리눅스 커널 소스 실행 분석기

최신 리눅스 커널 소스를 직접 실행하면서 분석할 수 있는

리눅스 커널 소스 실행 분석기

Menu Guide

커널연구회 (www.kernel.bz)

정재준 (rgbi3307@nate.com)

리눅스 커널 소스 실행 분석기

목차

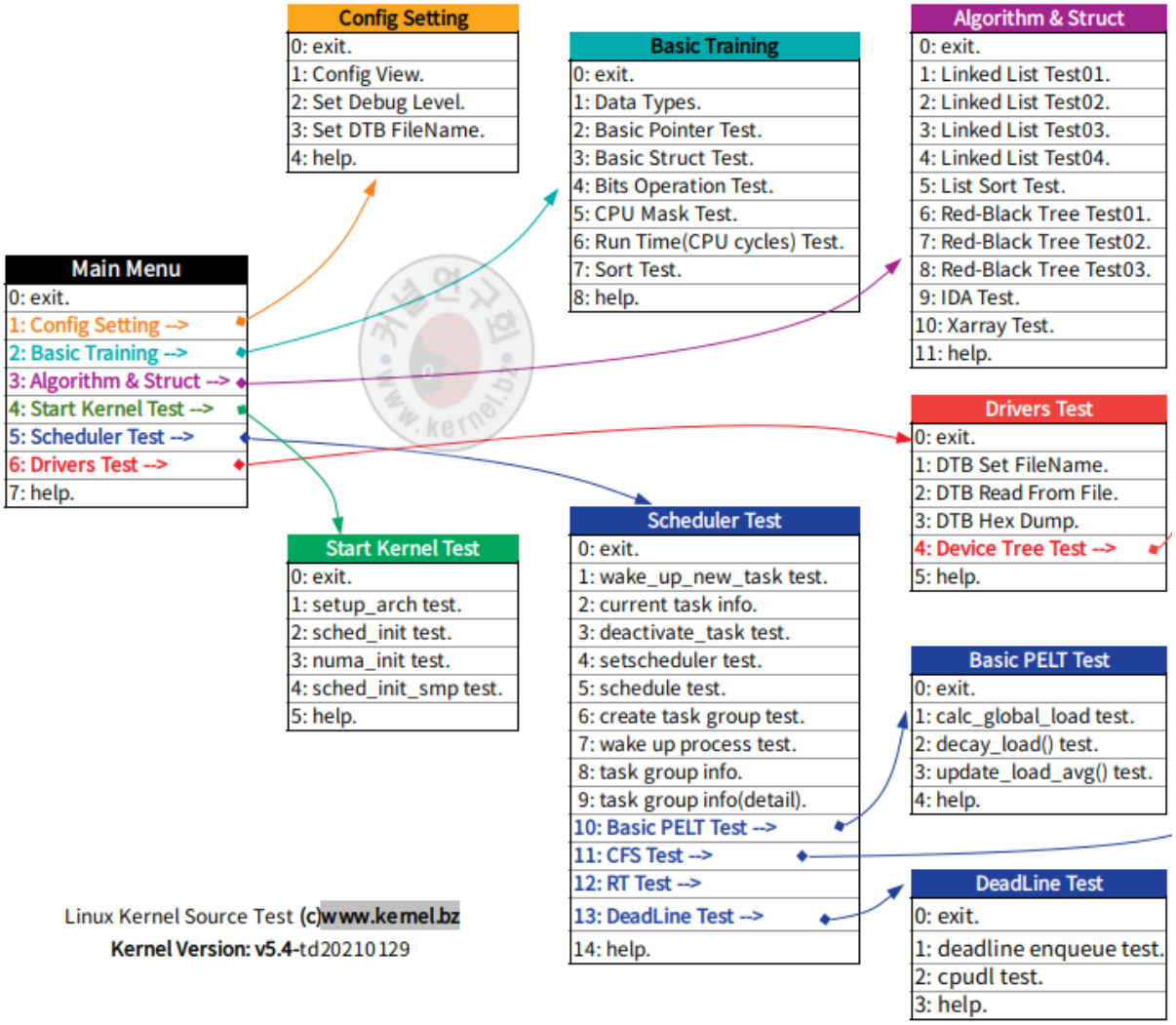
Table of Contents

목차

작업 메뉴별 기능 소개 (Menu Guide)

작업 메뉴의 전체적인 구성은 다음과 같습니다. 분석기는 qt 프로젝트에서 바로 실행해도 되고 리눅스 터미널 콘솔 화면에서 빌드된 실행파일(kernel-test)을 따로 실행해도 됩니다. 실행된 Main Menu 에서부터 메뉴번호를 입력하여 해당메뉴로 실행을 전개하는 방식으로 구성되어 있습니다.

작업 메뉴 전체 구성



2.1 메인 메뉴

먼저, 메인 메뉴는 다음과 같이 구성되어 있습니다.

Main Menu
0: exit.
1: Config Setting -->
2: Basic Training -->
3: Algorithm & Struct -->
4: Start Kernel Test -->
5: Scheduler Test -->
6: Drivers Test -->
7: help.

콘솔에서 다음과 같이 메뉴가 화면에 출력되며, 키보드에서 메뉴 번호를 입력하여 해당 기능을 실행합니다. 메인 메뉴는 하위 메뉴를 다시 호출하는 구조로 되어 있습니다. 하위 메뉴가 있는 메뉴명에는 --> 가 붙어 있습니다.

콘솔 화면에 출력된 메인 메뉴

```
[*] Linux Kernel Source Test (c)www.kernel.bz
0: exit.
1: Config Setting -->
2: Basic Training -->
3: Algorithm & Struct -->
4: Start Kernel Test -->
5: Scheduler Test -->
6: Drivers Test -->
7: help.
Kernel Version: v5.4-td20210129

Enter Menu Number[0,8]:
```

위의 메뉴별 기능을 요약 설명하면 다음과 같습니다.

0: exit. 프로그램 종료.

1: Config Setting --> 소스 환경설정 메뉴로 진행합니다.

2: Basic Training --> 커널 소스 기본(데이터 타입, 포인터, 구조체, 비트연산, CPU 마스크) 메뉴

3: Algorithm & Struct --> 커널 소스 알고리즘(링크드 리스트, Red-Black Tree, XArray) 메뉴

4: Start Kernel Test --> 커널 초기화 소스 테스트 메뉴

5: Scheduler Test --> 커널 스케줄러 소스 테스트 메뉴

6: Drivers Test --> 디바이스 트리 및 드라이버 테스트 메뉴

7: Help 간단한 도움말 출력.

Kernel Version: v5.4-td20210129

커널 v5.4 기준으로 포팅되었으며 td20210129 는 날짜(yyyymmmdd) 형태의 테스트 버전입니다.

이것은 날짜 형식으로 지속적으로 버전업 될 예정입니다.

2.2 환경 설정 메뉴

메인 메뉴에서 [1]을 입력하면 다음과 같이 **환경 설정(Config Setting) 메뉴**가 실행 됩니다.

Config Setting
0: exit.
1: Config View.
2: Set Debug Level.
3: Set DTB FileName.
4: help.

콘솔에서는 다음과 같이 메뉴가 화면에 출력되며, 키보드에서 메뉴 번호를 입력하여 해당 기능을 실행합니다.

환경 설정(Config Setting) 메뉴

```
[#]--> Config Setting Menu
0: exit.
1: Config View.
2: Set Debug Level.
3: Set DTB FileName.
4: help.

Enter Menu Number[0,5]:
```

위의 메뉴를 번호별로 요약 설명하면 다음과 같습니다.

0: exit. //이전 메뉴(메인 메뉴)로 돌아 갑니다.

1: Config View. //환경 설정된 내용들을 화면에 출력 합니다.

2: Set Debug Level. //디버그 메세지를 출력하는 레벨 범위를 설정 합니다.

3: Set DTB FileName. //읽어올 디바이스 트리 파일명을 설정 합니다.

4: help. //간단한 도움말을 화면에 출력 합니다.

메뉴 번호 [1] Config View 는 다음과 같이 환경 설정된 내용들을 화면에 출력 합니다.

Config View 실행 내용

```
Enter Menu Number[0,5]: 1

    <1> |                      ARCH_NAME : x86(__x86_64__)
    <1> |                      CONFIG_HZ : 100
    <1> |                 CONFIG_NR_CPUS : 8
    <1> |                 "CONFIG_ARM64" : CONFIG_ARM64
    <1> |                  "CONFIG_NUMA" : CONFIG_NUMA
```

```
    <1> |                CONFIG_BASE_SMALL : 0
    <1> |                   XA_CHUNK_SHIFT : 6
    <1> |                    XA_CHUNK_SIZE : 64
    <1> |              CONFIG_NODES_SHIFT : 2
    <1> |            CONFIG_USER_DTB_FILE : testcases.dtb
    <1> |                   dtb_file_name :
    <1> |                        dtb_size : 0
    <1> |                       DebugBase : 0
    <1> |                      DebugLevel : 200
    <1> |                     DebugEnable : 1
    <1> |     CONFIG_VERSION_1 : 5
    <1> |     CONFIG_VERSION_2 : 4
    <1> |     CONFIG_VERSION_3 : 20210210
```

이들의 의미에 대해서는 "제 3 장 환경설정 소스 설명"에서 자세히 설명 합니다만, 알고 가야 하는 것은 ARCH_NAME 입니다. 이것은 분석기 소스를 빌드하여 실행하는 CPU 아키텍쳐와 컴파일러에 따라서 자동으로 가지고 옵니다. 현재까지 필자는 x86 과 ARM 환경에서 소스를 테스트 했습니다.

x86(64 비트)은 필자의 노트북 PC 에서 소스를 빌드하여 테스트 했으며, 이것의 시스템 정보는 다음과 같습니다.

PC NoteBook (x86_64) 정보

```
$ uname -a
Linux HP7100U 4.14.146+ #1 SMP Fri Oct 4 22:04:13 KST 2019 x86_64 x86_64
x86_64 GNU/Linux
```

다음으로 ARM 환경에서 소스를 빌드하고 테스트 했는데, ARM 은 32 비트와 64 비트가 있습니다. ARM64 비트는 nvidia JETSON nano 보드에서 테스트 했으며 이것의 정보는 다음과 같습니다.

ARM64 (nvidia JETSON nano 보드)

```
$ uname -a
Linux kernel-bz 4.9.140-tegra #1 SMP PREEMPT Web Apr 8 18:10:49 PDT 2020
```

```
aarch64 aarch64 aarch64 GNU/Linux
```

ARM32 비트는 라즈베리파이 4 보드에서 테스트 했으며 이것의 정보는 다음과 같습니다.

ARM32 (라즈베리파이 4 보드)

```
$ uname -a
Linux 4.19.75-v7l+ #1270 SMP Tue Sep 24 18:51:41 BST 2019 armv7l GNU/Linux
```

디버그 메세지 범위 설정

메뉴 번호 [2] Set Debug Level 을 실행하여 디버그 메세지를 출력하는 범위에 해당하는 level 하한값(DebugBase)과 상한값(DebugLevel)을 설정할 수 있습니다.

디버그 메세지는 다음과 같이 해당 함수가 호출된 스택 깊이 만큼 오른쪽으로 들여쓰기하여 출력 됩니다.

