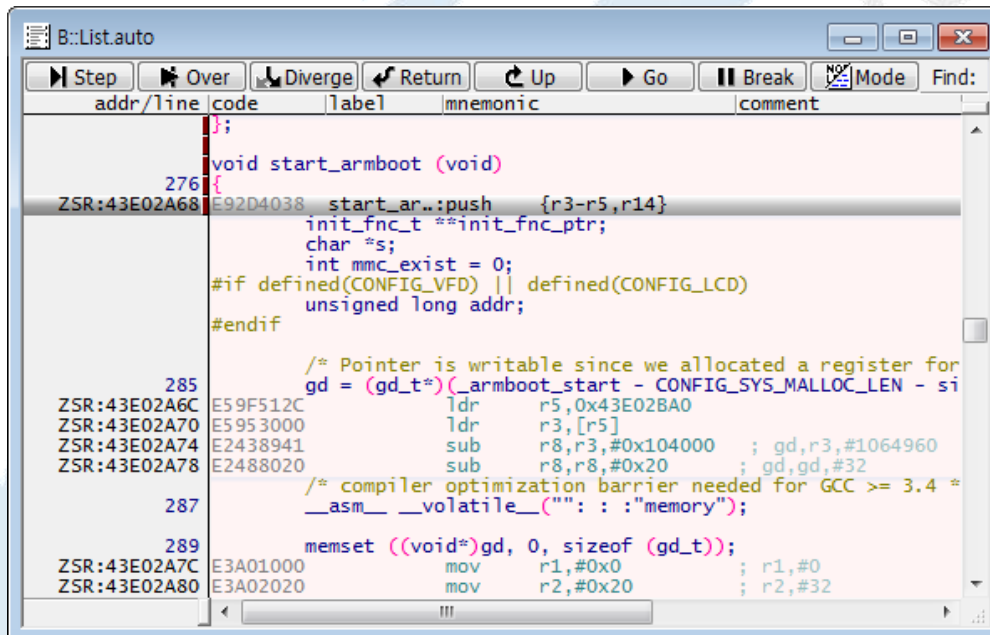
## 4-3. U-boot 디버깅 cmm 작성

### 기본 부트로더 디버깅 스크립트 작성

- start\_armboot 함수까지 수행

```
B.S start_armboot /Onchip  
GO  
Wait !RUN()
```

- 수행된 화면



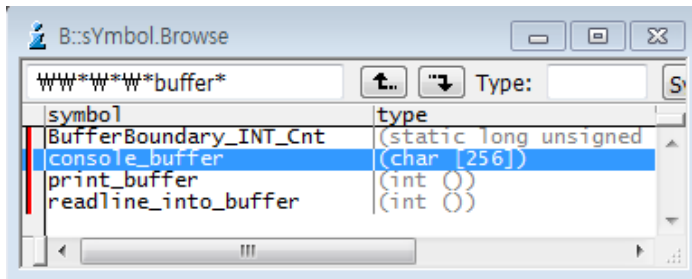
```
B::List.auto  
Step Over Diverge Return Up Go Break Mode Find:  
addr/line code label mnemonic comment  
};  
void start_armboot (void)  
276 {  
ZSR:43E02A68 E92D4038 start_ar...:push {r3-r5,r14}  
init_fnc_t **init_fnc_ptr;  
char *s;  
int mmc_exist = 0;  
#if defined(CONFIG_VFD) || defined(CONFIG_LCD)  
unsigned long addr;  
#endif  
  
/* Pointer is writable since we allocated a register for  
gd = (gd_t*)(_armboot_start - CONFIG_SYS_MALLOC_LEN - si  
285  
ZSR:43E02A6C E59F512C ldr r5,0x43E02BA0  
ZSR:43E02A70 E5953000 ldr r3,[r5]  
ZSR:43E02A74 E2438941 sub r8,r3,#0x104000 ; gd,r3,#1064960  
ZSR:43E02A78 E2488020 sub r8,r8,#0x20 ; gd,gd,#32  
/* compiler optimization barrier needed for GCC >= 3.4 *  
287 __asm__ __volatile__("" : : "memory");  
  
289 memset ((void*)gd, 0, sizeof (gd_t));  
ZSR:43E02A7C E3A01000 mov r1,#0x0 ; r1,#0  
ZSR:43E02A80 E3A02020 mov r2,#0x20 ; r2,#32
```

## 4-4. U-boot 디버깅 실습 1

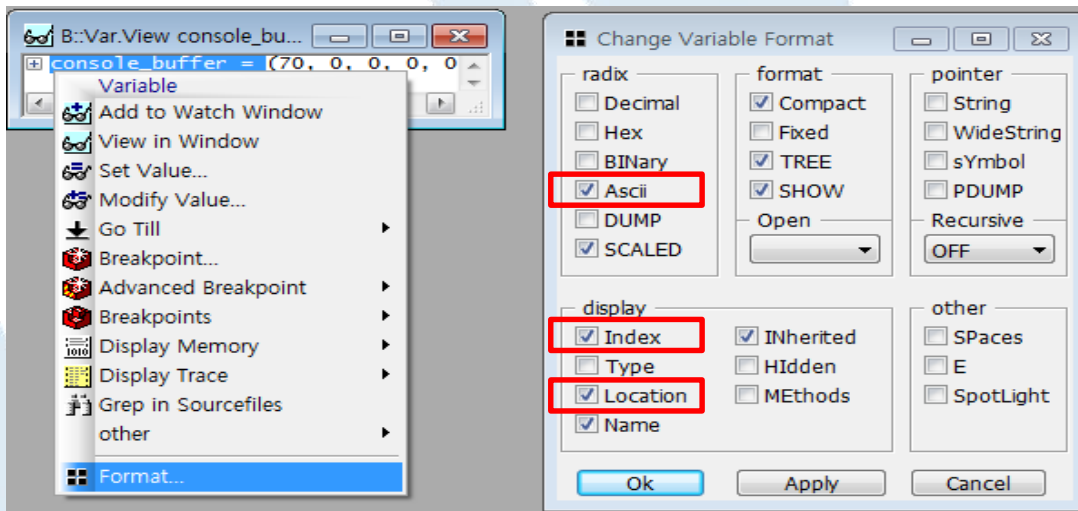
실습을 통해 bootloader 디버깅 및 R/W Break에 대해서 이해합니다.

### console\_buffer Format 설정

- sYmblo.Browse 창에서 console\_buffer를 선택



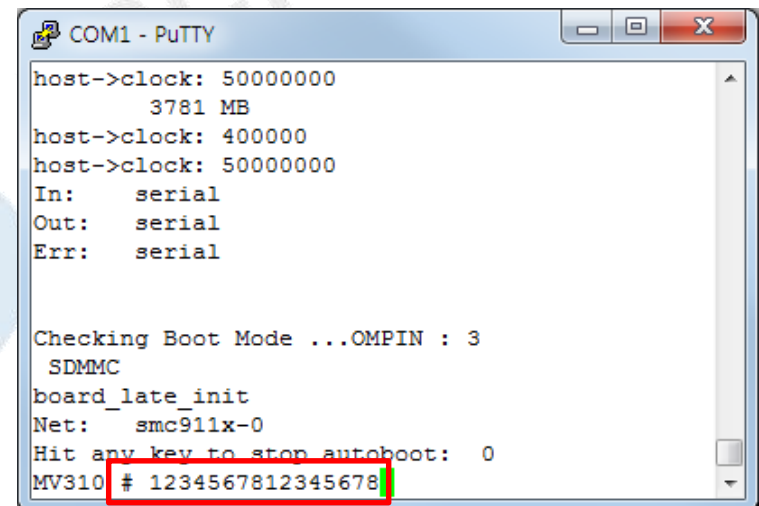
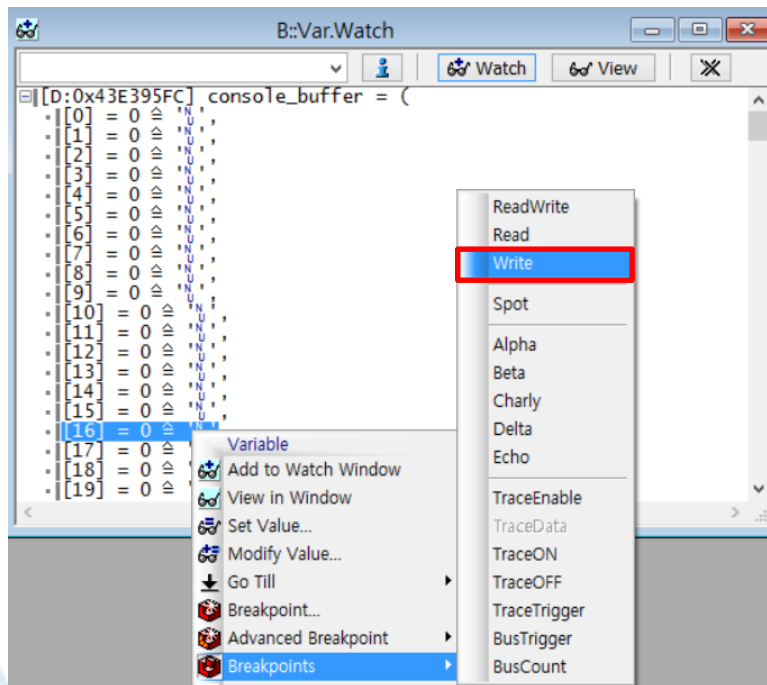
- Right 클릭으로 Format 창을 열어서 아래와 같이 옵션을 추가



## 4-4. U-boot 디버깅 실습 1

### console\_buffer 16번째 배열에 Write Break Point 설정

- Watch 창에서 아래와 같이 설정
- 터미널 창에서 문자를 입력



## 4-4. U-boot 디버깅 실습 1

### console\_buffer에 Write Break Point 된 위치 확인

- 코드 수행 위치 및 buffer, stack 등을 확인

The screenshot displays the TRACE32 PowerView for ARM 0 interface. The main window shows the source code of the `console_buffer` function, with the execution cursor at line 1070. The `B::Break.List` window shows a breakpoint set at address `Z:43E3960C--43E3960C` for the `write` operation. The `B::Register.view` window shows the current register values, including `R0` (13820000), `R1` (38), `R2` (43E39838), `R3` (2), `R4` (43E395FC), `R5` (43E3960C), `R6` (8), `R7` (38), `SPSR` (80000013), `R8` (43CFBFF0), `R9` (0), `R10` (2), `R11` (0), `R12` (0), `R13` (43FFEF98), `R14` (43E16D18), `PC` (43E11CB4), and `CPSR` (200001D3). The `B::Frame.view /Locals /Caller` window shows the current frame's locals, including `buffer` (0x43E395FC), `p` (0x43E3960C), `initd` (0), `n` (119), `plen` (8), `col` (25), and `c` (56). The status bar at the bottom indicates the current address is `ZSR:43E11CB4` and the program is stopped at a breakpoint.

# 5. Linux Kernel

---

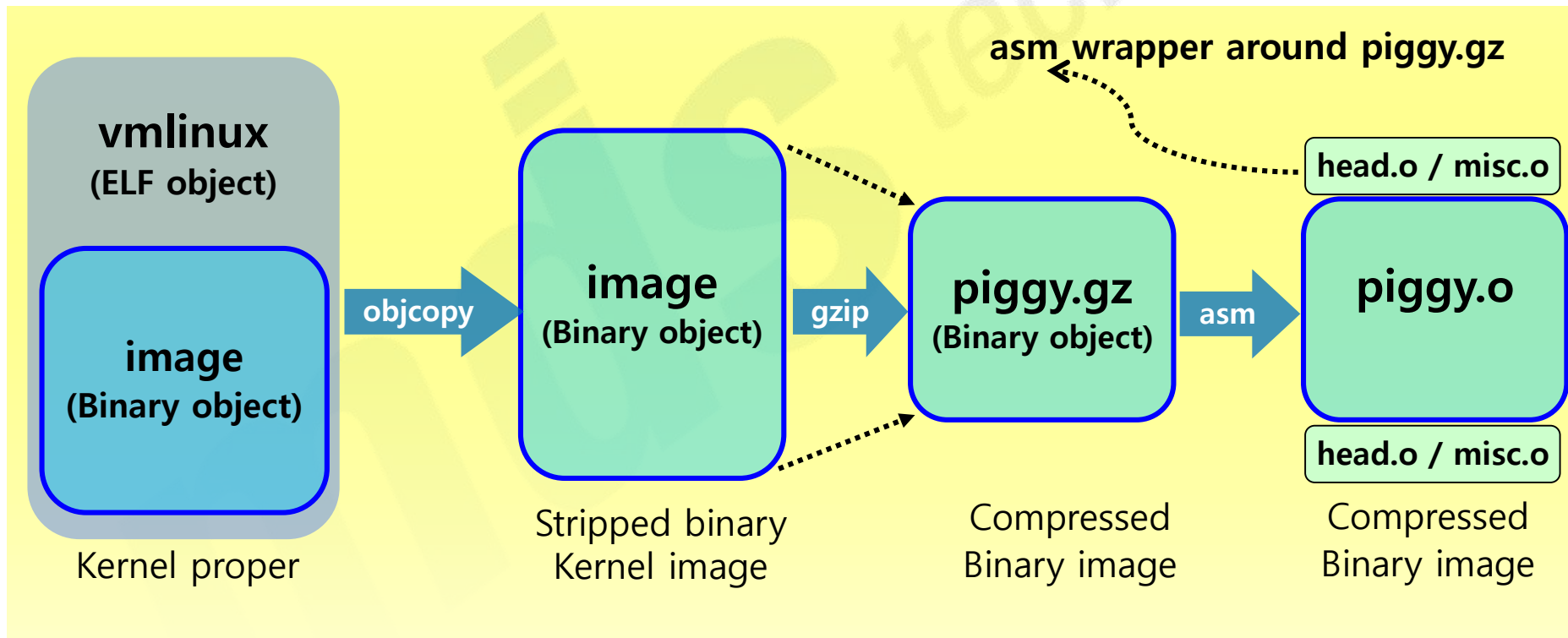
Linux Kernel과 Memory 구조를 이해하고 디버깅을 할 수 있습니다

1. Kernel 이미지 구조
2. Linux Memory Model
3. Kernel 디버깅 시작
4. Kernel 디버깅 실습

## 4-1. Kernel 이미지 구조

Linux Kernel의 경우 Compile이 완료되면 압축된 이미지 형태가 생성되는 구조  
zImage 구성

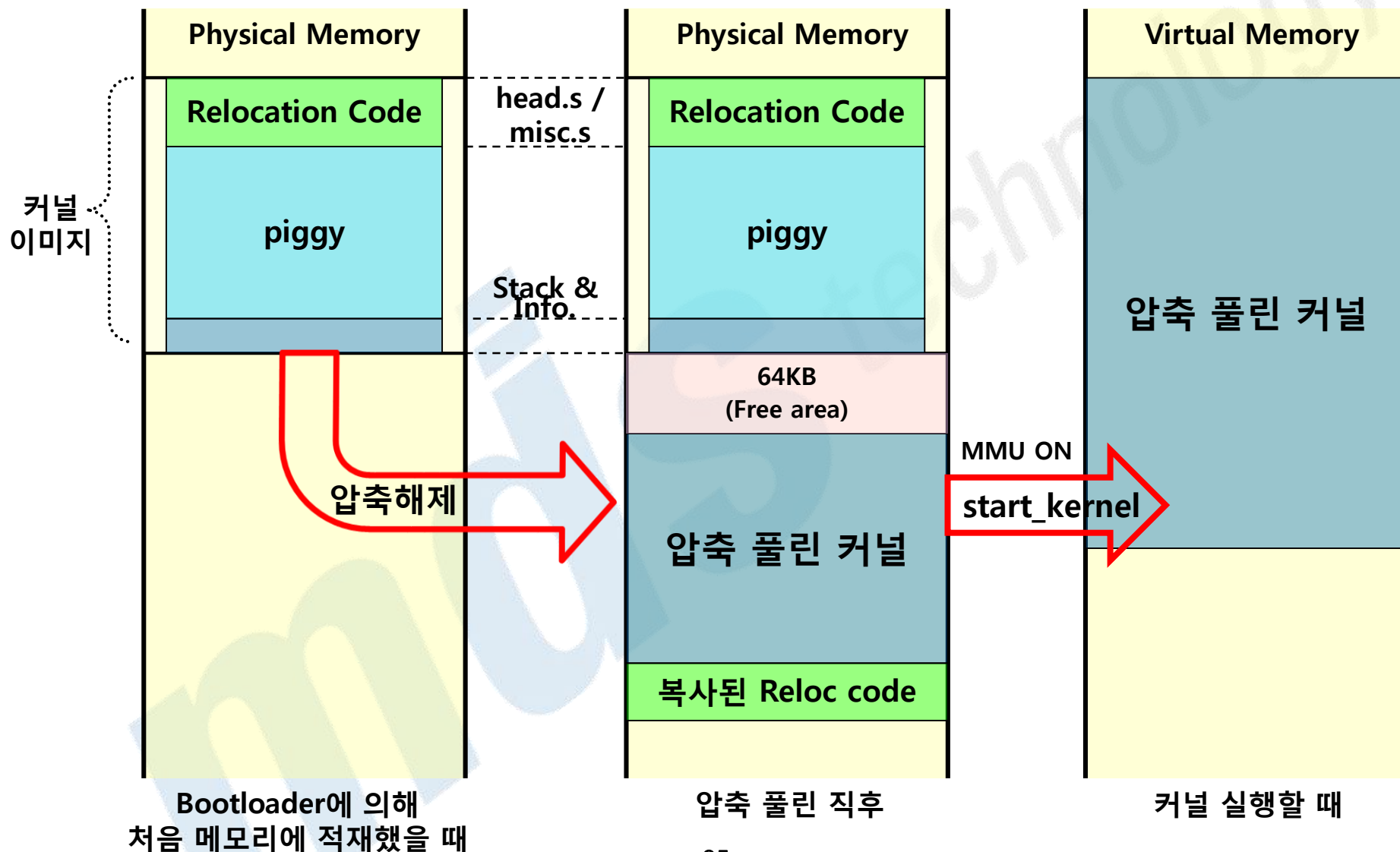
- Linux kernel Image는 크기를 줄이기 위해 압축된 형태로 저장
- 압축 해제 시에는 지정된 물리주소에 해제
  - kernel/arch/arm/mach-exynos/Makefile.boot
  - zreladdr-y= 0x40008000 값 참조





# 5-1. Kernel 이미지 구조

압축된 이미지는 Bootloader 또는 Kernel에 의해 압축 해제 및 재 배치됩니다



## 5-2. Linux Memory Model

Linux kernel은 Memory Model에 대해 이해 합니다.

MMU 사용으로 인해 Physical Memory가 아닌 Virtual Memory로 동작

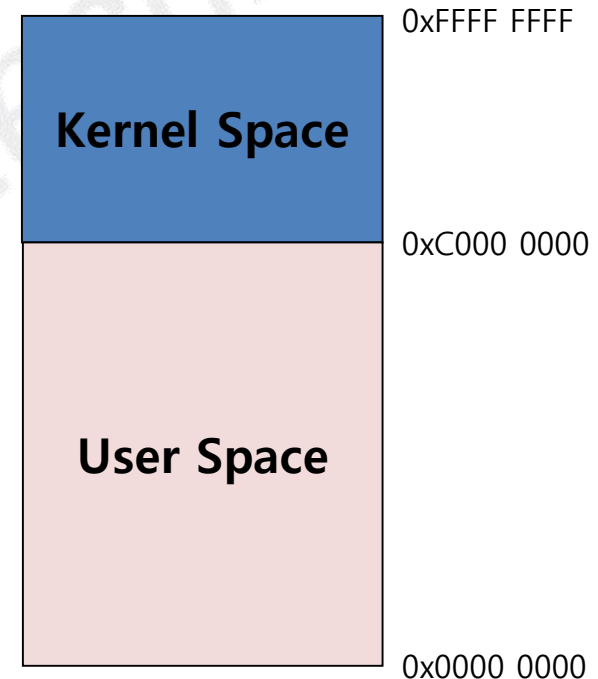
- 현재 ARM **32bit Architecture**로 인해 **Virtual Memory 4GB**를 사용함
- 통상 **User space**를 3GB, **Kernel Space**를 1GB 사용

✓ **0x0000 0000 ~ 0xBFFF FFFF :**

User Space 프로세스가 사용자 모드에서 동작 중일 때의 어드레스로 프로세스가 사용자 모드이든 커널 모드이든 접근 가능 (application, library, stack, heap 등 맵핑)

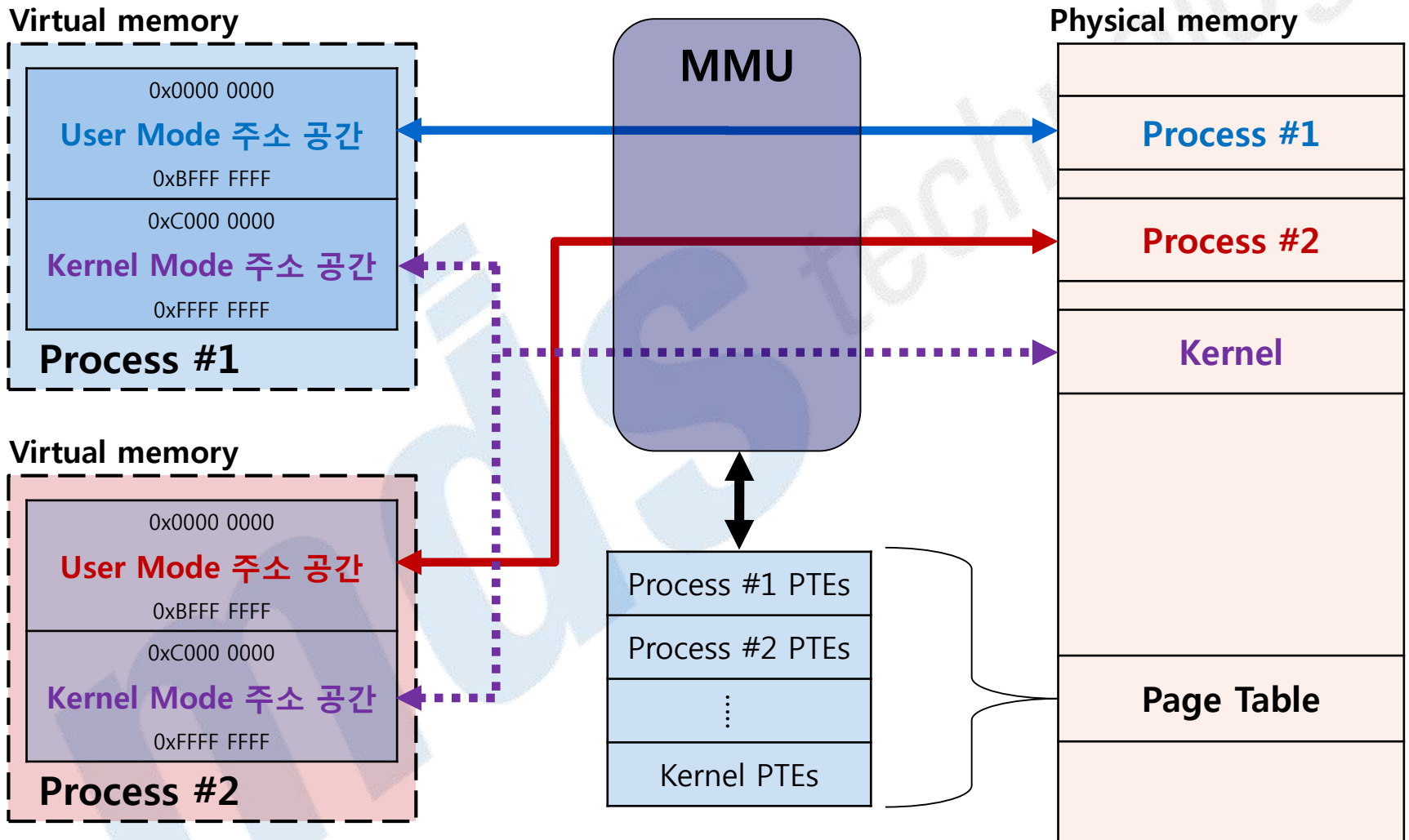
✓ **0xC000 0000 ~ 0xFFFF FFFF :**

Kernel Space 프로세서가 커널 모드에서 동작 중일 때의 어드레스로, 프로세스가 커널 모드에서만 접근 가능 (kernel, device driver, gpio 등 맵핑)



