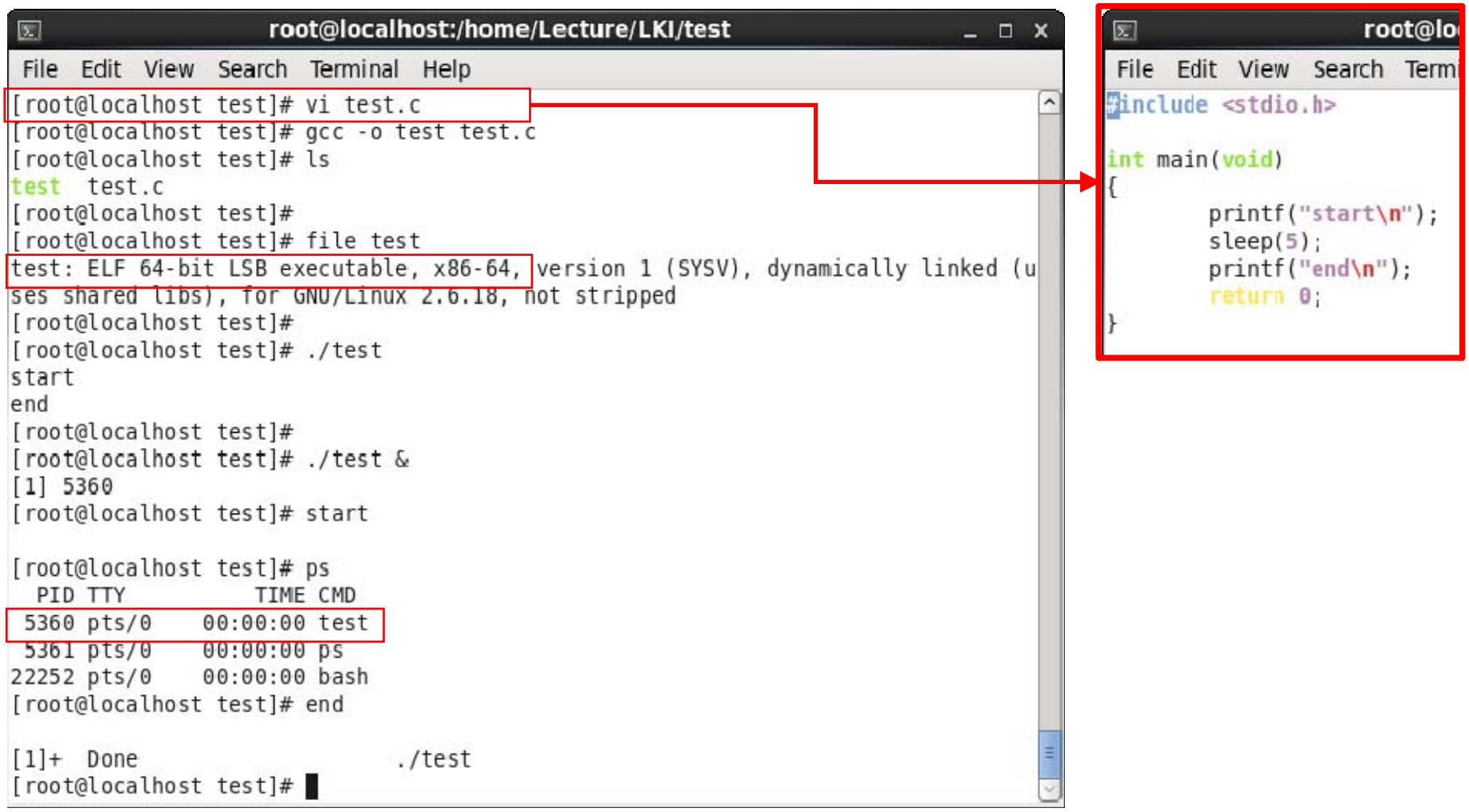
Task Management

March, 2016

Seungjae Baek

Dept. of software
Dankook University
http://embedded.dankook.ac.kr/~baeksj

Program & Process



Task, Process and Thread

Process

Thread

Task, Process and Thread

Process

- 실행 상태에 있는 프로그램의 **instance**
- 자원 소유권의 단위
- **...**

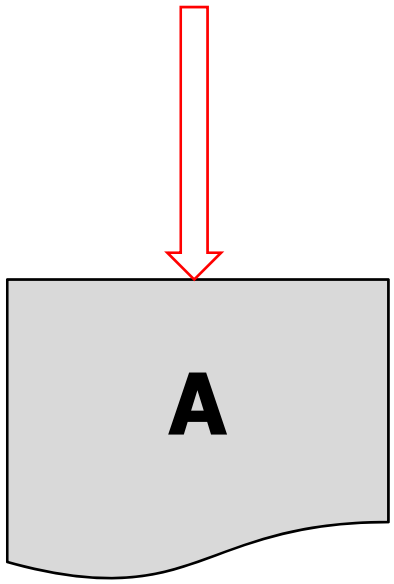
Thread

- 디스패칭의 단위
- 실행 흐름
- 수행의 단위
- **...**

Task, Process and Thread

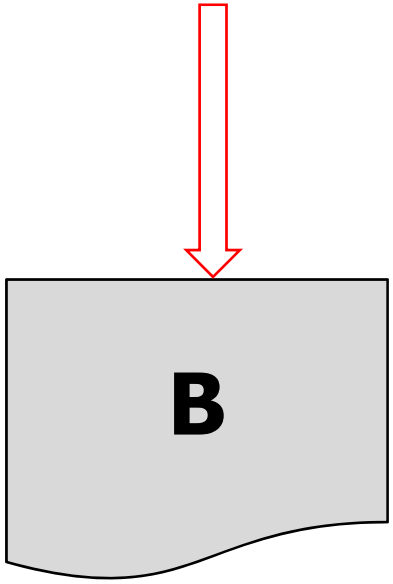
Process

- 실행 상태에 있는 프로그램의 **instance**
- 자원 소유권의 단위
- **...**

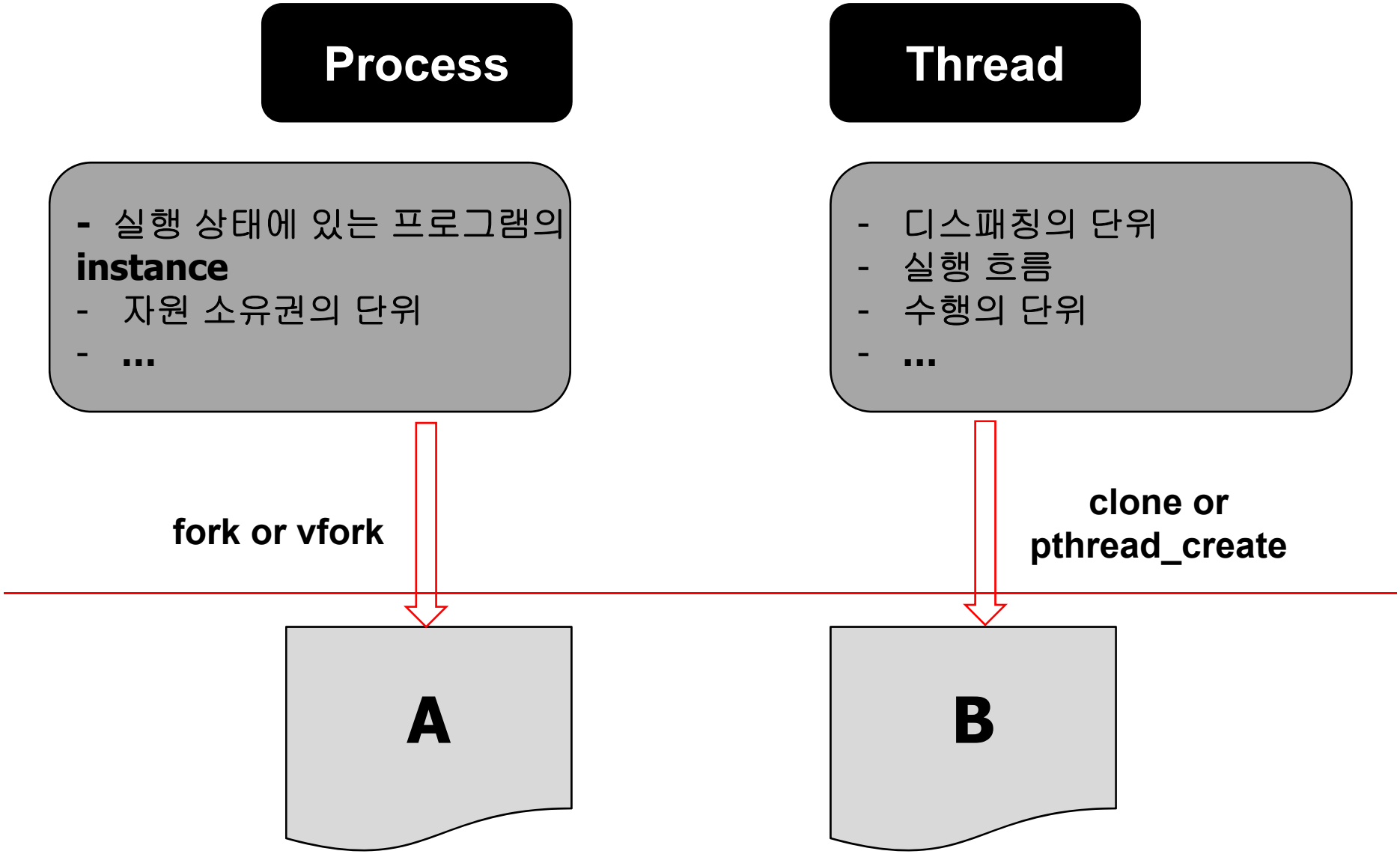


Thread

- 디스패칭의 단위
- 실행 흐름
- 수행의 단위
- **...**



Task, Process and Thread



Task, Process and Thread

Task, Process and Thread

1. PID

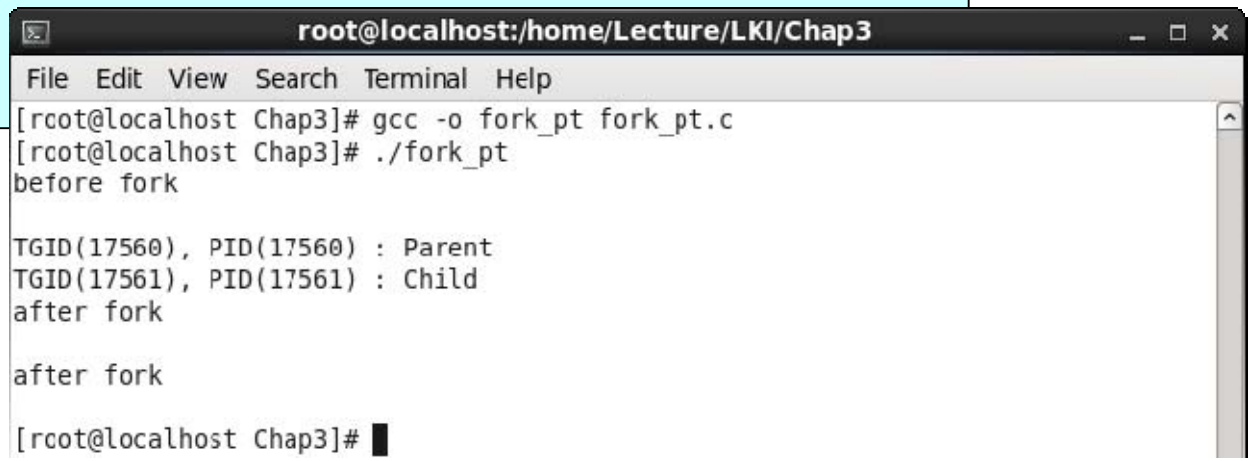
Task, Process and Thread

1. PID

2. POSIX: 한 **process**내의 **thread**는 동일한 **PID**를 공유해야 한다

Task example - fork

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <linux/unistd.h>

int main(void)
{
        int pid;

        printf("before fork₩n₩n");
        if((pid = fork()) < 0){
                printf("fork error₩n");
                exit(-2);
        }else if (pid == 0){
                printf("TGID(%d), PID(%d): Child₩n", getpid(), syscall(__NR_gettid));
        }else{
                printf("TGID(%d), PID(%d): Parent₩n", getpid(), syscall(__NR_gettid));
                sleep(2);
        }
        printf("after fork ₩n₩n");
        return 0;
}
```



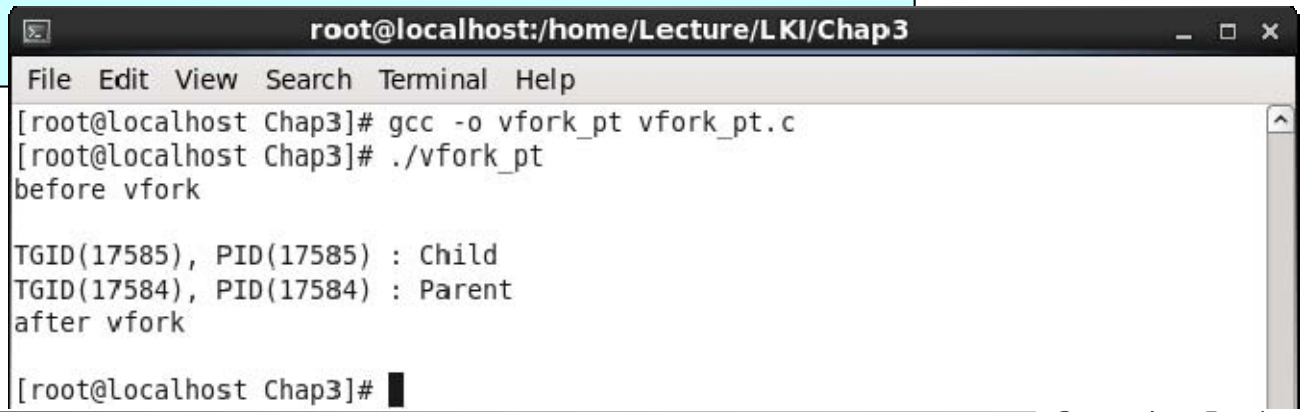
Task example - pthread


```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <linux/unistd.h>
void *t_function(void *data)
{
        int id;         int i=0;        pthread_t t_id;
        id = *((int *)data);
        printf("TGID(%d), PID(%d), pthread_self(%d): Child ₩n", getpid(), syscall(__NR_gettid), pthread_self());
        sleep(2);
}
int main()
{
        pthread_t p_thread[2];
        int thr_id;     int status;
        int a = 1;      int b = 2;
        printf("before pthread_create()₩n");
        if((thr_id = pthread_create(&p_thread[0], NULL, t_function, (void*)&a)) < 0){
                perror("thread create error : ");                       exit(1);
        }
        if((thr_id = pthread_create(&p_thread[1], NULL, t_function, (void*)&b)) < 0){
                perror("thread create error : ");                       exit(2);
        }
        pthread_join(p_thread[0], (void **)&status);
        printf("thread join : %d₩n", status);
        pthread_join(p_thread[1], (void **)&status);
        printf("thread join : %d₩n", status);

        printf("TGID(%d), PID(%d): Parent ₩n",
                getpid(), syscall(__NR_gettid));
        return 0;
}
```