디버그 메세지 출력 예

```
    3--> run_rebalance_domains()...
        4--> update_blocked_averages()...
            5--> update_rq_clock()...
            <5> |            rq->clock : 70000000
            <5> |       rq->clock_task : 70000000
            <5> |       rq->clock_pelt : 70000000
            <5> |               delta : 190000000
                6--> update_rq_clock_task()...
                <6> |       rq->clock_task : 260000000
                    7--> update_rq_clock_pelt()...
                    <7> |      (void*)rq->curr : 0x55968e5da6e0
                    <7> |    rq_clock_task(rq) : 260000000
                    <7> |               delta : 190000000
                    <7> |               delta : 190000000
```

```
                    <7> |       rq->clock_pelt : 260000000
                    7<-- update_rq_clock_pelt().

                6<-- update_rq_clock_task().

            <5> |            rq->clock : 260000000
            5<-- update_rq_clock().

                6--> ___update_load_sum()...
                <6> |                  now : 260000000
                <6> |                 load : 0
                <6> |             runnable : 0
                <6> |              running : 0
                <6> |       (s64)delta(ns) : 260000000
                <6> |            delta(us) : 253906
                <6> | sa->last_update_time : 259999744
                    7--> accumulate_sum()...
                    <7> |                 contrib(delta) : 253906
                    <7> |             sa->period_contrib : 0
                    <7> |                 delta(+old_d3) : 253906
                    <7> |                    periods(ms) : 247
                    <7> |                   sa->load_sum : 0
                    <7> |          sa->runnable_load_sum : 0
                    <7> |                   sa->util_sum : 0
                    <7> |                   sa->load_sum : 0
                    <7> |          sa->runnable_load_sum : 0
                    <7> |                   sa->util_sum : 0
```

위의 디버그 메세지 출력에서 행 앞에 있는 <번호>는 이 함수가 호출된 스택 깊이(level)에 해당합니다.

여기서 DebugBase 값을 4 로 하고 DebugLevel 값을 5 로 설정하면 스택 깊이(level)가 이 범위에 해당하는 디버그 메세지만 다음과 같이 출력 됩니다.

디버그 메세지 출력 범위 설정 (4 <= depth <= 5)

```
4--> update_blocked_averages()...
    5--> update_rq_clock()...
    <5> |            rq->clock : 530000000
    <5> |       rq->clock_task : 530000000
```

```
    <5> |       rq->clock_pelt : 530000000
    <5> |               delta : 270000000
    <5> |           rq->clock : 800000000
    5<-- update_rq_clock().

    5--> cfs_rq_clock_pelt()...
    <5> |                           rq_of(cfs_rq) : 0x55968da9f860
    <5> |            rq_clock_pelt(rq_of(cfs_rq)) : 800000000
    <5> |                    clock_pelt_throttled : 800000000
    5<-- cfs_rq_clock_pelt().

    5--> update_cfs_rq_load_avg()...
    <5> |   cfs_rq->removed.nr : 0
    <5> |              decayed : 1
    5<-- update_cfs_rq_load_avg().

    5--> update_tg_load_avg()...
    <5> |                          delta : 2047
    <5> |                 (void*)cfs_rq : 0x55968da9f8c0
    <5> |              (void*)cfs_rq->tg : 0x55968daa5280
4<-- update_blocked_averages().
```

따라서 디버그 메세지는 다음의 조건을 만족하는 범위에서 출력 되도록 설정할 수 있습니다.

디버그 메세지 출력 범위 조건

```
DebugBase <= 스택깊이(level) <=  DebugLevel
```

스택 깊이 계산과 디버그 메세지 출력에 대한 소스는 “제 3 장 환경설정 소스 설명”을 참고 하시기 바랍니다.

DTB 파일명 설정

보통 DTB(Device Tree Binary) 파일은 제조사에서 제공을 합니다. 커널을 빌드하면 arch/*/boot/dts/ 경로에 해당하는 보드 모델별로 dtb 파일이 컴파일 됩니다. 필자의 분석기는 교육 및 학습용으로 “testcases.dtb” 파일을 arch/dts/ 경로에 저장해 두었습니다. 이 DTB 파일을 분석기가 실행되고 있는 경로에 복사하여 사용하면 됩니다.

DTB 파일명은 메뉴번호 [3]을 입력하여 설정할 수 있습니다. DTB 파일은 분석기가 실행되고 있는 경로에 있어야 합니다. 이곳에 있는 DTB 파일명을 설정할 수 있고 현재는 분석용으로 “testcases.dtb” 파일로 설정하여 이것을 읽어서 DTB 정보를 파싱하도록 되어 있습니다. DTB 파일을 변경하고자 한다면 분석기가 실행되고 있는 경로에 DTB 파일을 복사하고 그 파일명을 입력 합니다.

DTB 파일명 설정

```
[#]--> Config Setting Menu
0: exit.
1: Config View.
2: Set Debug Level.
3: Set DTB FileName.
4: help.

Enter Menu Number[0,5]: 3
Enter DTB File Name[testcases.dtb]:
```

위의 메뉴에서는 DTB 파일 명칭만 설정합니다. 이 파일을 가지고 DTB 을 파싱하고 디바이스 드라이버에 적용하는 내용들은 “제 8 장 드라이버 소스 설명”에서 기술 됩니다.

2.3 기본 훈련 메뉴

메인 메뉴에서 [2]을 입력하면 다음과 같이 **Basic Training 메뉴**가 실행 됩니다. 이 메뉴는 커널 소스를 이해하기 위해서 반드시 선행 학습해야 하는 커널 데이터 타입, 포인터, 구조체, 비트연산, CPU 마스크 등의 소스를 연습하기 위한 용도로 만들어 졌습니다.

Basic Training
0: exit.
1: Data Types.
2: Basic Pointer Test.
3: Basic Struct Test.
4: Bits Operation Test.
5: CPU Mask Test.
6: Run Time(CPU cycles) Test.
7: Sort Test.
8: help.

콘솔에서는 다음과 같이 메뉴가 화면에 출력되며, 키보드에서 메뉴 번호를 입력하여 해당 기능을 실행합니다.

Basic Training 메뉴

```
[#]--> Basic Training Menu
0: exit.
1: Data Types.
2: Basic Pointer Test.
3: Basic Struct Test.
4: Bits Operation Test.
5: CPU Mask Test.
6: Run Time(CPU cycles) Test.
7: Sort Test.
8: help.
```

위의 메뉴를 번호별로 요약 설명하면 다음과 같습니다.

0: exit. //이전 메뉴(메인 메뉴)로 이동 합니다.

1: Data Types. //커널 데이터 타입을 연습하는 기능 입니다.

2: Basic Pointer Test. //커널 만의 포인터 연산 소스를 이해하는 기능 입니다.

3: Basic Struct Test. //커널 구조체를 이해하는 소스 입니다.

4: Bits Operation Test. //커널의 비트연산을 테스트 하는 소스로 구성되어 있습니다.

5: CPU Mask Test. //커널의 CPU 마스크를 테스트 하는 소스 입니다.

6: Run Time(CPU cycles) Test. //커널 소스 실행 시간을 벤치마킹 테스트할 수 있는 소스.

7: Sort Test. //커널의 정렬 알고리즘을 테스트하는 소스 입니다.

8: help. //간단한 도움말을 화면에 보여 줍니다.

Data Types

메뉴 번호 [1] Data Types 는 다음과 같이 커널에서 데이터 타입을 정의한 내용들을 화면에 출력 합니다. 이들의 소스에 대해서는 “제 4 장 기본 훈련 소스 설명”에서 설명 됩니다.

Data Types 실행 내용

```
Enter Menu Number[0,9]: 1
[#] Basic Types Test...
    2--> _basic_type1_test()...
    <2> |            sizeof(s8) : 1
    <2> |            sizeof(u8) : 1
    <2> |           sizeof(s16) : 2
    <2> |           sizeof(u16) : 2
    <2> |           sizeof(s32) : 4
    <2> |           sizeof(u32) : 4
```

```
    <2> |           sizeof(s64) : 8
    <2> |           sizeof(u64) : 8
    2<-- _basic_type1_test().

    2--> _basic_type2_test()...
    <2> |                U8_MAX : 255
    <2> |                S8_MAX : 127
    <2> |                S8_MIN : -128
    <2> |               U16_MAX : 65535
    <2> |               S16_MAX : 32767
    <2> |               S16_MIN : -32768
    <2> |               U32_MAX : 4294967295
    <2> |               S32_MAX : 2147483647
    <2> |               S32_MIN : -2147483648
    <2> |               U64_MAX : 18446744073709551615
    <2> |               S64_MAX : 9223372036854775807
    <2> |               S64_MIN : -9223372036854775808
    2<-- _basic_type2_test().

    2--> _basic_type3_test()...
    <2> |          __CHAR_BIT__ : 8
    <2> |      __SIZEOF_LONG__ : 8
    <2> |        BITS_PER_LONG : 64
    <2> |      __BITS_PER_LONG : 64
    <2> |   BITS_PER_LONG_LONG : 64
    2<-- _basic_type3_test().
```

Basic Pointer Test

메뉴 번호 [2] Basic Pointer Test 는 다음과 같이 실행 내용들을 화면에 출력 합니다. 리눅스 커널 소스에서 가장 빈번하게 등장하는 포인터 연산 매크로인 offsetof 와 container_of 에 대해서 이해하기 위한 소스로 구성되어 있습니다. 이들의 의미에 대해서는 4 장에서 소스 기술 할때 자세히 설명 하도록 하겠습니다.

Basic Pointer Test 실행 내용

```
        2--> _ptr_offsetof_test()...
        <2> |     offsetof(struct sample, a) : 0
        <2> |     offsetof(struct sample, b) : 4
        <2> |     offsetof(struct sample, c) : 8
        <2> |     offsetof(struct sample, d) : 16
        <2> |     offsetof(struct sample, e) : 24
        2<-- _ptr_offsetof_test().

        2--> _ptr_container_of_test()...
        <2> |              (void*)&sample_st : 0x7ffe7c2b21e0
        <2> | container_of(&sample_st.a, struct sample, a) :
0x7ffe7c2b21e0
        <2> | container_of(&sample_st.b, struct sample, b) :
0x7ffe7c2b21e0
        <2> | container_of(&sample_st.c, struct sample, c) :
0x7ffe7c2b21e0
        <2> | container_of(&sample_st.d, struct sample, d) :
0x7ffe7c2b21e0
        2<-- _ptr_container_of_test().
```

Basic Struct Test

메뉴 번호 [3] Basic Struct Test 는 다음과 같이 실행 내용들을 화면에 출력 합니다. 이 예제는 커널 소스에 있는 구조체들이 가장 일반적으로 작업되는 형식을 참조하여 구조체 멤버변수들을 할당하고 값을 초기화한 후 조회하는 작업을 연습하는 용도 입니다. 커널 소스는 구조체와 멤버변수들간에 연결 구조가 많아서 복잡해 보입니다. 그래서 처음에 접근하기에 힘들 수 있습니다. 하지만 구조체와 멤버변수들의 연결관계가 규칙성 있게 나타나므로 이것을 처음에 잘 파악해 두면 아무리 연결 구조가 복잡해도 쉽게 분석해 나갈 수 있습니다. 이곳에서는 복잡한 커널 구초체를 간략화하여 연결 구조의 규칙성을 핵심 위주로 익힐 수 있도록 테스트 소스를 구성 했습니다.

Basic Struct Test 실행 내용

```
Enter Menu Number[0,9]: 3
        2--> _struct_alloc_test1()...
            3--> _struct_alloc()...
            <3> |                          level : 5
            <3> |     sizeof(struct test_struct) : 48
            <3> |                   sizeof(**ts) : 48
            <3> |                           size : 240
            <3> |                             ts : 0x7fffedbaeed8
            3<-- _struct_alloc().

        <2> |           ts[i].name : test0
        <2> |             ts[i].id : 0
        <2> |           ts[i].name : test
        <2> |             ts[i].id : 1
        <2> |           ts[i].name : test
        <2> |             ts[i].id : 2
        <2> |           ts[i].name : test
        <2> |             ts[i].id : 3
        <2> |           ts[i].name : test
        <2> |             ts[i].id : 4
        2<-- _struct_alloc_test1().

        2--> _struct_alloc_test2()...
        <2> |                level : 5
        <2> |    sizeof(&ts->span) : 8
        <2> |                span : 0x560c1eb26b80
        <2> |          sizeof(*ts) : 48
        <2> |        sizeof(*span) : 8
        <2> |                  span[i].value : 0
        <2> |                  span[i].value : 1
        <2> |                  span[i].value : 2
        <2> |                  span[i].value : 3
        <2> |                  span[i].value : 4
        2<-- _struct_alloc_test2().
```

Bits Operation Test

메뉴 번호 [4] Bits Operation Test 는 다음과 같이 실행 내용들을 화면에 출력 합니다. 이 예제는 커널 소스에서 사용하고 있는 비트맵 변수들을 어떻게 정의하고 작업하는지 연습하는 소스로 구성되어 있습니다.