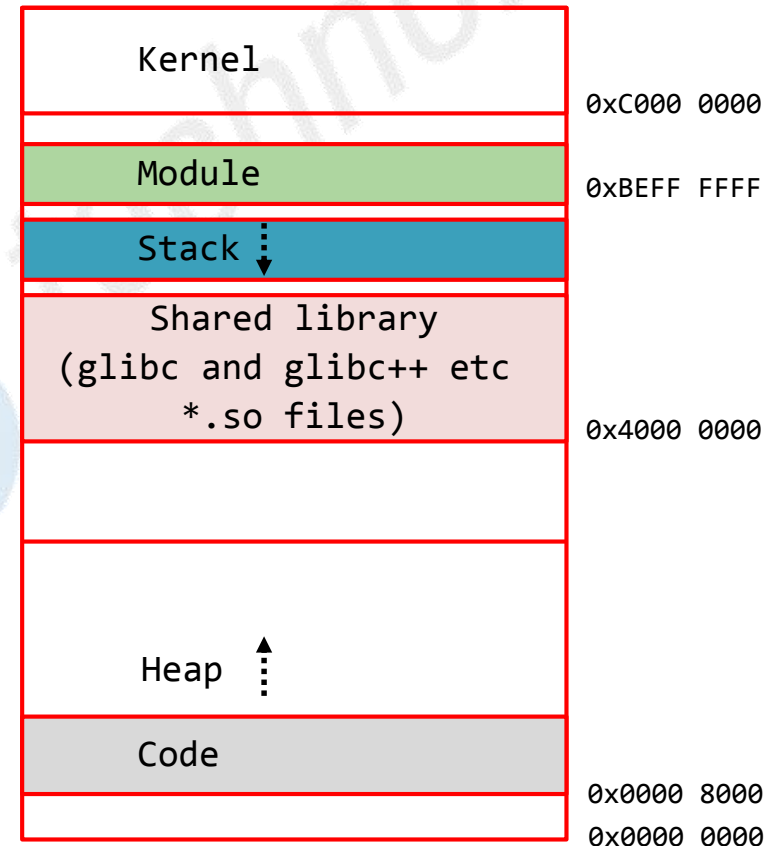
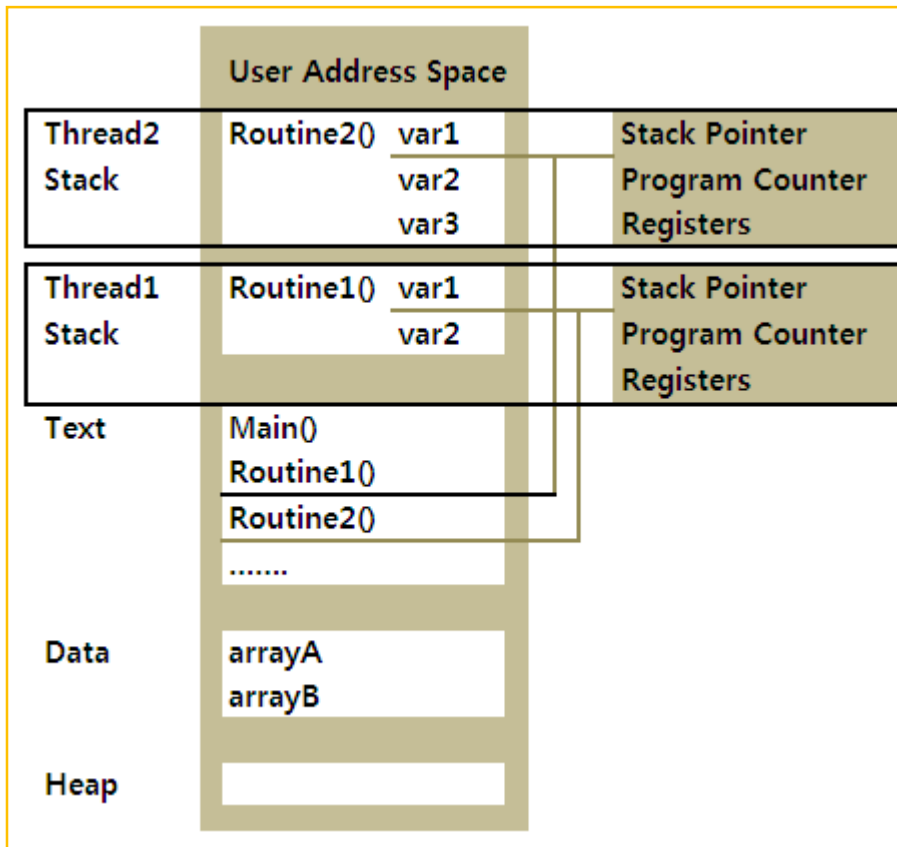
## 5-2. Linux Memory Model

Linux에서는 가상 메모리를 각 Process 마다 4GB씩 할당 받고, TRACE32에서는 이 메모리 공간을 **Space ID**라는 값으로 구분합니다



## 5-2. Linux Memory Model

**Linux**는 **Multi-thread**방식을 사용하는데 Thread들은 메모리 영역 중 3GB의 User Space에서 할당된 Stack 영역을 나눠서 사용합니다



## 5-3. Kernel 디버깅 시작

Kernel/Application/Library/Module 디버깅을 위한 cmm 작성합니다.

### 기본 커널 디버깅 스크립트 작성

- Target Interface (add options)

```
SYStem.CPU.S5PV310  
SYStem.Option.ResBreak OFF  
SYStem.Option.WaitReset 100.ms
```

```
SYStem.Option MMUSPACES ON
```

```
TrOnchip.Set DABORT OFF  
TrOnchip.Set PABORT OFF  
TrOnchip.Set UNDEF OFF
```

```
SYStem.Option DACR ON  
SETUP.IMASKASM ON  
SETUP.IMASKHLL ON
```

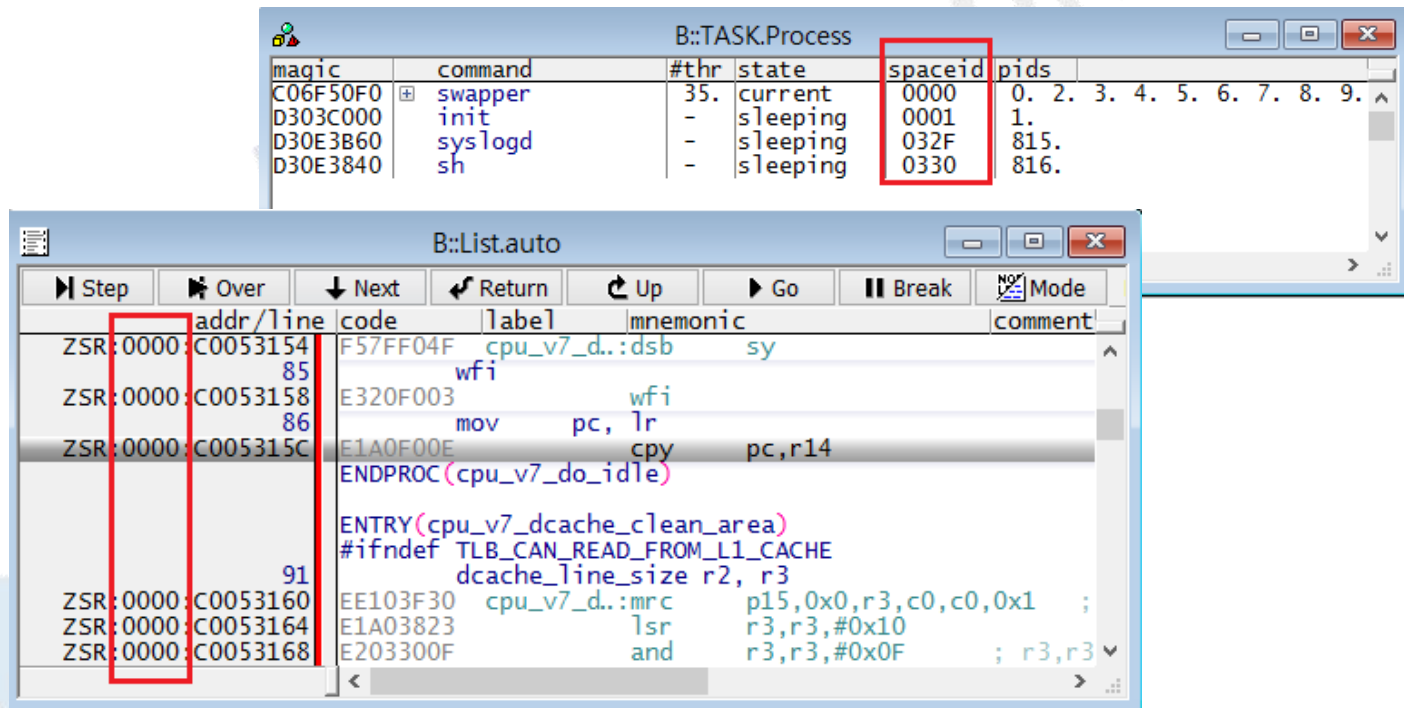
```
SYStem.Up
```

## 5-3. Kernel 디버깅 시작

- Target Interface (add options)

**SYStem.Option MMUSPACES ON**

- PID를 구분하는 용도로 사용(TRACE32에서는 spaceid 라고 부름)

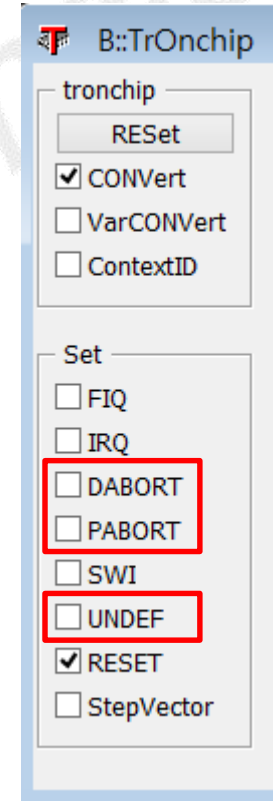


## 5-3. Kernel 디버깅 시작

- Target Interface (add options)

```
TrOnchip.Set DABORT OFF  
TrOnchip.Set PABORT OFF  
TrOnchip.Set UNDEF OFF
```

- Vector exception 기능 OFF (on-demand paging)
- TRACE32 는 기본적으로 exception 발생시 break하도록 되어 있어 해당 기능을 OFF



## 5-3. Kernel 디버깅 시작

- Target Interface (add options)

SYStem.Option DACR ON  
 SETUP.IMASKASM ON  
 SETUP.IMASKHLL ON

- DACR(Domain Access Control Register) 를 Domain 권한 상관없이 볼 수 있도록 설정
- step(F2) 시 Interrupt 발생 무시 하도록 설정(HLL 과 ASM Mode)

The screenshot shows two windows from a debugger. The top window, titled 'B:mmu', displays system configuration for various registers. The bottom window, titled 'B:mmu.listpt', displays a memory dump with columns for address, physical address, size, permissions, and other attributes. A red box highlights the DACR register settings in the top window and the 'd' column in the bottom window.

Register	Value	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8
SCTLR	10C5387D	TE	ARM	AFE	Disabled	TRE	Enabled		
		NMFI	Disabled	EE	Little	RR	Random		
		V	0xFFFF0000	I	Enabled	Z	Enabled		
		SW	Disabled	C	Enabled	A	Disabled		
		M	Enabled						
TTBR0	50B6804A	TTB0	50B68000		IRGN[1:0]	Back/allocated			
		RGN	Back/allocated	S	Shared				
TTBR1	4000404A	TTB1	40004000		IRGN[1:0]	Back/allocated			
		RGN	Back/allocated	S	Shared				
TTBCR	00000000	PDI	Enable	PDO	Enable	N	off		
DACR	00000015	D15	Denied	D14	Denied	D13	Denied	D12	Denied
		D11	Denied	D10	Denied	D9	Denied	D8	Denied
		D7	Denied	D6	Denied	D5	Denied	D4	Denied
		D3	Denied	D2	Client	D1	Client	D0	Client

Register	Value	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8	Field 9	Field 10	Field 11	Field 12
DFSR	00000817	EXT	DOMAIN	DE									
DFAR	4014C000	DFA	40										
IFSR	00000007	SD	DE										
IFAR	403EE100	IFA	40										
DAFSR	00000000	DAFS	00										
IAFSR	00000000	IAFS	00										

address	physical	sec	d	size	permissions	q1b	shr	pagefla
C:0001:00000000--00007FFF		s	01	00001000	P:readonly U:readonly exec	no	yes	I:wr-ba
C:0001:00008000--00009FFF	A:53DF0000--53DF1FFF	s	01	00001000	P:readonly U:readonly exec	no	yes	I:wr-ba
C:0001:0000A000--0000DFFF	A:40037000--4003AFFF	s	01	00001000	P:readonly U:readonly exec	no	yes	I:wr-ba
C:0001:0000E000--0000FFFF	A:53DEB000--53DECFFF	s	01	00001000	P:readonly U:readonly exec	no	yes	I:wr-ba
C:0001:00010000--0001FFFF		s	01	00001000	P:readonly U:readonly exec	no	yes	I:wr-ba
C:0001:00018000--00019FFF	A:53D77000--53D78FFF	s	01	00001000	P:readonly U:readonly exec	no	yes	I:wr-ba
C:0001:0001A000--0001BFFF	A:53D7B000--53D7BFFF	s	01	00001000	P:readonly U:readonly exec	no	yes	I:wr-ba
C:0001:0001C000--0001DFFF	A:53C17000--53C17FFF	s	01	00001000	P:readonly U:readonly exec	no	yes	I:wr-ba
C:0001:0001E000--0001FFFF	A:53D22000--53D23FFF	s	01	00001000	P:readonly U:readonly exec	no	yes	I:wr-ba
C:0001:0001F000--00020FFF	A:53D27000--53D28FFF	s	01	00001000	P:readonly U:readonly exec	no	yes	I:wr-ba
C:0001:00020000--00021FFF	A:53D2B000--53D2BFFF	s	01	00001000	P:readonly U:readonly exec	no	yes	I:wr-ba
C:0001:00022000--00023FFF	A:53D68000--53D69FFF	s	01	00001000	P:readonly U:readonly exec	no	yes	I:wr-ba



## 5-3. Kernel 디버깅 시작

---

### 기본 커널 디버깅 스크립트 작성

- 커널 심볼 로드

```
Data.Load.ELF C:\T32\T32_Linux_Edu\bsp_src\linux-3.0.15\vmlinux /nocode
```

- 소스패스 맞추기

```
sYmbol.SourcePATH.Translate "\Exynos4210_Linux" "C:\T32\T32_Linux_Edu"
```

Format:            **sYmbol.SourcePATH.Translate** <original\_string> <new\_string>

## 5-3. Kernel 디버깅 시작

### 기본 커널 디버깅 스크립트 작성

- MMU 설정(Target의 MMU정보를 TRACE32에게 통보, Kernel 영역과 공통 사용영역 설정)

```
PRINT "initializing debugger MMU..."
```

```
MMU.FORMAT LINUXSWAP3 swapper_pg_dir 0xC0000000--0xC07FFFFFFF 0x40000000
```

```
TRANSlation.COMMON 0xBF000000--0xFFFFFFFF
```

```
TRANSlation.TableWalk ON
```

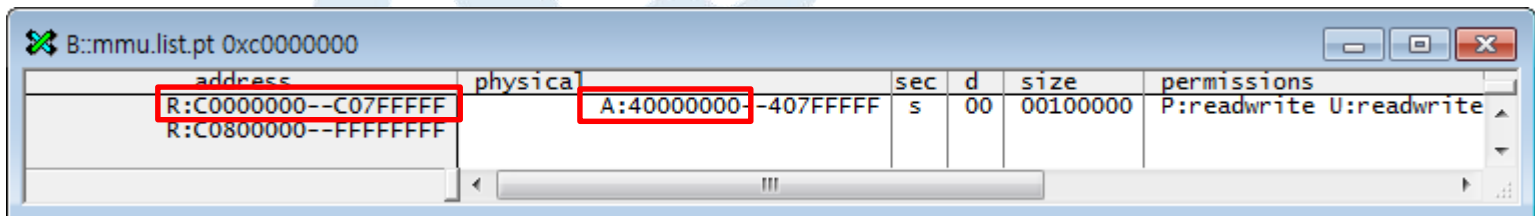
```
TRANSlation.ON
```

Format: **MMU.FORMAT** <format> <base address> <kernel translation>

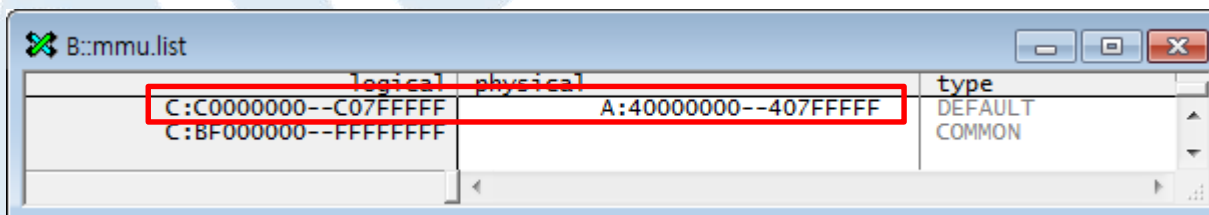
- MMU 설정할 영역을 지정

Format: **TRANSlation.COMMON** <logical\_range>

- MMU Common 영역을 지정



address	physical	sec	d	size	permissions
R:C0000000--C07FFFFFFF	A:40000000--407FFFFFFF	s	00	00100000	P:readwrite U:readwrite
R:C0800000--FFFFFFFF					



logical	physical	type
C:C0000000--C07FFFFFFF	A:40000000--407FFFFFFF	DEFAULT
C:BF000000--FFFFFFFF		COMMON

## 5-3. Kernel 디버깅 시작

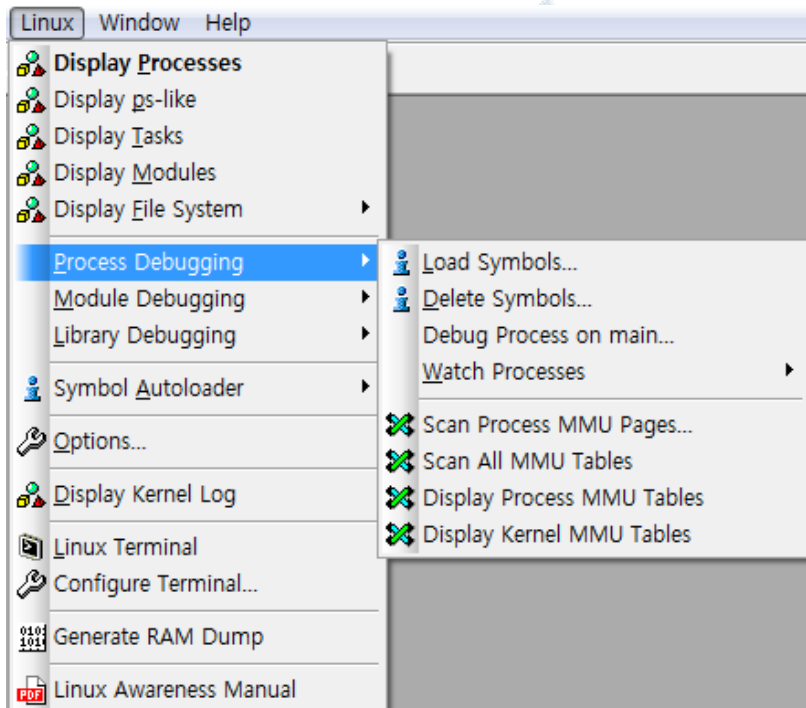
### 기본 커널 디버깅 스크립트 작성

- Linux Awareness 설정(OS 디버깅을 할 수 있도록 Linux Awareness 기능 활성화)

```
PRINT "initializing RTOS support..."
```

```
TASK.CONFIG ~/demo/arm/kernel/linux/linux-3.x/linux3.t32
```

```
MENU.ReProgram ~/demo/arm/kernel/linux/linux-3.x/linux.men
```



Linux kernel version 에 따라 awareness 파일을 다르게 설정

Kernel 2.x : ~/demo/arm/kernel/linux/linux-2.x/

Kernel 3.x : ~/demo/arm/kernel/linux/linux-3.x/

## 5-3. Kernel 디버깅 시작

### 기본 커널 디버깅 스크립트 작성

- start\_kernel() 까지 수행된 화면

The screenshot displays the TRACE32 PowerView for ARM 0 interface. The main window shows the assembly code for the `__init start_kernel(void)` function, starting at address 457. The code includes instructions like `start_ke...cpy r12,r13` and `push {r4-r7,r11-r12,r14,pc}`. A task window titled "B::TASK.Process" shows the `swapper` process in a `running` state. A terminal window titled "COM1 - PuTTY" shows the boot process output, including "Starting kernel ..." and "Uncompressing Linux... done, booting the kernel."

TRACE32 PowerView for ARM 0 [Power Debug USB @ ]

File Edit View Var Break Run CPU Misc Trace Perf Cgv CortexA9MPCORE Linux Window Help

B::List.auto

Step Over Diverge Return Up Go Break Mode Find:

addr/line code label mnemonic comment

457 { asmlinkage void \_\_init start\_kernel(void)

ZSR:FFFF:C0008778 E1A0C00D start\_ke...cpy r12,r13

ZSR:FFFF:C000877C E92DD8F0 push {r4-r7,r11-r12,r14,pc}

ZSR:FFFF:C0008780

ZSR:FFFF:C0008784

B::TASK.Process

magic	command	#thr	state	spaceid	pids
C06F50F0	swapper	-	running	0000	0.

ZSR:FFFF:C0008778

trigger devices trace Data Var List PERF SYSte

ZSR:FFFF:C0008778 W\\m\\linux\\init\\main\\start\_kernel (task error) 0 stopped at breakpoint

COM1 - PuTTY

MMC: host->clock: 400000

host->clock: 50000000

3781 MB

host->clock: 400000

host->clock: 50000000

In: serial

Out: serial

Err: serial

Checking Boot Mode ...OMPIN : 3

SDMMC

board\_late\_init

Net: smc911x-0

Hit any key to stop autoboot: 0

reading kernel.. 1120, 10240

MMC read: dev # 0, block # 1120, count 10240 ...10240 blocks read: OK

completed

Boot with zImage

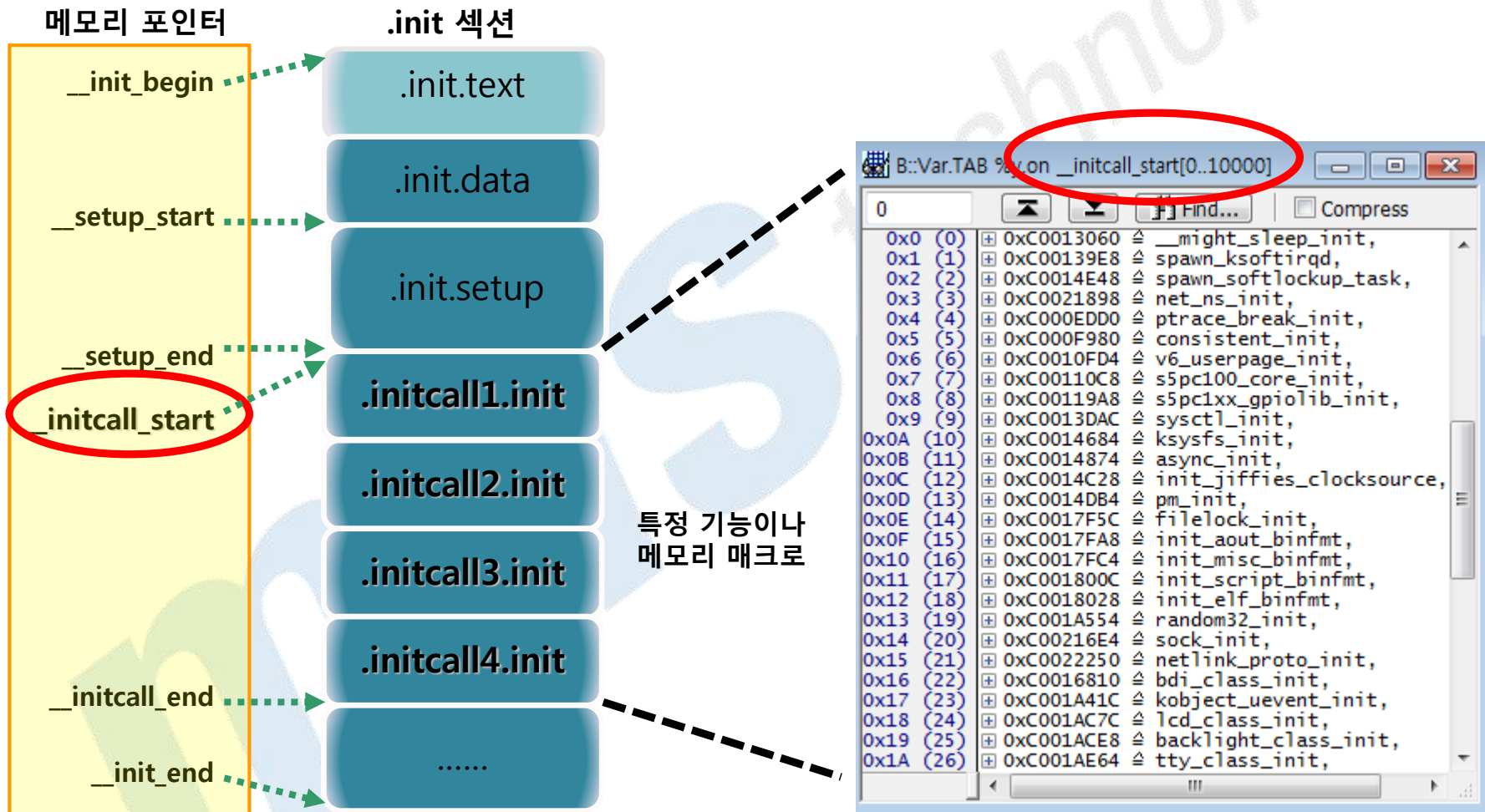
Starting kernel ...