```
root@localhost:/home/Lecture/LKI/Chap3
File  Edit  View  Search  Terminal  Help
[root@localhost Chap3]# gcc -o pthread_pt pthread_pt.c -lpthread
[root@localhost Chap3]# ./pthread_pt
before pthread_create

TGID(17672), PID(17673), pthread_self(355116224) : Child
TGID(17672), PID(17674), pthread_self(347723520) : Child
pthread_join(0)
pthread_join(0)
TGID(17672), PID(17672) : Parent
[root@localhost Chap3]#
```

Task example - vfork


```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <linux/unistd.h>

int main(void)
{
        pid_t pid;

        printf("before vfork₩n");

        if((pid = vfork()) < 0){
                printf("fork error₩n");
                exit(-2);
        }else if (pid == 0){
                printf("TGID(%d), PID(%d): Child ₩n", getpid(), syscall(__NR_gettid));
                _exit(0);
        }else{
                printf("TGID(%d), PID(%d): Parent ₩n", getpid(), syscall(__NR_gettid));
        }
        printf("after vfork ₩n ₩n");
        exit(0);
}
```



Task example - clone

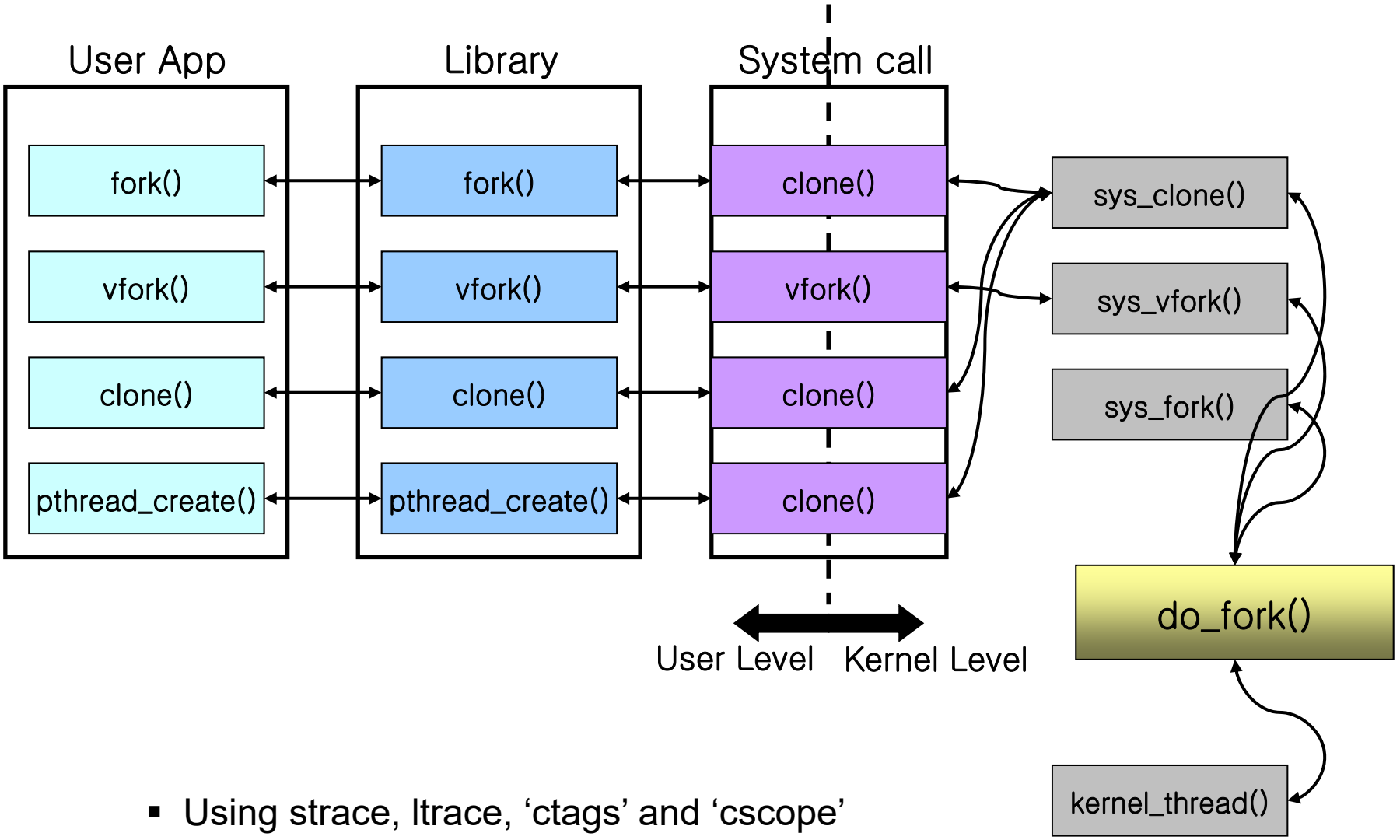

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <linux/unistd.h>
#include <sched.h>
int sub_a(void *arg)
{
        printf("TGID(%d), PID(%d): Child ₩n", getpid(), syscall(__NR_gettid));
        sleep(2);
        return 0;
}
int main(void)
{
        int child_a_stack[4096], child_b_stack[4096];

        printf("before clone₩n");
        printf("TGID(%d), PID(%d)₩n", getpid(), syscall(__NR_gettid));

        clone (sub_a, (void *)(child_a_stack+4095), CLONE_CHILD_CLEARTID | CLONE_CHILD_SETTID, NULL);
        clone (sub_a, (void *)(child_b_stack+4095), CLONE_CHILD_CLEARTID | CLONE_CHILD_SETTID, NULL);
        sleep(1);
        printf("after clone ₩n ₩n");
        return 0;
}
```

CLONE_THREAD option 추가 시 같은 tgid를 가짐

```
root@localhost:/home/Lecture/LKI/Chap3
File  Edit  View  Search  Terminal  Help
[root@localhost Chap3]# gcc -o clone_pt clone_pt.c
[root@localhost Chap3]# ./clone_pt
before clone

TGID(17750), PID(17750) : Parent
TGID(17750), PID(17752) : Child
TGID(17751), PID(17751) : Child
after clone

[root@localhost Chap3]#
```

Task implementation in Linux

- Flow controls



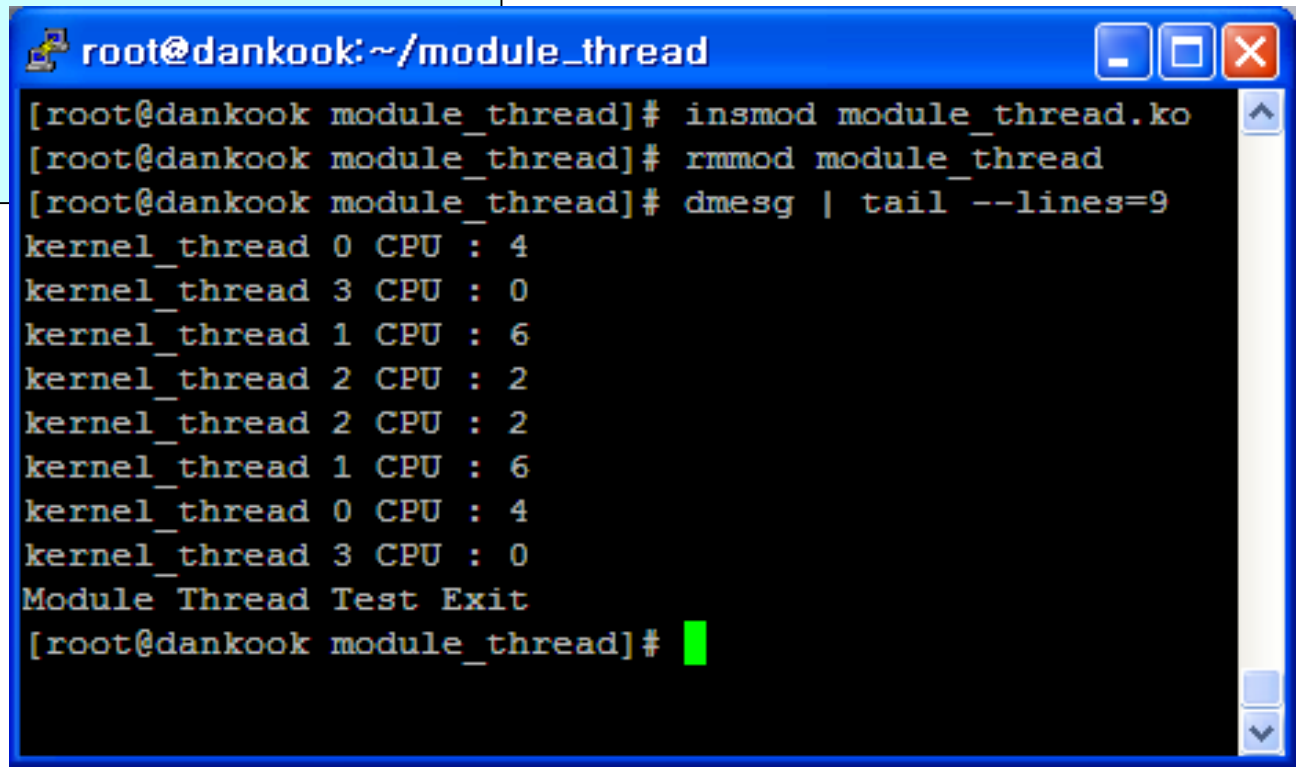
- Using strace, ltrace, ‘ctags’ and ‘cscope’

Task example – kernel thread


```
int __init module_thread_init(void);
void __exit module_thread_exit(void);

void test_kernel_thread(void *arg){
        int val, i, j;
        val = (int)(long)arg;
        printk("<0>kernel_thread %d CPU : %d₩n", val, smp_processor_id());
        for(j=0 ; j < 100000000 ; j++);
                for(i=0 ; i < 100000000 ; i++);
        printk("<0>kernel_thread %d CPU : %d₩n", val, smp_processor_id());
}

int __init module_thread_init(){

        int i;
        for(i = 0 ; i < 4 ; i++)
                kernel_thread((int (*)(void *))test_kernel_thread, (void *)(long)i, 0);
        return 0;
}

void __exit module_thread_exit(){
        printk("<0>Module Thread Test Exit ₩n");
}

module_init(module_thread_init);
module_exit(module_thread_exit);
```

Task example – kernel thread

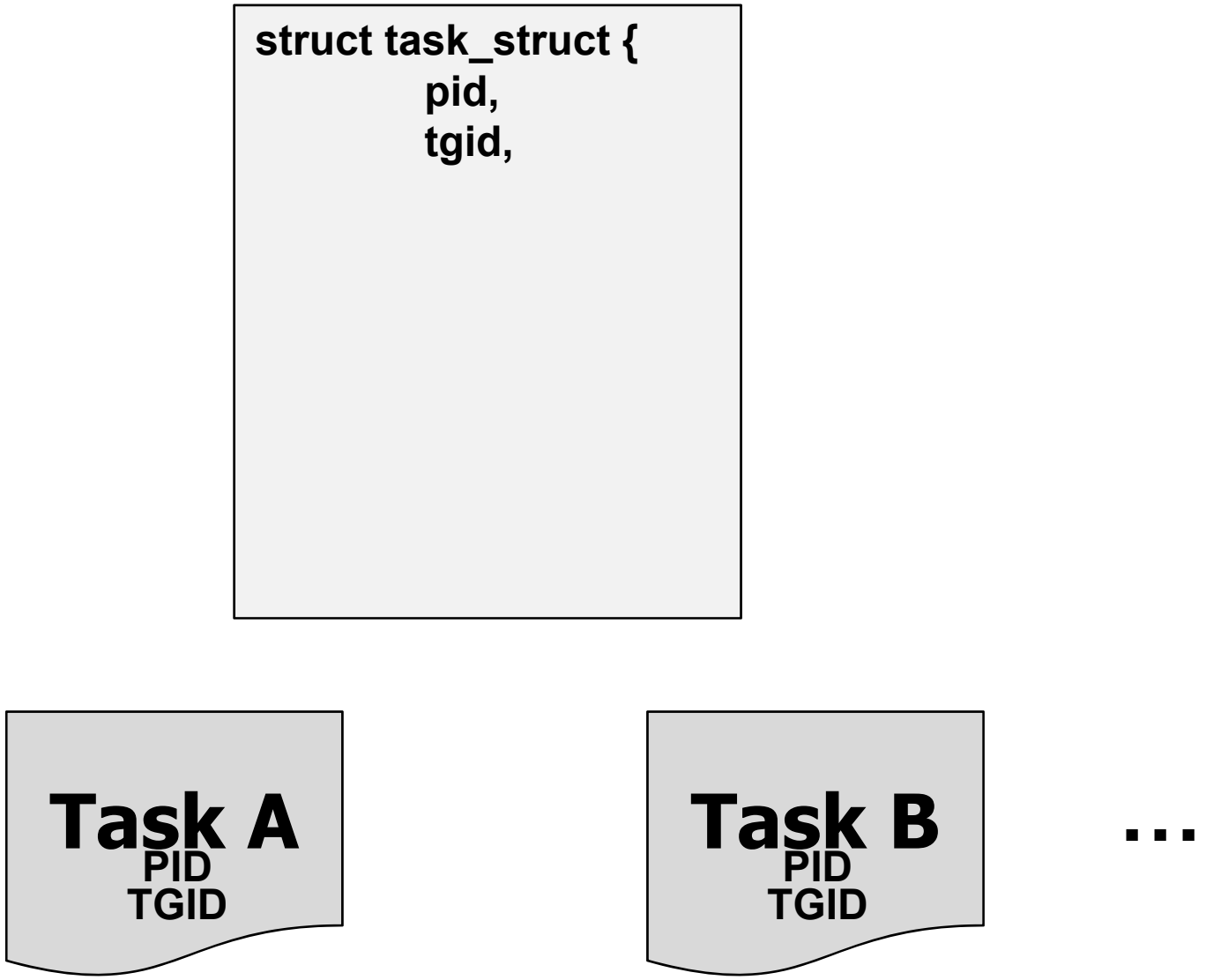

```
obj-m       := module_thread.o

KDIR        := /lib/modules/$(shell uname -r)/build
PWD         := $(shell pwd)

default:
            $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules

clean:
            rm -rf *.ko
            rm -rf *.mod.*
            rm -rf .*.cmd
            rm -rf *.o
```