Bits Operation Test 실행 내용

```
Enter Menu Number[0,9]: 4
        2--> _bitmap_test_01()...
        <2> |  BITS_PER_TYPE(char) : 8 bits
        <2> |   BITS_PER_TYPE(int) : 32 bits
        <2> |  BITS_PER_TYPE(long) : 64 bits
        <2> |                    i : 0
        <2> |     BITS_TO_LONGS(i) : 0
        <2> |                    i : 32
        <2> |     BITS_TO_LONGS(i) : 1
        <2> |                    i : 64
        <2> |     BITS_TO_LONGS(i) : 1
        <2> |                    i : 96
        <2> |     BITS_TO_LONGS(i) : 2
        <2> |                    i : 128
        <2> |     BITS_TO_LONGS(i) : 2
        <2> |                    i : 160
        <2> |     BITS_TO_LONGS(i) : 3
        <2> |                    i : 192
        <2> |     BITS_TO_LONGS(i) : 3
        <2> |                    i : 224
        <2> |     BITS_TO_LONGS(i) : 4
        <2> |                    i : 256
        <2> |     BITS_TO_LONGS(i) : 4
        <2> |                    i : 288
        <2> |     BITS_TO_LONGS(i) : 5
        <2> |                    i : 320
        <2> |     BITS_TO_LONGS(i) : 5
        <2> |                  sizeof(bitv) : 40
        <2> |   sizeof(bitv) * BITS_PER_BYTE : 320
        2<-- _bitmap_test_01().
```

```
        2--> _bitmap_test_02()...
        <2> |                nbits : 32 nbits
        <2> |                    v : 1 array
        <2> |                  len : 8 bytes
        <2> |    len*BITS_PER_BYTE : 64 bits

v[0]=00000000000000000000000000000000000000000000000000000000000000001

v[0]=00000000000000000000000000000000000000000000000000000000000000000

v[0]=00000000000000000000000000000001111111111111111111111111111111111

        2<-- _bitmap_test_02().
```

CPU Mask Test

메뉴 번호 [5] CPU Mask Test 는 다음과 같이 실행 내용들을 화면에 출력 합니다. 커널의 CPU 마스크는 비트 연산을 사용하여 비트단위로 CPU 을 선택할 수 있도록 합니다. 이 소스에 대한 자세한 설명은 다음장에서 이어 집니다.

CPU Mask Test 실행 내용

```
Enter Menu Number[0,9]: 5
    1--> cpus_mask_test()...
        2--> _cpus_mask_info()...
        <2> |          __CHAR_BIT__ :   8
        <2> |       __SIZEOF_LONG__ :   8
        <2> |         BITS_PER_LONG :  64
        <2> |    BITS_PER_LONG_LONG :  64
        <2> |            nr_cpu_ids :   8
        <2> |       nr_cpumask_bits :   8
        <2> |     __cpu_possible_mask.bits[0] : 0xFF
        <2> |      __cpu_present_mask.bits[0] : 0x0
        <2> |       __cpu_online_mask.bits[0] : 0x0
        <2> |       __cpu_active_mask.bits[0] : 0x0
        2<-- _cpus_mask_info().
```

```
    <1> |                          NR_CPUS : 8
    <1> |                       nr_cpu_ids : 8
    <1> | BITS_TO_LONGS(nr_cpumask_bits) : 1
    <1> |                               nr : 1
    <1> |             cpus_mask->bits[i] : 0x0

    <1> |                              cpu : 0
    <1> |             cpus_mask->bits[0] : 0x1
    <1> |                              cpu : 1
    <1> |             cpus_mask->bits[0] : 0x2
    <1> |                              cpu : 2
    <1> |             cpus_mask->bits[0] : 0x4
    <1> |                              cpu : 3
    <1> |             cpus_mask->bits[0] : 0x8
    <1> |                              cpu : 4
    <1> |             cpus_mask->bits[0] : 0x10
    <1> |                              cpu : 5
    <1> |             cpus_mask->bits[0] : 0x20
    <1> |                              cpu : 6
    <1> |             cpus_mask->bits[0] : 0x40
    <1> |                              cpu : 7
    <1> |             cpus_mask->bits[0] : 0x80
    1<-- cpus_mask_test().
```

Run Time(CPU cycles) Test

메뉴 번호 [6] Run Time(CPU cycles) Test 는 다음과 같이 실행 내용들을 화면에 출력 합니다. 이 메뉴에 있는 소스는 커널 소스의 실행 시간을 벤치마킹 테스트할때 유용하게 사용할 수 있습니다.

Run Time(CPU cycles) 실행 내용

```
Enter Loop Iteration Number: 100
Enter Run Counter: 10
        2--> _run_time_test()...
        <2> | rdtsc 1: run_cycles=2544, one call:25 cycles
        <2> | rdtsc 2: run_cycles=1600, one call:16 cycles
```

```
        <2> | rdtsc 3: run_cycles=1888, one call:18 cycles
        <2> | rdtsc 4: run_cycles=1576, one call:15 cycles
        <2> | rdtsc 5: run_cycles=1548, one call:15 cycles
        <2> | rdtsc 6: run_cycles=1400, one call:14 cycles
        <2> | rdtsc 7: run_cycles=1636, one call:16 cycles
        <2> | rdtsc 8: run_cycles=1664, one call:16 cycles
        <2> | rdtsc 9: run_cycles=1564, one call:15 cycles
        <2> | rdtsc 10: run_cycles=1628, one call:16 cycles
        <2> | *Total Average: one call=17 cycles
        2<-- _run_time_test().
```

Sort Test

메뉴 번호 [7] Sort Test 는 다음과 같이 실행 내용들을 화면에 출력 합니다. 이 메뉴에 있는 소스는 커널의 라이브러리 공통 소스에 있는 정렬 알고리즘을 이해하기 위한 것입니다. 자세한 소스 설명은 다음장에서 진행 합니다.

Sort Test 실행 내용

```
Enter Menu Number[0,9]: 7
Enter Sorting Data Counter: 100
Sorting DataType Size: 4 bytes
Random Source Data[100] ---->
6570, 5893, -870, -1166, -5780, 4192, -4333, 1823, 6524, 628, 1585, 228,
6586, 5429, -612, -2049, -6569, -5922, -4888, 4720, -6447, -2861, -2818, -
4065, 650, 947, 5532, -3599, 334, 3512, -4410, 4056, -389, -1917, -3798,
6105, 6180, 4005, -5509, 2932, 759, 4385, -3331, 5760, -3612, 711, 5777, -
4105, 1810, 302, 4440, -4926, 5822, -5410, 61, 4830, -3038, 3862, -1362, -
96, -3815, 6598, 5081, -3718, 3785, -5728, 2684, 1352, 386, 2004, 1275,
2619, 3237, -3034, 3746, -4597, 2880, 2267, 247, 6035, 1611, 6073, 509,
5036, -2413, -2612, -3440, 2322, 5251, -2049, -6569, -5922, -4888, 4720, -
6447, -2861, -2818, -4065, 650, 947,

Sorting DataType Size: 4 bytes
Sorting Data[100] ---->
-6569, -6569, -6447, -6447, -5922, -5922, -5780, -5728, -5509, -5410, -
```

```
4926, -4888, -4888, -4597, -4410, -4333, -4105, -4065, -4065, -3815, -
3798, -3718, -3612, -3599, -3440, -3331, -3038, -3034, -2861, -2861, -
2818, -2818, -2612, -2413, -2049, -2049, -1917, -1362, -1166, -870, -612,
-389, -96, 61, 228, 247, 302, 334, 386, 509, 628, 650, 650, 711, 759, 947,
947, 1275, 1352, 1585, 1611, 1810, 1823, 2004, 2267, 2322, 2619, 2684,
2880, 2932, 3237, 3512, 3746, 3785, 3862, 4005, 4056, 4192, 4385, 4440,
4720, 4720, 4830, 5036, 5081, 5251, 5429, 5532, 5760, 5777, 5822, 5893,
6035, 6073, 6105, 6180, 6524, 6570, 6586,
FUNC | test_sort_init : Sorting test passed.
FUNC | lib_sort_test : err = 0
FUNC | test_sort_init : Sorting test passed.
FUNC | lib_sort_test : err = 0
```

2.4 알고리즘 및 구조체 학습 메뉴

메인 메뉴에서 [3]을 입력하면 다음과 같이 **Algorithm & Struct 메뉴**가 실행 됩니다. 이 메뉴는 커널 소스에서 가장 많이 등장하는 알고리즘과 자료구조를 이해하기 위한 소소들로 구성되어 있습니다. 이 메뉴에 있는 소스들을 이해하시면 커널 소스 분석의 머나먼 항해를 떠나기 위한 준비물을 갖추게 됩니다. 이 메뉴에서는 커널 소스의 핵심 자료구조 알고리즘인 Linked List 에 대해서 확실히 이해할 수 있도록 다섯가지 종류의 메뉴로 예제 소스를 준비 했습니다. 또한 Red-Black 트리도 커널 소스의 근간을 이루는 자료구조로서 이것을 이해하기 위한 메뉴도 세가지를 준비하여 소스를 테스트 할 수 있습니다. 그리고 비교적 최근에 커널 소스에 추가된 XArray 에 대한 메뉴도 실행하여 소스를 테스트, 분석 및 이해할 수 있도록 예제를 준비 했습니다.

Algorithm & Struct
0: exit.
1: Linked List Test01.
2: Linked List Test02.
3: Linked List Test03.
4: Linked List Test04.
5: List Sort Test.
6: Red-Black Tree Test01.
7: Red-Black Tree Test02.
8: Red-Black Tree Test03.
9: IDA Test.
10: Xarray Test.
11: help.

콘솔에서는 다음과 같이 메뉴가 화면에 출력되며, 키보드에서 메뉴 번호를 입력하여 해당 기능을 실행합니다.

Algorithm & Struct 메뉴

```
[#]--> Algorithm & Struct Menu
 0: exit.
 1: Linked List Test01.
 2: Linked List Test02.
 3: Linked List Test03.
 4: Linked List Test04.
 5: List Sort Test.
 6: Red-Black Tree Test01.
 7: Red-Black Tree Test02.
 8: Red-Black Tree Test03.
 9: XArray Test -->
10: help.
```

위의 메뉴를 번호별로 요약 설명하면 다음과 같습니다.

0: exit. //이전 메뉴(메인)로 이동 합니다.

1: Linked List Test01. //링크드 리스트 구조체와 offsetof, container_of 소스 테스트.

2: Linked List Test02. //링크드 리스트 헤더에 추가 및 탐색 소스 테스트.

3: Linked List Test03. //링크드 리스트 테일에 추가 및 탐색 소스 테스트.

4: Linked List Test04. //링크드 리스트 분리, 붙임 소스 테스트.

5: List Sort Test. //링크드 리스트 정렬 소스 테스트.

6: Red-Black Tree Test01. //Red-Black Tree 삽입, 삭제, 출력 소스 테스트.