Uncompressing Linux... done, booting the kernel.

## 5-4. Kernel 디버깅 실습

Kernel build시 등록된 디바이스 드라이버들을 디버깅하는 방법을 학습합니다

Built-in device driver들은 대부분 initcall table에 등록되어 있으며, initcall table의 구조는 아래 그림과 같습니다 (<kernel>/include/linux/init.h 파일 참조)

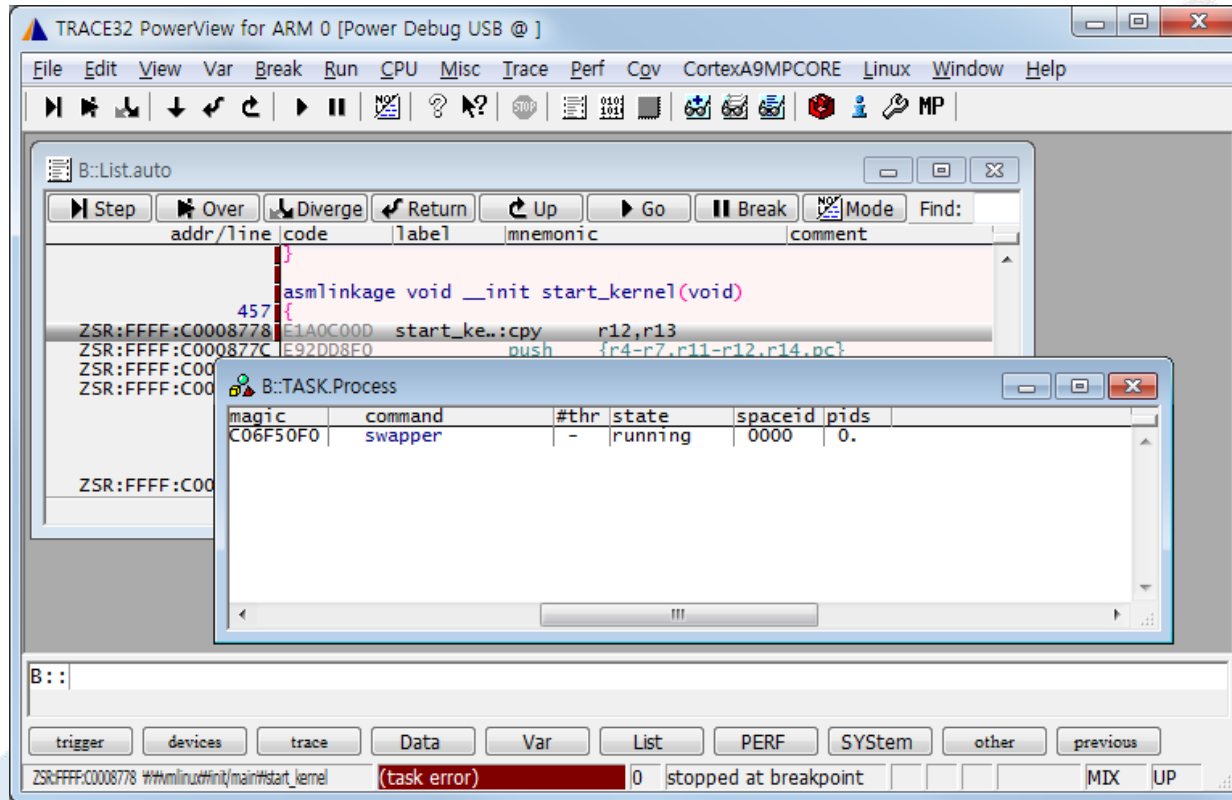


## 5-4. Kernel 디버깅 실습

실습을 통해 built-in device driver 에 대해서 이해합니다

### Built-in Driver 디버깅

- linux\_debug.cmm 실행

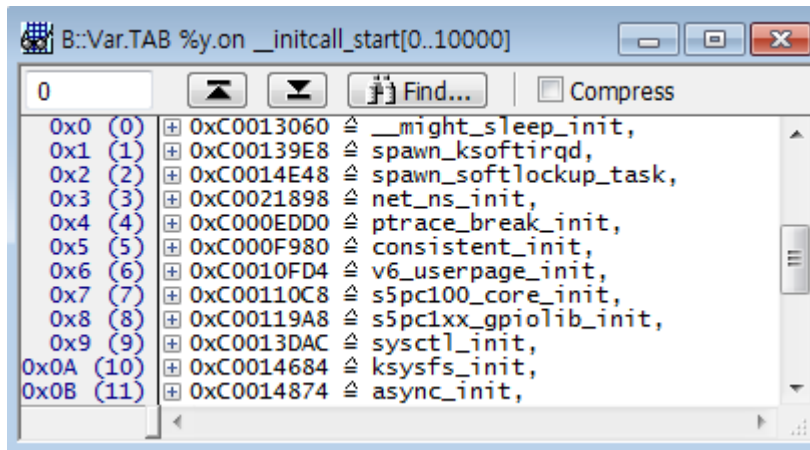


## 5-4. Kernel 디버깅 실습

### Built-in Driver 디버깅

- do\_one\_initcall.cmm 실행

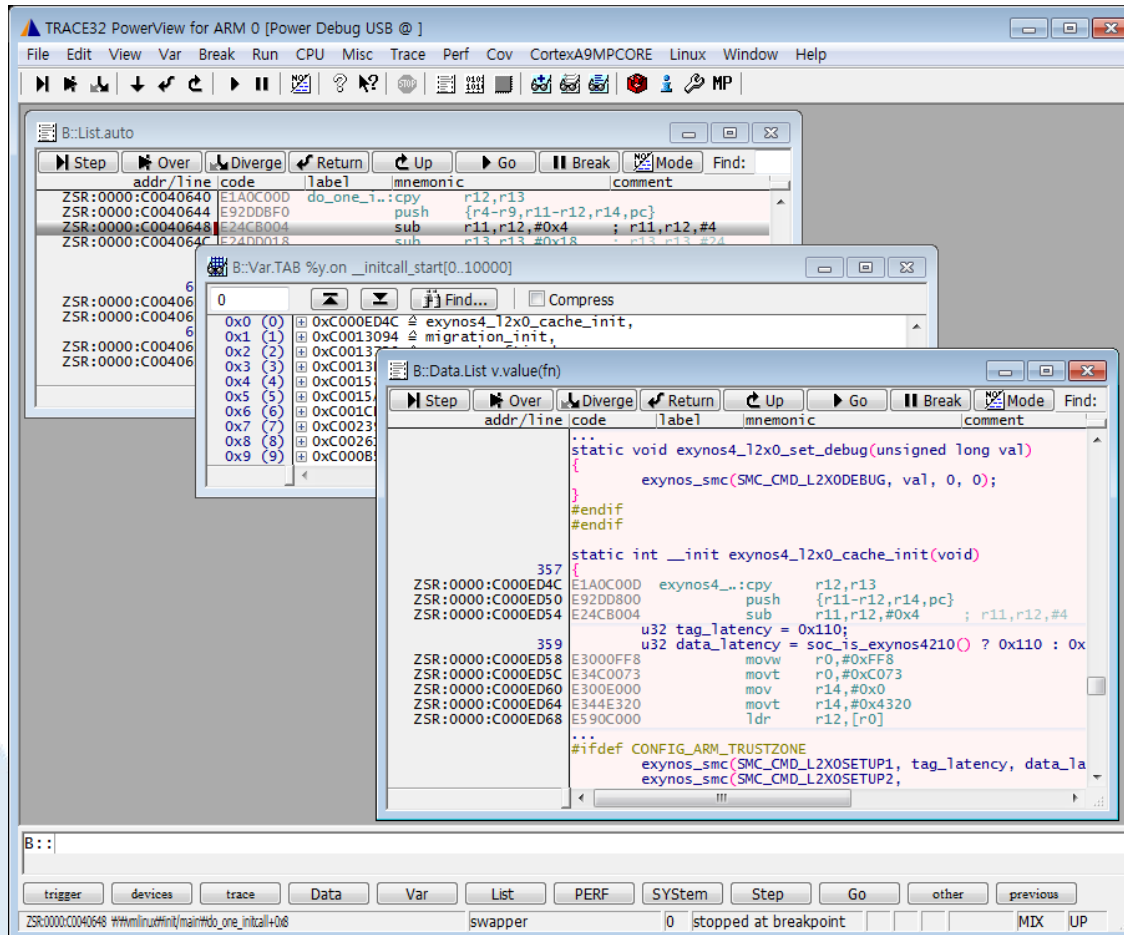
```
B::do do_one_initcall.cmm
```



## 5-4. Kernel 디버깅 실습

### Built-in Driver 디버깅

- Target을 실행하여 수행되는 Build-in Driver 확인





# 6. Linux Awareness

---

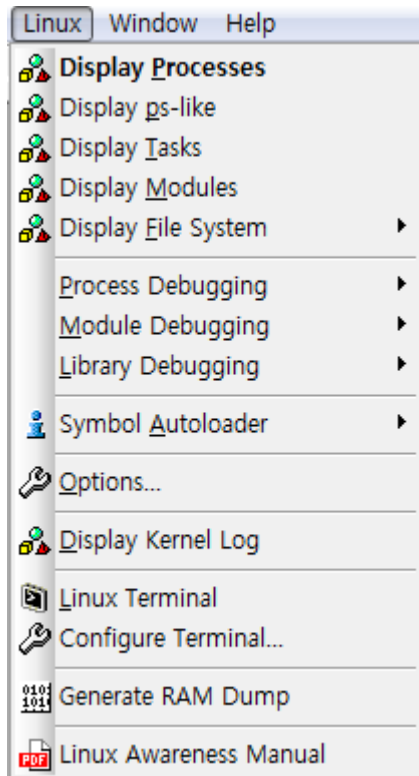
Linux Awareness 기능을 이해하고, Autoloader를 통해 디버깅 방법을 학습합니다.

1. Linux Awareness

2. Autoloader 활용

# 6-1. Linux Awareness

Linux Awareness는 Linux kernel의 정보를 제공하는 TRACE32의 기능으로 현재 CPU의 상태 및 디버깅을 위한 Symbol Load등을 할 수 있습니다



\* Linux Kernel의 특정 정보를 Display

\* Process/Library/Module 디버깅 할 수 있도록 정보제공

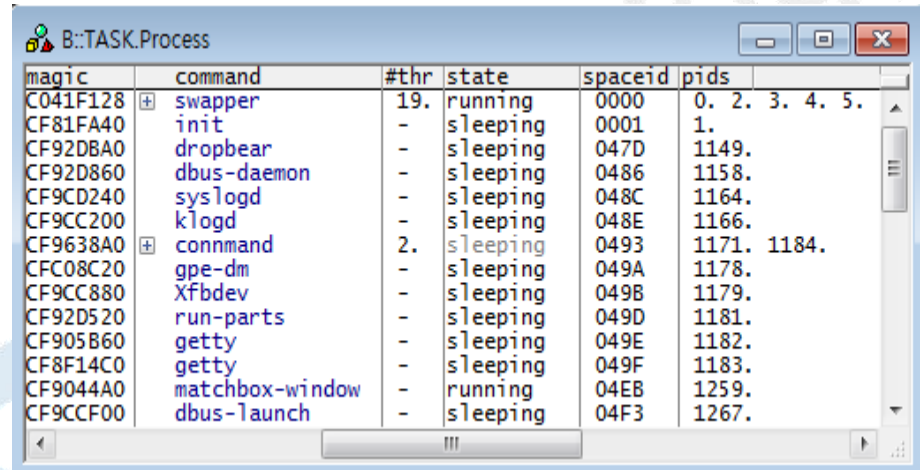
- **Display Processes:** 현재 실행되고 있는 Process
- **Display ps-like:** Linux Terminal에서 ps명령결과값
- **Display Tasks:** 실행중인 Thread를 모두 Display
- **Display Modules:** 동적으로(insmod) Load된 Display
- **Display File System:** Mount된 File System Display
- **Process Debugging:** Process(Daemon) Debugging
- **Module Debugging:** Module (Device driver) Debugging
- **Library Debugging:** Library Debug Function
- **Autoloader :** Symbol Autoload (6-2. 참고)

# 6-1. Linux Awareness

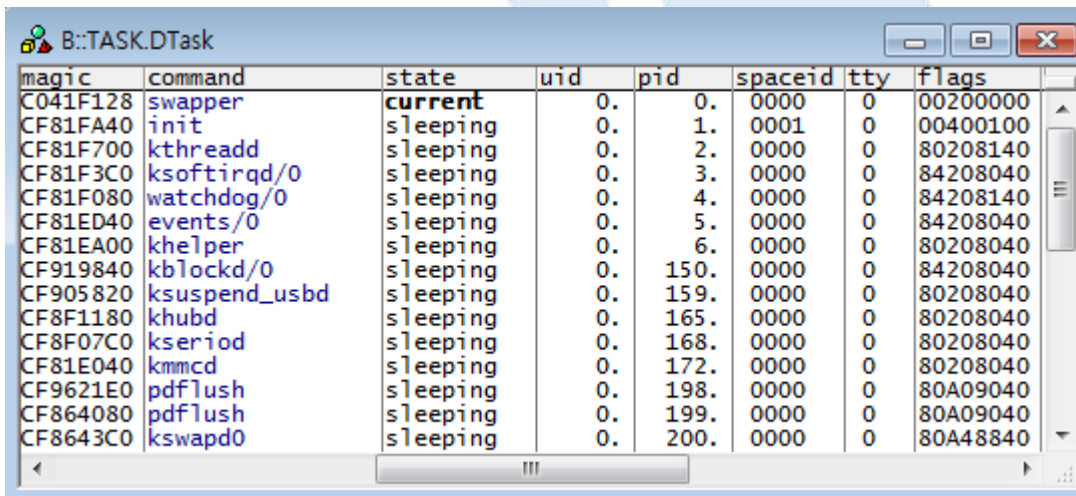
Display Processes / ps-like / Tasks 관련 기능을 실습해 봅니다

모든 Task (Process 또는 Thread)  
목록을 확인하고 관련기능과  
연동되는 윈도우

TCB 정보 및 라이브러리 목록 등을 확인



magic	command	#thr	state	spaceid	pids
C041F128	swapper	19.	running	0000	0. 2. 3. 4. 5.
CF81FA40	init	-	sleeping	0001	1.
CF92DBA0	dropbear	-	sleeping	047D	1149.
CF92D860	dbus-daemon	-	sleeping	0486	1158.
CF9CD240	syslogd	-	sleeping	048C	1164.
CF9CC200	klogd	-	sleeping	048E	1166.
CF9638A0	connmand	2.	sleeping	0493	1171. 1184.
CFC08C20	gpe-dm	-	sleeping	049A	1178.
CF9CC880	Xfbdev	-	sleeping	049B	1179.
CF92D520	run-parts	-	sleeping	049D	1181.
CF905B60	getty	-	sleeping	049E	1182.
CF8F14C0	getty	-	sleeping	049F	1183.
CF9044A0	matchbox-window	-	running	04EB	1259.
CF9CCF00	dbus-launch	-	sleeping	04F3	1267.

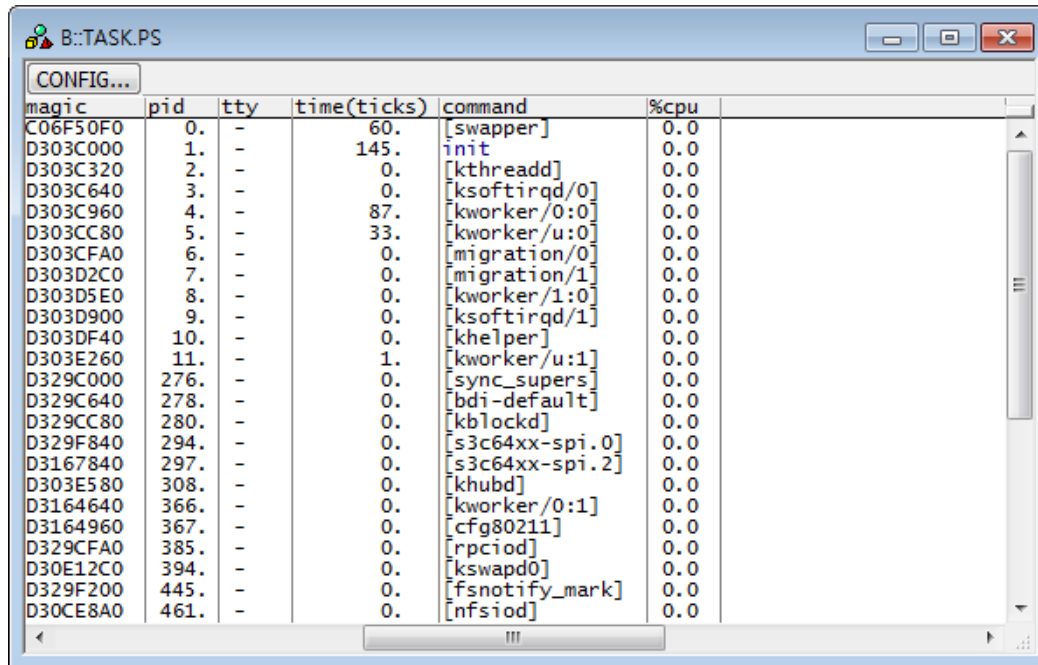


magic	command	state	uid	pid	spaceid	tty	flags
C041F128	swapper	current	0.	0.	0000	0	00200000
CF81FA40	init	sleeping	0.	1.	0001	0	00400100
CF81F700	kthreadd	sleeping	0.	2.	0000	0	80208140
CF81F3C0	ksoftirqd/0	sleeping	0.	3.	0000	0	84208040
CF81F080	watchdog/0	sleeping	0.	4.	0000	0	84208140
CF81ED40	events/0	sleeping	0.	5.	0000	0	84208040
CF81EA00	khelper	sleeping	0.	6.	0000	0	80208040
CF919840	kblockd/0	sleeping	0.	150.	0000	0	84208040
CF905820	ksuspend_usbd	sleeping	0.	159.	0000	0	80208040
CF8F1180	khubd	sleeping	0.	165.	0000	0	80208040
CF8F07C0	kseriod	sleeping	0.	168.	0000	0	80208040
CF81E040	kmmcd	sleeping	0.	172.	0000	0	80208040
CF9621E0	pdflush	sleeping	0.	198.	0000	0	80A09040
CF864080	pdflush	sleeping	0.	199.	0000	0	80A09040
CF8643C0	kswapd0	sleeping	0.	200.	0000	0	80A48840

# 6-1. Linux Awareness

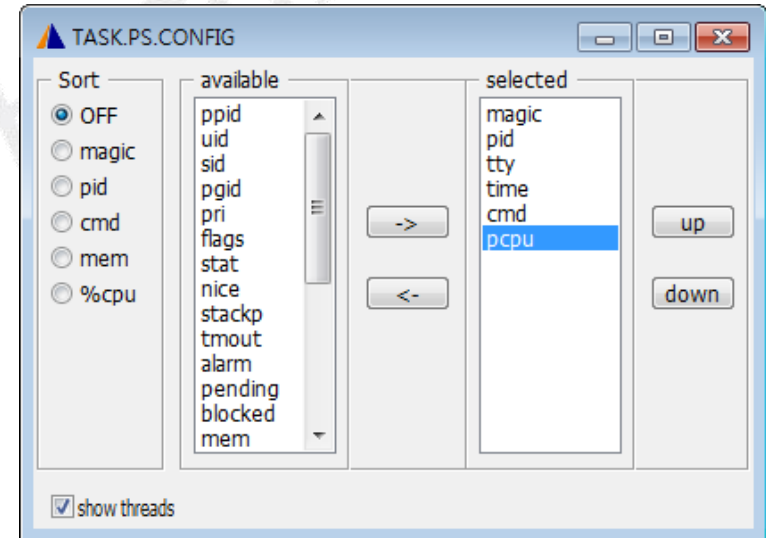
Display Processes / ps-like / Tasks 관련 기능을 실습해 봅니다

모든 Task (Process 또는 Thread) 목록을 확인하고 관련기능과 연동되는 윈도우  
TCB 정보 및 라이브러리 목록 등을 확인 가능



The screenshot shows a window titled "B::TASK.PS" with a "CONFIG..." button. It displays a table of processes with columns: magic, pid, tty, time(ticks), command, and %cpu. The table lists various system processes like swapper, init, kthreadd, ksoftirqd, kworker, migration, khelper, sync\_supers, bdi-default, kblockd, s3c64xx-spi, khubd, kworker, cfg80211, rpciod, kswapd0, fsnotify\_mark, and nfsiod.

magic	pid	tty	time(ticks)	command	%cpu
C06F50F0	0.	-	60.	[swapper]	0.0
D303C000	1.	-	145.	[init]	0.0
D303C320	2.	-	0.	[kthreadd]	0.0
D303C640	3.	-	0.	[ksoftirqd/0]	0.0
D303C960	4.	-	87.	[kworker/0:0]	0.0
D303CC80	5.	-	33.	[kworker/u:0]	0.0
D303CFA0	6.	-	0.	[migration/0]	0.0
D303D2C0	7.	-	0.	[migration/1]	0.0
D303D5E0	8.	-	0.	[kworker/1:0]	0.0
D303D900	9.	-	0.	[ksoftirqd/1]	0.0
D303DF40	10.	-	0.	[khelper]	0.0
D303E260	11.	-	1.	[kworker/u:1]	0.0
D329C000	276.	-	0.	[sync_supers]	0.0
D329C640	278.	-	0.	[bdi-default]	0.0
D329CC80	280.	-	0.	[kblockd]	0.0
D329F840	294.	-	0.	[s3c64xx-spi.0]	0.0
D3167840	297.	-	0.	[s3c64xx-spi.2]	0.0
D303E580	308.	-	0.	[khubd]	0.0
D3164640	366.	-	0.	[kworker/0:1]	0.0
D3164960	367.	-	0.	[cfg80211]	0.0
D329CFA0	385.	-	0.	[rpciod]	0.0
D30E12C0	394.	-	0.	[kswapd0]	0.0
D329F200	445.	-	0.	[fsnotify_mark]	0.0
D30CE8A0	461.	-	0.	[nfsiod]	0.0



# 6-1. Linux Awareness

특정 Process 또는 Thread의 task\_struct 요약 정보를 확인할 수 있습니다

프로세스의 상태  
(eg. SUPERPRIV : super-user 권한)

가족 관계

Argument 정보

환경 변수 정보

Open한 파일  
(eg. Device node file)

Code/Data/Stack  
정보

library 관련 정보  
(section 정보 포함)

시작 시간, sleep기간

magic	command	state	uid	pid	spaceid	tty	flags	cpu
D3165F40	syslogd	sleeping	0.	816.	0330	0	00400040	0.

gid	vm size	ttb	tty name	path
0.	0000031B	D0BFC000	-	/sbin/syslogd

flags: FORKNOEXEC RANDOMIZE

parent: youngest child  
init: sh

arguments: /sbin/syslogd

environment: USER=root, HOME=/, TERM=vt102, PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin, SHELL=/bin/sh, PWD=/

open files: /dev/null, /var/log/messages

code	addr/size	data	addr/size	stack	start
00008000	/ 000DB9F4	000EBF04	/ 0000086E	BEDEFEB0	

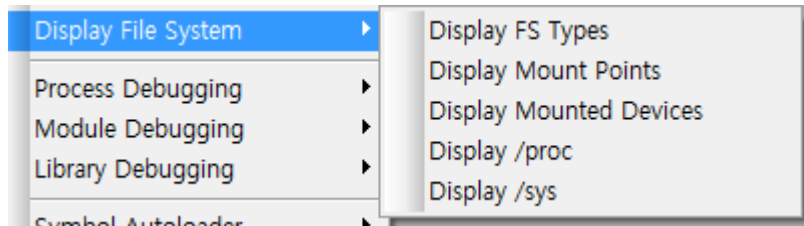
code file	start address	flags
busybox	00008000	EX RD
busybox	000EB000	RD
busybox	000EC000	WR RD
ld-2.12.2.so	400C0000	EX RD
ld-2.12.2.so	400E4000	RD
ld-2.12.2.so	400E5000	WR RD
libm-2.12.2.so	401B3000	EX RD
libm-2.12.2.so	40251000	
libm-2.12.2.so	40258000	RD
libm-2.12.2.so	40259000	WR RD
libc-2.12.2.so	4025A000	EX RD
libc-2.12.2.so	4037C000	
libc-2.12.2.so	40383000	RD
libc-2.12.2.so	40385000	WR RD

start time	sleep time	user time	system time
0AB29A13	000001FF	00000000	00000001

# 6-1. Linux Awareness

## Display File System 정보를 확인하고 이해합니다



커널에 로드 되어 있는 파일 시스템  
또는 모듈의 목록을 확인할 수  
있습니다

**B::TASK.FS.Types**

magic	name	#devs
C042F59C	sysfs	1.
C042F788	rootfs	1.
C042EEC0	bdev	1.
C042F290	proc	1.
C042F104	binfmt_mi	0.
C0448A38	sockfs	1.
C04434EC	usbfs	1.
C042E884	pipefs	1.
C042F034	anon_inod	1.
C042E380	tmpfs	5.
C042EF94	inotifyfs	1.