Task, Process and Thread

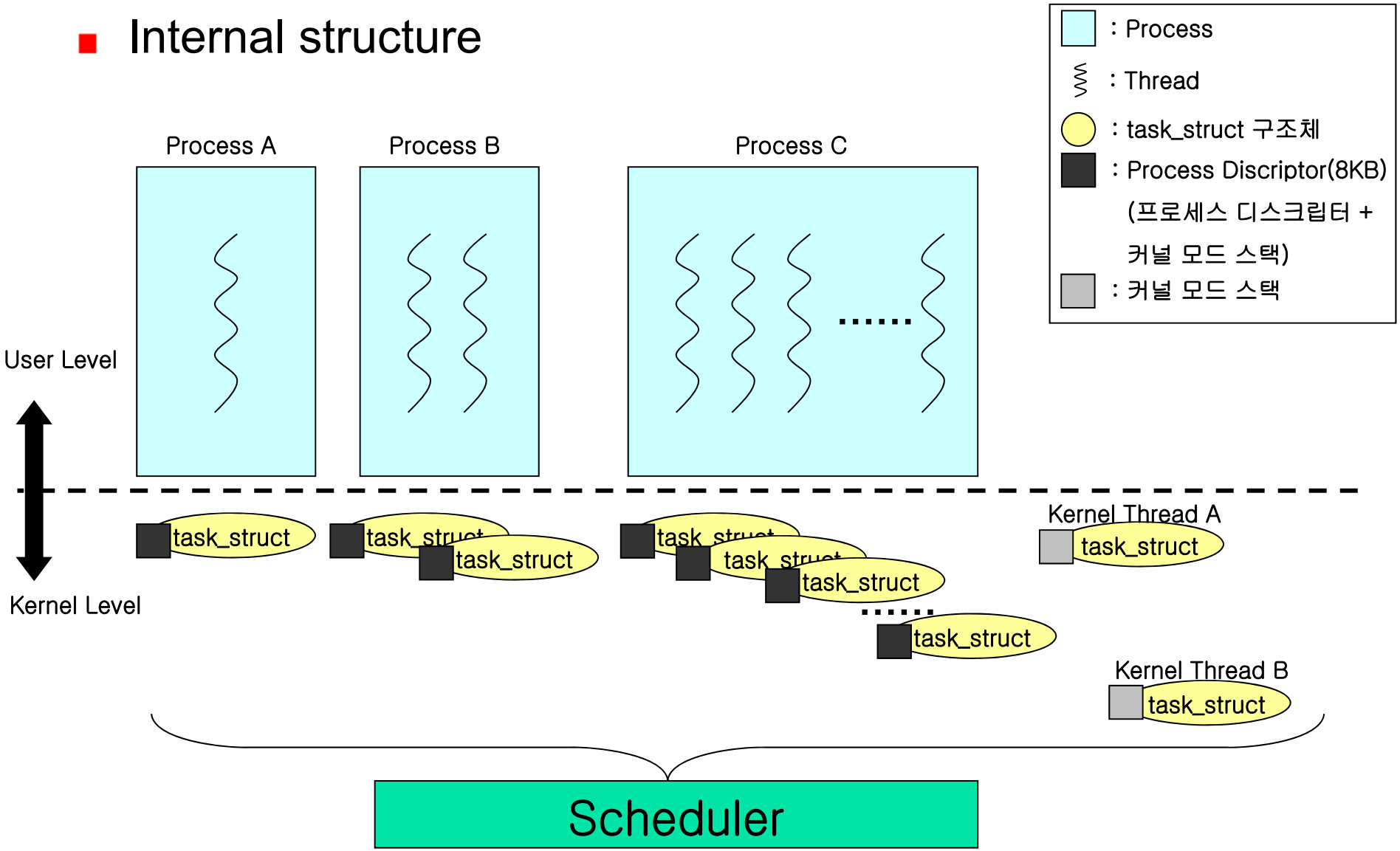
1. **PID**
2. **POSIX:** 한 **process**내의 **thread**는 동일한 **PID**를 공유해야 한다

Task, Process and Thread



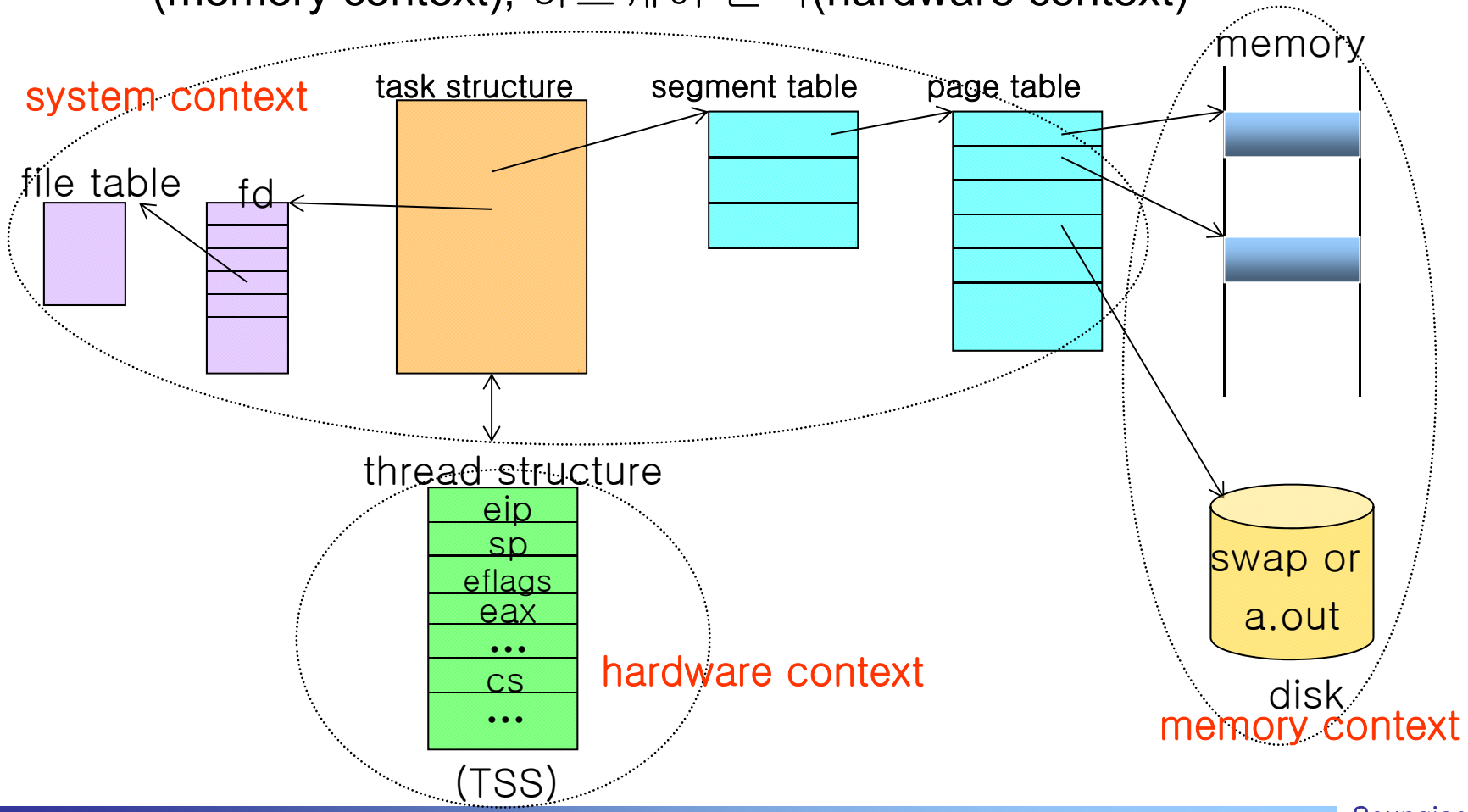
Linux Thread Model

- Internal structure



Context

- 문맥 (Context) :
  - ✓ 커널이 관리하는 태스크의 자원과 수행 환경 집합
  - ✓ 3 부분으로 구성 :시스템 문맥 (system context), 메모리 문맥 (memory context), 하드웨어 문맥(hardware context)



System Context

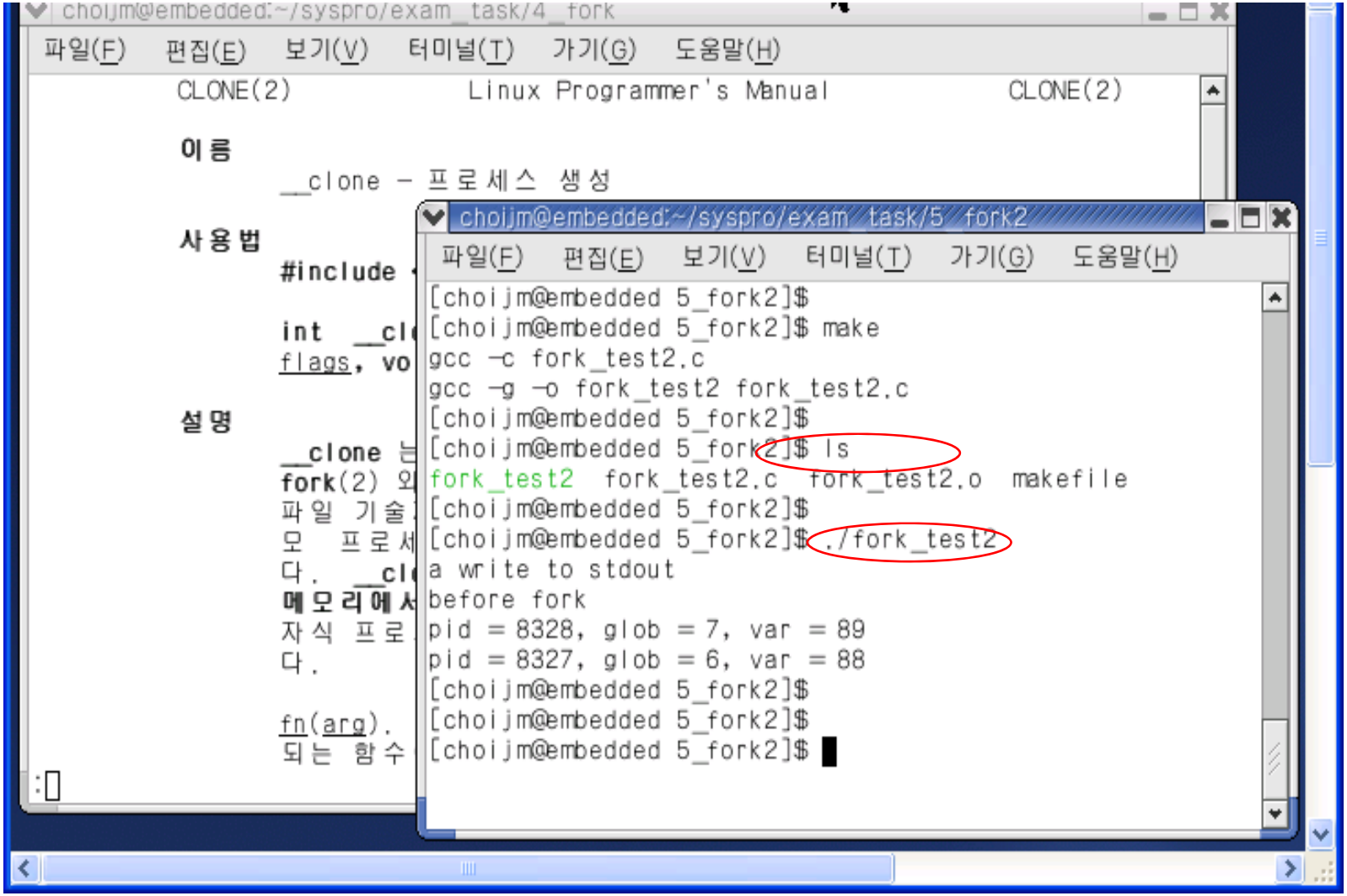
- 태스크를 관리하는 정보 집합
  - ✓ 태스크 정보: pid, uid, euid, suid, …
  - ✓ 태스크 상태: 실행 상태, READY 상태, 수면 상태, …
  - ✓ 태스크의 가족 관계: p_pptr, p_cptr, next_task, next_run
  - ✓ 스케줄링 정보: policy, priority, counter, rt_priority, need_resched
  - ✓ 태스크가 접근한 파일 정보: file descriptor
  - ✓ 시그널 정보
  - ✓ 그 외: 수행 시간 정보, 수행 파일 이름, 등등 (kernel dependent)

태스크 생성과 전이(커널 수준으로 진입/수면)의 예

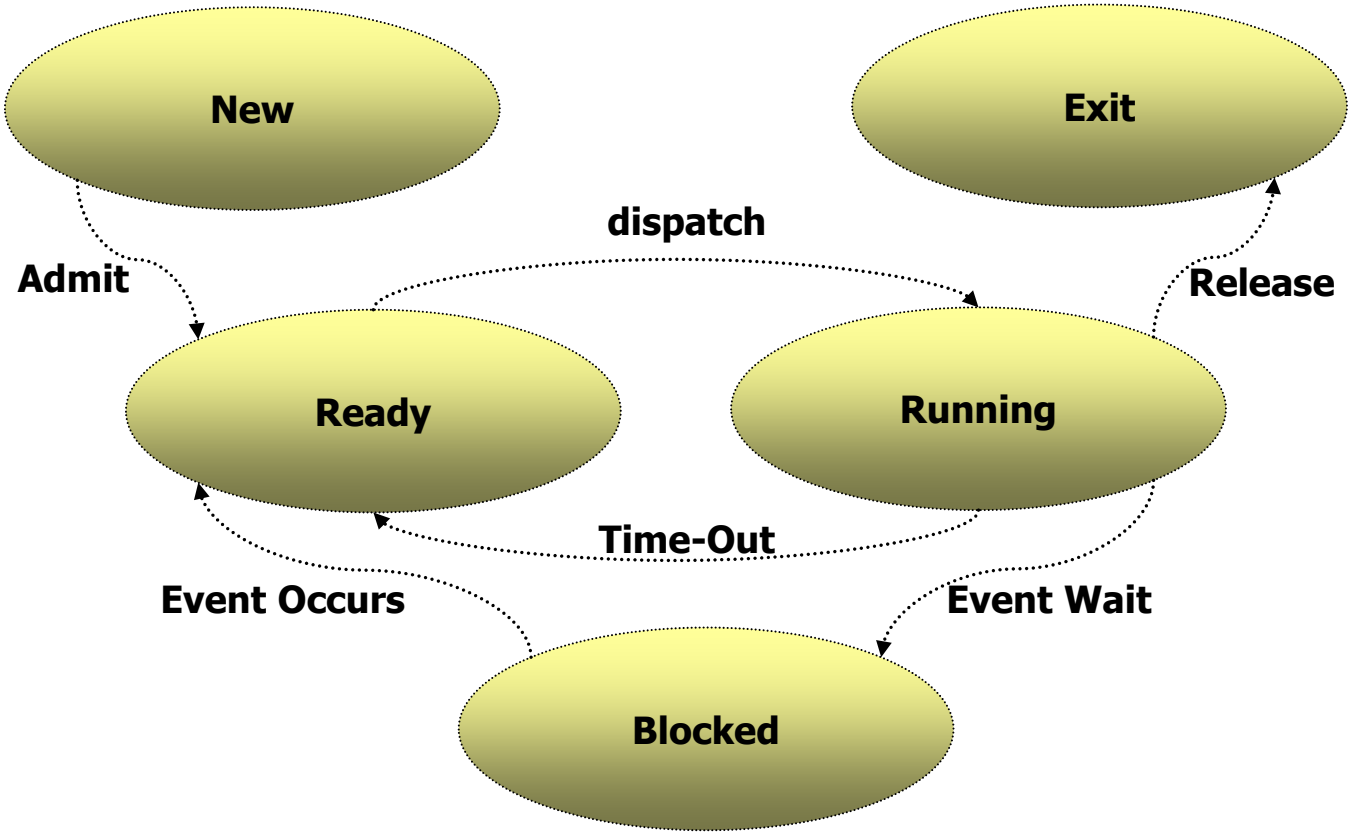

```
/* test.c */

int         glob = 6;
char        buf[] = "a write to stdout₩n";

int main(void)
{
            int  var;
            pid_t pid;

            var = 88;
            write(STDOUT_FILENO, buf, sizeof(buf)-1);
            printf("before fork₩n");

            if ((pid = fork()) == 0) {          /* child */
                        glob++; var++;
            } else
                        sleep(2);               /* parent */

            printf("pid = %d, glob = %d, var = %d₩n", getpid(), glob, var);
            exit (0);
}
```