7: Red-Black Tree Test02. //Red-Black Tree 검색 소스 테스트.

8: Red-Black Tree Test03. //Red-Black Tree 삽입, 삭제 실행 속도 테스트.

9: XArray Test --> //XArray 을 테스트 하는 하위 메뉴 호출

10: help. //간단한 도움말 출력.

위에서 메뉴 번호 [9]을 입력하면 XArray 을 테스트하는 하위 메뉴가 다음과 같이 출력 됩니다.

Algorithm --> XArray Test Menu

```
[#]--> Algorithm --> XArray Test Menu
0: exit.
1: XArray Check Test.
2: XArray MultiOrder Test.
3: IDR Simple Test.
4: IDA Simple Test.
5: IDR, IDA Check Test.
6: help.
```

1: XArray Check Test. //xarray 오류 체크, 삽입, 삭제, 탐색 테스트

2: XArray MultiOrder Test. //xarray 의 인덱스 order 테스트

3: IDR Simple Test. //IDR 할당, 삽입, 검색, 삭제 테스트

4: IDA Simple Test. //IDA 할당, 해제 테스트

5: IDR, IDA Check Test. //IDR, IDA 할당 해제 전체 테스트

Linked List Test

메뉴 번호 [1] Linked List Test01 는 다음과 같이 실행 내용들을 화면에 출력 합니다. 링크드 리스트 헤더가 포함되어 있는 구조체 멤버 변수들의 크기와 주소, offsetof, container_of 포인터 연산 결과를 확인할 수 있습니다. 소스는 다음장부터 설명 합니다.

Linked List Test01 실행 내용

```
    1--> list_test01()...
    <1> | size: 40, 8, 4, 16
    <1> | first_fox value: 40, 10, 0
    <1> | first_fox addr: 0x562fd5df1a80, 0x562fd5df1a80, 0x562fd5df1a88,
0x562fd5df1a90, 0x562fd5df1a98
    <1> | offset: 0, 8, 16, 24
    <1> | container_of: 0x562fd5df1a80, 0x562fd5df1a80
    1<-- list_test01().
```

메뉴 번호 [2] Linked List Test02 는 다음과 같이 실행 내용들을 화면에 출력 합니다. 링크드 리스트 구조체 헤더에 노드들을 삽입하고 출력하는 결과(LIFO, Stack 구조)를 확인할 수 있습니다.

Linked List Test02 실행 내용

```
Enter Menu Number[0,12]: 2
    1--> list_test02()...
    <1> | fox value:   fourth_fox, 44, 40, 1
    <1> | fox value:    third_fox, 33, 30, 0
    <1> | fox value:   second_fox, 22, 20, 1
    <1> | fox value:    first_fox, 11, 10, 0
    1<-- list_test02().
```

메뉴 번호 [3] Linked List Test03 는 다음과 같이 실행 내용들을 화면에 출력 합니다. 링크드 리스트 구조체 테일에 노드들을 삽입하고 출력하는 결과(FIFO 구조)를 확인할 수 있습니다. 소스는 다음장부터 설명 합니다.

Linked List Test03 실행 내용

```
Enter Menu Number[0,12]: 3
    <1> | fox value:    first_fox, 11, 10, 0
    <1> | fox value:   second_fox, 22, 20, 1
    <1> | fox value:    third_fox, 33, 30, 0
    <1> | fox value:   fourth_fox, 44, 40, 1
```

메뉴 번호 [4] Linked List Test04 는 다음과 같이 실행 내용들을 화면에 출력 합니다. 링크드 리스트를 분리하고 다시 병합하는 결과를 확인할 수 있고 해당 소스를 분석 및 테스트할 수 있습니다.

Linked List Test04 실행 내용

```
Enter Menu Number[0,12]: 4
        2--> list_output()...
        <2> | fox value:   fourth_fox, 44, 40, 3
        <2> | fox value:    third_fox, 33, 30, 2
        <2> | fox value:   second_fox, 22, 20, 1
        <2> | fox value:    first_fox, 11, 10, 0
        2<-- list_output().

        2--> list_output()...
        <2> | fox value:   second_fox, 22, 20, 1
        <2> | fox value:    first_fox, 11, 10, 0
        2<-- list_output().

        2--> list_output()...
        <2> | fox value:   fourth_fox, 44, 40, 3
        <2> | fox value:    third_fox, 33, 30, 2
        2<-- list_output().

        2--> list_output()...
        <2> | fox value:   fourth_fox, 44, 40, 3
        <2> | fox value:    third_fox, 33, 30, 2
```

```
        <2> | fox value:   second_fox, 22, 20, 1
        <2> | fox value:    first_fox, 11, 10, 0
        2<-- list_output().
```

List Sort Test

메뉴 번호 [5] List Sort Test 는 다음과 같이 실행 내용들을 화면에 출력 합니다. 링크드 리스트를 정렬하는 함수(list_sort)가 실행된 결과를 확인할 수 있고 해당 소스를 분석 및 테스트할 수 있습니다.

List Sort Test 실행 내용

```
Enter Menu Number[0,14]: 5
Enter Sorting Data Counter: 100
start testing list_sort()
Random List Data[100] ---->
6570, 5893, -870, -1166, -5780, 4192, -4333, 1823, 6524, 628, 1585, 228,
6586, 5429, -612, -2049, -6569, -5922, -4888, 4720, -6447, -2861, -2818, -
4065, 650, 947, 5532, -3599, 334, 3512, -4410, 4056, -389, -1917, -3798,
6105, 6180, 4005, -5509, 2932, 759, 4385, -3331, 5760, -3612, 711, 5777, -
4105, 1810, 302, 4440, -4926, 5822, -5410, 61, 4830, -3038, 3862, -1362, -
96, -3815, 6598, 5081, -3718, 3785, -5728, 2684, 1352, 386, 2004, 1275,
2619, 3237, -3034, 3746, -4597, 2880, 2267, 247, 6035, 1611, 6073, 509,
5036, -2413, -2612, -3440, 2322, 5251, -2049, -6569, -5922, -4888, 4720, -
6447, -2861, -2818, -4065, 650, 947,

Sorting List Data ---->
-6569, -6569, -6447, -6447, -5922, -5922, -5780, -5728, -5509, -5410, -
4926, -4888, -4888, -4597, -4410, -4333, -4105, -4065, -4065, -3815, -
3798, -3718, -3612, -3599, -3440, -3331, -3038, -3034, -2861, -2861, -
2818, -2818, -2612, -2413, -2049, -2049, -1917, -1362, -1166, -870, -612,
-389, -96, 61, 228, 247, 302, 334, 386, 509, 628, 650, 650, 711, 759, 947,
947, 1275, 1352, 1585, 1611, 1810, 1823, 2004, 2267, 2322, 2619, 2684,
2880, 2932, 3237, 3512, 3746, 3785, 3862, 4005, 4056, 4192, 4385, 4440,
4720, 4720, 4830, 5036, 5081, 5251, 5429, 5532, 5760, 5777, 5822, 5893,
6035, 6073, 6105, 6180, 6524, 6570, 6586,
FUNC | lib_list_sort_test : err = 0
```

Red-Black Tree Test

메뉴 번호 [6] Red-Black Tree Test01 는 다음과 같이 실행 내용들을 화면에 출력 합니다. Red-Black Tree 에 노드를 삽입하고 삭제하는 결과를 확인할 수 있고, 해당 소스를 분석 및 테스트할 수 있습니다.

Red-Black Tree Test01 실행 내용

```
Enter Menu Number[0,12]: 6
    output rb_node: 0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 5, 5, 6, 6,
7, 7,
    searched value=4
    erasing value=4
    output rb_node: 0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 5, 5, 6, 6, 7,
7,
    drop rb_node: rb_test_root->rb_node: (nil)
    output rb_node:
    output rb_node: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 18, 19,
```

메뉴 번호 [7] Red-Black Tree Test02 는 다음과 같이 실행 내용들을 화면에 출력 합니다. Red-Black Tree 에 8 개의 노드를 삽입하고 각각에 대해서 탐색 테스트를 수행 합니다.

Red-Black Tree Test02 실행 내용

```
Enter Menu Number[0,12]: 7
datatype size: u16=2, u32=4, u64=8
Nodes Size: 320 / 40, Count: 8
rbtree_nodes_init(0) ==============================
0: 0xD5E00868: 10
1: 0xD5E00890: 20
2: 0xD5E008B8: 30
3: 0xD5E008E0: 40
4: 0xD5E00908: 50
5: 0xD5E00930: 60
6: 0xD5E00958: 70
```

```
7: 0xD5E00980: 80
rbtree insert testing...wait...
search level=4
search level=4
rbtree_search ----------------------------------
0: 0xD5E00868: 10
1: 0xD5E00890: 20
2: 0xD5E008B8: 30
3: 0xD5E008E0: 40
4: 0xD5E00908: 50
5: 0xD5E00930: 60
6: 0xD5E00958: 70
7: 0xD5E00980: 80

rbtree erase testing...wait...
rbtree_search ----------------------------------

*runtime: 0 seconds
augmented rbtree insert testing...wait...
rbtree_search ----------------------------------
0: 0xD5E00868: 10
1: 0xD5E00890: 20
2: 0xD5E008B8: 30
3: 0xD5E008E0: 40
4: 0xD5E00908: 50
5: 0xD5E00930: 60
6: 0xD5E00958: 70
7: 0xD5E00980: 80

augmented rbtree erase testing...wait...
rbtree_search ----------------------------------

*runtime: 0 seconds
rbtree_nodes_init(1) ==============================
0: 0xD5E00868: 80
1: 0xD5E00890: 70
2: 0xD5E008B8: 60
3: 0xD5E008E0: 50
4: 0xD5E00908: 40
5: 0xD5E00930: 30
6: 0xD5E00958: 20
7: 0xD5E00980: 10
rbtree insert testing...wait...
search level=4
```

```
search level=4
rbtree_search ---------------------------------
0: 0xD5E00980: 10
1: 0xD5E00958: 20
2: 0xD5E00930: 30
3: 0xD5E00908: 40
4: 0xD5E008E0: 50
5: 0xD5E008B8: 60
6: 0xD5E00890: 70
7: 0xD5E00868: 80

rbtree erase testing...wait...
rbtree_search ---------------------------------

*runtime: 0 seconds
augmented rbtree insert testing...wait...
rbtree_search ---------------------------------
0: 0xD5E00980: 10
1: 0xD5E00958: 20
2: 0xD5E00930: 30
3: 0xD5E00908: 40
4: 0xD5E008E0: 50
5: 0xD5E008B8: 60
6: 0xD5E00890: 70
7: 0xD5E00868: 80

augmented rbtree erase testing...wait...
rbtree_search ---------------------------------

*runtime: 0 seconds
rbtree_nodes_init(2) ==============================
0: 0xD5E00868: 10
1: 0xD5E00890: 30
2: 0xD5E008B8: 20
3: 0xD5E008E0: 50
4: 0xD5E00908: 40
5: 0xD5E00930: 25
6: 0xD5E00958: 22
7: 0xD5E00980: 35
rbtree insert testing...wait...
search level=4
search level=4
rbtree_search ---------------------------------
0: 0xD5E00868: 10
```

```
1: 0xD5E008B8: 20
2: 0xD5E00958: 22
3: 0xD5E00930: 25
4: 0xD5E00890: 30
5: 0xD5E00980: 35
6: 0xD5E00908: 40
7: 0xD5E008E0: 50

rbtree erase testing...wait...
rbtree_search ---------------------------------

*runtime: 0 seconds
augmented rbtree insert testing...wait...
rbtree_search ---------------------------------
0: 0xD5E00868: 10
1: 0xD5E008B8: 20
2: 0xD5E00958: 22
3: 0xD5E00930: 25
4: 0xD5E00890: 30
5: 0xD5E00980: 35
6: 0xD5E00908: 40
7: 0xD5E008E0: 50

augmented rbtree erase testing...wait...
rbtree_search ---------------------------------

*runtime: 0 seconds
test end.
```

메뉴 번호 [8] Red-Black Tree Test03 는 다음과 같이 실행 내용들을 화면에 출력 합니다. Red-Black Tree 에 노드를 삽입하고 삭제하는 시간을 테스트하고 해당 소스를 분석할 수 있습니다.