**B::TASK.FS.Mount**

magic	device	mountpoint
CF80E7A0	rootfs	/
CF80E220	/dev/root	/
CF80EF20	/proc	proc
CF80EEA0	sysfs	sys
CF80EE20	ramfs	dev
CF80EDA0	tmpfs	/
CF80ED20	devpts	/pts
CF80ECA0	usbfs	/bus/usb
CF80EC20	tmpfs	var/volat
CF80EBA0	tmpfs	/shm
CF80EB20	tmpfs	media/ram

**B::TASK.FS.MountDevs**

magic	dev#	fsmagic	type	ro
-------	------	---------	------	----

**B::TASK.FS.PROC**

name	address	mode	links	uid	gid
/proc	C042F228	dr-xr-xr-x	12.	0.	
asound	CF9686C0	dr-xr-xr-x	3.	0.	
mtdev	CFBFE1A0	-r--r--r--	1.	0.	
sysrq-trigger	CFBFEEA0	--w-----	1.	0.	
partitions	CF867D60	-r--r--r--	1.	0.	
diskstats	CF867DE0	-r--r--r--	1.	0.	
crypto	CF867E60	-r--r--r--	1.	0.	
yaffs	CF9B6140	-r--r--r--	1.	0.	
kpageflags	CF9B6540	-r--r--r--	1.	0.	
kpagecount	CF9B65C0	-r--r--r--	1.	0.	
kmsg	CF9B6640	-r--r--r--	1.	0.	

**B::TASK.FS.SYS**

name	address	mode	count
/sys	C042F570	drwxr-xr-x	
fs	CF812428	drwxr-xr-x	
devices	CF8123F8	drwxr-xr-x	
dev	CF8123C8	drwxr-xr-x	
bus	CF812338	drwxr-xr-x	
class	CF812308	drwxr-xr-x	
firmware	CF8122D8	drwxr-xr-x	
kernel	CF812D88	drwxr-xr-x	
power	CF812C68	drwxr-xr-x	
module	CF84DD28	drwxr-xr-x	
block	CF93EB48	drwxr-xr-x	



## 6-2. Autoloader 활용

Autoloader는 Kernel Symbol인 VMLinux를 제외한 Process, Module, Library 대한 Symbol 정보를 자동으로 Load하는 기능입니다

The screenshot shows the 'B::TASK.Process' window with a list of processes. The 'kload' process is selected, and a context menu is open. The menu options are:

- Display detailed
- Display task struct
- Display Stack Frame
- Display Registers
- Switch Context
- Load Process Symbols
- Delete Process Symbols
- Add Libraries to Symbol Autoloader
- Add ALL Libraries to Symbol Autoloader

magic	command	#thr	state	spaceid	pids
C041F128	swapper	20.	current	0000	0. 2.
CF81FA40	init	-	sleeping	0001	1.
CF88D440	dropbear	-	sleeping	0481	1153.
CF9054E0	dbus-daemon	-	sleeping	048A	1162.
CF8506E0	syslogd	-	sleeping	0490	1168.
CF8F0B00	kload	-	sleeping	0492	1170.

The 'Switch Context' menu is open, showing the following options:

- Load Process Symbols
- Delete Process Symbols
- Add Libraries to Symbol Autoloader
- Add ALL Libraries to Symbol Autoloader

The 'libXau.so.6.0.0' library entry is selected, and a context menu is open. The menu options are:

- Display Library struct
- Load Library Symbols
- Delete Library Symbols

libX11.so.6.2.0	40330000
libX11.so.6.2.0	4035D000
libgcc_s.so.1	40361000
libgcc_s.so.1	40368000
libgcc_s.so.1	40372000
libXau.so.6.0.0	40373000
libXau.so.6.0.0	
libXau.so.6.0.0	
libXdmcp.so.6.0.0	
libXdmcp.so.6.0.0	
libXdmcp.so.6.0.0	
libdl-2.6.1.so	

이 기능은 Process의 목록에서 특정 디버깅 심볼 정보가 필요할 때 직관적으로 사용할 수 있다는 장점이 있고 또한 다음 장의 Autoloader 메뉴의 List Components 기능과도 관련이 있습니다

## 6-2. Autoloader 활용

TASK.Process 또는 TASK.DTask 를 통해 Symbol load 하는 방법을 익힙니다

The screenshot shows the Android Studio interface with the following components:

- Linux Dalvik Window Help** menu bar.
- Display Processes** button (highlighted with a red box).
- Display Tasks** button (highlighted with a red box).
- B::TASK.Process** window showing a list of processes. The **mediaserver** process is highlighted with a red box.
- B::TASK.DTask** window showing a list of tasks. The **mediaserver** task is highlighted with a red box.
- B::Task.DTask 0xE743C840** window showing detailed information for the selected task, including flags, relationship, arguments, environment, open files, addresses, code files, and times.

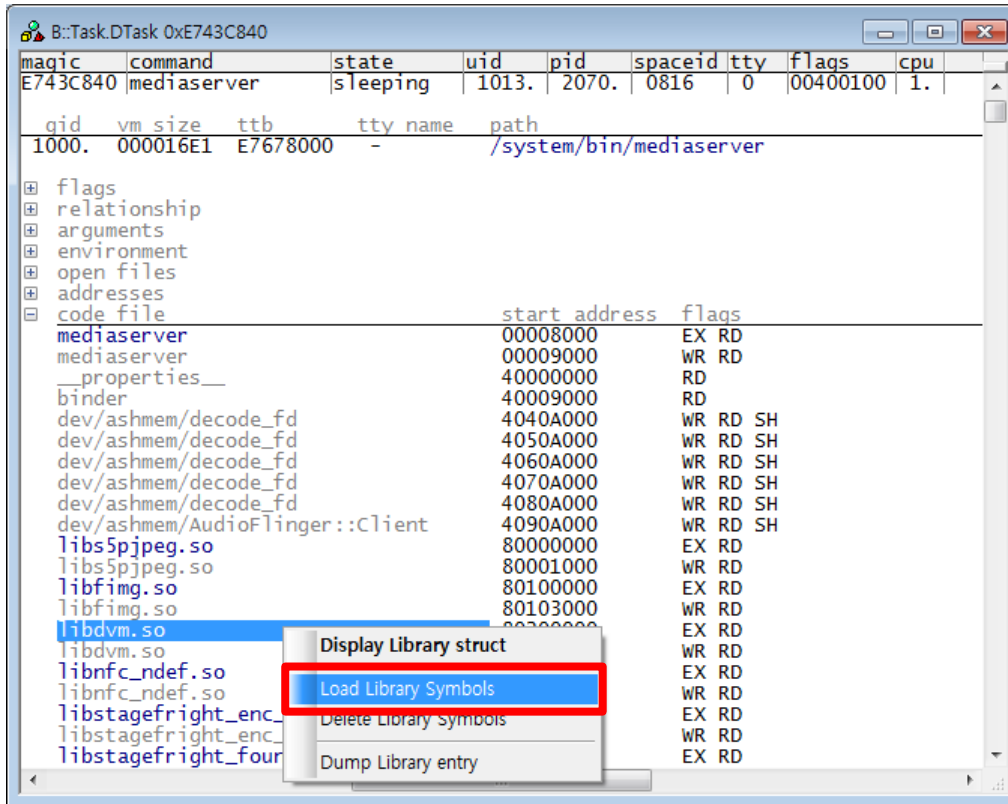
Red arrows indicate the flow of interaction: from the **Display Processes** button to the **B::TASK.Process** window, then to the **mediaserver** process, and finally to the **B::TASK.DTask** window.

디버깅을 원하는 라이브러리를 사용하는 프로세스 이름 또는 Magic number(address of *task\_struct*)를 더블 클릭하여 특정 프로세스의 자세한 정보를 확인하도록 합니다

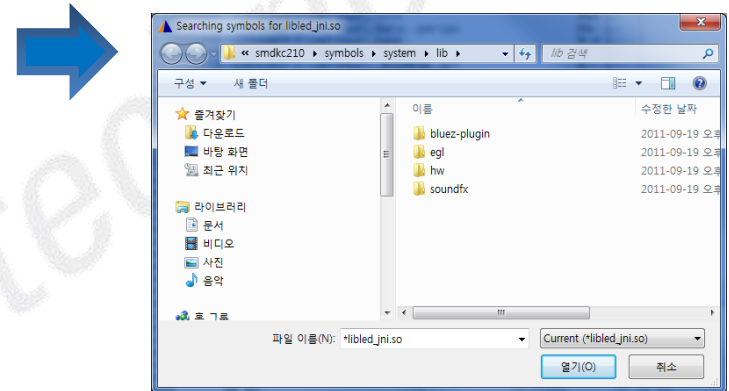


## 6-2. Autoloader 활용

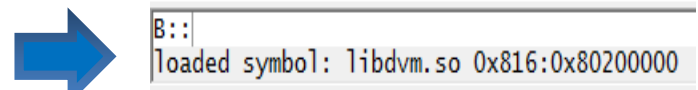
TASK.Process 또는 TASK.DTask 를 통해 Symbol load 하는 방법을 익힙니다



파일이 존재하지 않을 경우



파일이 존재 할 경우



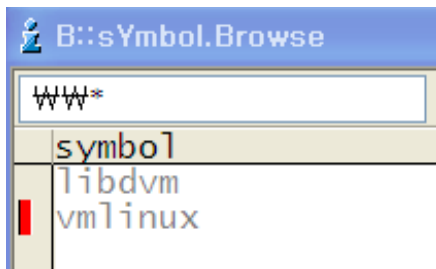
특정 프로세스의 자세한 태스크 정보 중에서 **code files**부분을 확장시켜서 확인하고 디버깅을 원하는 라이브러리를 선택하여 마우스 오른쪽 버튼을 누르고 Load Library symbols를 선택하여 심볼 이미지를 선택하고 로드 합니다

## 6-2. Autoloader 활용

TASK.Process 또는 TASK.DTask 를 통해 Symbol load 하는 방법을 익힙니다

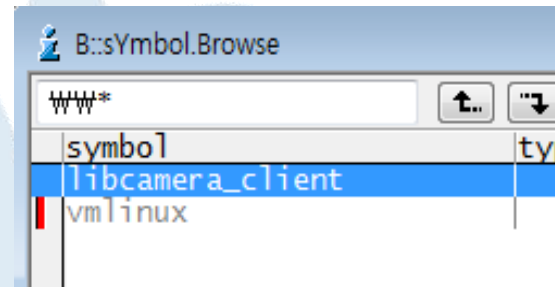
Zygote process의 libdvm.so를 load하는 경우

```
B::task.symbol.loadlib "zygote" "libdvm.so"
```



Mediaserver의 libcamera\_client.so를 load하는 경우

```
B::task.symbol.loadlib "mediaserver" "libcamera_client.so"
```

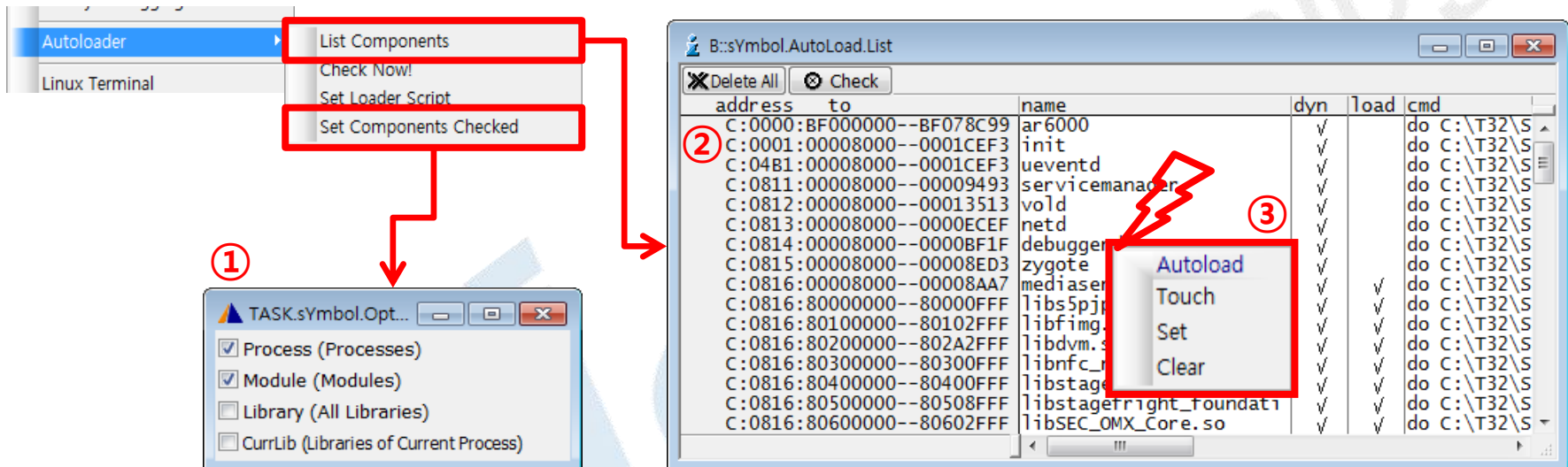


추가적으로 명령어를 통한 Library Symbol Loading 하는 방법이 있다.

Display Processes window는 data를 계속 access하는 window로 open이 되면 느려지는 현상이 발생 할 수 있다. 이 때 symbol을 load하면 느리다는 느낌이 들기 때문에 명령어를 통해 빠른 symbol loading을 할 수 있다.

## 6-2. Autoloader 활용

List Components (sYmbol.AutoLoad.List) 와 Set Components Checked (TASK.sYmbol.Option AutoLoad) 기능을 통한 Symbol load 방법을 익힙니다



디버깅하길 원하는 구성 요소(Component)를 선택하여 선택 디버깅하는 기능

- Check : 수동적으로 목록의 새로 고침을 수행합니다.
- Touch : 수동적으로 심볼 정보 파일을 로드 하도록 합니다.
- Set : 특정 심볼 정보가 이미 로드 된 것처럼 마크합니다  
(심볼 파일이 없을 경우 유용합니다.)
- Clear : Load로 마크된 특정 심볼 정보를 언-마크합니다.

# 7. Process(Daemon)

---

Linux에서 Daemon(Process) 디버깅을 어떻게 하는지 이해합니다

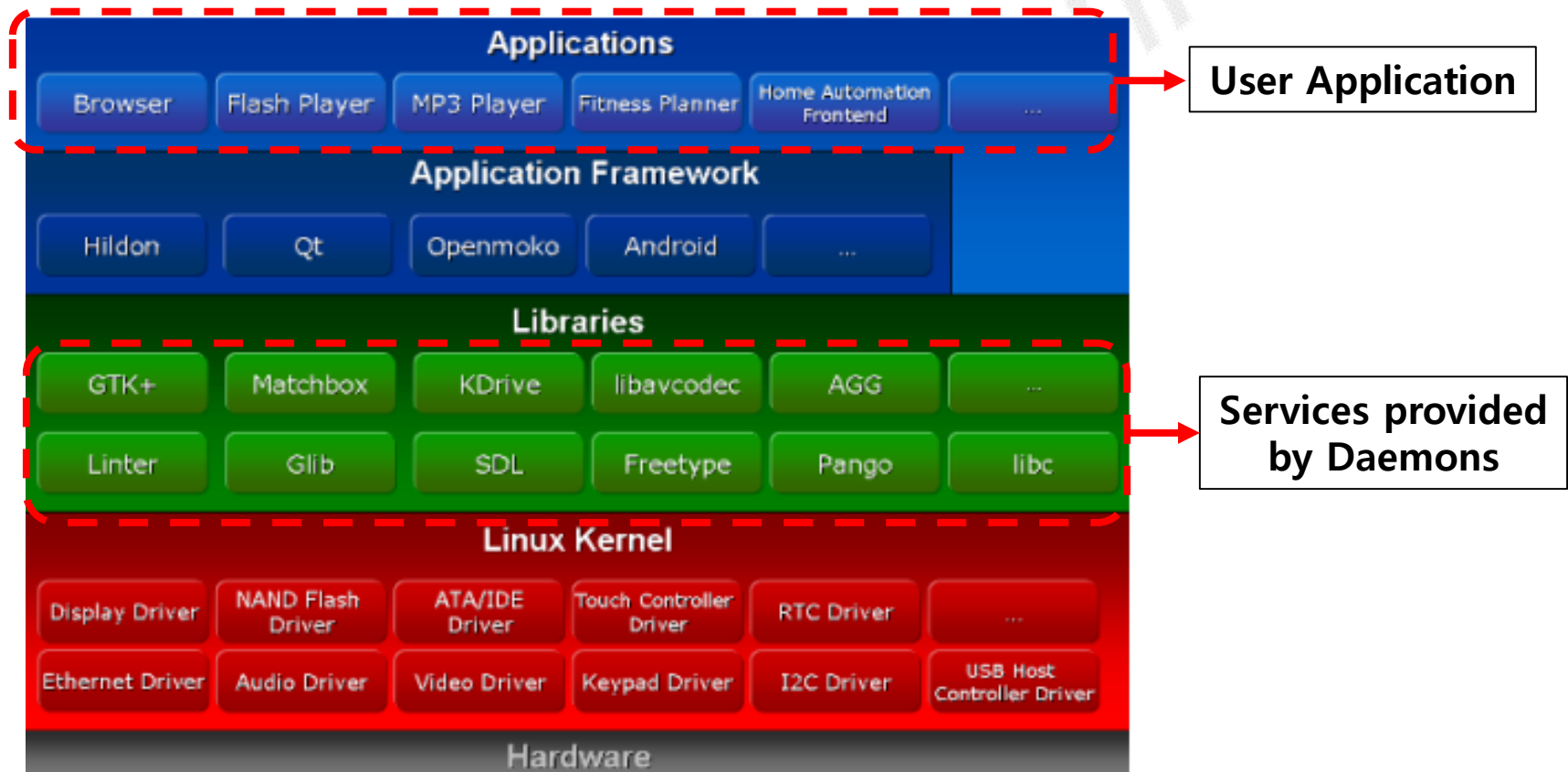
1. Linux Application & Daemon
2. Application main 함수부터 디버깅 실습

# 7-1. Linux application & Daemon

User Application과 Native Process인 Daemon으로 구분

Linux 에서는 두 종류의 Process(application & daemon)가 존재합니다

- Linux application은 부팅 후 사용자에게 의해 실행되는 User Process이고  
Daemon은 보통 Native Library Load를 위한 Native Process 입니다

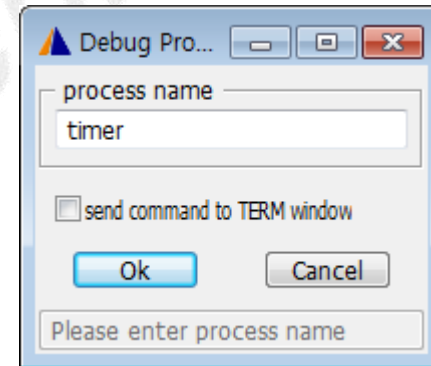
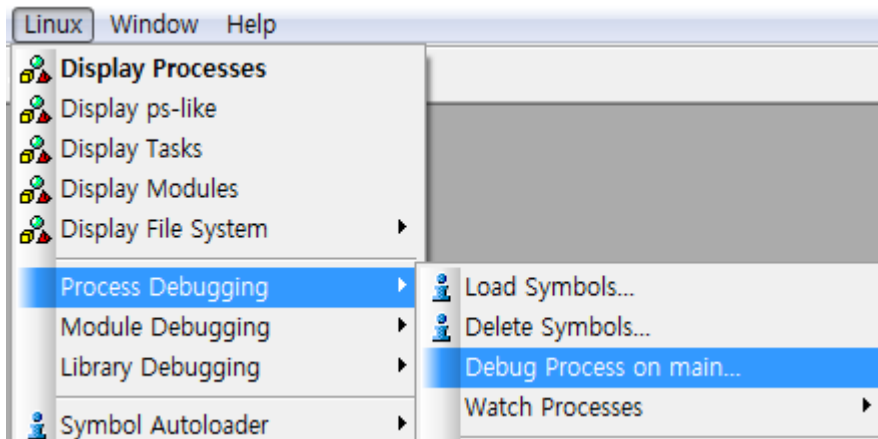


## 7-2. APP main 함수부터 디버깅 실습

실습을 통해 Application의 main 부터 디버깅 방법을 이해합니다

### Timer 어플리케이션 디버깅

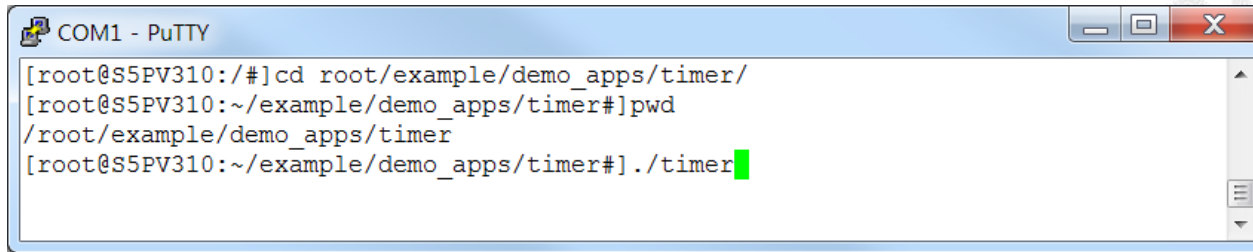
- linux\_debug.cmm 실행
- 부팅 완료 후 main 함수 디버깅 실행



## 7-2. APP main 함수부터 디버깅 실습

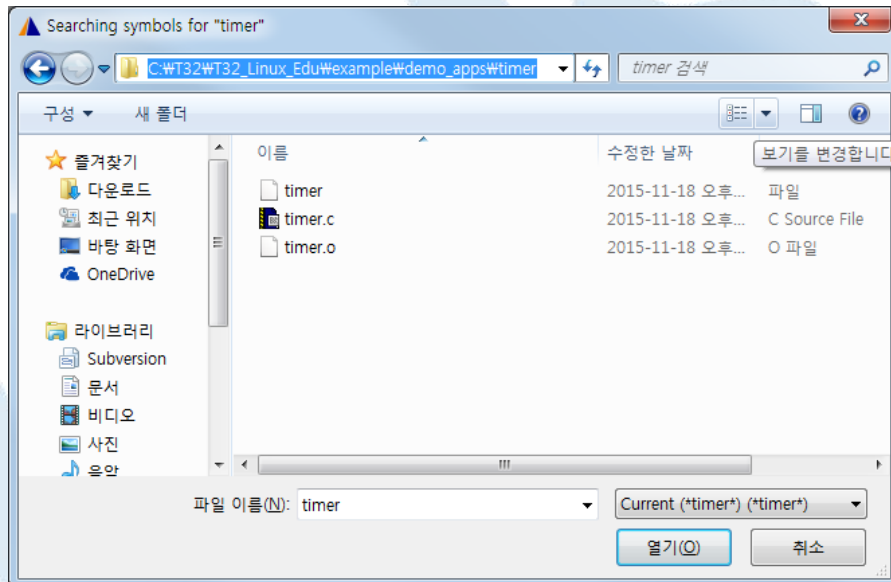
### Timer 어플리케이션 디버깅

- 터미널에서 timer 어플리케이션 실행



```
COM1 - PuTTY
[root@S5PV310:/#]cd root/example/demo_apps/timer/
[root@S5PV310:~/example/demo_apps/timer#]pwd
/root/example/demo_apps/timer
[root@S5PV310:~/example/demo_apps/timer#] ./timer
```