(Source : Adv. programming in the UNIX Env., pgm 8.1)

Task 생성과 주소공간 예제 결과



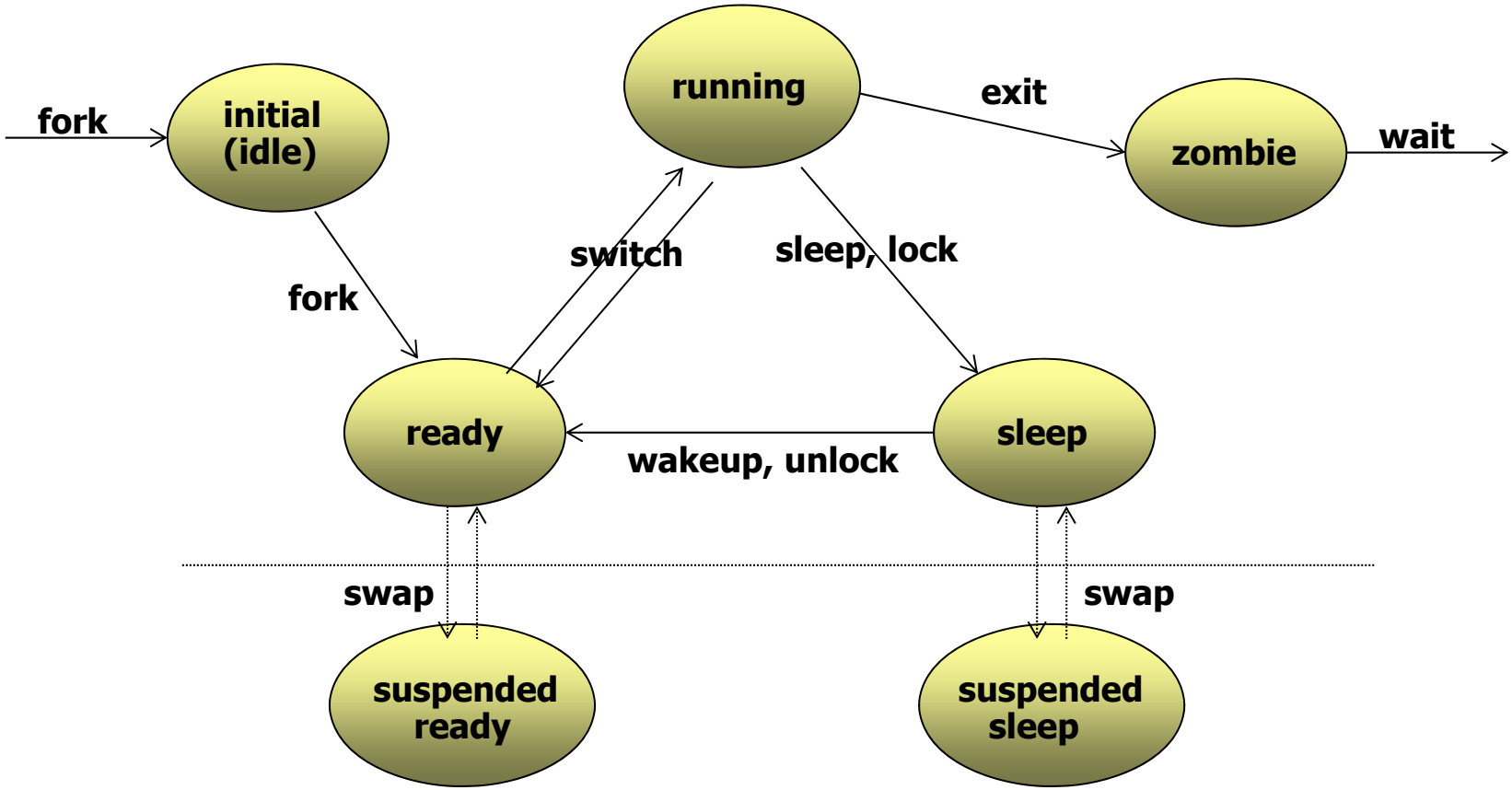
Process State Diagram(1/2)



☞ **Five-State Process Model**

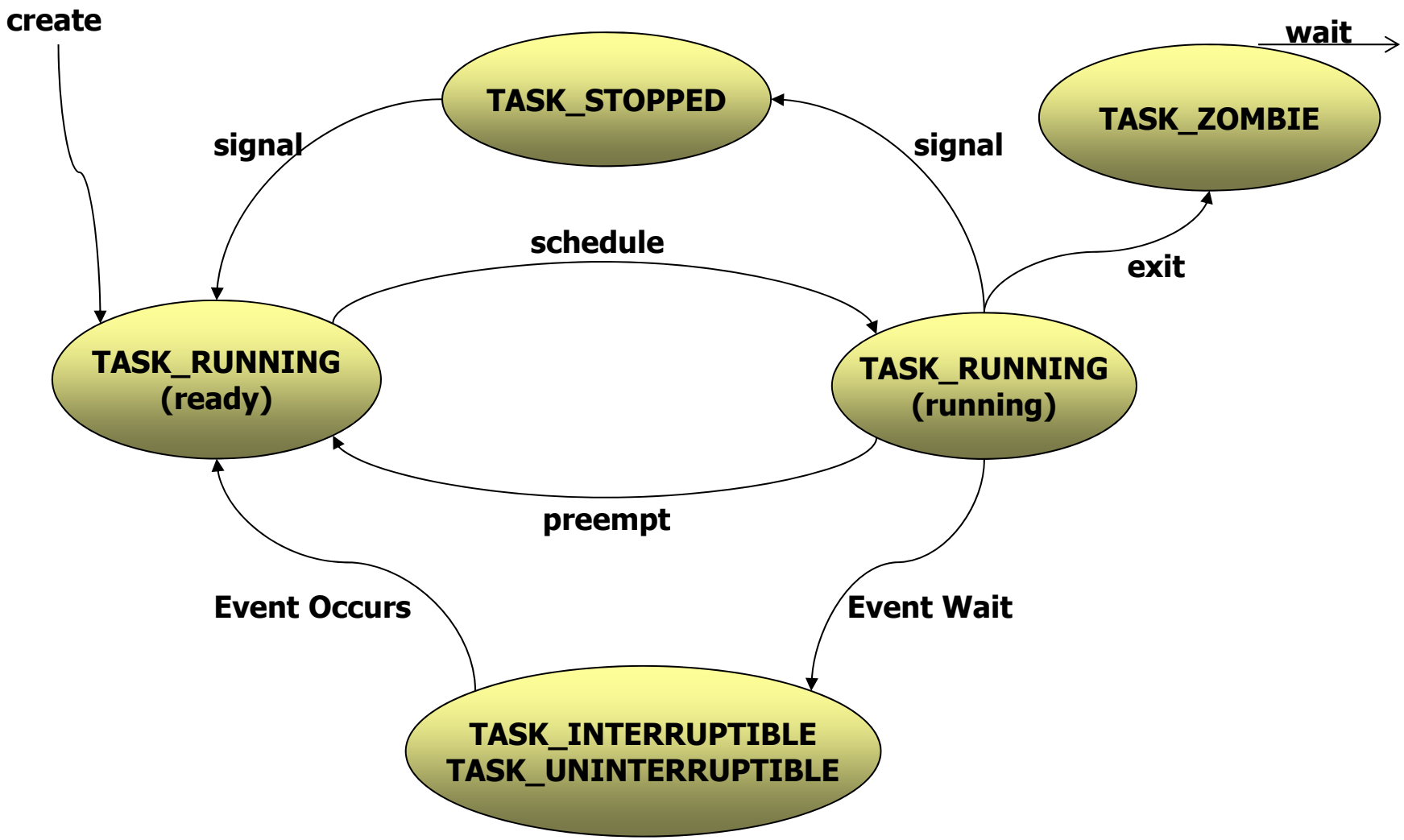
(Source : OS(stallings))

Process State Diagram(2/2)

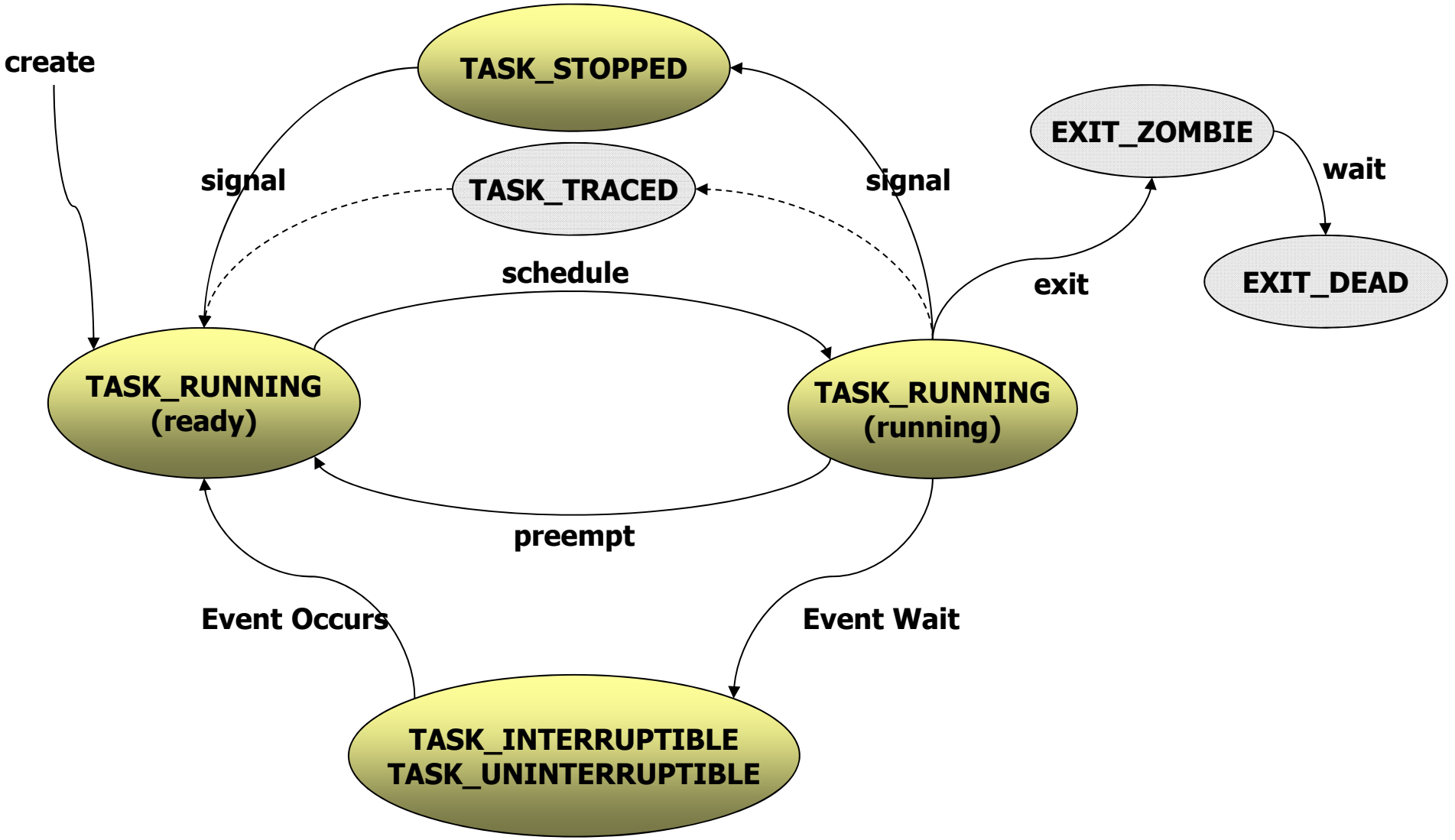


(Source : UNIX Internals)

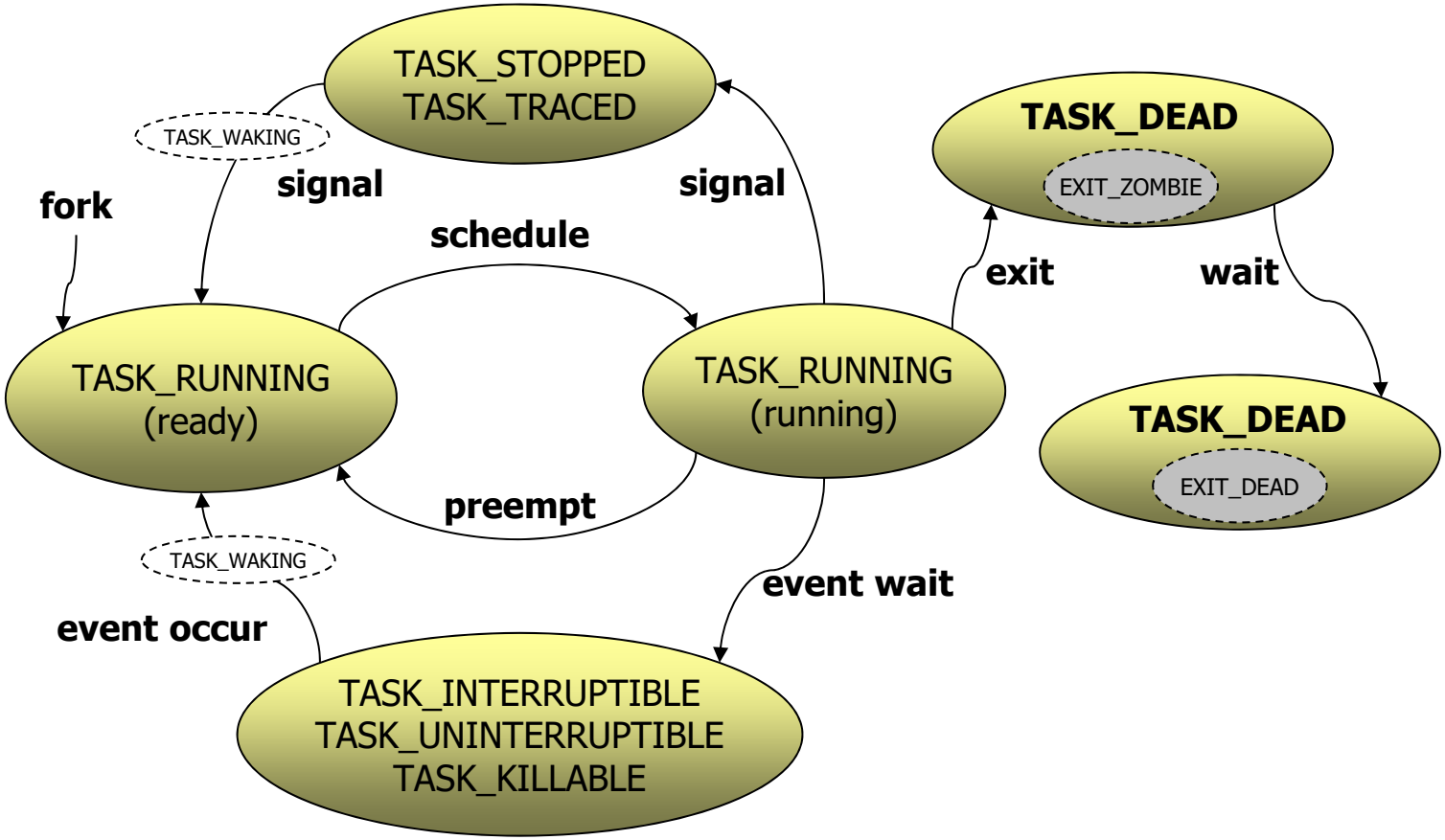
Task State Diagram of LINUX(2.4)