Red-Black Tree Test03 실행 내용

```
Enter Menu Number[0,12]: 8
datatype size: u16=2, u32=4, u64=8, time_t=8
nodes size: 40 / 4000
random number initialize...
rnd: 1082932233, 1082932233, 1082932233, 1082932233
rbtree testing...wait...
*runtime: 1 seconds
search level=9
```

```
search level=9
augmented rbtree testing...wait...
*runtime: 2 seconds
test exit.
```

XArray Test

XArray Test 메뉴는 다음과 같은 하위 메뉴로 구성되어 있습니다. XArray 을 각각의 메뉴 항목별로 소스를 테스트하고 분석 및 디버깅 할 수 있습니다.

Algorithm --> XArray Test Menu

```
[#]--> Algorithm --> XArray Test Menu
0: exit.
1: XArray Check Test.
2: XArray MultiOrder Test.
3: IDR Simple Test.
4: IDA Simple Test.
5: IDR, IDA Check Test.
6: help.
```

메뉴 번호 [1] XArray Check Test 는 다음과 같이 실행 내용들을 화면에 출력 합니다. xarray 에 오류 사항이 발생 했는지 여부를 체크하고 노드들을 삽입, 삭제, 탐색한 결과를 단계적으로 보여 줍니다.

XArray Check Test 실행 내용

```
Enter Menu Number[0,7]: 1
check_xa_err()...
check_xas_retry()...
check_xa_load()...
check_xa_mark()...
check_xa_shrink()...
check_xas_erase()...
check_insert()...
```

```
check_cmpxchg()...
check_reserve()...
check_reserve()...
check_multi_store()...
check_xa_alloc()...
check_find()...
check_find_entry()...
check_account()...
check_destroy()...
check_move()...
check_create_range()...
check_store_range()...
check_store_iter()...
check_align()...
check_workingset()...
XArray: 21005221 of 21005221 tests passed
FUNC | lib_xarray_test : result = 0
nr_allocated = 328171
```

메뉴 번호 [2] XArray MultiOrder Test.는 다음과 같이 실행 내용들을 화면에 출력 합니다. xarray 의 인덱스 order 들을 테스트한 결과를 확인하고 소스를 분석할 수 있습니다.

XArray MultiOrder Test 실행 내용

```
Enter Menu Number[0,7]: 2
            3--> multiorder_checks()...
                4--> multiorder_iteration()...
                <4> |           NUM_ENTRIES : 11
                <4> |       XA_CHUNK_SHIFT : 6
                4<-- multiorder_iteration().

                4--> multiorder_tagged_iteration()...
                4<-- multiorder_tagged_iteration().

                4--> radix_tree_cpu_dead()...
                4<-- radix_tree_cpu_dead().
```

```
            3<-- multiorder_checks().
```

메뉴 번호 [3] IDR Simple Test 는 다음과 같이 실행 내용들을 화면에 출력 합니다. 리눅스 커널의 IDR 은 “ID Radix” 입니다. 정수형 ID(인덱스)를 관리하기 위한 “integer ID management” 입니다. 이러한 IDR 을 할당하고 노드들을 삽입, 검색, 삭제하는 과정을 테스트하고 소스를 분석 및 디버깅 할 수 있습니다.

IDR Simple Test 실행 내용

```
Enter Menu Number[0,7]: 3

idr_alloc()..........................
i=0, id=0
i=1, id=1
i=2, id=2
i=3, id=3
i=4, id=4
i=5, id=5
i=6, id=6
i=7, id=7
i=8, id=8
i=9, id=9
idr_for_each_entry().................
id=0, item->index=0, order=0
id=1, item->index=1, order=10
id=2, item->index=2, order=20
id=3, item->index=3, order=30
id=4, item->index=4, order=40
id=5, item->index=5, order=50
id=6, item->index=6, order=60
id=7, item->index=7, order=70
id=8, item->index=8, order=80
id=9, item->index=9, order=90
idr_find()...........................
found id=5, item->index=5, order=50
idr_remove()........................
```

```
Not found(id=5)
idr_destroy().....................
idr is empty.
```

메뉴 번호 [4] IDA Simple Test 는 다음과 같이 실행 내용들을 화면에 출력 합니다. IDA 는 ID 할당자인 “ID Allocator” 입니다. ID 는 식별번호(인덱스 키) 역할을 합니다. IDR 은 ID 을 생성하여 사용자 데이터까지 연결하여 관리하지만 IDA 는 ID(인덱스)만 생성하여 사용합니다. 아래의 실행 내용은 IDA 을 할당하고 노드들을 해제하는 과정을 테스트한 것입니다. 소스에 대해서는 “제 5 장 알고리즘 및 구조체 소스 설명”의 XArray 와 IDA, IDR 설명에서 자세히 기술 됩니다.

IDA Simple Test 실행 내용

```
Enter Menu Number[0,7]: 4

ida_alloc().........................
id=0
id=1
id=2
id=3
id=4
id=5
id=6
id=7
id=8
id=9
ida_free().........................
id=3
id=5
id=10
id=11
id=12
ida_destroy().....................
ida is empty.
```

메뉴 번호 [5] IDR, IDA Check Test 는 다음과 같이 실행 내용들을 화면에 출력 합니다. IDR, IDA 할당하고 노드들을 해제하는 과정을 전체적으로 테스트하고 소스를 분석 및 디버깅 할 수 있습니다.

IDR, IDA Check Test

```
Enter Menu Number[0,7]: 5

ida_check_alloc()...
ida_check_destroy()...
ida_check_leaf(0)...
ida_check_leaf(1024)...
ida_check_leaf(1024 * 64)...
ida_check_max()...
ida_check_conv()...
IDA: 147353 of 147353 tests passed
FUNC | lib_ida_test : result = 0
```

지금까지 XArray 를 테스트하는 메뉴를 기능 위주로 요약 했습니다. 이곳에서 실행되는 소스들은 "제 5 장 알고리즘 및 구조체 소스 설명"에서 XArray 소스 설명에서 기술 됩니다.

2.5 커널 시작 소스 테스트 메뉴

앞에서 커널 소스에 대한 기본을 이해했다면, 이제부터는 본격적으로 커널 소스 테스트를 시작합니다. “Start Kernel Test” 메뉴는 커널이 처음에 부팅될때 실행하는 소스들을 테스트하는 내용으로 구성되어 있습니다. 여기서 테스트 하는 소스들은 리눅스 커널 소스 init/main.c 에 있는 내용들입니다. 이 소스들은 아래 메뉴와 같이 구성하여 단계적으로 실행하면서 소스 안의 내용들을 분석 및 디버그할 수 있습니다. **(참고로 아래 메뉴는 메뉴번호 순서대로 실행해야 합니다.)**

Start Kernel Test
0: exit.
1: setup_arch test.
2: sched_init test.
3: numa_init test.
4: sched_init_smp test.
5: help.

콘솔에서는 다음과 같이 메뉴가 화면에 출력되며, 키보드에서 메뉴 번호를 입력하여 해당 기능을 실행합니다.

Start Kernel Test 메뉴

```
[#]--> Start Kernel Test Menu
0: exit.
1: setup_arch test.
2: sched_init test.
3: numa_init test.
4: sched_init_smp test.
5: help.
```

위의 메뉴를 번호별로 요약 설명하면 다음과 같습니다.

0: exit. //이전 메뉴로 돌아 갑니다.

1: setup_arch test. //커널이 부팅될때 실행하는 setup_arch() 함수안의 소스 테스트.

2: sched_init test. //sched_init() 함수에서 실행되는 소스 테스트.

3: numa_init test. //numa_init() 함수에서 실행되는 소스 테스트.

4: sched_init_smp test. //sched_init_smp() 함수에서 실행되는 소스 테스트.

5: help.

setup_arch test

리눅스 커널 소스 init/main.c 의 start_kernel() 함수에서 실행되는 **setup_arch()** 소스를 테스트 합니다. 소스가 함수 단위로 실행되는 과정과 그때마다 처리되는 데이터들을 스택깊이별로 확인할 수 있습니다. 특히 이곳에서 아키텍쳐별로 제공하는 DTB 을 파싱하여 device_node 에 적용하는 과정을 상세히 확인할 수 있습니다. 소스는 "제 6 장 커널 시작 소스 설명"에서 기술 됩니다.

setup_arch test 실행 내용

```
Enter Menu Number[0,6]: 1
    1--> test_setup_arch()...
        2--> setup_arch()...
            3--> parse_dtb()...
                4--> dtb_test_read_file()...
                <4> |        dtb_file_name : testcases.dtb
                <4> |             dtb_size : 7380 bytes
                4<-- dtb_test_read_file().

                4--> early_init_dt_scan()...
                    5--> early_init_dt_verify()...
                        6--> fdt_check_header()...
```

```
                        <6> |       fdt_magic(fdt) : 0xd00dfeed
                        <6> |              hdrsize : 40
                        <6> |     fdt_version(fdt) : 0x11
                        <6> |   fdt_totalsize(fdt) : 7380
                        6<-- fdt_check_header().

                    <5> |                                    params :
0x55c40ba30a80
                    <5> |                       initial_boot_params :
0x55c40ba30a80
                    <5> |       fdt_totalsize(initial_boot_params) : 7380
                    5<-- early_init_dt_verify().

                    5--> early_init_dt_scan_nodes()...
                    <5> |    boot_command_line :
                        6--> of_scan_flat_dt()...
                        <6> |  initial_boot_params : 0x55c40ba30a80
                        <6> |         dtb_early_va : 0x55c40ba30a80


//이하 생략...
```

sched_init test

리눅스 커널 소스 init/main.c 의 start_kernel() 함수에서 실행되는 **sched_init()** 소스를 테스트 합니다. 여기서는 CPU 들이 설정 되고 스케쥴링을 실행하기 위한 구조체들(runqueue, task_group, sched_entity)을 메모리 할당하고 초기화 합니다. sched_init() 함수 안의 소스가 실행되는 과정을 스택 깊이별로 단계적으로 출력하고 그때마다 실행되는 데이터들을 확인할 수 있습니다.