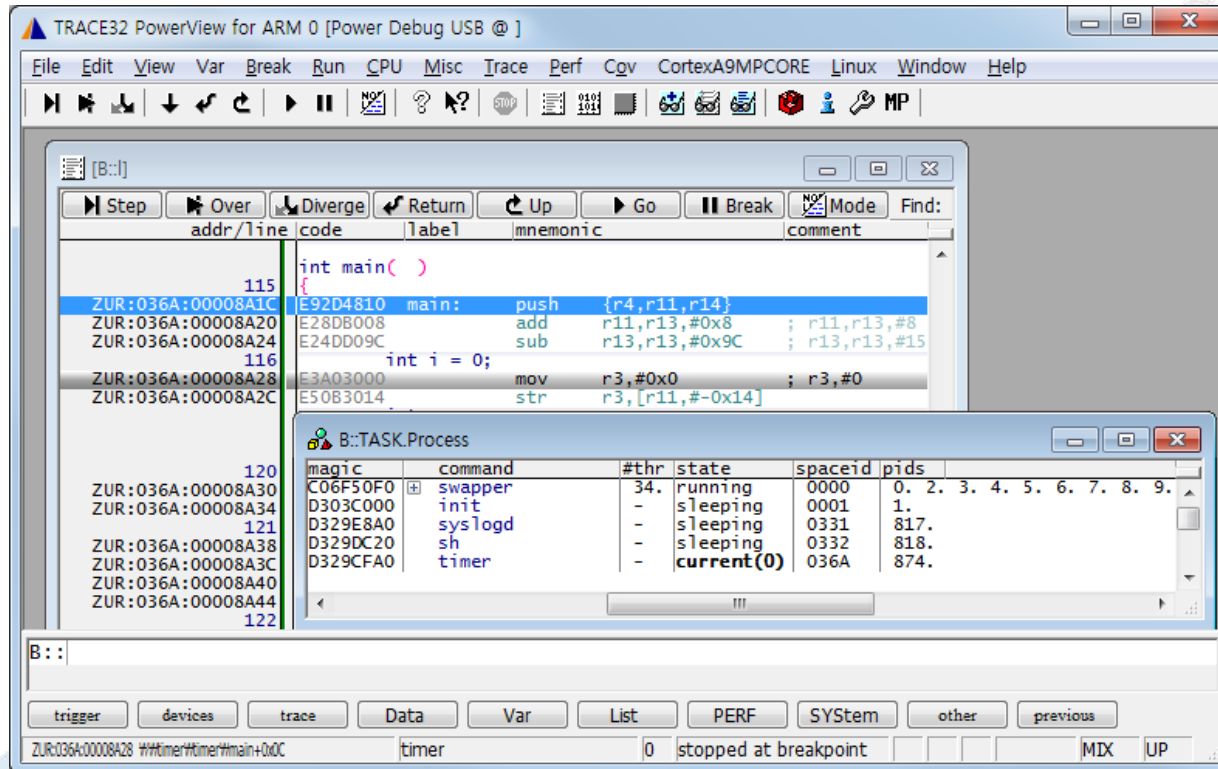
- PowerView에서 timer에 대한 심볼 선택



## 7-2. APP main 함수부터 디버깅 실습

### Timer 어플리케이션 디버깅

- Main 함수부터 디버깅 가능

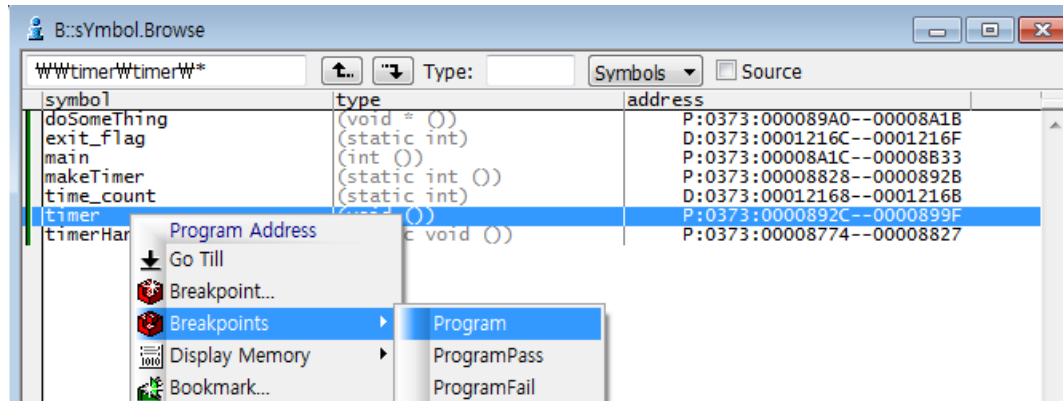




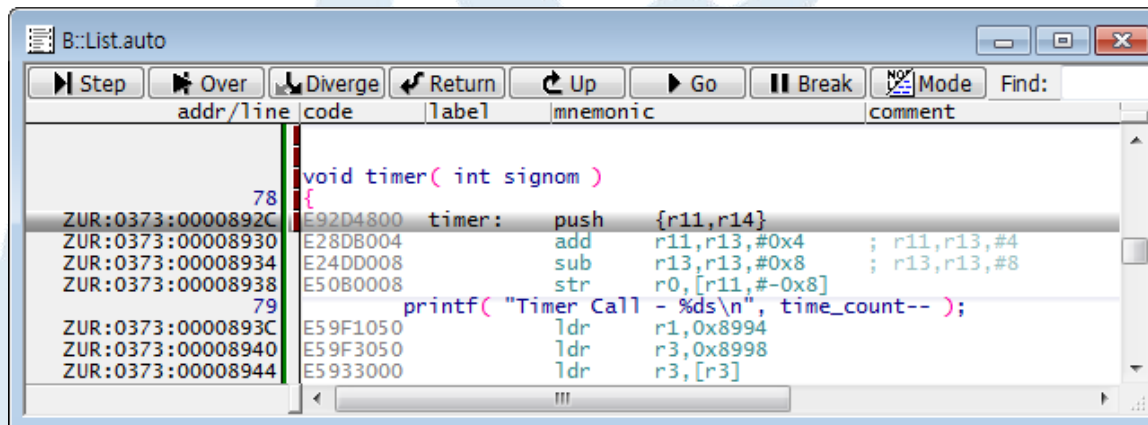
## 7-2. APP main 함수부터 디버깅 실습

### Timer 어플리케이션 디버깅

- Timer 어플리케이션의 timer 함수에 Break Point 설정



- Timer 함수 Break 된 모습



# 8. Library

---

Linux Library에 대해서 이해하고 Library 디버깅을 할 수 있습니다.

1. Linux Library
2. Library 종류
3. Static Library 실습
4. Shared Library - Dynamic Linking 실습
5. Shared Library - Dynamic Loading 실습

# 8-1. Linux Library

---

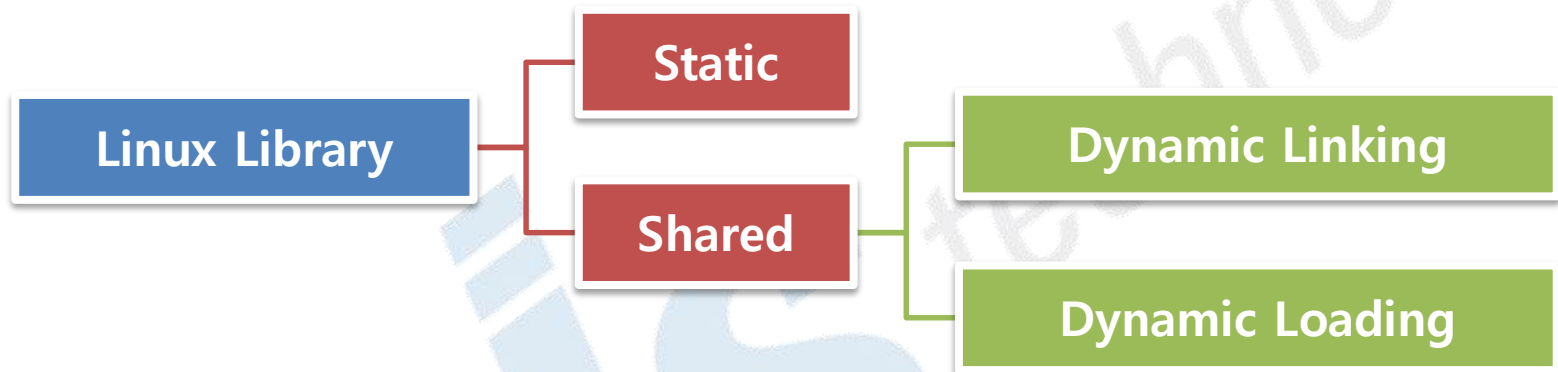
Linux 에서 실행되는 대부분의 Application들은 System Resource들을 사용하기 위해 Library에 있는 코드들을 수행하게 됩니다

- 대부분의 Application들은 Library들을 사용하게 됩니다
- 이 Library들은 부팅 중 Daemon에 의해 대부분 Memory에 Load됩니다

## 8-2. Library 종류

Library의 종류에 대해서 알아보고 이해합니다

Library는 Linking과 loading 방식에 따라 구분할 수 있다.



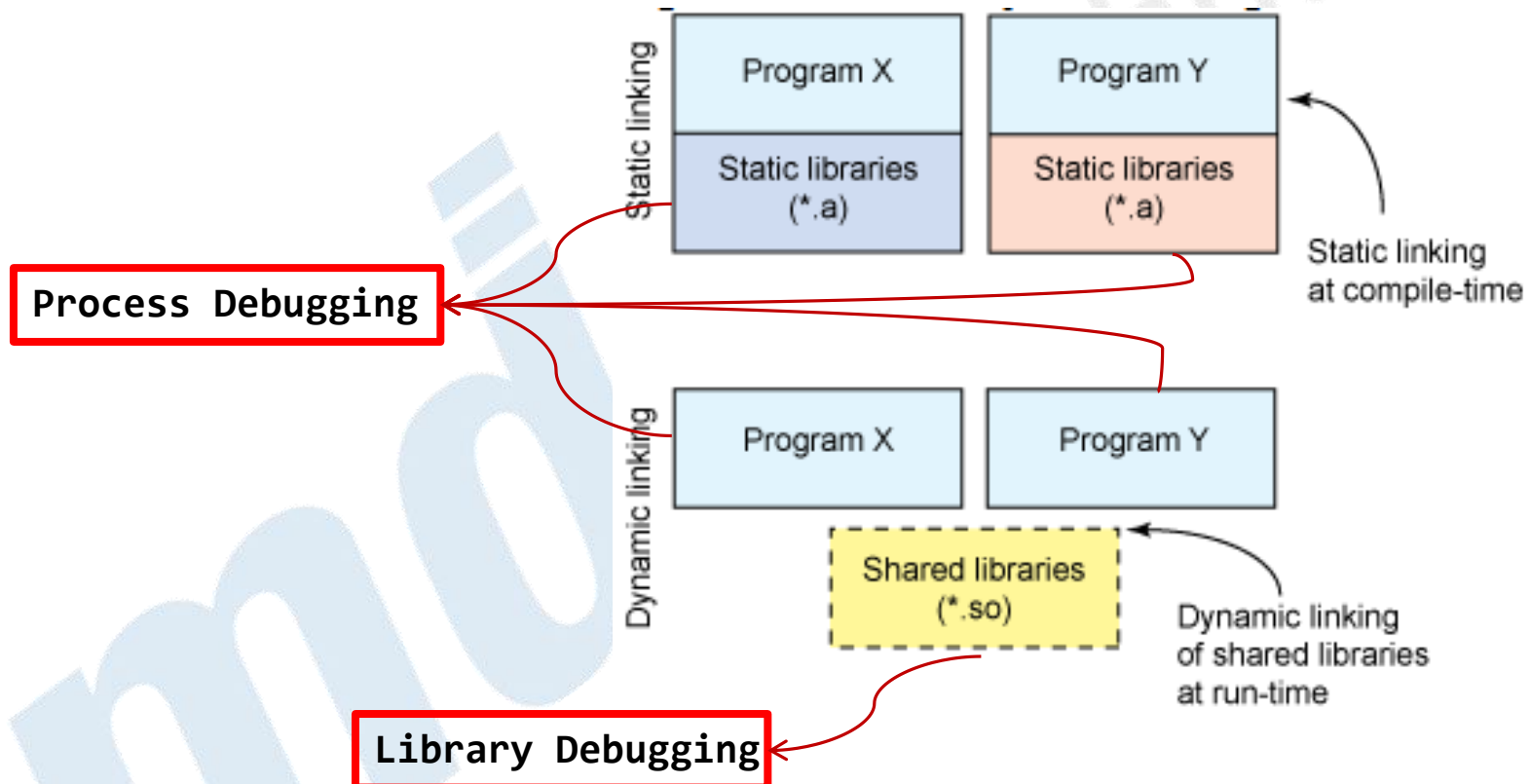
종 류	프로그램에 라이브러리 적재 시기	라이브러리 사용
정적 라이브러리 (Static Library)	컴파일 타임	컴파일 시 라이브러리를 적재한 프로그램만 라이브러리 코드를 사용
공유 라이브러리 (Dynamic Linking)	런타임(프로그램 메모리에 적재 시)	메모리에 라이브러리 적재 되 있으면 라이브러리 사용하는 프로그램끼리 메모리영역을 공유
동적 라이브러리 (Dynamic Loading)	런타임(프로그램 실행 중 필요할 때) 즉 사용하는 응용프로그램이 결정	메모리에 라이브러리 적재 되 있으면 라이브러리 사용하는 프로그램끼리 한 메모리 영역을 공유

## 8-3. Static Library 실습

Static Library의 경우 Process 디버깅 하는 것과 동일하게 디버깅이 가능합니다.

### Static Library 디버깅

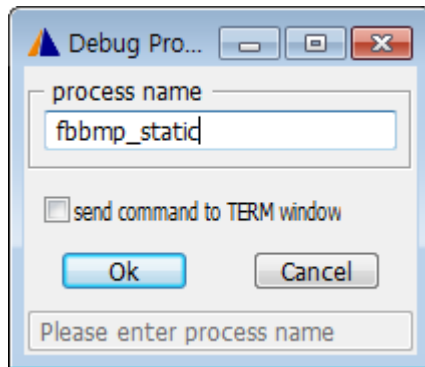
- Static Library 디버깅은 Process 디버깅과 동일



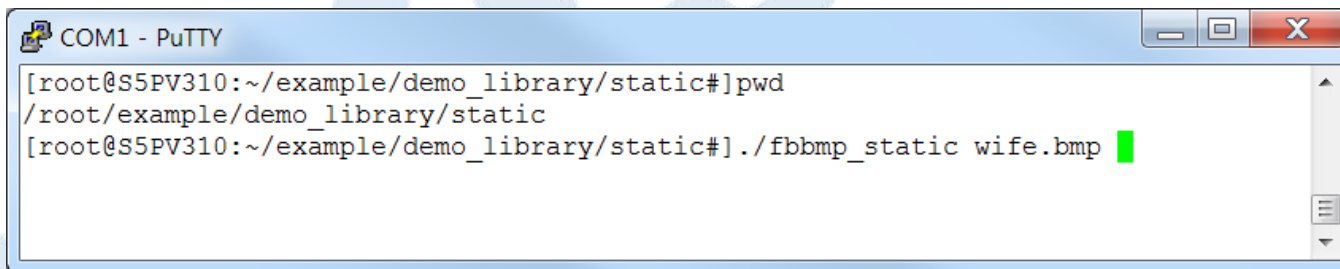
## 8-3. Static Library 실습

### Static Library 디버깅

- fbbmp\_static 을 어플리케이션 디버깅과 동일하게 진행



- 터미널에서 fbbmp\_static 어플리케이션을 실행



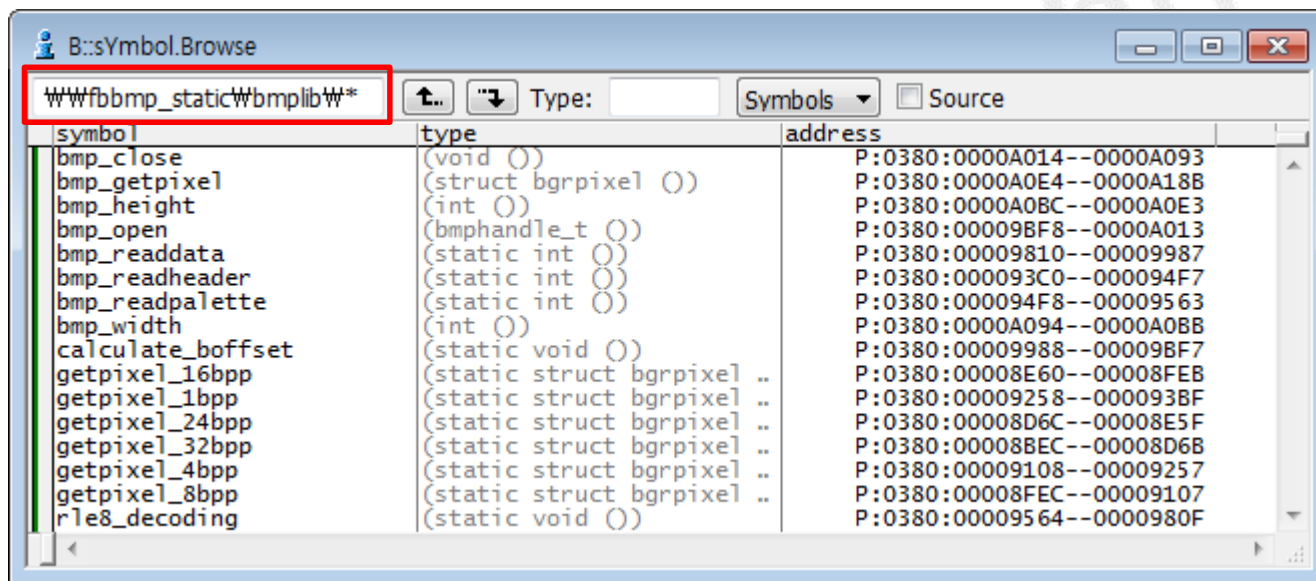
```
COM1 - PuTTY
[root@S5PV310:~/example/demo_library/static#]pwd
/root/example/demo_library/static
[root@S5PV310:~/example/demo_library/static#] ./fbbmp_static wife.bmp
```

## 8-3. Static Library 실습

### Static Library 디버깅

- fbbmp\_static의 어플리케이션의 static library 함수를 확인

심볼위치 : C:\T32\T32\_Linux\_Edu\example\demo\_library\static

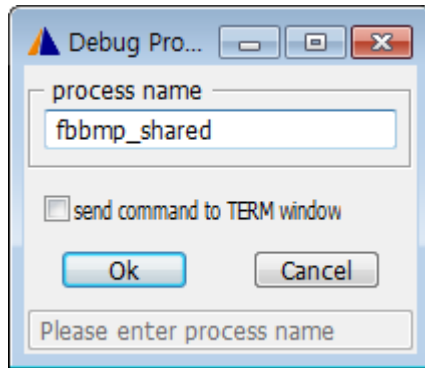


## 8-4. Shared Library - Dynamic Linking 실습

실습을 통해 Dynamic Linking library 디버깅을 이해합니다.

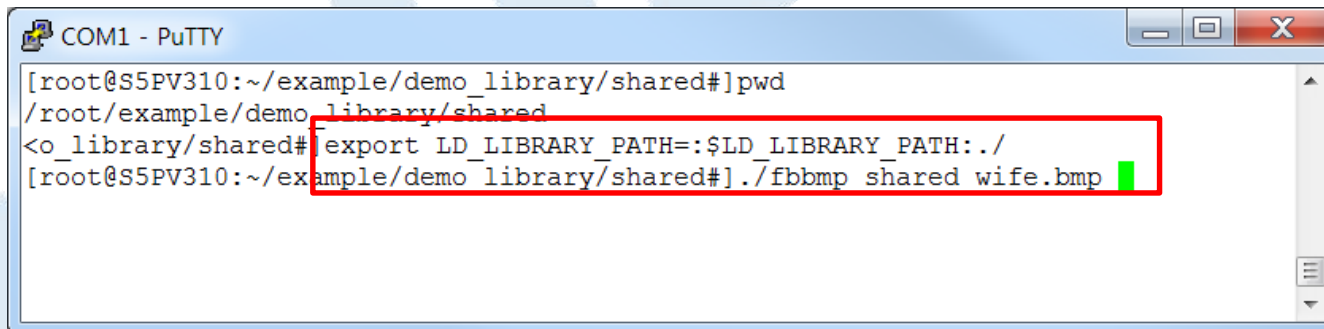
### Dynamic Linking 디버깅

- Dynamic Linking Library 디버깅 하기 위해 프로세스 디버깅 진행



- Library path 등록 후 어플리케이션 실행

심볼위치 C:WT32\WT32\_Linux\_Edu\example\demo\_library\shared



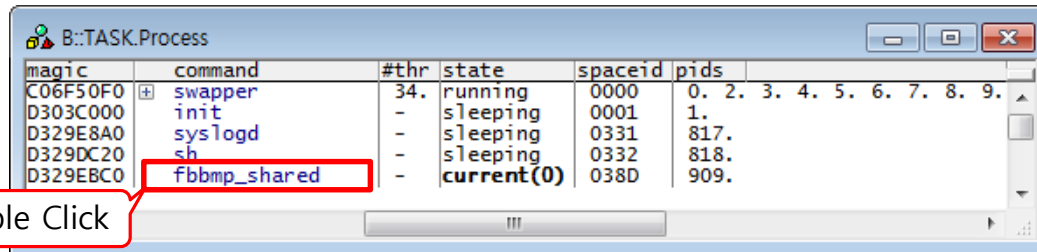
```
COM1 - PuTTY
[root@S5PV310:~/example/demo_library/shared#]pwd
/root/example/demo_library/shared
<root@S5PV310:~/example/demo_library/shared# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./
[root@S5PV310:~/example/demo_library/shared#] ./fbbmp shared wife.bmp
```



## 8-4. Shared Library - Dynamic Linking 실습

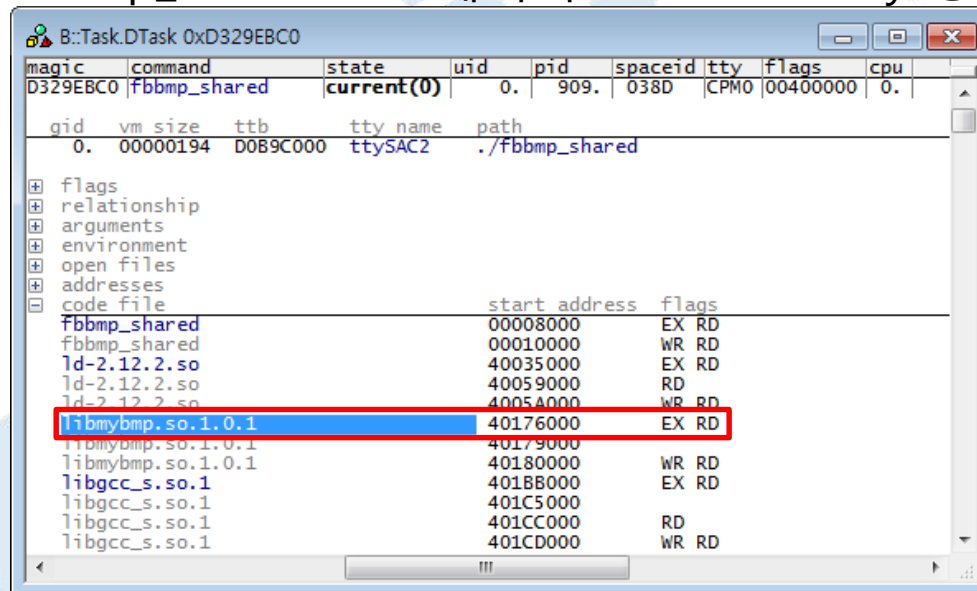
### Dynamic Linking 디버깅

- Task.Process 명령어로 Process 확인 가능



magic	command	#thr	state	spaceid	pids
C06F50F0	swapper	34.	running	0000	0. 2. 3. 4. 5. 6. 7. 8. 9.
D303C000	init	-	sleeping	0001	1.
D329E8A0	syslogd	-	sleeping	0331	817.
D329DC20	sh	-	sleeping	0332	818.
D329EBC0	fbmp_shared	-	current(0)	038D	909.