Task State Diagram of LINUX(2.6)

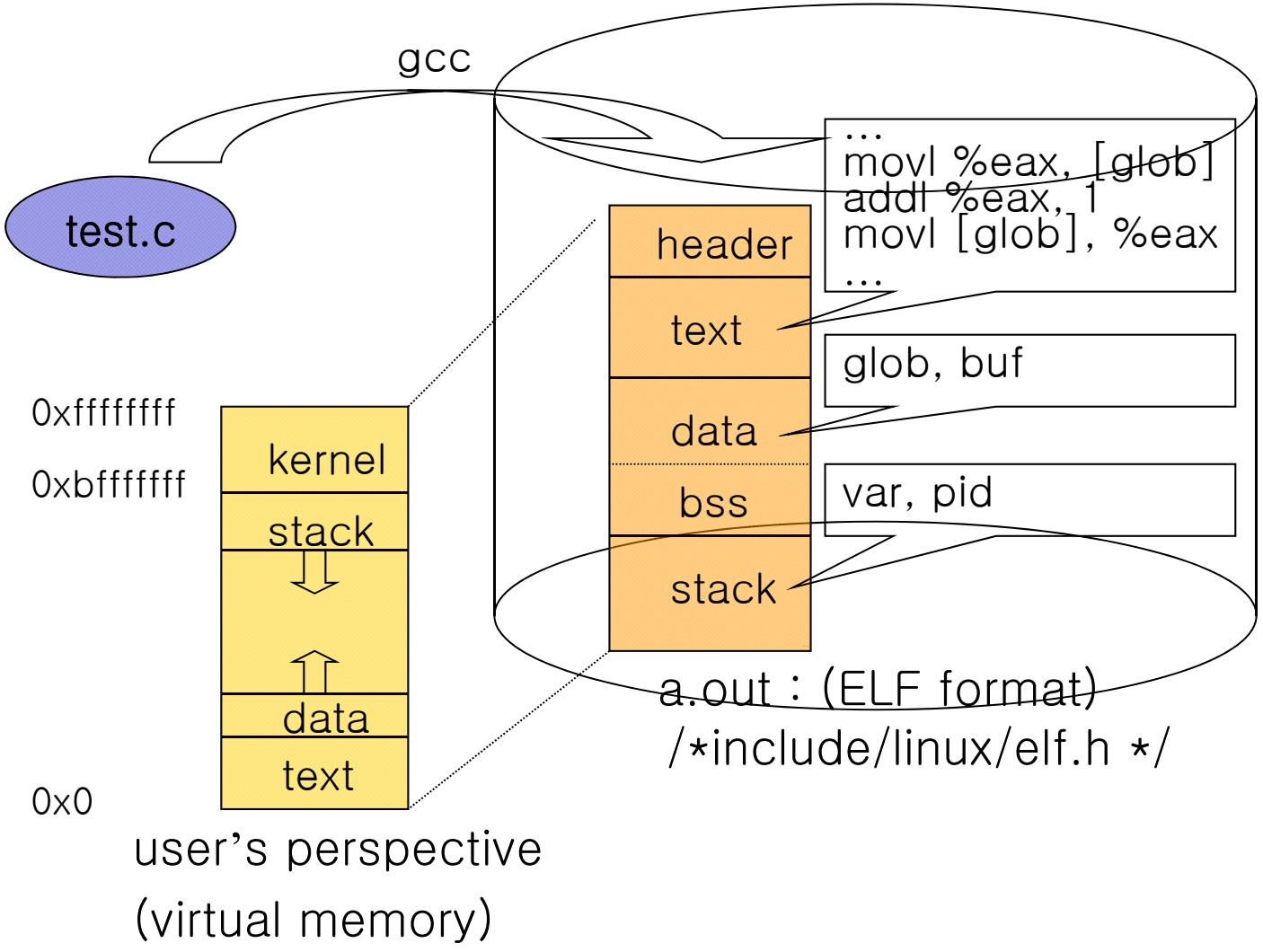


Task State Diagram of LINUX(3.18)



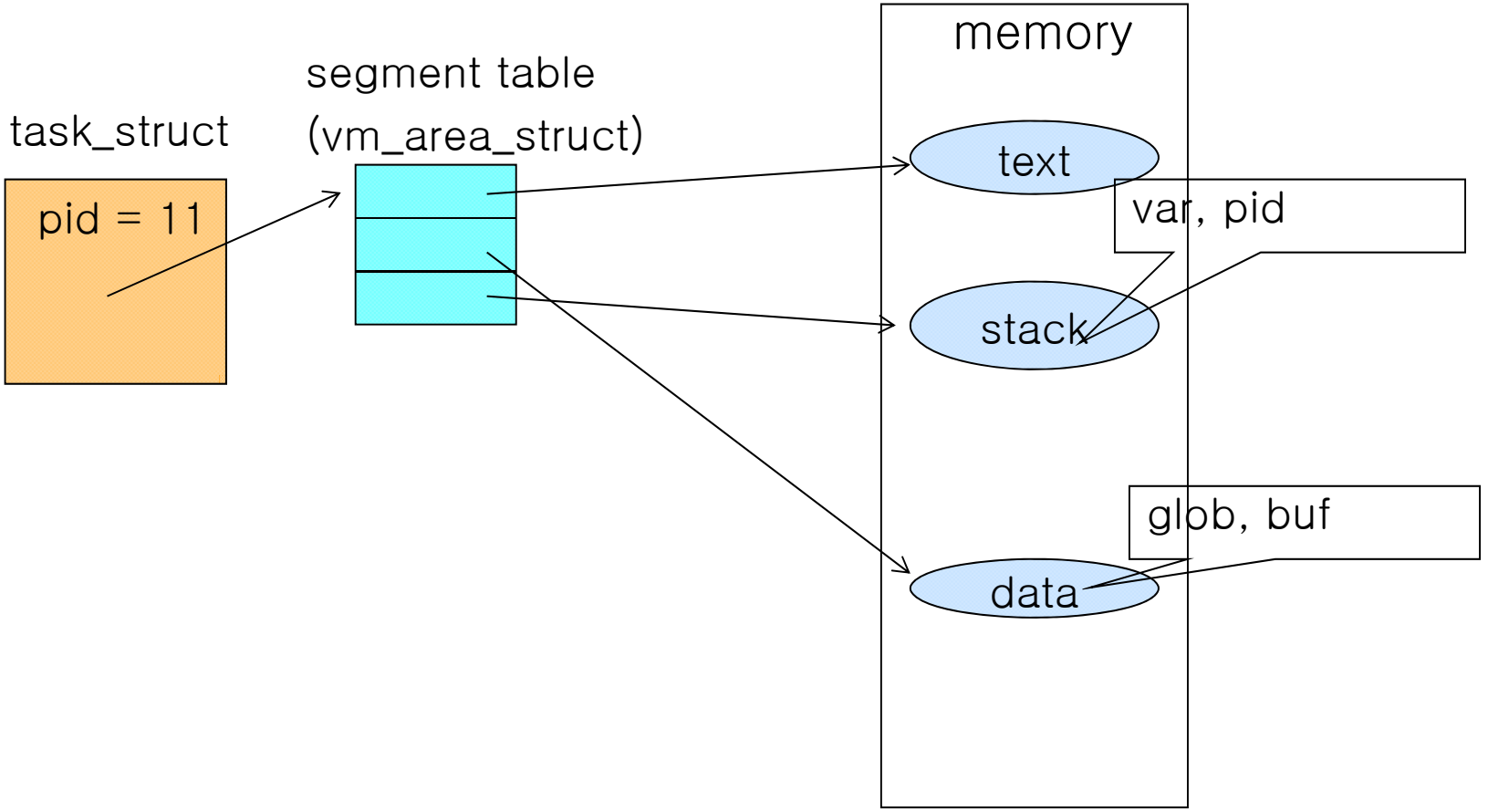
Memory Context (1/5)

- fork internal : compile results



Memory Context (2/5)

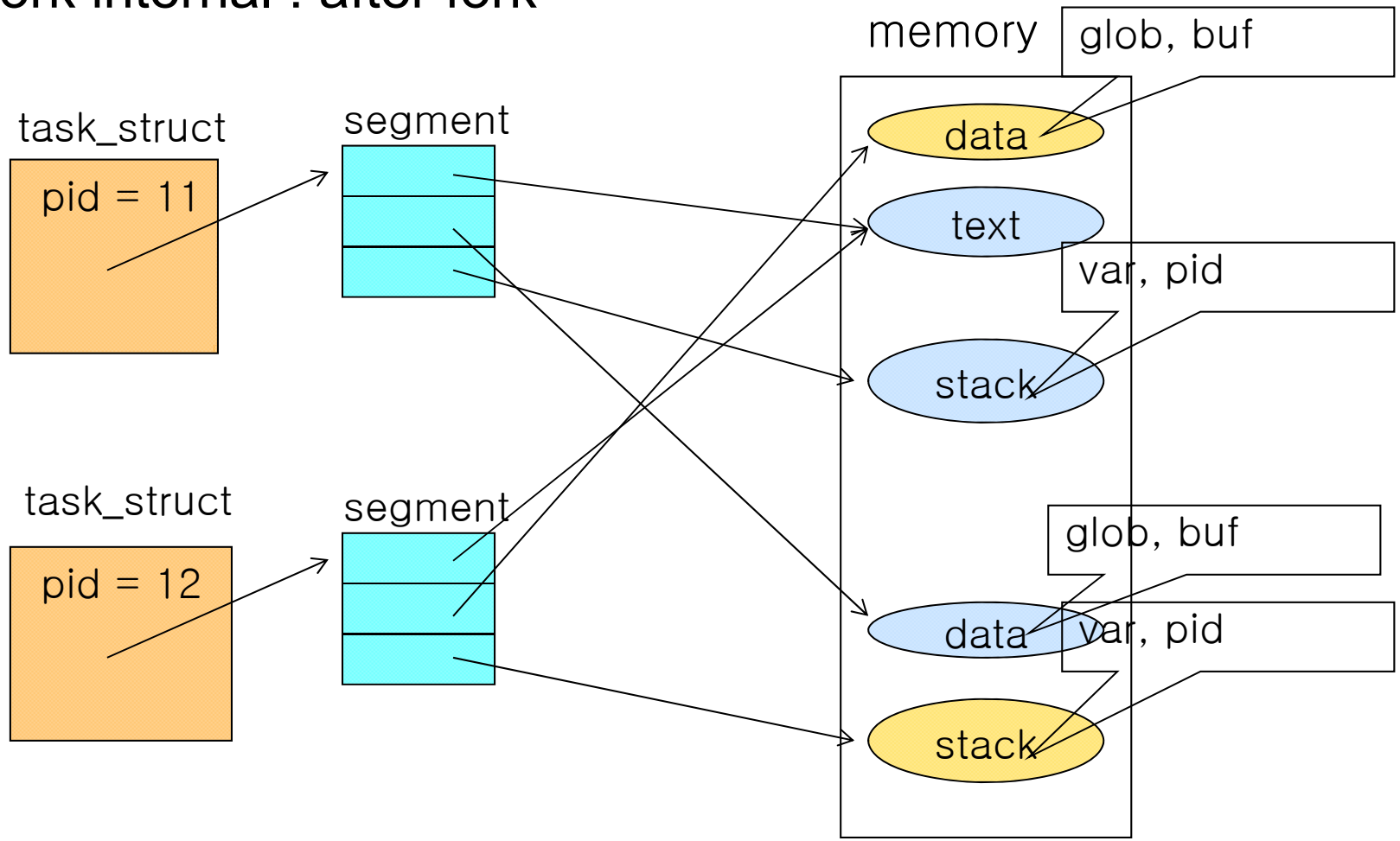
- fork internal : after loading (after run a.out) & before fork



✓ In this figure, we assume that there is no paging mechanism.

Memory Context (3/5)