sched_init test 실행 내용

```
Enter Menu Number[0,6]: 2
        2--> sched_init()...
```

```
        <2> |                              ptr : 256
        <2> |        (void*)&root_task_group : 0x55c40b8f4280
        <2> |                     (void*)ptr : 0x55c40ba32810
        <2> |      (void*)root_task_group.se : 0x55c40ba32810
        <2> |  (void*)root_task_group.cfs_rq : 0x55c40ba32850
        <2> |   (void*)root_task_group.rt_se : 0x55c40ba32890
        <2> |   (void*)root_task_group.rt_rq : 0x55c40ba328d0
            3--> init_defrootdomain()...
                4--> init_rootdomain()...
                    5--> cpudl_init()...
                    <5> |                    i : 8
                    5<-- cpudl_init().

                    5--> cpupri_init()...
                    5<-- cpupri_init().

            3<-- init_defrootdomain().

        <2> |    __cpu_possible_mask.bits[0] : 0xFF
        <2> |              (void*)&runqueues : 0x55c40b8ee860
        <2> |                               i : 0
        <2> |                     (void*)rq : 0x55c40b8ee860
        <2> |                     rq->clock : 0
        <2> |                (void*)&rq->cfs : 0x55c40b8ee8c0
        <2> |                 (void*)&rq->rt : 0x55c40b8eea40
        <2> |                 (void*)&rq->dl : 0x55c40b8ef120
            3--> init_cfs_rq()...
            <3> |           cfs_rq->min_vruntime : 18446744073708503040
            3<-- init_cfs_rq().

            3--> init_rt_rq()...
            3<-- init_rt_rq().

            3--> init_dl_rq()...
            3<-- init_dl_rq().

        <2> |   (void*)&rq->leaf_cfs_rq_list : 0x55c40b8ef188
        <2> |    (void*)rq->tmp_alone_branch : 0x55c40b8ef188
            3--> init_cfs_bandwidth()...
            <3> |       cfs_b->runtime: 0
            <3> |         cfs_b->quota: 18446744073709551615
            <3> |        cfs_b->period: 100000000
```

```
            3<-- init_cfs_bandwidth().


//이하 생략...
```

numa_init test

이 메뉴에서는 Bit-Little 구조처럼 CPU 여러개가 다중적으로 동작하는 NUMA 환경을 설정하는 소스를 테스트 합니다. NUMA 환경은 CPU 아키텍쳐별로 소스가 구성되어 있습니다. 여기서는 ARM64 기준으로 arch/arm64/mm/numa.c 에 있는 **arm64_numa_init()** 함수 내부의 소스들이 실행되는 과정을 스택 깊이별로 출력하고 그때마다 데이터들을 확인할 수 있습니다.

numa_init test 실행 내용

```
Enter Menu Number[0,6]: 3
        <2> |                          nr_cpu : 8
        <2> |     cpu_possible_mask->bits[0] : 0xFF
        <2> |       cpu_online_mask->bits[0] : 0xFF
        <2> |      cpu_present_mask->bits[0] : 0xFF
        <2> |       cpu_active_mask->bits[0] : 0xFF
        2--> numa_usr_set_node()...
        <2> |                  step : 2
        <2> |                     i : 0
        <2> |                   nid : 0
        <2> |                     i : 1
        <2> |                   nid : 0
        <2> |                     i : 2
        <2> |                   nid : 1
        <2> |                     i : 3
        <2> |                   nid : 1
        <2> |                     i : 4
        <2> |                   nid : 2
        <2> |                     i : 5
```

```
        <2> |                  nid : 2
        <2> |                    i : 6
        <2> |                  nid : 3
        <2> |                    i : 7
        <2> |                  nid : 3
        2<-- numa_usr_set_node().

        2--> arm64_numa_init()...
Initialized distance table, cnt=4
Node to cpumask map for 4 nodes
            3--> numa_usr_set_cpumask_map()...
            <3> |                  cpu : 0
            <3> |                  nid : 0
            <3> |     cpumask->bits[0] : 0x3
            <3> |                  cpu : 1
            <3> |                  nid : 0
            <3> |     cpumask->bits[0] : 0x3
            <3> |                  cpu : 2
            <3> |                  nid : 1
            <3> |     cpumask->bits[0] : 0xC
            <3> |                  cpu : 3
            <3> |                  nid : 1
            <3> |     cpumask->bits[0] : 0xC
            <3> |                  cpu : 4
            <3> |                  nid : 2
            <3> |     cpumask->bits[0] : 0x30
            <3> |                  cpu : 5
            <3> |                  nid : 2
            <3> |     cpumask->bits[0] : 0x30
            <3> |                  cpu : 6
            <3> |                  nid : 3
            <3> |     cpumask->bits[0] : 0xC0
            <3> |                  cpu : 7
            <3> |                  nid : 3
            <3> |     cpumask->bits[0] : 0xC0
            3<-- numa_usr_set_cpumask_map().

        2<-- arm64_numa_init().

        2--> numa_distance_info()...
0: 10, 20, 30, 20,
1: 20, 10, 20, 30,
```

```
2: 30, 20, 10, 20,
3: 20, 30, 20, 10,
        2<-- numa_distance_info().
```

sched_init_smp test

SMP 환경에서 스케쥴링 도메인 설정에 관련되는 소스를 테스트 합니다. 먼저 sched_init_numa() 함수가 실행되는 과정을 분석하고, sched_init_domains() 함수가 실행되는 과정을 스택 깊이별로 자세히 분석 및 디버깅할 수 있습니다. 소스에 대한 설명은 "제 6 장 커널 시작 소스 설명"에서 부터 기술 합니다.

sched_init_smp test 실행 내용

```
5--> pr_debug_sd_rq_info()...
    <5> | ================================
    <5> |                   cpu : 0
    <5> | --------------------------------
    <5> | --------------------------------
    <5> |                           cnt++ : rq->sd->parent: 0
    <5> | --------------------------------
    <5> |                        (void*)sd : d.sd: 0x55c40ba3a3a0
    <5> |                         sd->name : DIE
    <5> |                 sd->span_weight : 2
    <5> |                        sd->flags : 0x102F
    <5> |               (void*)sd->parent : 0x55c40ba3dc00
    <5> |                                    gcnt++ : 0
    <5> | --------------------------------
    <5> |                           (void*)sd_sg_next : 0x55c40ba360e0
    <5> |                    sd_sg_next->ref.counter : 2
    <5> |               sd_sg_next->asym_prefer_cpu : 0
    <5> |                  sd_sg_next->group_weight : 1
    <5> |                    sd_sg_next->cpumask[0] : 0x1
    <5> |                                (void*)sd_sg_next->sgc :
0x55c40ba373d0
    <5> |                                   sd_sg_next->sgc->id : 0
```

```
    <5> |                          sd_sg_next->sgc->ref.counter : 2
    <5> |                             sd_sg_next->sgc->capacity : 1024
    <5> |                           sd_sg_next->sgc->cpumask[0] : 0x1
    <5> |                                       gcnt++ : 1
    <5> | ---------------------------------
    <5> |                            (void*)sd_sg_next : 0x55c40ba36b60
    <5> |                       sd_sg_next->ref.counter : 2
    <5> |                  sd_sg_next->asym_prefer_cpu : 0
    <5> |                     sd_sg_next->group_weight : 1
    <5> |                       sd_sg_next->cpumask[0] : 0x2
    <5> |                                (void*)sd_sg_next->sgc :
0x55c40ba36b90
    <5> |                                   sd_sg_next->sgc->id : 1
    <5> |                          sd_sg_next->sgc->ref.counter : 2
    <5> |                             sd_sg_next->sgc->capacity : 1024
    <5> |                           sd_sg_next->sgc->cpumask[0] : 0x2
    <5> | ---------------------------------
    <5> |                             cnt++ : rq->sd->parent: 1
    <5> | ---------------------------------
    <5> |                         (void*)sd : d.sd: 0x55c40ba3dc00
    <5> |                          sd->name : NUMA
    <5> |                   sd->span_weight : 6
    <5> |                         sd->flags : 0x642F
    <5> |                 (void*)sd->parent : 0x55c40ba3ed00
    <5> |                                       gcnt++ : 0
    <5> | ---------------------------------
    <5> |                            (void*)sd_sg_next : 0x55c40ba478d0
    <5> |                       sd_sg_next->ref.counter : 1
    <5> |                  sd_sg_next->asym_prefer_cpu : 0
    <5> |                     sd_sg_next->group_weight : 2
    <5> |                       sd_sg_next->cpumask[0] : 0x3
    <5> |                                (void*)sd_sg_next->sgc :
0x55c40ba3d300
    <5> |                                   sd_sg_next->sgc->id : 0
    <5> |                          sd_sg_next->sgc->ref.counter : 6
    <5> |                             sd_sg_next->sgc->capacity : 2048
    <5> |                           sd_sg_next->sgc->cpumask[0] : 0x3
    <5> |                                       gcnt++ : 1
    <5> | ---------------------------------
    <5> |                            (void*)sd_sg_next : 0x55c40ba47900
    <5> |                       sd_sg_next->ref.counter : 1
```

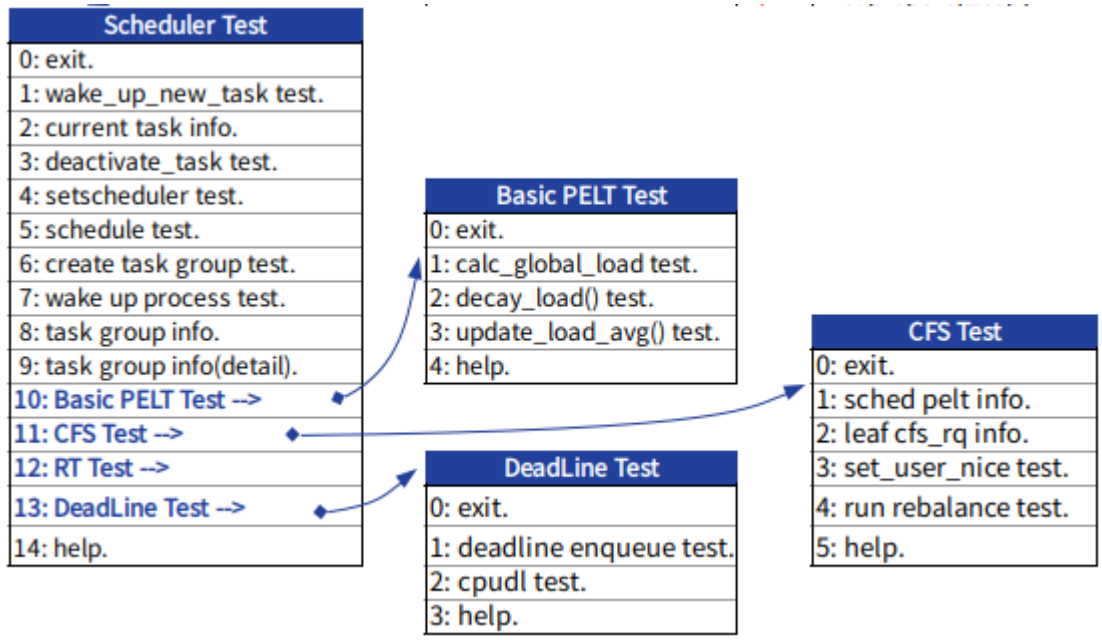
```
    <5> |               sd_sg_next->asym_prefer_cpu : 0
    <5> |                  sd_sg_next->group_weight : 2
    <5> |                    sd_sg_next->cpumask[0] : 0xC
    <5> |                              (void*)sd_sg_next->sgc :
0x55c40ba3d680
    <5> |                                 sd_sg_next->sgc->id : 2
    <5> |                       sd_sg_next->sgc->ref.counter : 6
    <5> |                          sd_sg_next->sgc->capacity : 2048
    <5> |                        sd_sg_next->sgc->cpumask[0] : 0xC
    <5> |                                    gcnt++ : 2
    <5> | ---------------------------------
    <5> |                         (void*)sd_sg_next : 0x55c40ba47930
    <5> |                     sd_sg_next->ref.counter : 1
    <5> |               sd_sg_next->asym_prefer_cpu : 0
    <5> |                  sd_sg_next->group_weight : 2
    <5> |                    sd_sg_next->cpumask[0] : 0xC0
    <5> |                              (void*)sd_sg_next->sgc :
0x55c40ba3dd80
    <5> |                                 sd_sg_next->sgc->id : 6
    <5> |                       sd_sg_next->sgc->ref.counter : 6
    <5> |                          sd_sg_next->sgc->capacity : 2048
    <5> |                        sd_sg_next->sgc->cpumask[0] : 0xC0
    <5> | ---------------------------------
    <5> |                          cnt++ : rq->sd->parent: 2
    <5> | ---------------------------------
    <5> |                       (void*)sd : d.sd: 0x55c40ba3ed00
    <5> |                        sd->name : NUMA
    <5> |                 sd->span_weight : 8
    <5> |                       sd->flags : 0x642F
    <5> |               (void*)sd->parent : (nil)
    <5> |                                    gcnt++ : 0
    <5> | ---------------------------------
    <5> |                         (void*)sd_sg_next : 0x55c40ba47960
    <5> |                     sd_sg_next->ref.counter : 1
    <5> |               sd_sg_next->asym_prefer_cpu : 0
    <5> |                  sd_sg_next->group_weight : 6
    <5> |                    sd_sg_next->cpumask[0] : 0xCF
    <5> |                              (void*)sd_sg_next->sgc :
0x55c40ba3c3c0
    <5> |                                 sd_sg_next->sgc->id : 0
    <5> |                       sd_sg_next->sgc->ref.counter : 4
```

```
    <5> |                              sd_sg_next->sgc->capacity : 6144
    <5> |                            sd_sg_next->sgc->cpumask[0] : 0x3
    <5> |                                       gcnt++ : 1
    <5> | --------------------------------
    <5> |                            (void*)sd_sg_next : 0x55c40ba47990
    <5> |                       sd_sg_next->ref.counter : 1
    <5> |                  sd_sg_next->asym_prefer_cpu : 0
    <5> |                     sd_sg_next->group_weight : 6
    <5> |                       sd_sg_next->cpumask[0] : 0xFC
    <5> |                                 (void*)sd_sg_next->sgc :
0x55c40ba3cac0
    <5> |                                    sd_sg_next->sgc->id : 4
    <5> |                          sd_sg_next->sgc->ref.counter : 4
    <5> |                             sd_sg_next->sgc->capacity : 6144
    <5> |                            sd_sg_next->sgc->cpumask[0] : 0x30
```