- fbmp\_shared 프로세서의 Shared Library 정보 확인



magic	command	state	uid	pid	spaceid	tty	flags	cpu
D329EBC0	fbmp_shared	current(0)	0.	909.	038D	CPM0	00400000	0.

qid	vm size	ttb	tty name	path
0.	00000194	D0B9C000	ttySAC2	./fbmp_shared

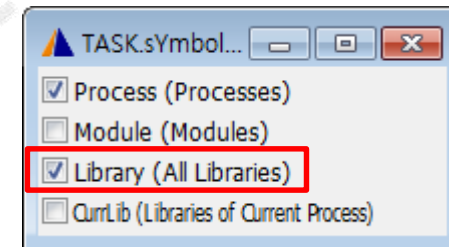
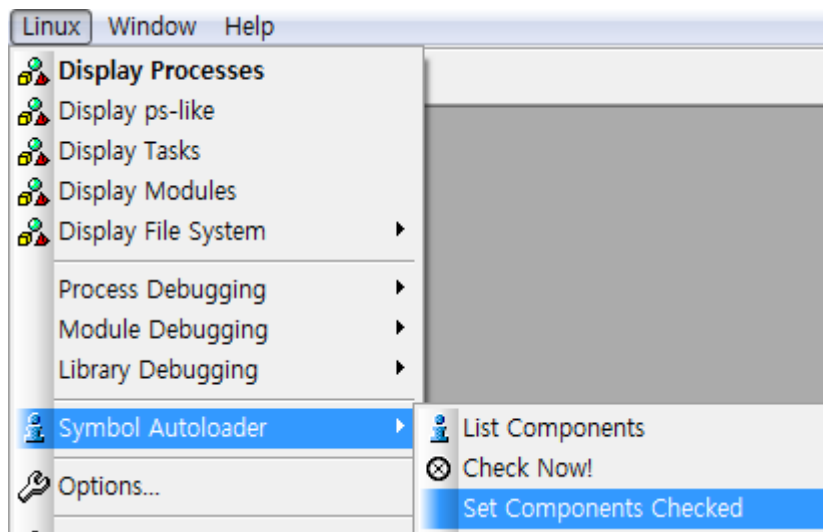
  

code file	start address	flags
fbmp_shared	00008000	EX RD
fbmp_shared	00010000	WR RD
ld-2.12.2.so	40035000	EX RD
ld-2.12.2.so	40059000	RD
ld-2.12.2.so	4005A000	WR RD
libmybmp.so.1.0.1	40176000	EX RD
libmybmp.so.1.0.1	40179000	
libmybmp.so.1.0.1	40180000	WR RD
libgcc_s.so.1	40188000	EX RD
libgcc_s.so.1	401C5000	
libgcc_s.so.1	401CC000	RD
libgcc_s.so.1	401CD000	WR RD

## 8-4. Shared Library - Dynamic Linking 실습

### Dynamic Linking 디버깅

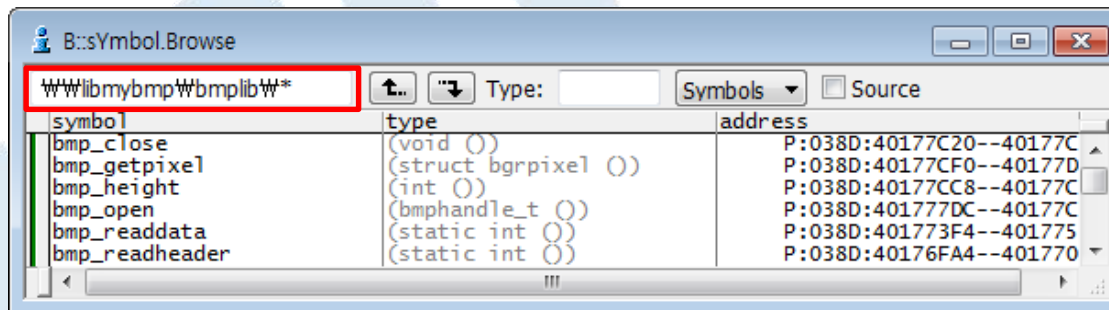
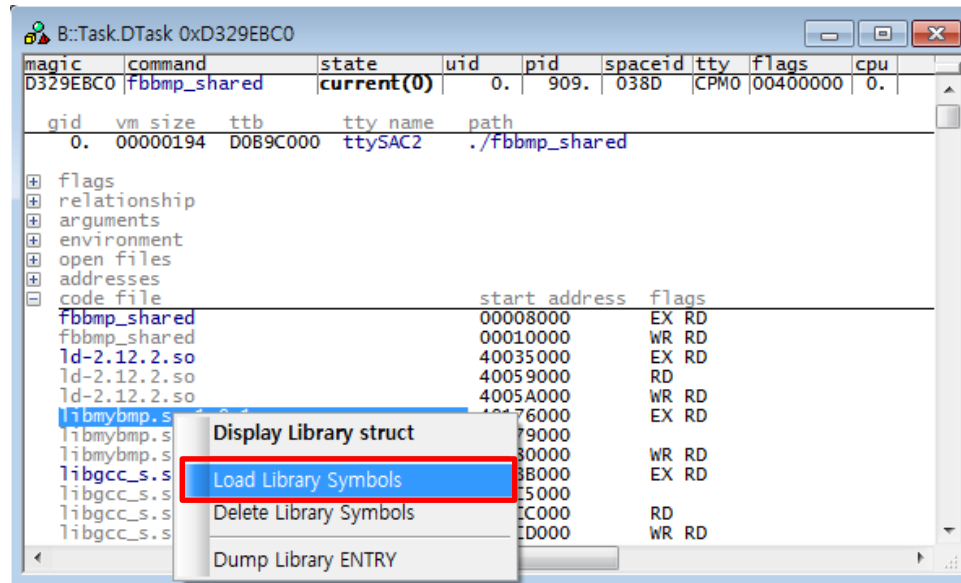
- Library 디버깅 하기 위해서 아래와 같이 Library에 체크



## 8-4. Shared Library - Dynamic Linking 실습

### Dynamic Linking 디버깅

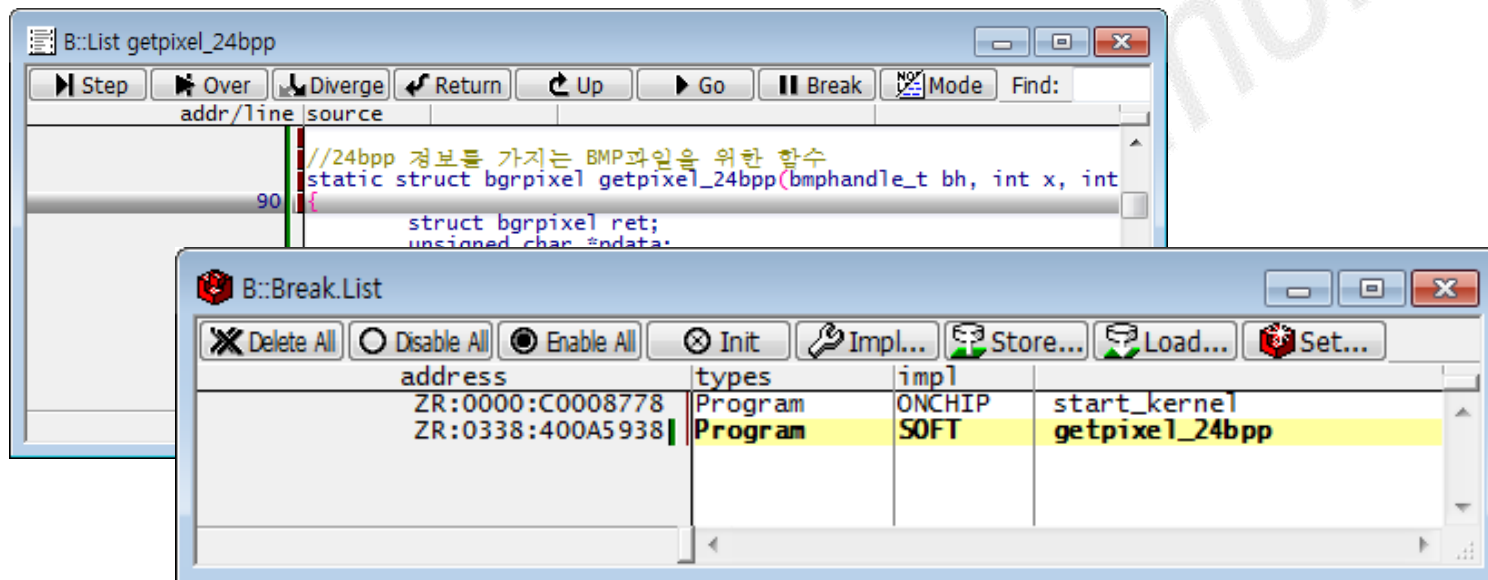
- 해당 Library에서 우측클릭 하여 "Load Library Symbols" 를 통해 심볼 로딩



## 8-4. Shared Library - Dynamic Linking 실습

### Dynamic Linking 디버깅

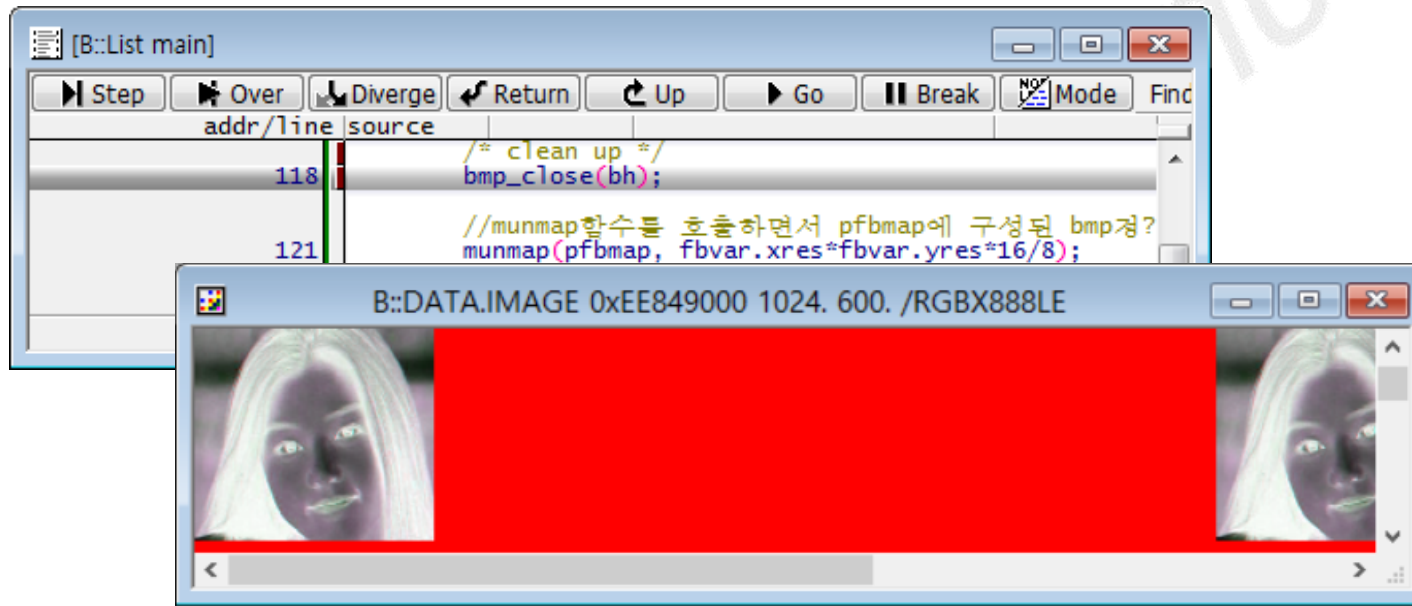
- getpixel\_24bpp 함수에 Break Point를 설정해 확인 가능



## 8-4. Shared Library - Dynamic Linking 실습

### Dynamic Linking 디버깅

- bmp\_close 함수까지 수행하면 LCD화면 확인 가능

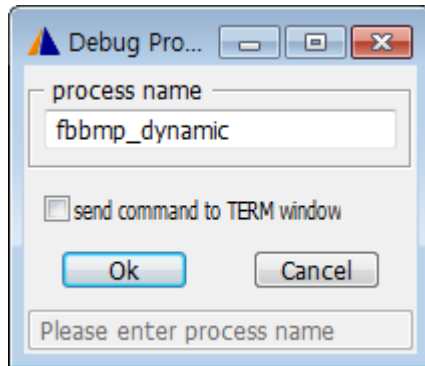


## 8-4. Shared Library - Dynamic Loading 실습

실습을 통해 Dynamic Loading library 디버깅을 이해합니다.

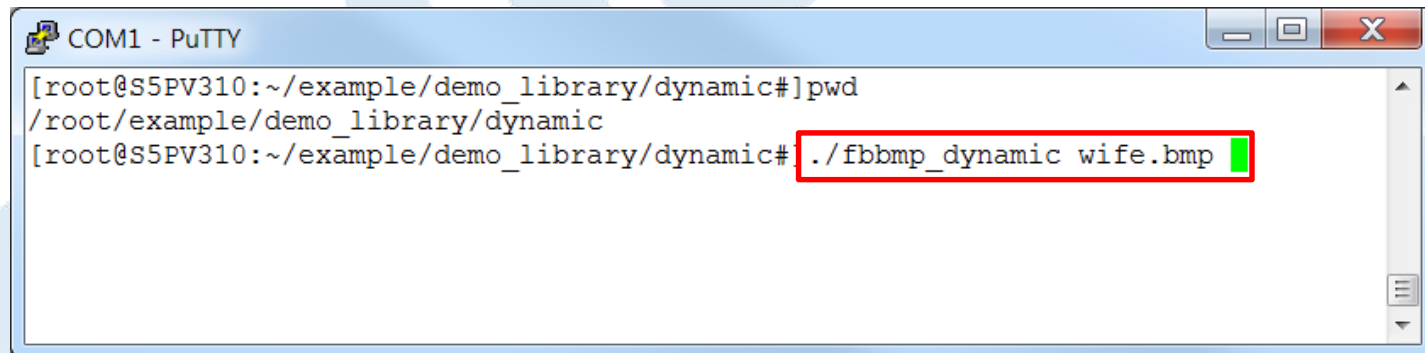
### Dynamic Loading Library 디버깅

- Dynamic Loading Library 디버깅 하기 위해 프로세스 디버깅 진행



- 어플리케이션 실행

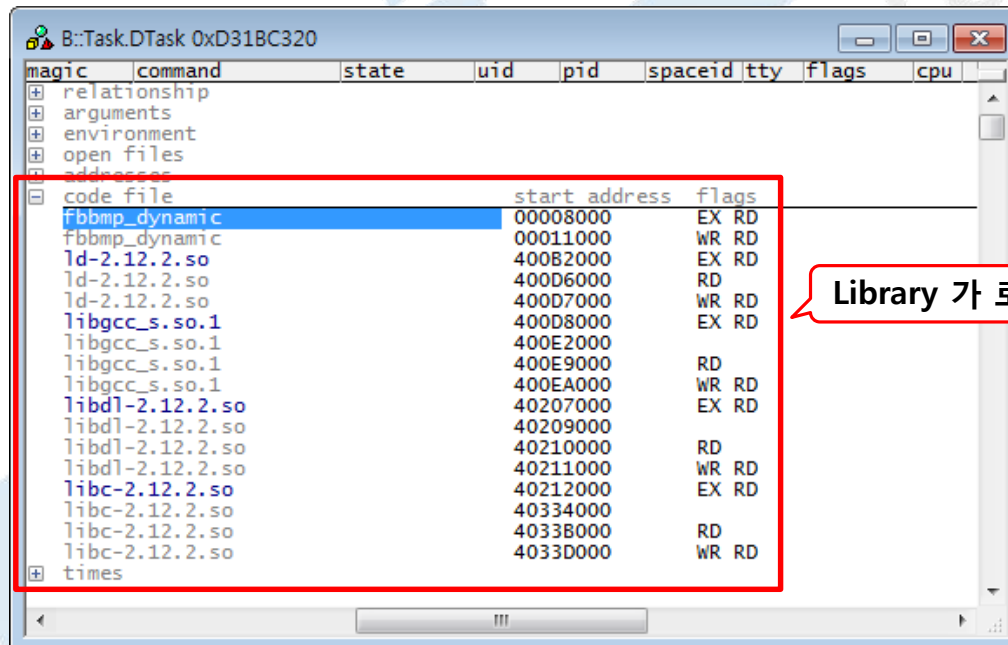
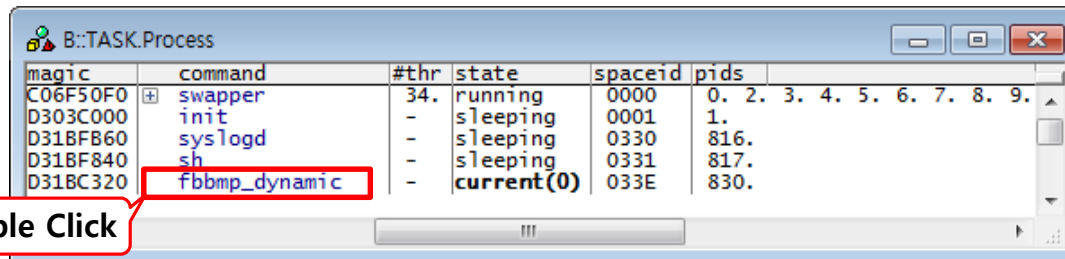
심볼위치 C:WT32\WT32\_Linux\_Edu\example\demo\_library\dynamic



## 8-4. Shared Library - Dynamic Loading 실습

### Dynamic Loading Library 디버깅

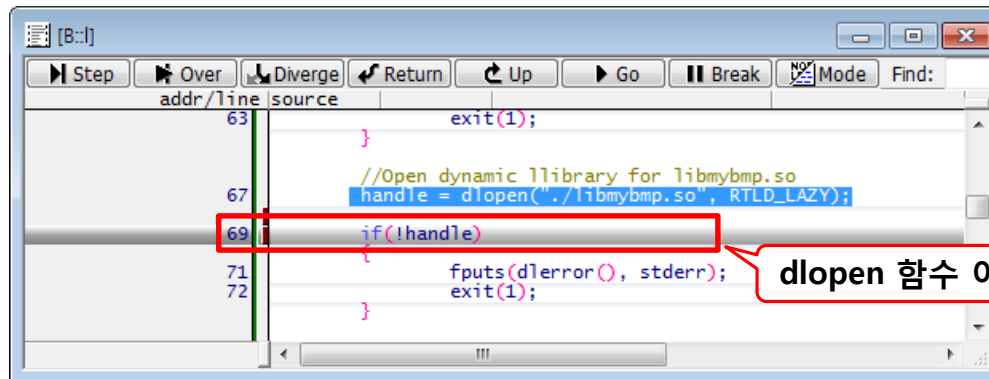
- Main 함수에서 Library 로딩 확인



## 8-4. Shared Library - Dynamic Loading 실습

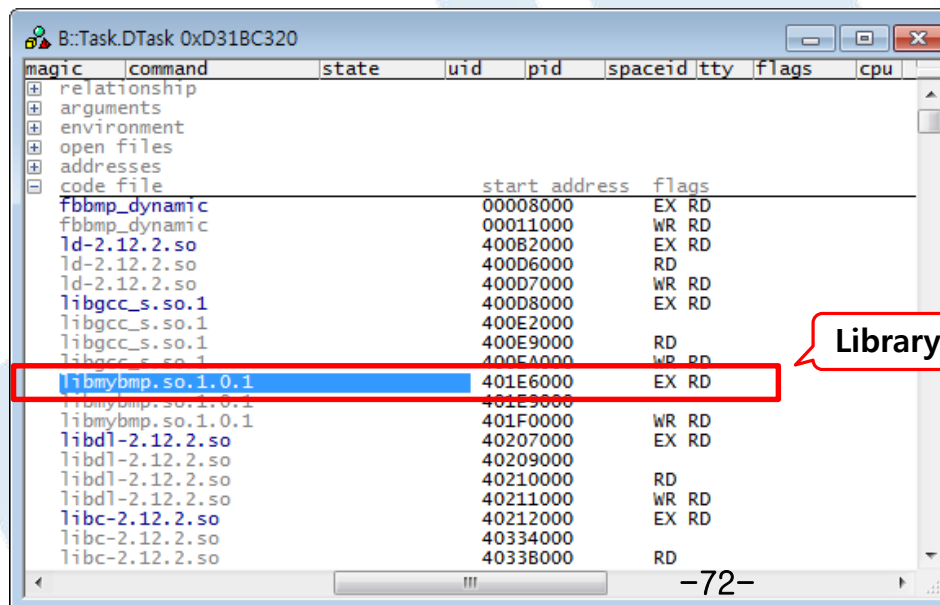
### Dynamic Loading Library 디버깅

- dlopen 함수 이후에 Break 한 후 Library 로드 된 것을 확인.



```
[B::l]
Step Over Diverge Return Up Go Break Mode Find:
addr/line source
63         exit(1);
67         //Open dynamic library for libmybmp.so
        handle = dlopen("./libmybmp.so", RTLD_LAZY);
69         if(!handle)
71         {
72             fputs(dlerror(), stderr);
            exit(1);
        }
```

dlopen 함수 이후까지 Target을 수행



magic	command	state	uid	pid	spaceid	tty	flags	cpu
+	relationship							
+	arguments							
+	environment							
+	open files							
+	addresses							
+	code file							
					start address		flags	
	fbmp_dynamic				00008000		EX RD	
	fbmp_dynamic				00011000		WR RD	
	ld-2.12.2.so				400B2000		EX RD	
	ld-2.12.2.so				400D6000		RD	
	ld-2.12.2.so				400D7000		WR RD	
	libgcc_s.so.1				400D8000		EX RD	
	libgcc_s.so.1				400E2000			
	libgcc_s.so.1				400E9000		RD	
	libgcc_s.so.1				400FA000		WR RD	
	libmybmp.so.1.0.1				401E6000		EX RD	
	libmybmp.so.1.0.1				401E9000			
	libmybmp.so.1.0.1				401F0000		WR RD	
	libdl-2.12.2.so				40207000		EX RD	
	libdl-2.12.2.so				40209000			
	libdl-2.12.2.so				40210000		RD	
	libdl-2.12.2.so				40211000		WR RD	
	libc-2.12.2.so				40212000		EX RD	
	libc-2.12.2.so				40334000			
	libc-2.12.2.so				4033B000		RD	

-72-



## 9. Kernel Module

---

Linux에서는 Kernel에 Built-In되지 않고, 원하는 시점에 동적으로 Load할 수 있는 Module 디버깅 방법에 대해 학습하도록 합니다

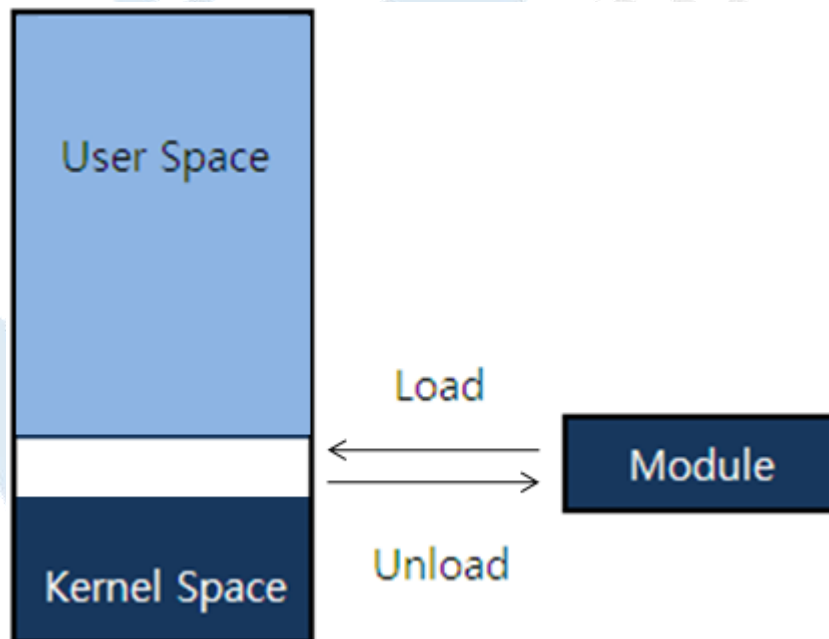
1. Kernel Module의 이해
2. Kernel Module의 init 부터 디버깅

# 9-1. Kernel Module의 이해

Linux 커널에 동적으로 프로그램 코드를 적재하는 LKM에 대해 학습합니다

Device Driver는 H/W를 다루기 위한 Kernel 코드의 종류이며, Linux에서는 동적으로 Load할 수 있는 Kernel Object 형태인 Module로 제공되기도 합니다

- 통상 Device Driver를 Module 방식으로 작성하기 때문에 같은 의미로 통용
- 부팅 중 Kernel에 의해 Device Driver가 등록되는 경우가 많음.
- 일부 동적으로 Memory에 Load하는 Module도 존재(eg. Wifi, GPS)
- 동적으로 동작되는 Module은 insmod/rmmod/lsmmod 등의 명령을 사용



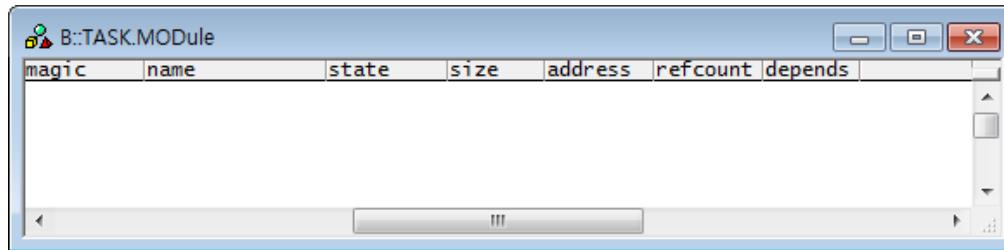
## 9-2. Kernel Module의 init 부터 디버깅

실습을 통해 module의 init 함수 디버깅을 이해합니다

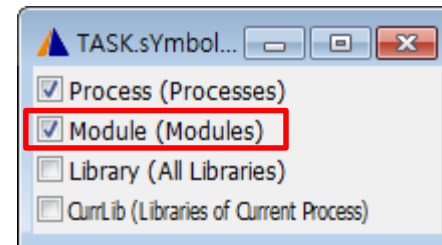
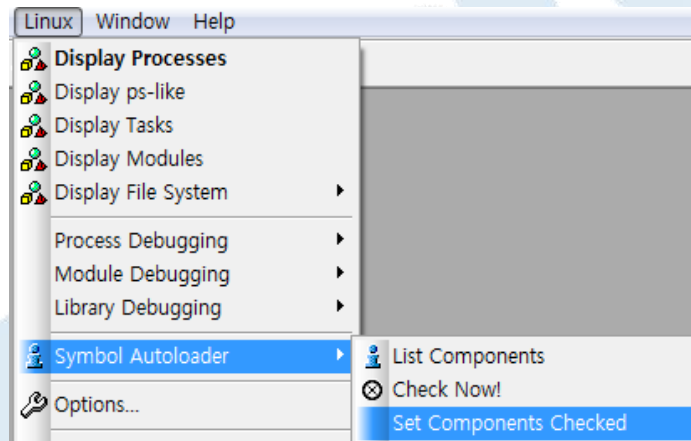
### Module 디버깅

- 로드되어 있는 Module 확인

Module init 부터 디버깅 하기 위해서는 Module이 로드되어 있으면 안됨.