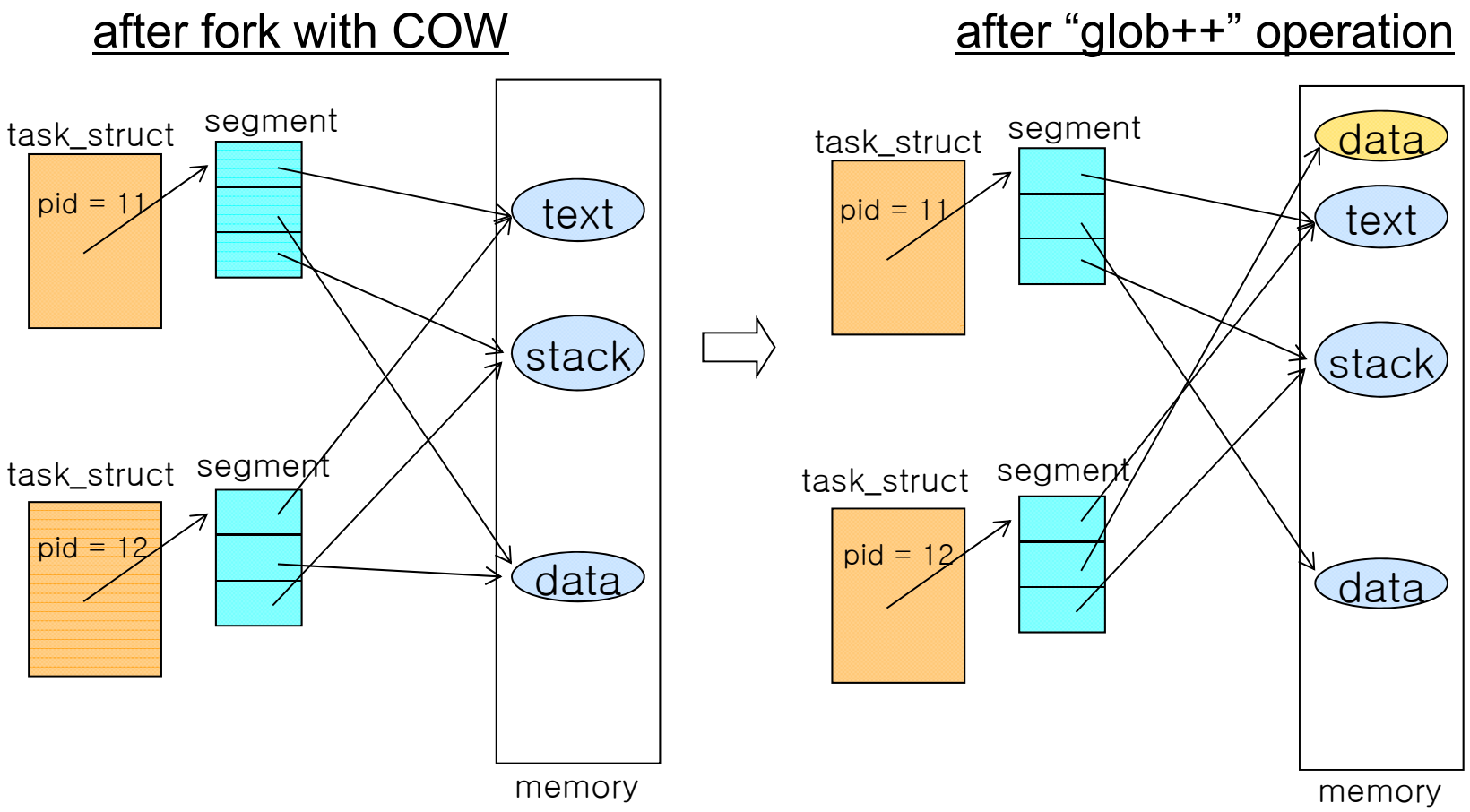
- fork internal : after fork



✓ address space : basic protection barrier

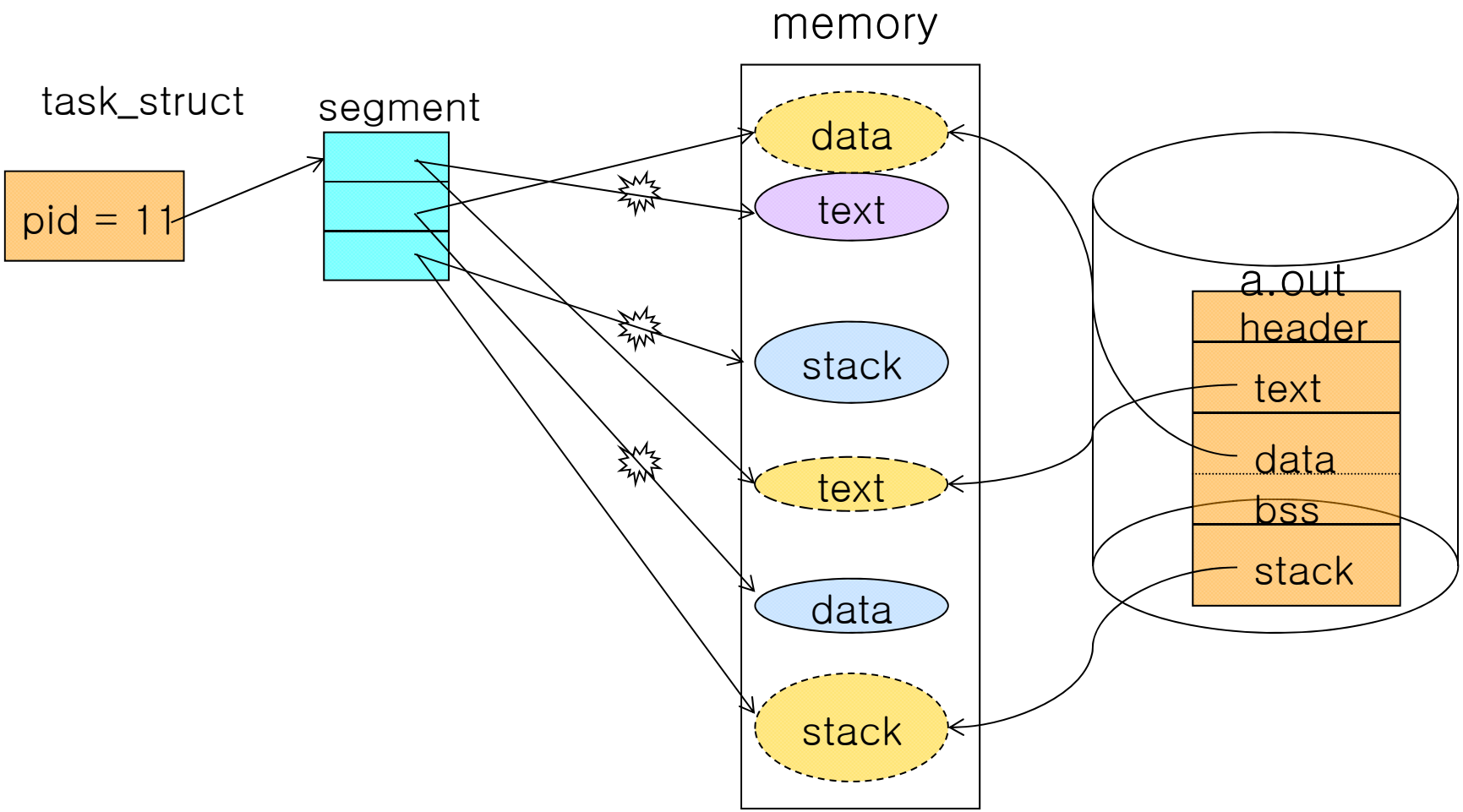
Memory Context (4/5)

- fork internal : with COW (Copy on Write) mechanism



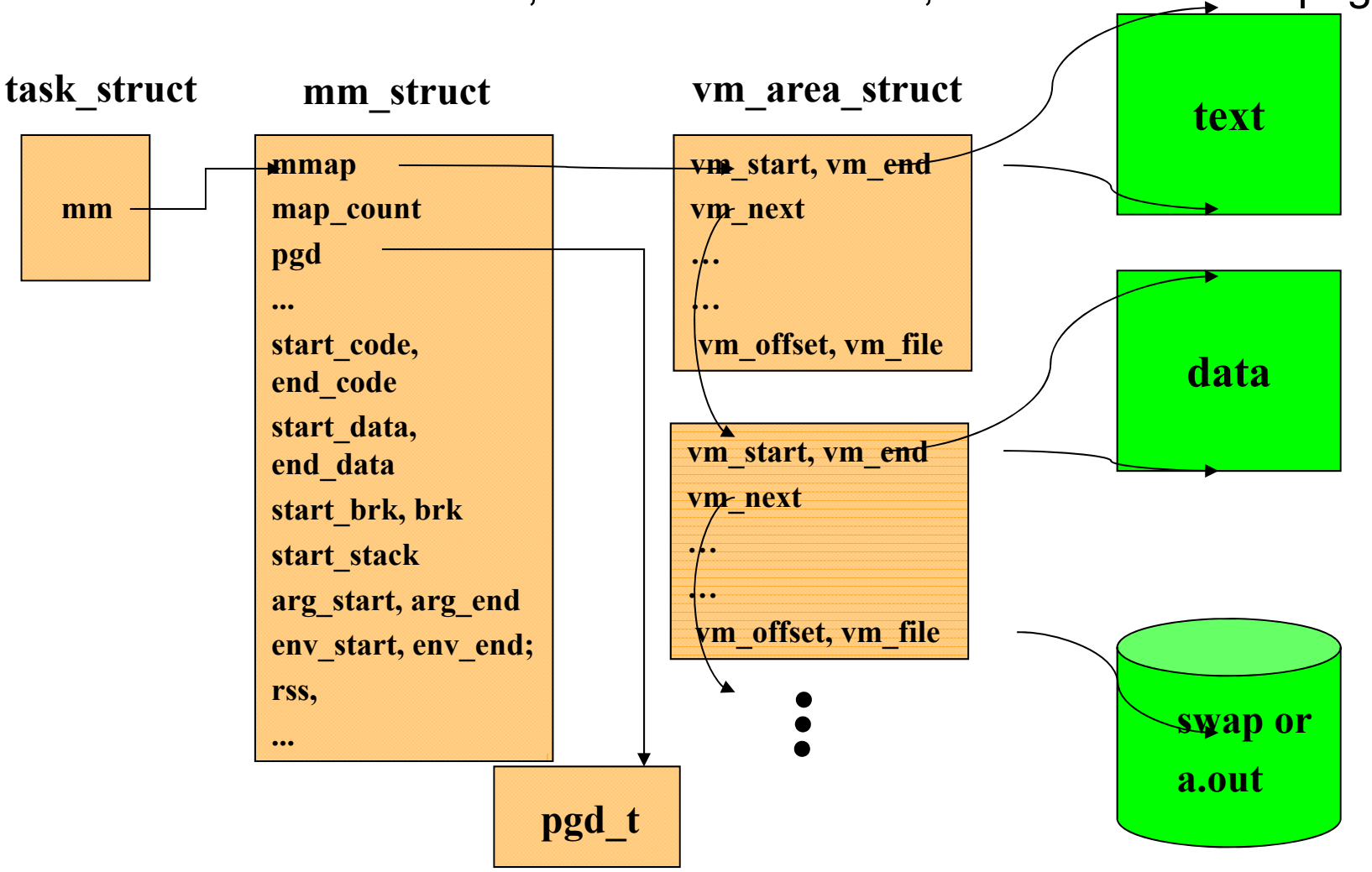
Memory Context (5/5)

- 새로운 프로그램 수행: **execve() internal**



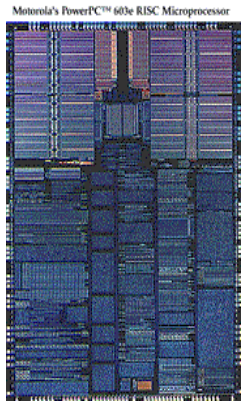
Virtual Memory Describing

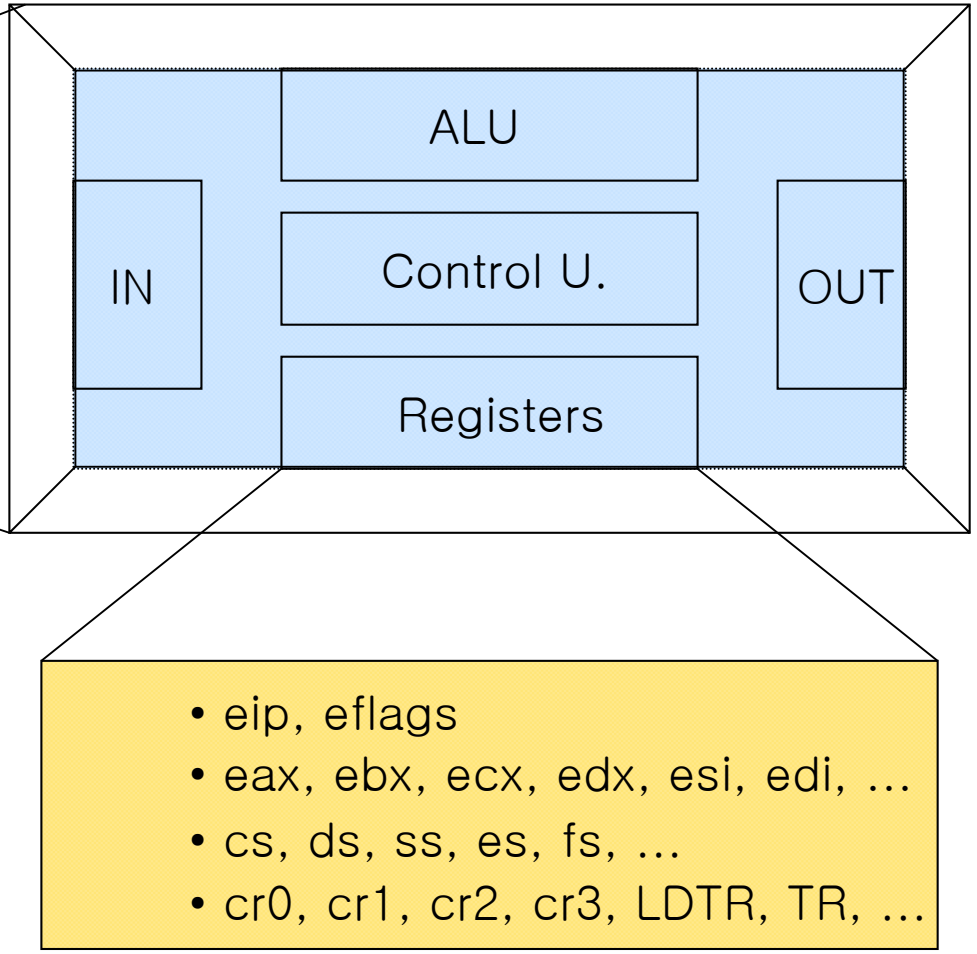
- 메모리 관리 자료 구조
  - ✓ include/linux/sched.h, include/linux/mm.h, include/asm-i386/page.h



Hardware Context (1/3)

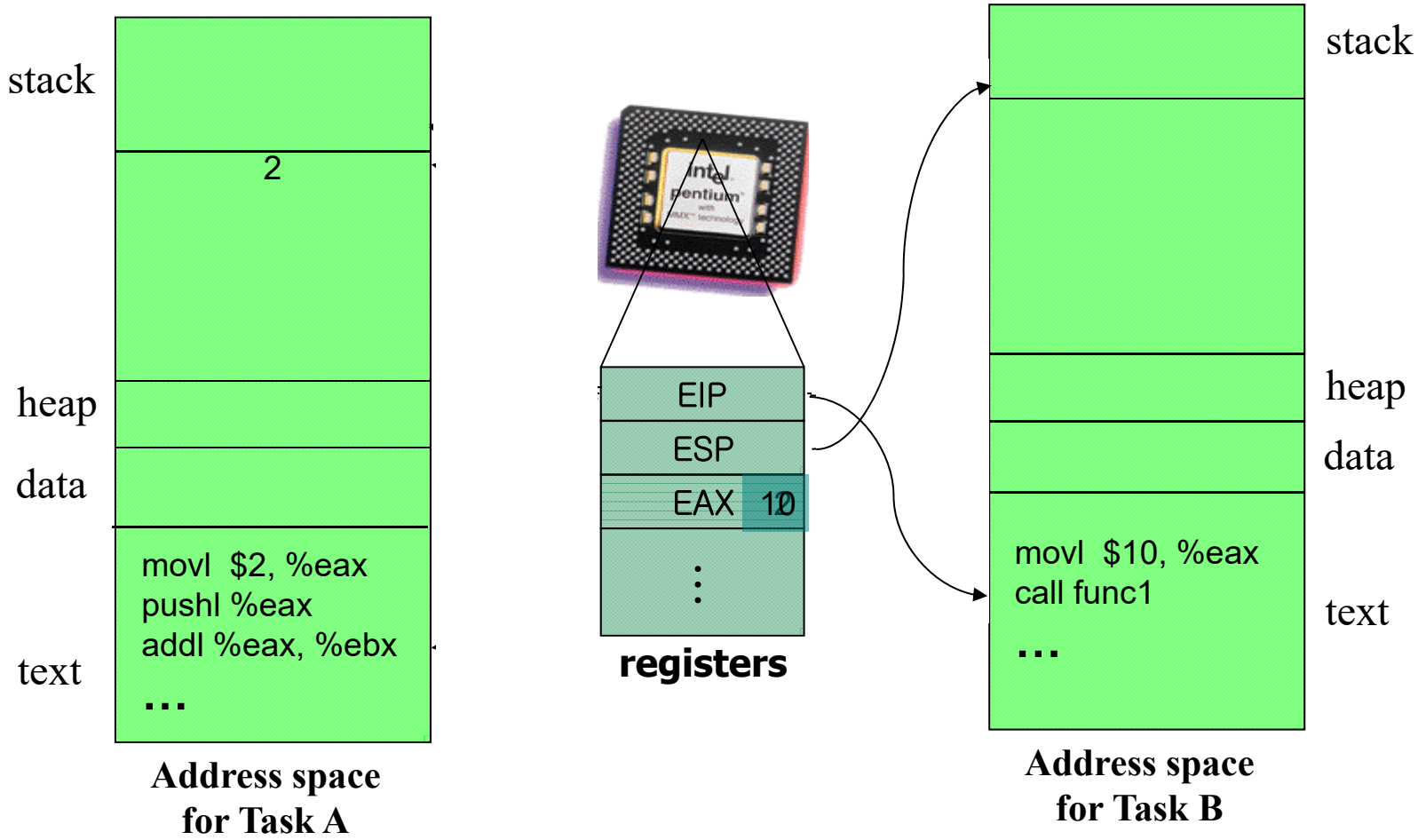
- brief reminds the 80x86 architecture





Hardware Context(2/3)