2.6 스케쥴러 소스 테스트 메뉴

지금 부터는 리눅스 커널의 핵심 기능인 스케쥴러 소스들을 테스트하는 메뉴입니다. 커널 스케쥴러 소스는 kernel/sched/ 경로에 소스 파일들이 모두 모여 있습니다. 소스 파일의 개수와 소스량이 상당히 많습니다. 스케쥴러는 정복 해야하는 리눅스 커널 소스들중에서 가장 크고 중요한 산맥중에 하나 입니다. 필자는 스케쥴러 소스들을 순서대로 단계적으로 등반하여 이해할 수 있도록 다음과 같은 메뉴로 구성 했습니다. **(참고로 아래 메뉴는 메뉴번호 순서대로 실행해야 합니다.)**

스케쥴러 소스 테스트 메뉴 흐름



콘솔에서는 다음과 같이 메뉴가 화면에 출력되며, 키보드에서 메뉴 번호를 입력하여 해당 기능을 실행합니다.

Scheduler Test 메뉴

```
[#]--> Scheduler Source Test Menu
 0: exit.
 1: wake_up_new_task test.
 2: current task info.
 3: deactivate_task test.
 4: setscheduler test.
 5: schedule test.
 6: create task group test.
 7: wake up process test.
 8: task group info.
 9: task group info(detail).
10: Basic PELT Test -->
11: CFS Test -->
12: RT Test -->
13: DeadLine Test -->
14: help.
```

위의 메뉴를 번호별로 요약 설명하면 다음과 같습니다.

0: exit. //이전 메뉴로 돌아 갑니다.

1: wake_up_new_task test. //새로운 태스크를 하나 만드는 함수 소스 테스트.

2: current task info. //현재 태스크 정보를 확인하는 소스.

3: deactivate_task test. //태스크 실행을 비활성화 하는 함수 소스 테스트.

4: setscheduler test. //스케쥴링 정보를 설정하는 함수 소스 테스트.

5: schedule test. //스케쥴을 실행하는 함수 소스 테스트.

6: create task group test. //태스크 그룹을 생성하는 함수 소스 테스트.

7: wake up process test. //프로세스를 깨우는 함수 소스.

8: task group info. //태스크 그룹 정보를 확인하는 소스.

9: task group info(detail). //태스크 그룹 정보를 상세히 디버깅 하는 소스.

10: Basic PELT Test --> 스케쥴링 PELT 을 이해하기 위한 소스(하위 메뉴로 이동)

11: CFS Test --> CFS 스케쥴러를 이해하기 위한 소스(하위 메뉴로 이동)

12: RT Test --> RT 스케쥴러를 이해하기 위한 소스(하위 메뉴로 이동)

13: DeadLine Test --> DeadLine 스케쥴러를 이해하기 위한 소스(하위 메뉴로 이동)

14: help.

위의 메뉴별로 소스가 실행되는 내용들은 함수가 호출되는 스택 깊이별로 함수 명칭이 출력되면서 그안의 데이터들도 같이 확인할 수 있습니다. 소스가 실행되는 과정을 디버깅 메세지를 보면서 분석할 수 있습니다. 지면 관계상 실행 내용들은 생략 하도록 하겠습니다.

Basic PELT Test 메뉴

Scheduler Test 메뉴에서 [10]을 입력하면 Basic PELT Test 메뉴로 진입합니다. 이 메뉴는 PELT(Per-Entity Load Tracking)에 관련되는 소스를 테스트 하기 위한 메뉴로 구성되어 있습니다.

Basic PELT Test
0: exit.
1: calc_global_load test.
2: decay_load() test.
3: update_load_avg() test.
4: help.

콘솔에서는 다음과 같이 메뉴가 화면에 출력되며, 키보드에서 메뉴 번호를 입력하여 해당 기능을 실행합니다.

```
[#]--> Scheduler --> Basic PELT Test Menu
0: exit.
1: calc_global_load test.
```

```
2: decay_load() test.
3: update_load_avg() test.
4: help.
```

위의 메뉴를 번호별로 요약 설명하면 다음과 같습니다.

0: exit.

1: calc_global_load test. //kernel/sched/loadavg.c 에 있는 calc_global_load() 함수 소스.

2: decay_load() test. //kernel/sched/pelt.c 에 있는 decay_load() 함수 소스 테스트.

3: update_load_avg() test. //kernel/sched/pelt.c 에 있는 ___update_load_avg() 함수 소스.

4: help.

위의 메뉴별로 소스가 실행되는 내용들은 함수가 호출되는 스택 깊이별로 함수 명칭이 출력되면서 그안의 데이터들도 같이 확인할 수 있습니다. 소스가 실행되는 과정을 디버깅 메세지를 보면서 분석할 수 있습니다. 지면 관계상 실행 내용들은 생략 하도록 하겠습니다.

CFS Test 메뉴

Scheduler Test 메뉴에서 [11]을 입력하면 CFS Test 메뉴로 진입합니다. CFS(Completely Fair Scheduler) 스케쥴러는 커널 스케쥴러들 중에서 가장 많이 사용하는 스케쥴러 입니다. 이 소스를 이해하기 위해서 다음과 같이 메뉴를 준비 했습니다.

CFS Test
0: exit.
1: sched pelt info.
2: leaf cfs_rq info.
3: set_user_nice test.
4: run rebalance test.
5: help.

콘솔에서는 다음과 같이 메뉴가 화면에 출력되며, 키보드에서 메뉴 번호를 입력하여 해당 기능을 실행합니다.

```
[#]--> Scheduler --> CFS Test Menu
0: exit.
1: sched pelt info.
2: leaf cfs_rq info.
3: set_user_nice test.
4: run rebalance test.
5: help.
```

위의 메뉴를 번호별로 요약 설명하면 다음과 같습니다.

0: exit.

1: sched pelt info. //스케쥴러에 의해서 계산되어 있는 PELT 정보를 확인 합니다.

2: leaf cfs_rq info. //leaf_cfs_rq 포인터를 따라가며 그곳에 있는 정보를 확인 합니다.

3: set_user_nice test. //스케쥴러 nice 값을 설정하고 테스트 합니다.

4: run rebalance test. //rebalance 을 실행하고 테스트 합니다.

5: help.

위의 메뉴별로 소스가 실행되는 내용들은 함수가 호출되는 스택 깊이별로 함수 명칭이 출력되면서 그안의 데이터들도 같이 확인할 수 있습니다. 소스가 실행되는 과정을 디버깅 메세지를 보면서 분석할 수 있습니다. 지면 관계상 실행 내용들은 생략 하도록 하겠습니다.

DeadLine Test 메뉴

Scheduler Test 메뉴에서 [13]을 입력하면 DeadLine Test 메뉴로 진입합니다. DeadLine 스케쥴러는 그렇게 많이 사용하는 스케쥴러는 아니지만 타이머를 활용하여 우선순위를 빠르게 처리하는 스케쥴러 입니다. 이 소스를 이해하기 위해서 다음과 같이 메뉴를 준비 했습니다.

DeadLine Test
0: exit.
1: deadline enqueue test.
2: cpudl test.
3: help.

콘솔에서는 다음과 같이 메뉴가 화면에 출력되며, 키보드에서 메뉴 번호를 입력하여 해당 기능을 실행합니다.

```
[#]--> Scheduler --> DeadLine Test Menu
0: exit.
1: deadline enqueue test.
2: cpudl test.
3: help.
```

위의 메뉴를 번호별로 요약 설명하면 다음과 같습니다.

0: exit.

1: deadline enqueue test. //deadline 스케쥴러에서 enqueue 하는 과정을 테스트 합니다.

2: cpudl test. //cpudl 구조체값을 계산하는 과정을 테스트 합니다.

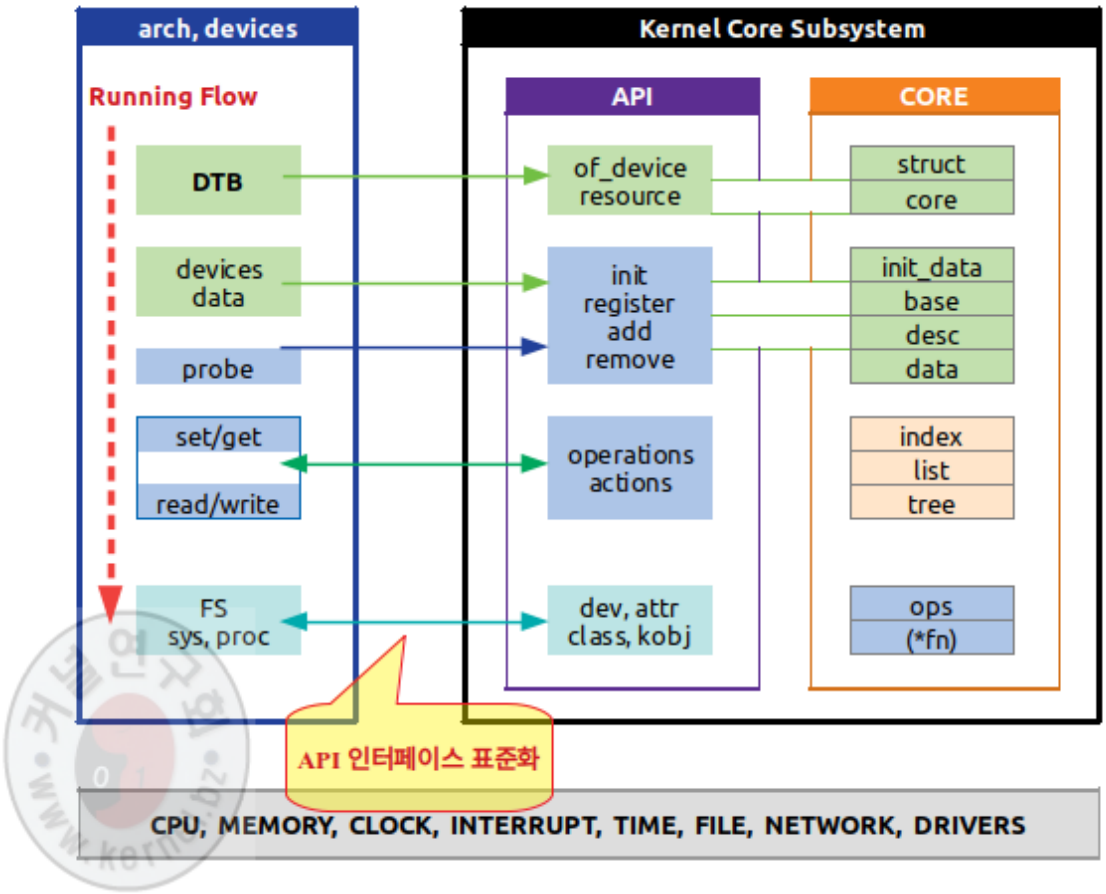
3: help.