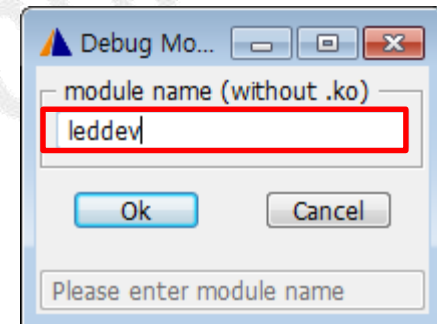
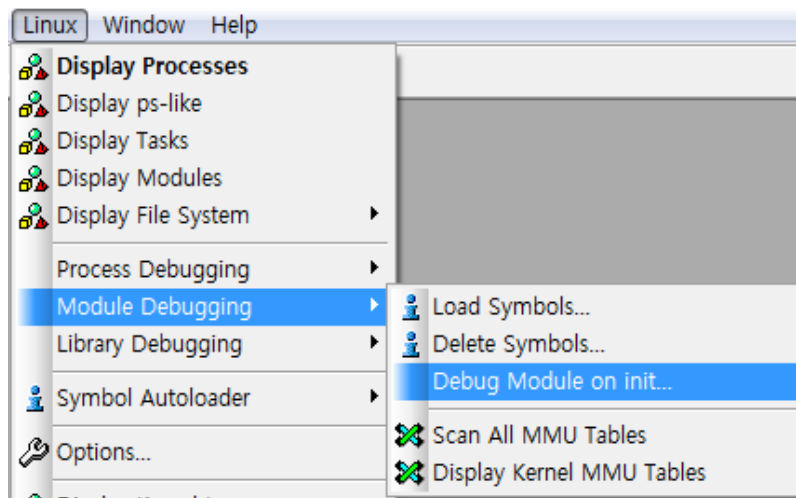
- Module 디버깅 하기 위해서 체크



## 9-2. Kernel Module의 init 부터 디버깅

### Module 디버깅

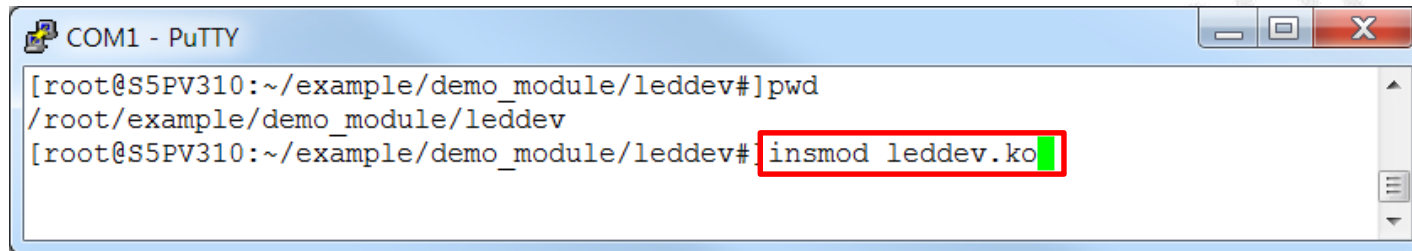
- Module init 부터 디버깅



## 9-2. Kernel Module의 init 부터 디버깅

### Module 디버깅

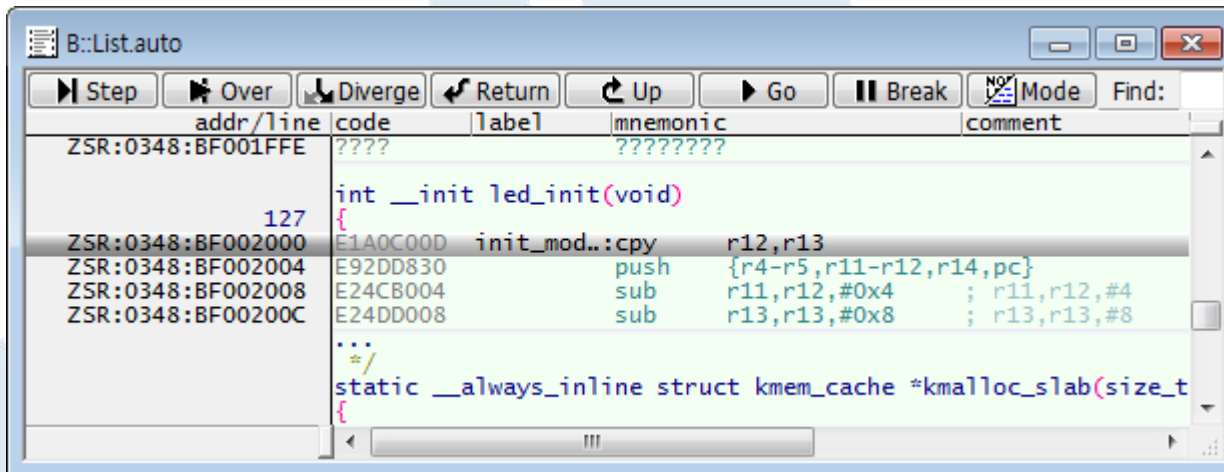
- Module init 부터 디버깅



```
COM1 - PuTTY
[root@S5PV310:~/example/demo_module/leddev#]pwd
/root/example/demo_module/leddev
[root@S5PV310:~/example/demo_module/leddev#]insmod leddev.ko
```

- 심볼로드 후 led\_init 부터 디버깅 가능

심볼위치 : C:\WT32\WT32\_Linux\_Edu\example\demo\_module\leddev



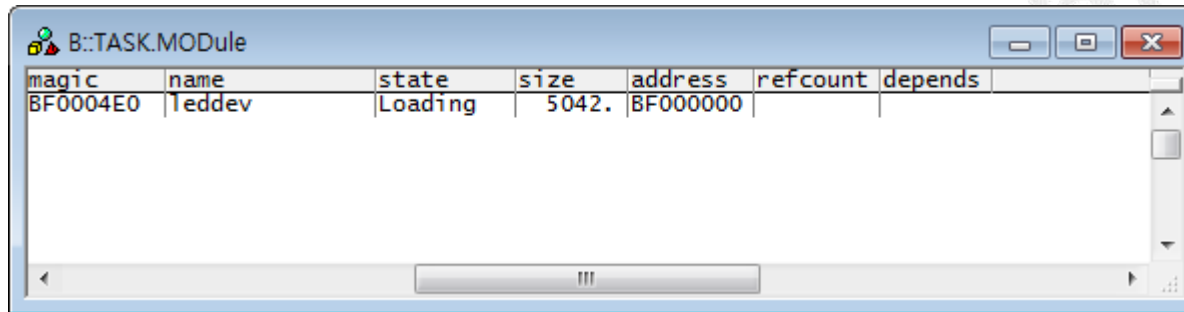
addr/line	code	label	mnemonic	comment
ZSR:0348:BF001FFE	????		????????	
127				
ZSR:0348:BF002000	E1A0C00D	init_mod::cpy	r12,r13	
ZSR:0348:BF002004	E92DD830	push	{r4-r5,r11-r12,r14,pc}	
ZSR:0348:BF002008	E24CB004	sub	r11,r12,#0x4	; r11,r12,#4
ZSR:0348:BF00200C	E24DD008	sub	r13,r13,#0x8	; r13,r13,#8
...				
*/				
			static __always_inline struct kmem_cache *kmalloclab(size_t	
			{	

## 9-2. Kernel Module의 init 부터 디버깅

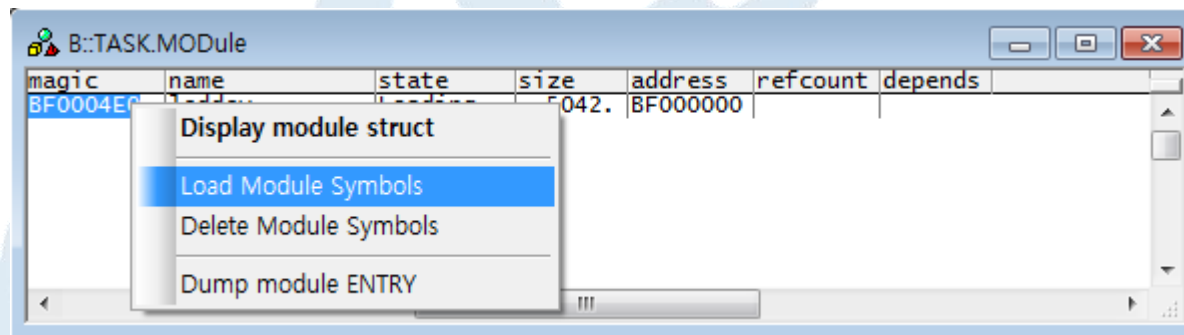
### Module 디버깅

- 로드 된 Module 리스트 확인 가능

TASK.MODulE 명령어



- 이미 로드 된 Module의 경우 magic에서 우측 클릭하여 심볼 로드 가능



# 10. Segmentation Fault Debugging

---

지금까지의 Debugging 기능을 이용하여 Exception Debugging을 할 수 있습니다

# 10-1. Segmentation Fault 디버깅

Linux에서는 Exception Vector Table을 통한 방식을 이용하지 않고 Kernel의 정보를 이용해서 Exception Debugging을 합니다

- Linux 기반의 Exception은 발생할 당시의 정보를 가지고 있는 Kernel 정보를 통해 이루어 짐.
- Application인 User Space에서 발생한 Exception부터 Kernel Panic 상황까지 디버깅이 가능.
- TRACE32 Script를 통해 미리 Breakpoint를 설정한 후 재현을 하면 됨.
- 자동으로 문제가 되는 Register 정보 복원을 통해 Exception이 발생한 위치를 확인할 수 있음.

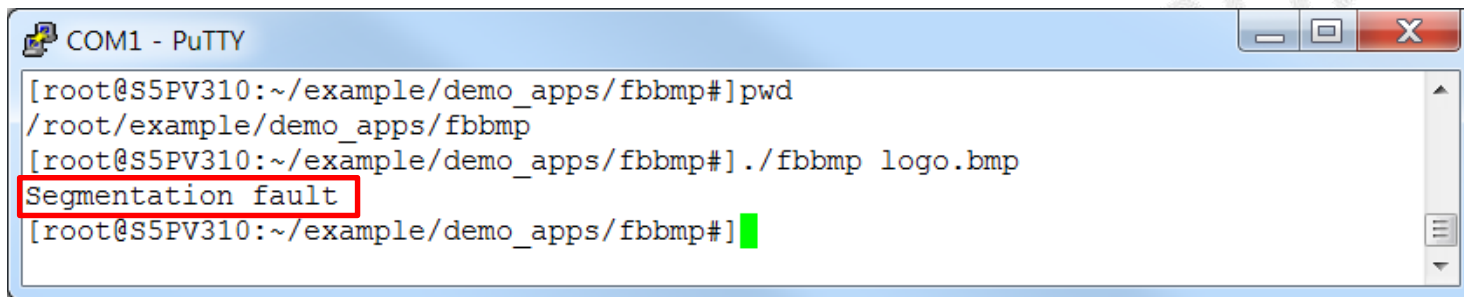


## 10-2. Segmentation Fault 디버깅 실습

Segmentation Fault를 디버깅 방법에 대해서 학습합니다.

### Segmentation Fault 디버깅

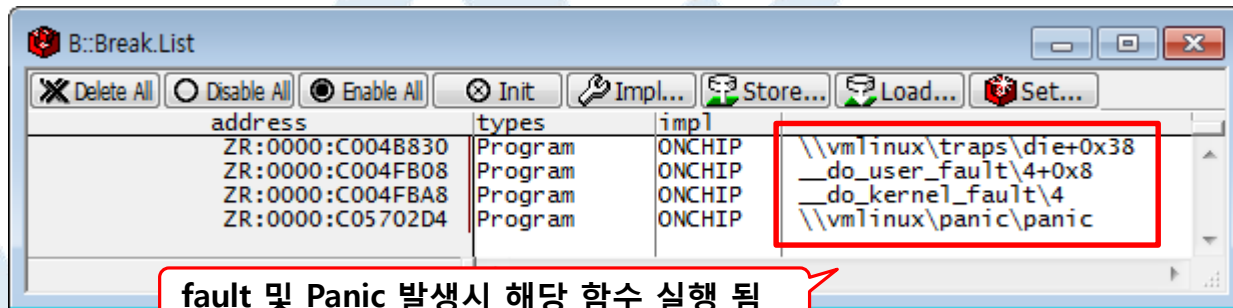
- fbbmp 어플리케이션을 실행하여 Fault 확인



```
COM1 - PuTTY
[root@S5PV310:~/example/demo_apps/fbbmp#] pwd
/root/example/demo_apps/fbbmp
[root@S5PV310:~/example/demo_apps/fbbmp#] ./fbbmp logo.bmp
Segmentation fault
[root@S5PV310:~/example/demo_apps/fbbmp#]
```

- segv.cmm 실행(아래 폴더에 스크립트 파일 존재)

C:\WT32\demo\arm\kernel\linux\linux-3.x



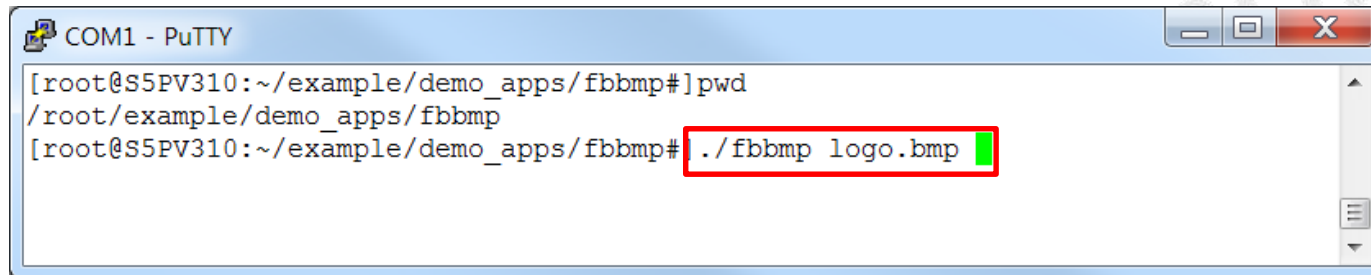
address	types	impl
ZR:0000:C004B830	Program	ONCHIP \\vmlinux\traps\die+0x38
ZR:0000:C004FB08	Program	ONCHIP __do_user_fault\4+0x8
ZR:0000:C004FBA8	Program	ONCHIP __do_kernel_fault\4
ZR:0000:C05702D4	Program	ONCHIP \\vmlinux\panic\panic

fault 및 Panic 발생시 해당 함수 실행 됨

## 10-2. Segmentation Fault 디버깅 실습

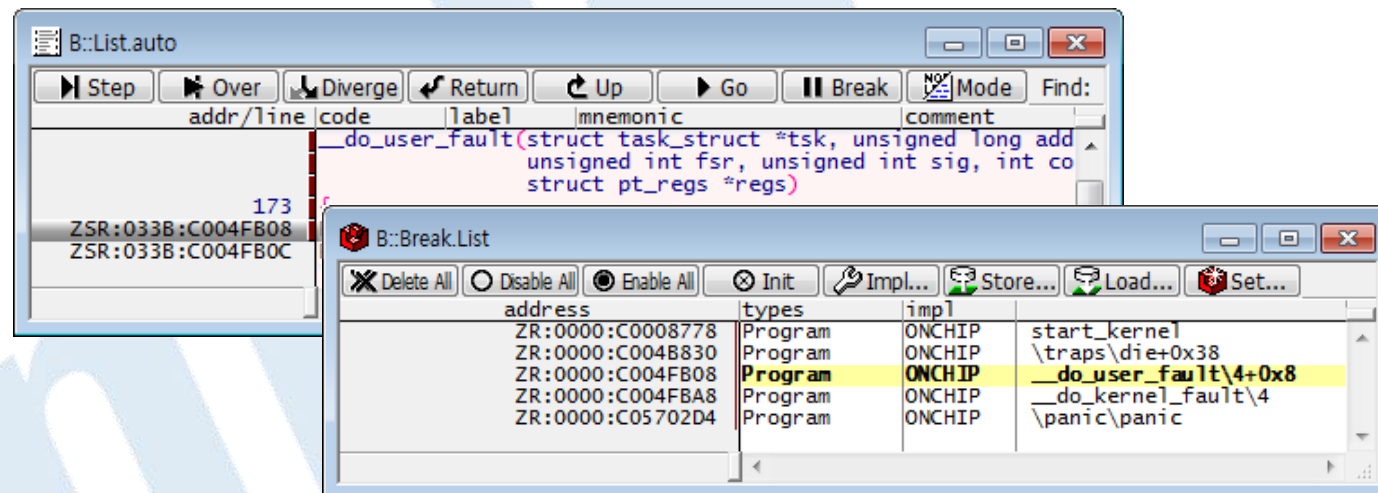
### Segmentation Fault 디버깅

- fbbmp 어플리케이션을 다시 실행



```
COM1 - PuTTY
[root@S5PV310:~/example/demo_apps/fbbmp] pwd
/root/example/demo_apps/fbbmp
[root@S5PV310:~/example/demo_apps/fbbmp] ./fbbmp logo.bmp
```

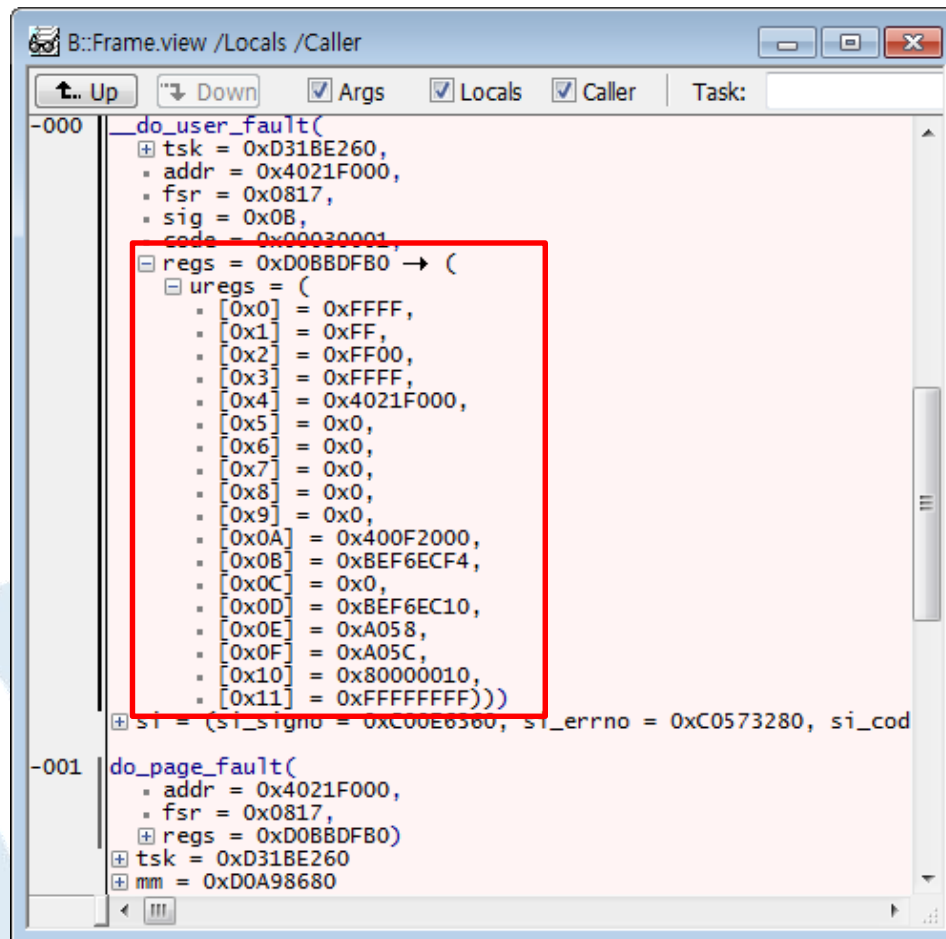
- PowerView에서 \_\_do\_user\_fault 에서 Break 된 것을 확인



## 10-2. Segmentation Fault 디버깅 실습

### Segmentation Fault 디버깅

- 콜 스택 화면에서 마지막 코드 수행 후 문제된 Register 값을 확인



```
B::Frame.view /Locals /Caller
[+] Up [-] Down [x] Args [x] Locals [x] Caller Task:
-000 do_user_fault(
  + tsk = 0xD31BE260,
  + addr = 0x4021F000,
  + fsr = 0x0817,
  + sig = 0x0B,
  + code = 0x00000001,
  + regs = 0xD0BBDFB0 -> (
    + uregs = (
      + [0x0] = 0xFFFF,
      + [0x1] = 0xFF,
      + [0x2] = 0xFF00,
      + [0x3] = 0xFFFF,
      + [0x4] = 0x4021F000,
      + [0x5] = 0x0,
      + [0x6] = 0x0,
      + [0x7] = 0x0,
      + [0x8] = 0x0,
      + [0x9] = 0x0,
      + [0x0A] = 0x400F2000,
      + [0x0B] = 0xBEF6ECF4,
      + [0x0C] = 0x0,
      + [0x0D] = 0xBEF6EC10,
      + [0x0E] = 0xA058,
      + [0x0F] = 0xA05C,
      + [0x10] = 0x80000010,
      + [0x11] = 0xFFFFFFFF)))
  + si = (si_signo = 0xC00E8360, si_errno = 0xC0573280, si_cod
-001 do_page_fault(
  + addr = 0x4021F000,
  + fsr = 0x0817,
  + regs = 0xD0BBDFB0)
  + tsk = 0xD31BE260
  + mm = 0xD0A98680
```

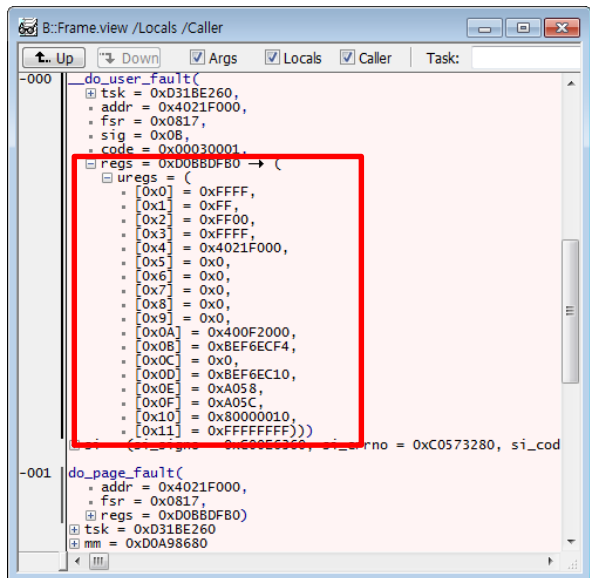
## 10-2. Segmentation Fault 디버깅 실습

### Segmentation Fault 디버깅

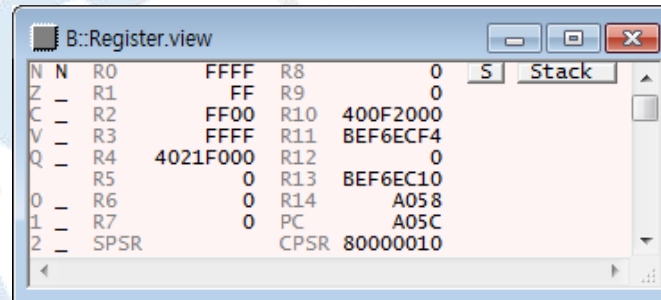
- 문제가 된 마지막 Register를 복원

```
B::do C:\T32\demo\arm\kernel\linux\linux-3.x\segv.cmm
```

segv.cmm을 다시 한번 실행하면  
regs regs 값으로 Register 값을 변경해 줌.



```
-000 do_user_fault(  
  * tsk = 0xD31BE260,  
  * addr = 0x4021F000,  
  * fsr = 0x0817,  
  * sig = 0x0B,  
  * code = 0x00030001,  
  * regs = 0xD08BDF80 → (  
    * uregs = (  
      * [0x0] = 0xFFFF,  
      * [0x1] = 0xFF,  
      * [0x2] = 0xFF00,  
      * [0x3] = 0xFFFF,  
      * [0x4] = 0x4021F000,  
      * [0x5] = 0x0,  
      * [0x6] = 0x0,  
      * [0x7] = 0x0,  
      * [0x8] = 0x0,  
      * [0x9] = 0x0,  
      * [0xA] = 0x400F2000,  
      * [0xB] = 0xBEF6ECF4,  
      * [0xC] = 0x0,  
      * [0xD] = 0xBEF6EC10,  
      * [0xE] = 0xA058,  
      * [0xF] = 0xA05C,  
      * [0x10] = 0x80000010,  
      * [0x11] = 0xFFFFFFFF)))  
-001 do_page_fault(  
  * addr = 0x4021F000,  
  * fsr = 0x0817,  
  * regs = 0xD08BDF80)  
  * tsk = 0xD31BE260  
  * mm = 0xD0A98680
```