- 쓰레드 자료 구조 : CPU 추상화

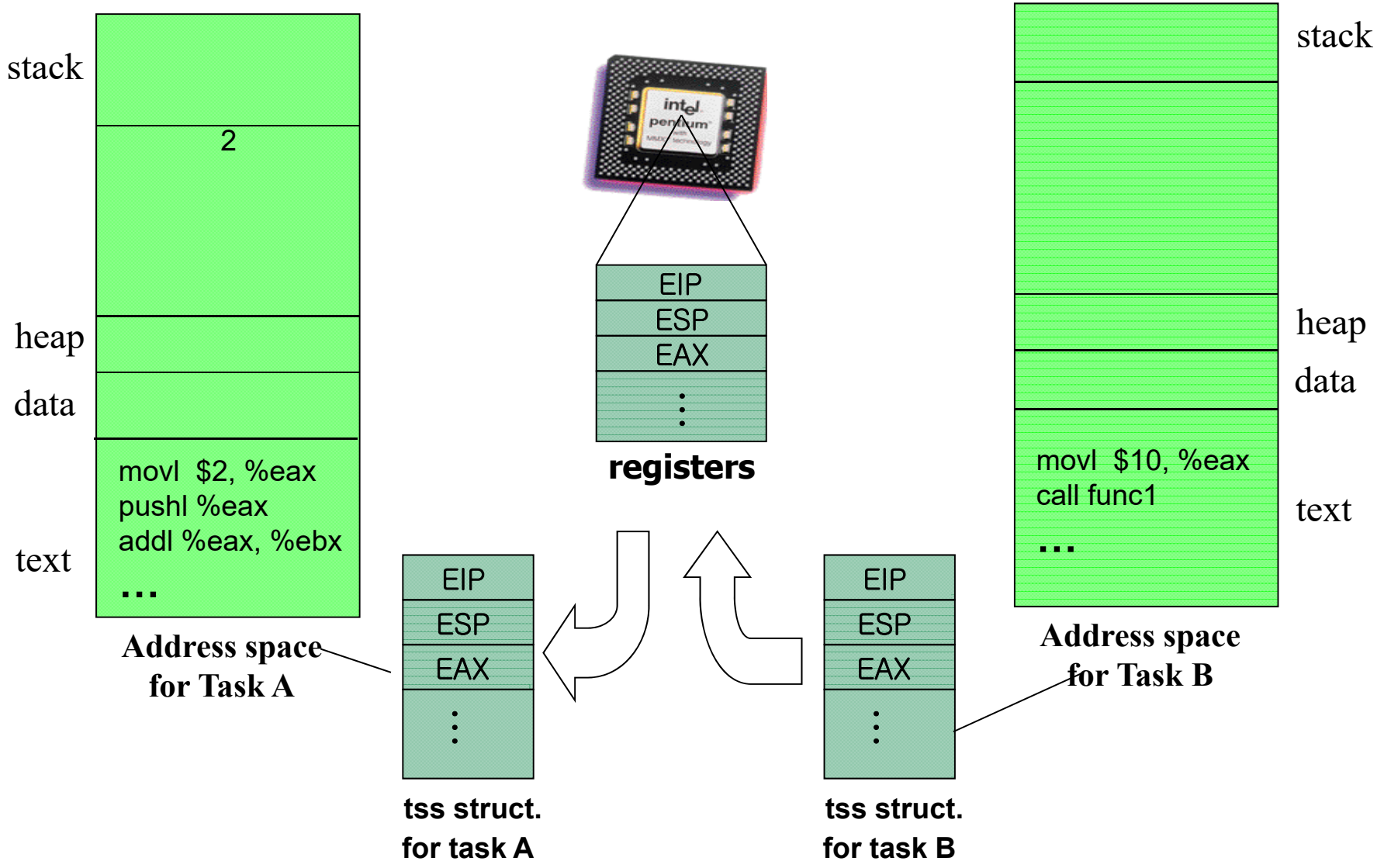


☞ 다시 **Task A**가 수행되려면?

☞ 문맥 교환(**Context Switch**) ➔ **thread structure**

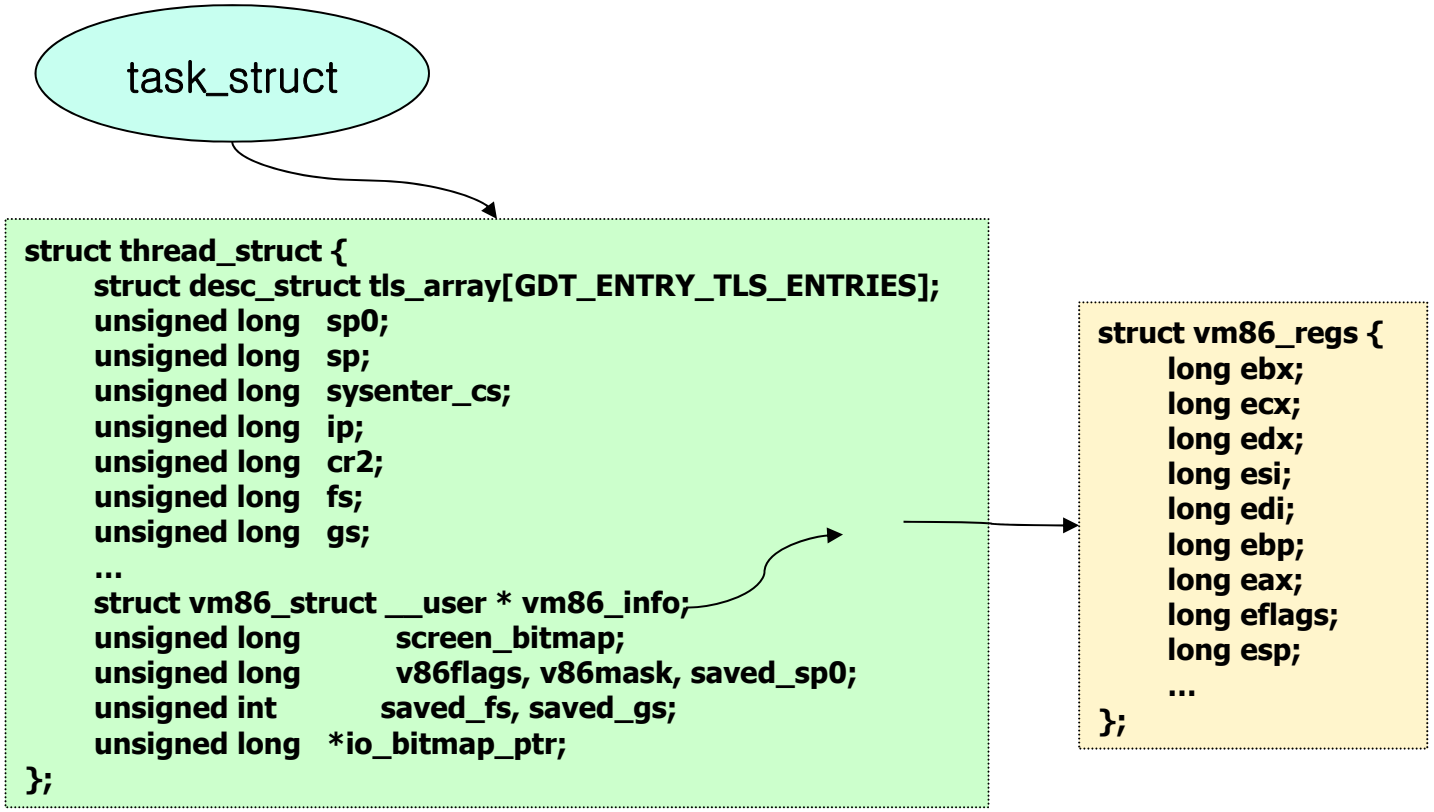
Hardware Context(3/3)

- 쓰레드 자료 구조 : CPU 추상화



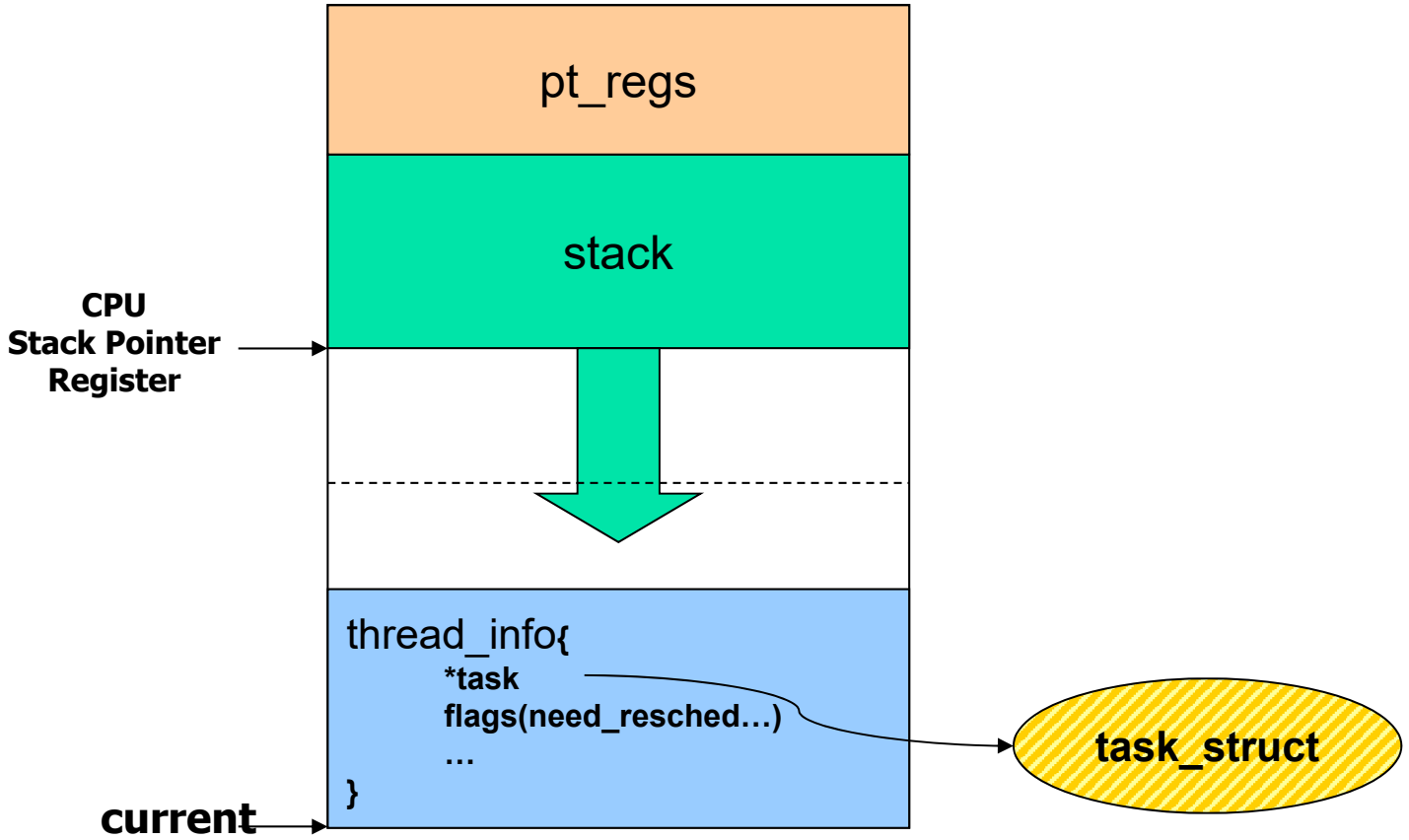
task_struct & H/W context

- 쓰레드 자료 구조 : CPU 추상화



Kernel Stack

- Thread union
  - ✓ 커널은 각 태스크 별로 8KB메모리 할당
  - ✓ thread_info 구조체와 kernel stack
  - ✓ alloc_thread_struct, free_thread_struct



User VS kernel mode

- INT, syscall ➔ kernel mode로 전환
  - ✓ control path : kernel mode로 진입하기 위한 일련의 명령어
- struct pt_regs ?
  - ✓ 현재 P의 상태를 커널 스택에 저장하기 위해 사용

```
struct pt_regs {
        long ebx;
        long ecx;
        long edx;
        long esi;
        long edi;
        long ebp;
        long eax;
        int  xds;
        int  xes;
        int  xfs;
        int  xgs;
        long orig_eax;
        long eip;
        int  xcs;
        long eflags;
        long esp;
        int  xss;
};
```

```
SYSCALL_DEFINE3(open, const char __user *, filename, int, flags, umode_t, mode)
{
        if (force_o_largefile())
                flags |= O_LARGEFILE;

        return do_sys_open(AT_FDCWD, filename, flags, mode);
}
```