위의 메뉴별로 소스가 실행되는 내용들은 함수가 호출되는 스택 깊이별로 함수 명칭이 출력되면서 그안의 데이터들도 같이 확인할 수 있습니다. 소스가 실행되는 과정을 디버깅 메세지를 보면서 분석할 수 있습니다. 지면 관계상 실행 내용들은 생략 하도록 하겠습니다.

2.7 드라이버 소스 테스트 메뉴

이제 디바이스 드라이버를 테스트하는 소스 입니다. 커널에서 디바이스 드라이버에 관련 되는 소스는 drivers/ 경로에 소스 파일들이 모여 있습니다. 커널 전체 소스 중에서 80%을 차지하는 가장 많은 분량으로 되어 있습니다. 장치들이 제조사마다 수많은 모델들이 있어서 소스 분량이 많아 졌지만, 커널 디바이스 드라이버들이 장치를 제어하는 핵심 API 소스를 점점 표준화하고 있어서 이것을 잘 파악하면 대부분의 드라이버 소스들을 이해할 수 있습니다.

디바이스 드라이버 요약 블럭도



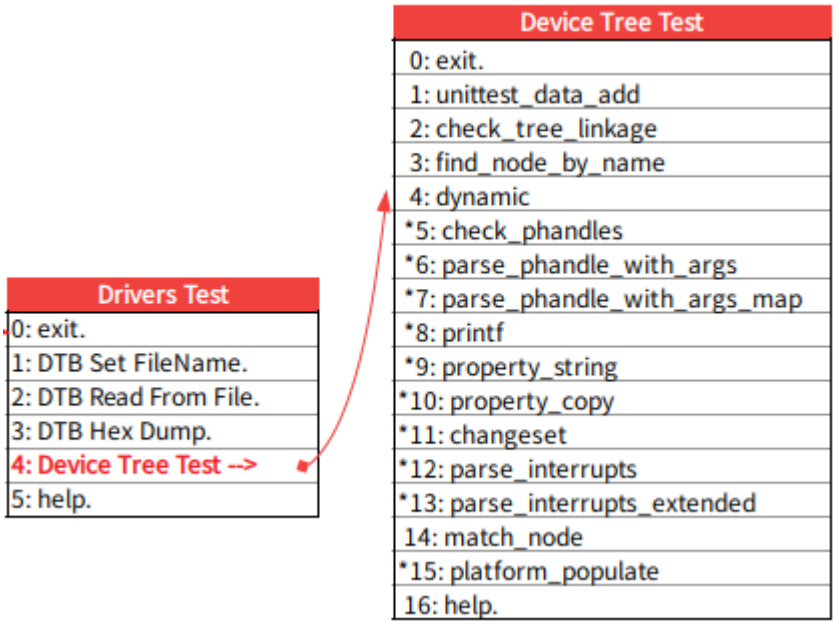
필자는 위와 같이 디바이스 드라이버 소스의 핵심에 해당하는 내용들을 블럭도로 정리 했습니다.

블럭도 내용을 보시면 다음과 같이 크게 3 가지 요소들로 요약할 수 있습니다.

- **디바이스 트리(장치 정의) 소스:** drivers/of 경로에 소스가 모여 있습니다.
- **드라이버 API 소스:** drivers/base 경로에 소스가 모여 있습니다.
- **드라이버 코어 소스:** drivers/base/core.c 에 소스가 모여 있습니다.

커널연구회의 "리눅스 커널 소스 실행 분석기"는 디바이스 드라이버 소스들중에서 장치들을 정의 하는 디바이스 트리 소스부터 이해할 수 있도록 다음과 같이 메뉴를 구성 했습니다.

드라이버 테스트 메뉴



콘솔에서는 다음과 같이 메뉴가 화면에 출력되며, 키보드에서 메뉴 번호를 입력하여 해당 기능을 실행합니다.

Drivers Test 메뉴

```
[#]--> Drivers Test Menu
0: exit.
1: DTB Set FileName.
2: DTB Read From File.
3: DTB Hex Dump.
4: Device Tree Test -->
5: help.
```

위의 메뉴를 번호별로 요약 설명하면 다음과 같습니다.

0: exit.

1: DTB Set FileName. //DTB 파일명을 설정 합니다.

2: DTB Read From File. //DTB 파일을 읽어서 메모리에 저장합니다.

3: DTB Hex Dump. //읽어온 DTB 파일을 Hexa 형태로 화면에 출력 합니다.

4: Device Tree Test --> //DTB 소스들을 테스트 합니다. (하위 메뉴 호출)

5: help.

소스가 실행되는 내용들은 함수가 호출되는 스택 깊이별로 함수 명칭이 출력되면서 그안의 데이터들을 확인할 수 있습니다. 소스가 실행되는 과정을 디버깅 메세지를 보면서 분석할 수 있습니다. 참고로 위의 메뉴 3 번에서 읽어온 DTB 내용은 다음과 같이 Hexa 로 덤프되어 출력 됩니다.

DTB Hex Dump 내용

```
Enter Menu Number[0,6]: 3
raw data: 00000000: d0 0d fe ed 00 00 1c d4 00 00 00 38 00 00 1a
00  ...........8....
```

```
raw data: 00000010: 00 00 00 28 00 00 00 11 00 00 00 10 00 00 00 00  ...
(............
raw data: 00000020: 00 00 02 d4 00 00 19 c8 00 00 00 00 00 00 00
00  ................
raw data: 00000030: 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00
00  ................
raw data: 00000040: 00 00 00 01 74 65 73 74 63 61 73 65 2d 64 61
74  ....testcase-dat
raw data: 00000050: 61 00 00 00 00 00 00 03 00 00 00 09 00 00 00 00
a...............
raw data: 00000060: 70 61 73 73 77 6f 72 64 00 00 00 00 00 00 00 03
password........
raw data: 00000070: 00 00 00 0a 00 00 00 12 64 75 70 6c 69 63 61
74  ........duplicat
raw data: 00000080: 65 00 00 00 00 00 00 03 00 00 00 04 00 00 00 21
e..............!
raw data: 00000090: 00 00 00 0b 00 00 00 01 63 68 61 6e 67 65 73
65  ........changese
raw data: 000000a0: 74 00 00 00 00 00 00 03 00 00 00 06 00 00 00 29
t..............)
raw data: 000000b0: 68 65 6c 6c 6f 00 00 00 00 00 00 03 00 00 00 06
hello...........
raw data: 000000c0: 00 00 00 35 77 6f 72 6c 64 00 00 00 00 00 00
01  ...5world.......
raw data: 000000d0: 6e 6f 64 65 2d 72 65 6d 6f 76 65 00 00 00 00 02  node-
remove.....
raw data: 000000e0: 00 00 00 02 00 00 00 01 64 75 70 6c 69 63 61
74  ........duplicat
raw data: 000000f0: 65 2d 6e 61 6d 65 00 00 00 00 00 02 00 00 00 01  e-
name..........
raw data: 00000100: 70 68 61 6e 64 6c 65 2d 74 65 73 74 73 00 00 00
phandle-tests...
raw data: 00000110: 00 00 00 01 70 72 6f 76 69 64 65 72 30 00 00
00  ....provider0...
raw data: 00000120: 00 00 00 03 00 00 00 04 00 00 00 41 00 00 00
00  ...........A....
raw data: 00000130: 00 00 00 03 00 00 00 04 00 00 00 21 00 00 00
02  ...........!....
raw data: 00000140: 00 00 00 02 00 00 00 01 70 72 6f 76 69 64 65
72  ........provider
raw data: 00000150: 31 00 00 00 00 00 00 03 00 00 00 04 00 00 00 41
1.............A
raw data: 00000160: 00 00 00 01 00 00 00 03 00 00 00 04 00 00 00
21  ...............!
```

```
raw data: 00000170: 00 00 00 01 00 00 00 02 00 00 00 01 70 72 6f
76  ............prov
raw data: 00000180: 69 64 65 72 32 00 00 00 00 00 00 03 00 00 00 04
ider2...........
raw data: 00000190: 00 00 00 41 00 00 00 02 00 00 00 03 00 00 00
04  ...A............
raw data: 000001a0: 00 00 00 21 00 00 00 04 00 00 00 02 00 00 00
01  ...!............
raw data: 000001b0: 70 72 6f 76 69 64 65 72 33 00 00 00 00 00 00 03
provider3.......
raw data: 000001c0: 00 00 00 04 00 00 00 41 00 00 00 03 00 00 00
03  .......A........
raw data: 000001d0: 00 00 00 04 00 00 00 21 00 00 00 03 00 00 00
02  .......!........
raw data: 000001e0: 00 00 00 01 70 72 6f 76 69 64 65 72 34 00 00
00  ....provider4...


//이하 생략...
```

Device Tree Test 메뉴

DTB 내용을 device_node 구조체에 파싱하고 장치 노드별로 내용들을 검색하고, 속성값들을 테스트하는 메뉴로 다음과 같이 구성되어 있습니다. **(참고로 아래 메뉴는 메뉴번호 순서대로 실행해야 합니다.)**

Device Tree Test
0: exit.
1: unittest_data_add
2: check_tree_linkage
3: find_node_by_name
4: dynamic
*5: check_phandles
*6: parse_phandle_with_args
*7: parse_phandle_with_args_map
*8: printf
*9: property_string
*10: property_copy
*11: changeset
*12: parse_interrupts
*13: parse_interrupts_extended
14: match_node
*15: platform_populate
16: help.

콘솔에서는 다음과 같이 메뉴가 화면에 출력되며, 키보드에서 메뉴 번호를 입력하여 해당 기능을 실행합니다.

```
[#]--> Device Tree --> UnitTest Menu
  0: exit.
  1: unittest_data_add
  2: check_tree_linkage
  3: find_node_by_name
  4: dynamic
```

```
 *5: check_phandles
 *6: parse_phandle_with_args
 *7: parse_phandle_with_args_map
 *8: printf
 *9: property_string
*10: property_copy
*11: changeset
*12: parse_interrupts
*13: parse_interrupts_extended
 14: match_node
*15: platform_populate
 16: help.
```

위의 메뉴를 번호별로 요약 설명하면 다음과 같습니다.

0: exit.

1: unittest_data_add //DTB 내용을 device_node 구조체에 파싱하여 적재 합니다.

2: check_tree_linkage //device_node 연결 정보를 점검 합니다.

3: find_node_by_name //장치 node 을 이름으로 검색합니다.

4: dynamic //장치 node 에 속성값을 추가, 수정, 조회 작업을 다이나믹 테스트 합니다.

위의 메뉴별로 소스가 실행되는 내용들은 함수가 호출되는 스택 깊이별로 함수 명칭이 출력되면서 그안의 데이터들도 같이 확인할 수 있습니다. 소스가 실행되는 과정을 디버깅 메세지를 보면서 분석할 수 있습니다. 지면 관계상 실행 내용들은 생략 하도록 하겠습니다.

참고로, 나머지 메뉴 번호에 있는 기능들은 미완성 단계여서 앞으로 기능을 계속 보완해 나가도록 하겠습니다.