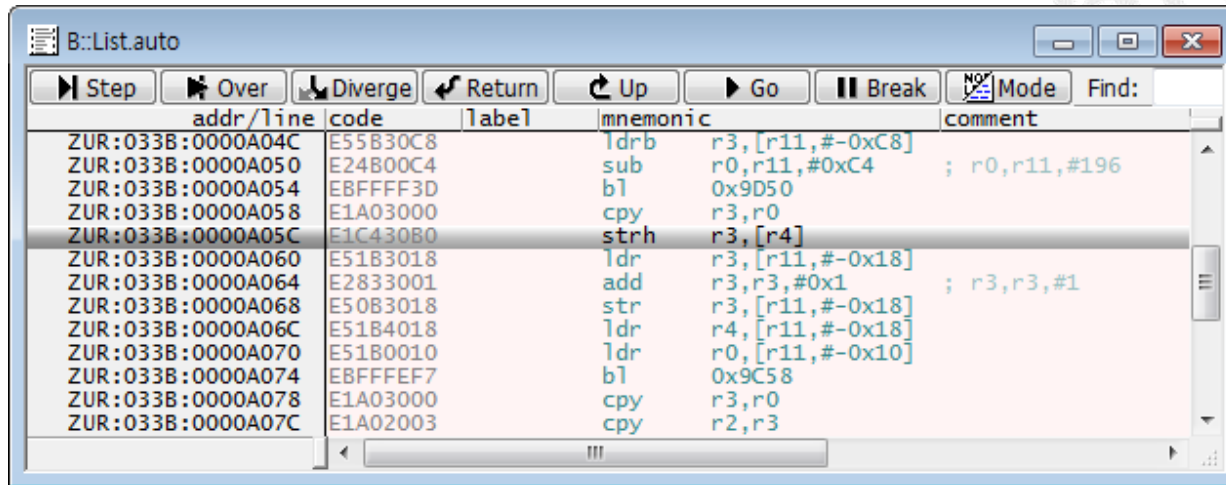


N	N	R0	FFFF	R8	0	S	Stack
Z	-	R1	FF	R9	0		
C	-	R2	FF00	R10	400F2000		
V	-	R3	FFFF	R11	BEF6ECF4		
Q	-	R4	4021F000	R12	0		
	-	R5	0	R13	BEF6EC10		
0	-	R6	0	R14	A058		
1	-	R7	0	PC	A05C		
2	-	SPSR		CPSR	80000010		

## 10-2. Segmentation Fault 디버깅 실습

### Segmentation Fault 디버깅

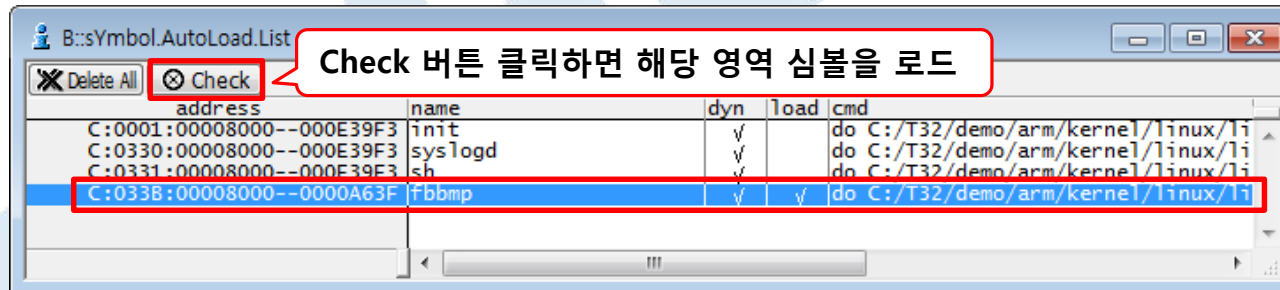
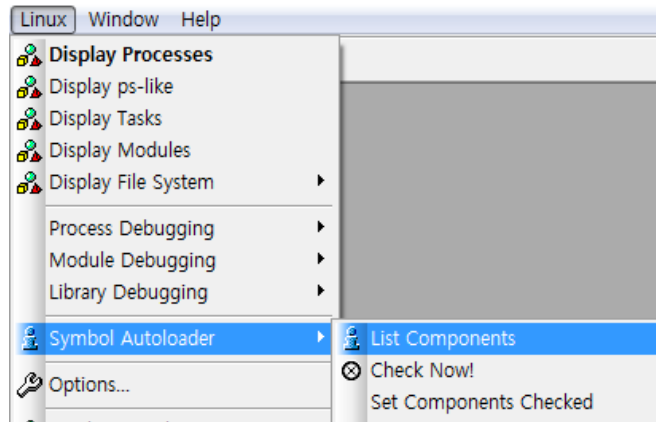
- List.auto 창을 통해 마지막 수행 코드 확인  
strh 명령어를 수행하면서 fault 발생



## 10-2. Segmentation Fault 디버깅 실습

### Segmentation Fault 디버깅

- 해당 ASM 코드 위치 맞는 심볼로드

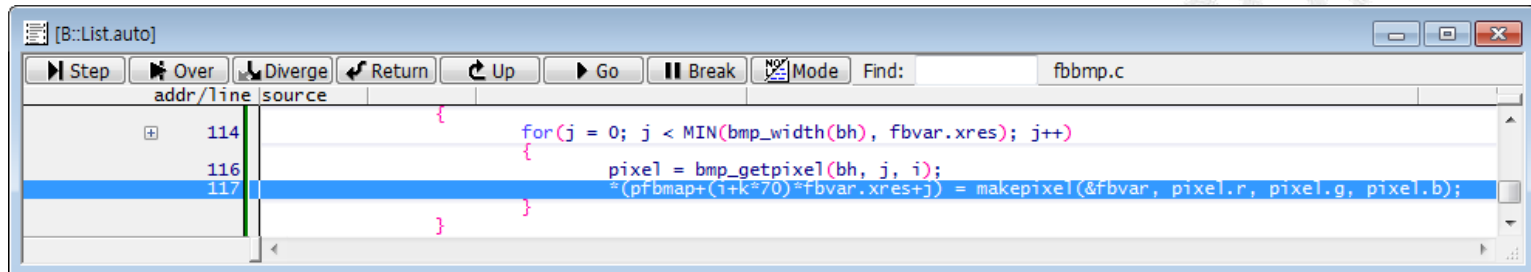


심볼위치 : C:\T32\Linux\_Edu\example\demo\_apps\fbmp

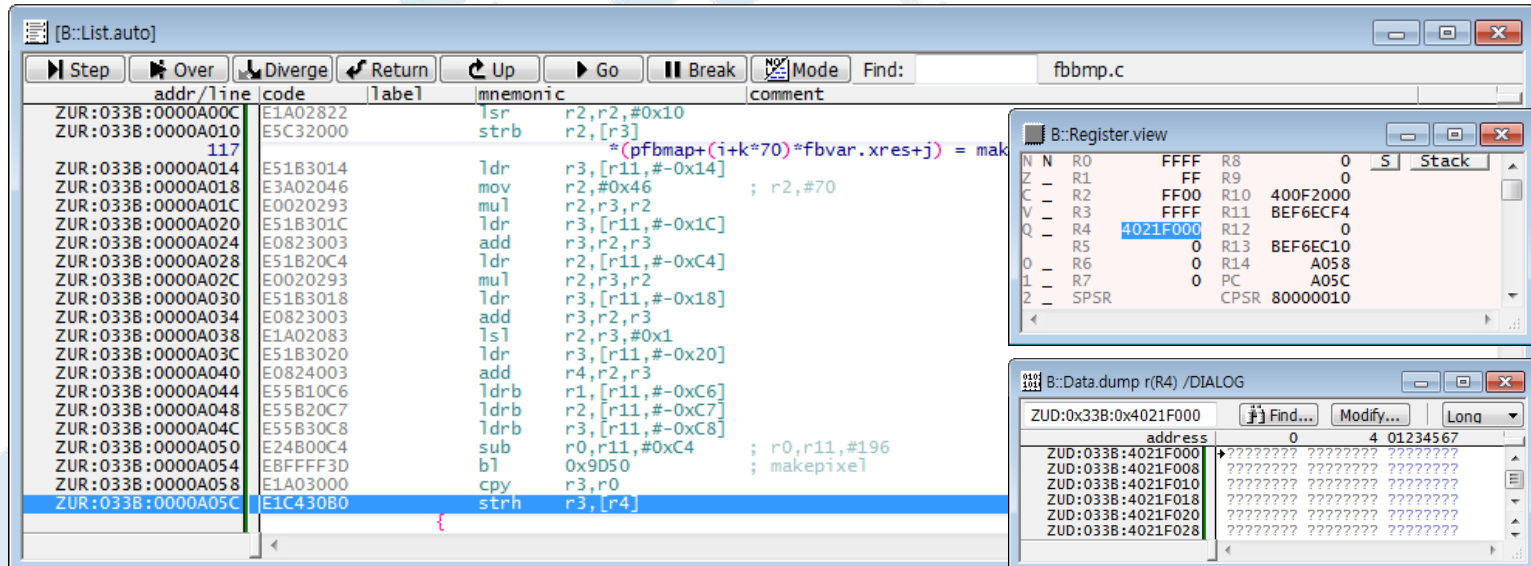
# 10-2. Segmentation Fault 디버깅 실습

## Segmentation Fault 디버깅

- 해당 C 소스 코드와 ASM을 확인하여 원인 파악



```
[B::List.auto]
Step Over Diverge Return Up Go Break Mode Find: fbbmp.c
addr/line source
114 {
116     for(j = 0; j < MIN(bmp_width(bh), fbvar.xres); j++)
117     {
118         pixel = bmp_getpixel(bh, j, i);
119         *(pfbmap+(i+k*70)*fbvar.xres+j) = makepixel(&fbvar, pixel.r, pixel.g, pixel.b);
120     }
121 }
```



**Assembly Code:**

addr/line	code	label	mnemonic	comment
ZUR:033B:0000A00C	E1A02822		lsr r2,r2,#0x10	
ZUR:033B:0000A010	E5C32000		strb r2,[r3]	
117				*(pfbmap+(i+k*70)*fbvar.xres+j) = mak
ZUR:033B:0000A014	E51B3014		ldr r3,[r11,#-0x14]	
ZUR:033B:0000A018	E3A02046		mov r2,#0x46	; r2,#70
ZUR:033B:0000A01C	E0020293		mul r2,r3,r2	
ZUR:033B:0000A020	E51B301C		ldr r3,[r11,#-0x1C]	
ZUR:033B:0000A024	E0823003		add r3,r2,r3	
ZUR:033B:0000A028	E51B20C4		ldr r2,[r11,#-0xC4]	
ZUR:033B:0000A02C	E0020293		mul r2,r3,r2	
ZUR:033B:0000A030	E51B3018		ldr r3,[r11,#-0x18]	
ZUR:033B:0000A034	E0823003		add r3,r2,r3	
ZUR:033B:0000A038	E1A02083		lsl r2,r3,#0x1	
ZUR:033B:0000A03C	E51B3020		ldr r3,[r11,#-0x20]	
ZUR:033B:0000A040	E0824003		add r4,r2,r3	
ZUR:033B:0000A044	E55B10C6		ldrb r1,[r11,#-0xC6]	
ZUR:033B:0000A048	E55B20C7		ldrb r2,[r11,#-0xC7]	
ZUR:033B:0000A04C	E55B30C8		ldrb r3,[r11,#-0xC8]	
ZUR:033B:0000A050	E24B00C4		sub r0,r11,#0xC4	; r0,r11,#196
ZUR:033B:0000A054	EBFFF3D		bl 0x9D50	; makepixel
ZUR:033B:0000A058	E1A03000		cpy r3,r0	
ZUR:033B:0000A05C	E1C43080		strh r3,[r4]	

**B::Register.view**

N	R0	R1	R2	R3	R4	R5	R6	R7	SPSR	R8	R9	R10	R11	R12	R13	R14	PC	CPSR
N	FFFF	FF	FF00	FFFF	4021F000	0	0	0	0	0	0	400F2000	BEF6ECF4	0	BEF6EC10	A058	A05C	80000010

**B::Data.dump r(R4) /DIALOG**

address	0	4	01234567
ZUD:033B:4021F000	????????	????????	????????
ZUD:033B:4021F008	????????	????????	????????
ZUD:033B:4021F010	????????	????????	????????
ZUD:033B:4021F018	????????	????????	????????
ZUD:033B:4021F020	????????	????????	????????
ZUD:033B:4021F028	????????	????????	????????

# 11. 추가 실습

---

추가 실습을 통해 TRACE32의 활용도를 더 확대해 봅니다

1. Kernel Log 확인하기
2. 함수 수행 시간 측정하기
3. 터미널(시리얼) 프로그램

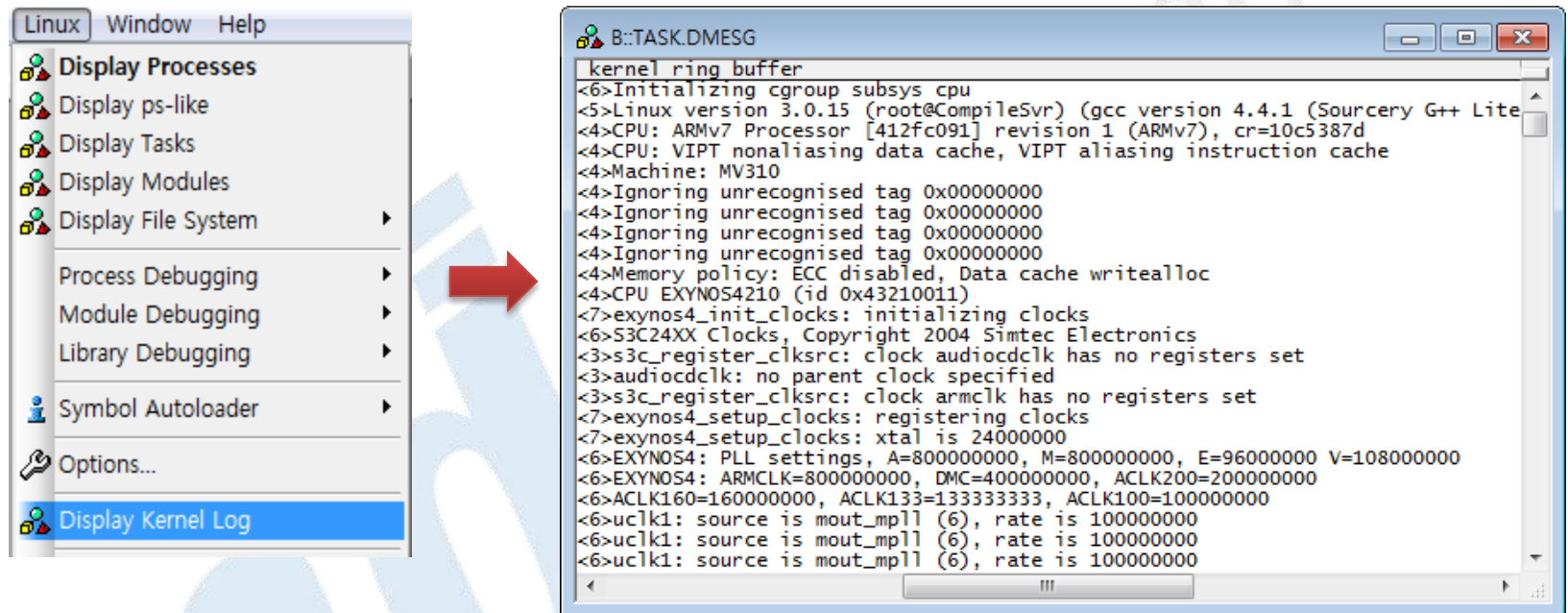


# 11-1. Kernel Log 확인하기

Kernel Log를 JTAG을 통해 확인 할 수 있습니다.

## Kernel Log 확인

- 아래 Display Kernel Log 버튼을 통해 확인 가능

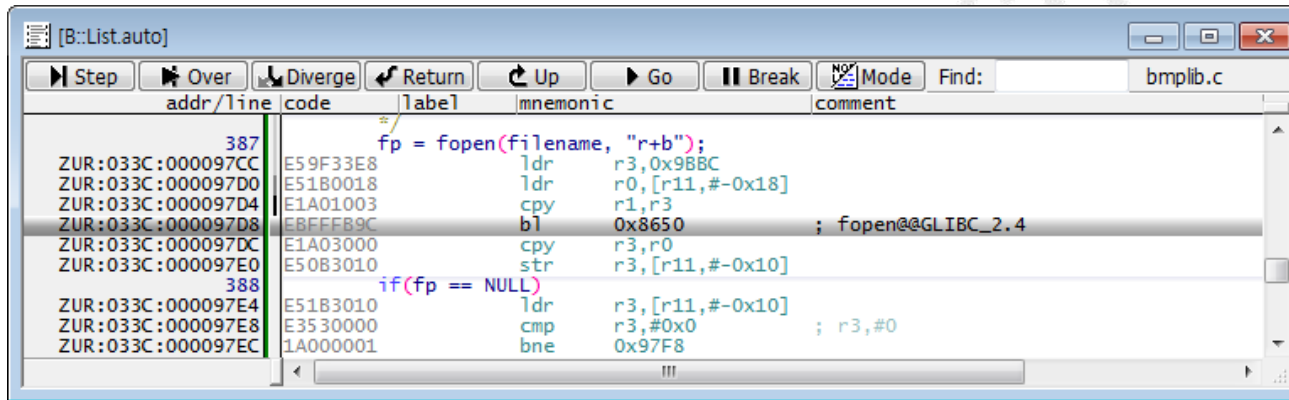


# 11-2. 함수 수행 시간 측정하기

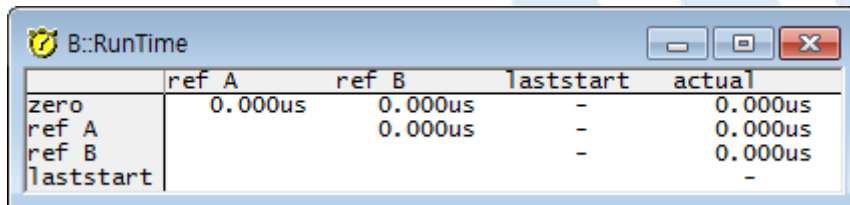
함수의 수행 시간 측정을 할 수 있습니다.

## 함수 수행 시간 측정

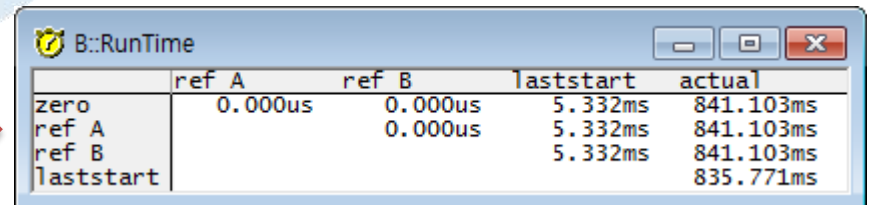
- Runtime 명령어로 함수 수행 시간 측정  
측정하고자 하는 위치에서 Break 한 후 원하는 코드 위치까지 Target을 Running 하면 수행된 시간을 Runtime 창에서 확인 가능



```
[B::List:auto]
Step Over Diverge Return Up Go Break Mode Find: bmpilib.c
addr/line code label mnemonic comment
387 fp = fopen(filename, "r+b");
ZUR:033C:000097CC E59F33E8 ldr r3,0x9B8C
ZUR:033C:000097D0 E51B0018 ldr r0,[r11,#-0x18]
ZUR:033C:000097D4 E1A01003 cpy r1,r3
ZUR:033C:000097D8 EBFFFB9C bl 0x8650 ; fopen@@GLIBC_2.4
ZUR:033C:000097DC E1A03000 cpy r3,r0
ZUR:033C:000097E0 E50B3010 str r3,[r11,#-0x10]
388 if(fp == NULL)
ZUR:033C:000097E4 E51B3010 ldr r3,[r11,#-0x10]
ZUR:033C:000097E8 E3530000 cmp r3,#0x0 ; r3,#0
ZUR:033C:000097EC 1A000001 bne 0x97F8
```



	ref A	ref B	laststart	actual
zero	0.000us	0.000us	-	0.000us
ref A		0.000us	-	0.000us
ref B			-	0.000us
laststart				-



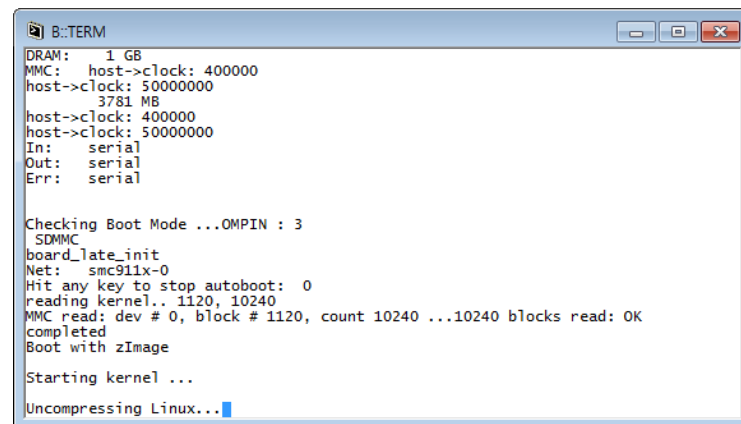
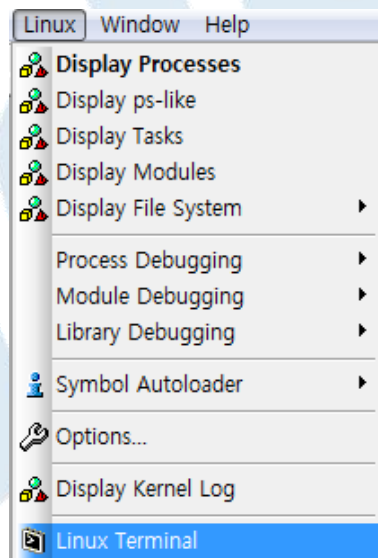
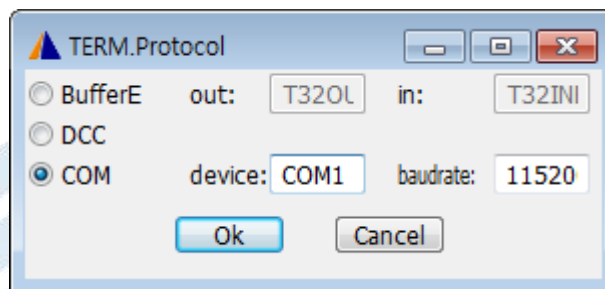
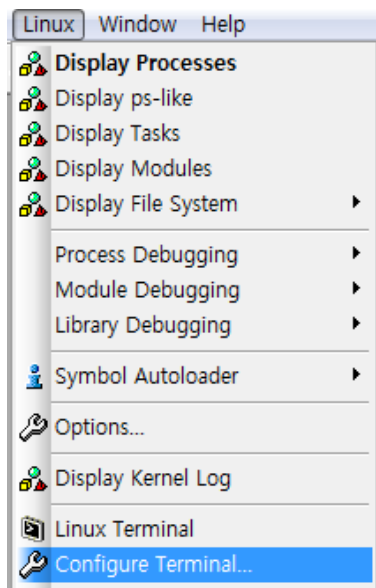
	ref A	ref B	laststart	actual
zero	0.000us	0.000us	5.332ms	841.103ms
ref A		0.000us	5.332ms	841.103ms
ref B			5.332ms	841.103ms
laststart				835.771ms

# 11-3. 터미널 프로그램

PowerView에서 터미널을 오픈 할 수 있습니다.

## 터미널 오픈

- COM 포트와 속도 설정 후 사용 가능




[메인홈](#) | [로그인](#) | [회원가입](#)
 [SEARCH](#)
[제품정보](#)[교육신청](#)[자료실](#)[Q&A](#)[고객지원](#)

## Debug Specific PowerDebug

전 세계 가장 많은 개발자가 사용하는 디버거로서의 갖는 다양한 기능  
문제의 원인을 단순화하여 빠른 디버깅이 가능한 Advanced Breakpoint  
프로세서 특성이나 사용자 환경에 최적화한 자동화 언어 CMM Script

[Solutions](#)

&lt; &gt;

[PowerDebug](#)[PowerTrace](#)[PowerProbe](#)[CombiProbe](#)[μTrace](#)

### Latest News

+

- [무료방문세미나] 리눅스 디버깅/동적 메모리 관리 방안 11-23
- [퀴즈이벤트] 실시간 트레이스를 통한 중대한 버그 검출 솔루션 Power... 11-18
- [세미나] Value Chain 기반의 Enterprise IoT 구현 사례 세미나에 초대... 10-13
- [세미나] 글로벌 진출을 위한 의료기기 SW 개발 및 품질 향상 세미나... 10-05
- [교육안내] TRACE32 10월 정규 교육 과정 안내 09-23
- [신제품출시] Linux Application 개발자를 위한 최강의 SW 디버깅 솔... 09-21
- [신제품출시] 실차환경에서 CAN통신으로 TRACE32 활용하자! 09-09

[지원 Processor](#)[지원 RTOS](#)[전적/기술지원](#)[iTSP](#)[제품 동영상](#)[제품 브로슈어](#)

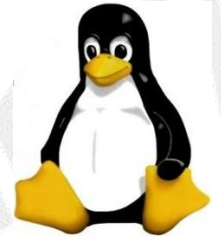
고객지원



고객과 함께 생각하고 문제를 해결할때  
진정한 보람을 얻는 것이 우리의 고객정신입니다.

## Technical Support Request

- Ref. E-mail : [trace32@mdstec.com](mailto:trace32@mdstec.com)
- Homepage [www.trace32.com](http://www.trace32.com) : Q&A or 기술지원 버튼
  - Training Course : 홈페이지 내부 교육 메뉴
  - Repair Support Service Tel : 031-627-3119



# Linux Debugging

감사합니다

MDS Technology  
DT1 Team