Thread Model 비교

- LinuxThreads vs NPTL

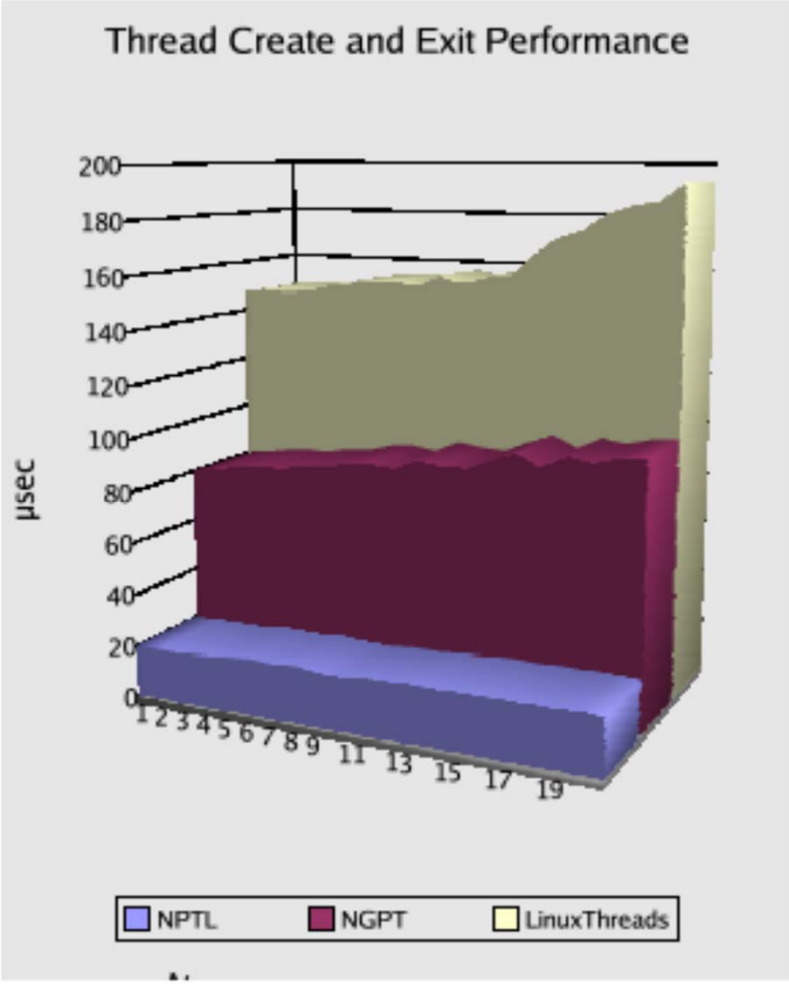


Figure 1: Varying number of Toplevel Threads

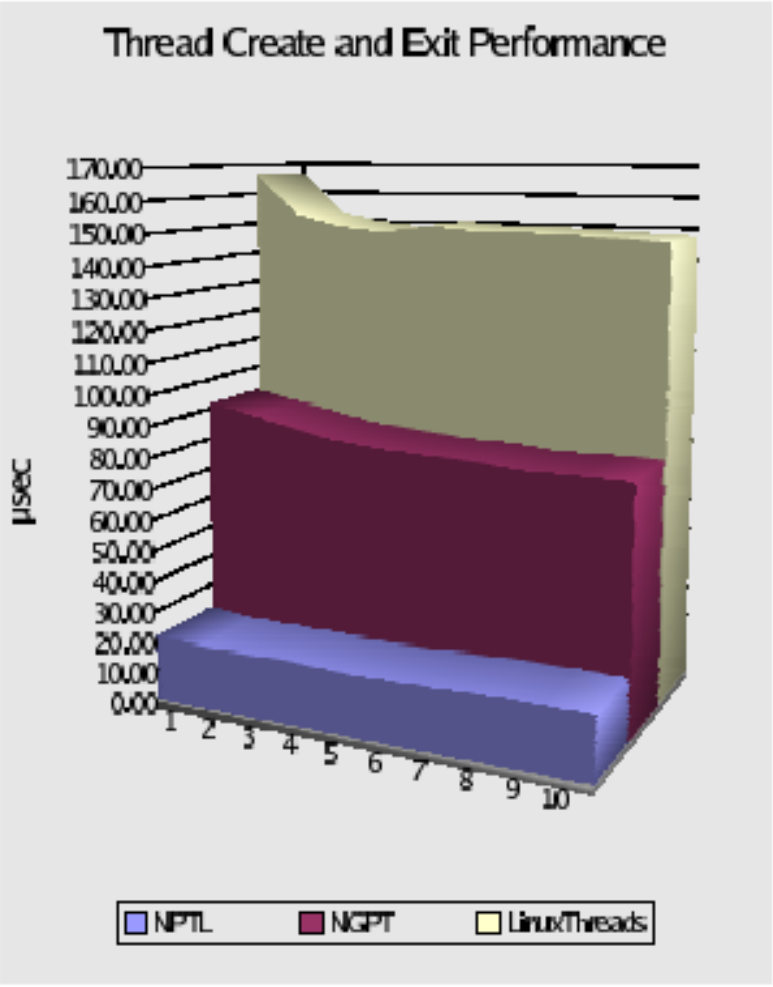
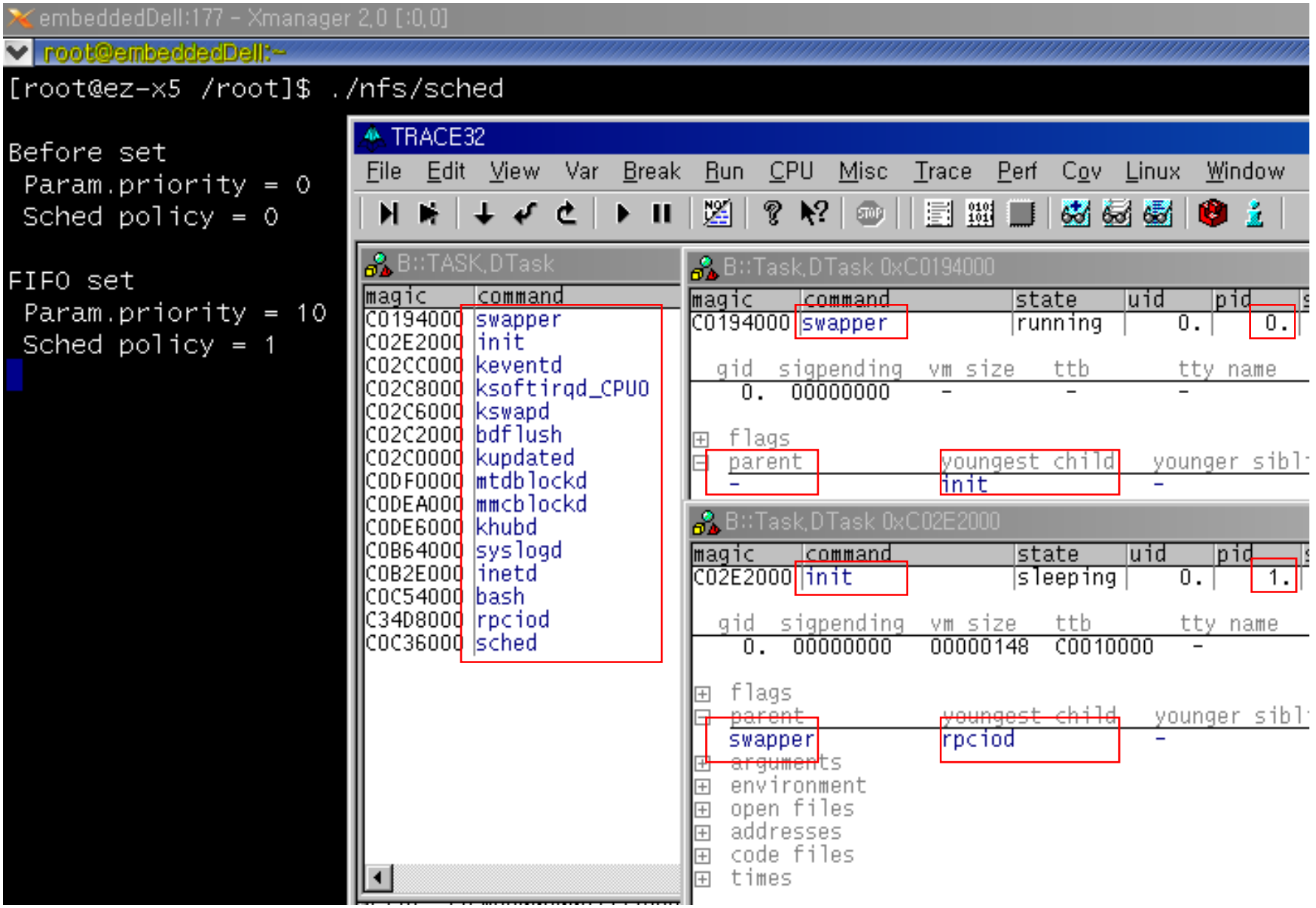
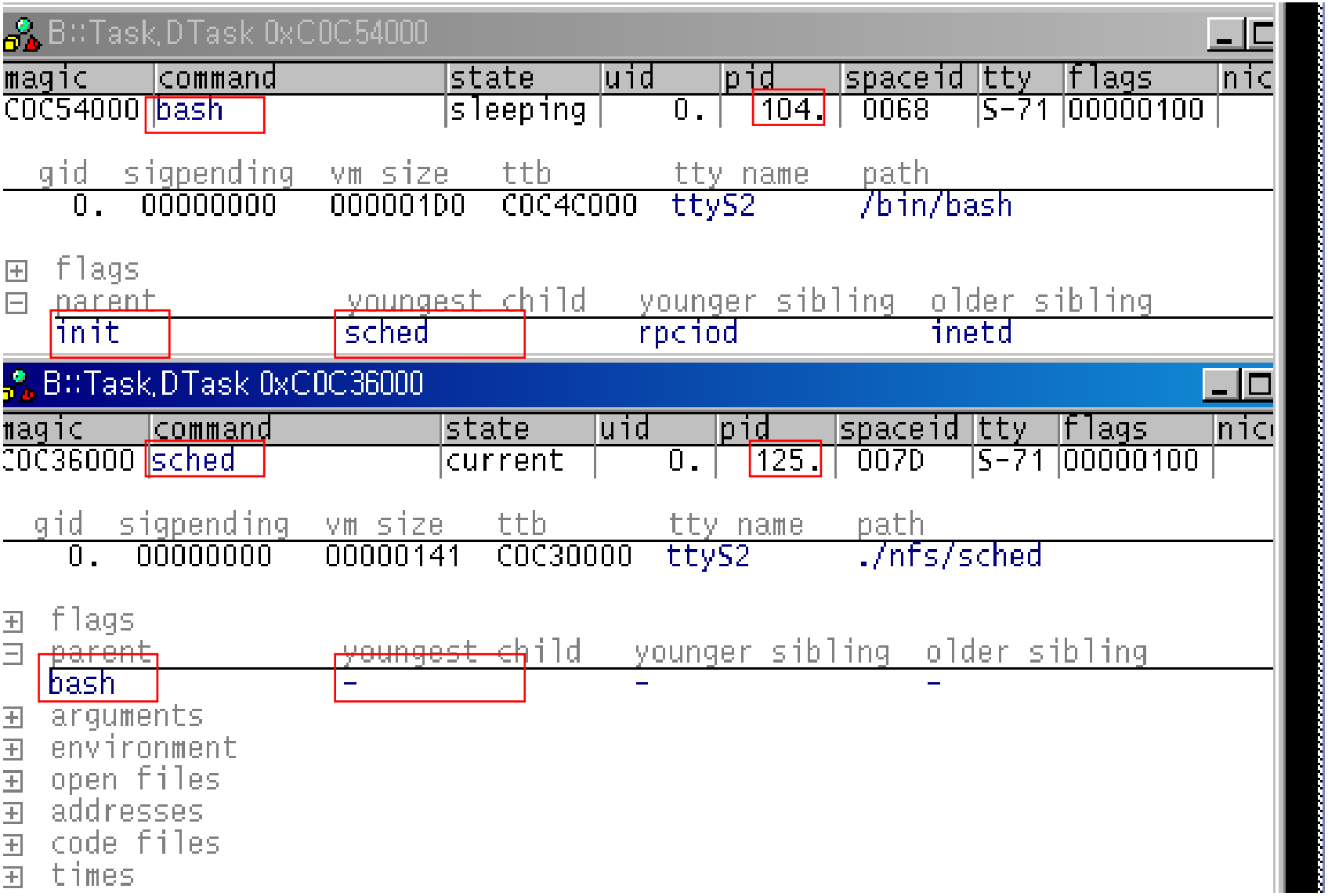


Figure 2: Varying number of Concurrent Children

Task 계층 구조 확인

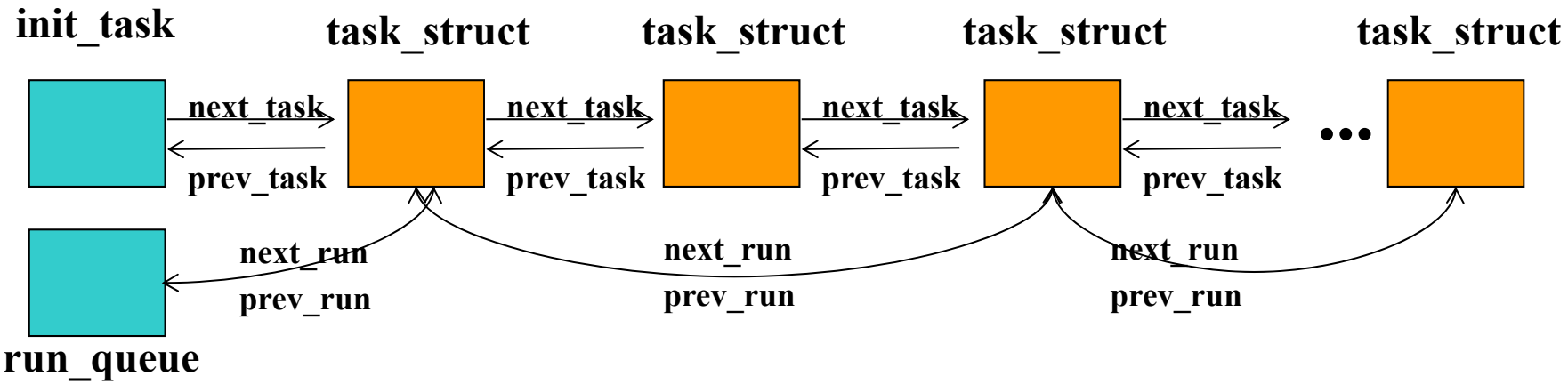

embeddedDell:177 - Xmanager 2.0 [:0.0]
root@embeddedDell:~
[root@ez-x5 /root]$ ./nfs/sched
Before set
Param.priority = 0
Sched policy = 0
FIFO set
Param.priority = 10
Sched policy = 1
TRACE32
File Edit View Var Break Run CPU Misc Trace Perf Cov Linux Window
B::TASK.DTask
magic command
C0194000 swapper
C02E2000 init
C02CC000 keventd
C02C8000 ksoftirqd_CPU0
C02C6000 kswapd
C02C2000 bdflush
C02C0000 kupdated
C0DF0000 mtdblockd
C0DEA000 mmcblockd
C0DE6000 khubd
C0B64000 syslogd
C0B2E000 inetd
C0C54000 bash
C34D8000 rpciod
C0C36000 sched
B::Task.DTask 0xC0194000
magic command state uid pid
C0194000 swapper running 0. 0.
gid sigpending vm size ttb tty name
0. 00000000 - - -
flags
parent youngest child younger sibl
- init -
B::Task.DTask 0xC02E2000
magic command state uid pid
C02E2000 init sleeping 0. 1.
gid sigpending vm size ttb tty name
0. 00000000 00000148 C0010000 -
flags
parent youngest child younger sibl
swapper rpciod -
arguments
environment
open files
addresses
code files
times

Task 계층 구조 확인


B::Task,DTask 0xC0C54000
magic command state uid pid spaceid tty flags nic
C0C54000 bash sleeping 0. 104. 0068 S-71 00000100
gid sigpending vm size ttb tty name path
0. 00000000 00001D0 C0C4C000 ttyS2 /bin/bash
flags
parent youngest child younger sibling older sibling
init sched rpciod inetd
B::Task,DTask 0xC0C36000
magic command state uid pid spaceid tty flags nic
C0C36000 sched current 0. 125. 007D S-71 00000100
gid sigpending vm size ttb tty name path
0. 00000000 00000141 C0C30000 ttyS2 ./nfs/sched
flags
parent youngest child younger sibling older sibling
bash - - -
arguments
environment
open files
addresses
code files
times

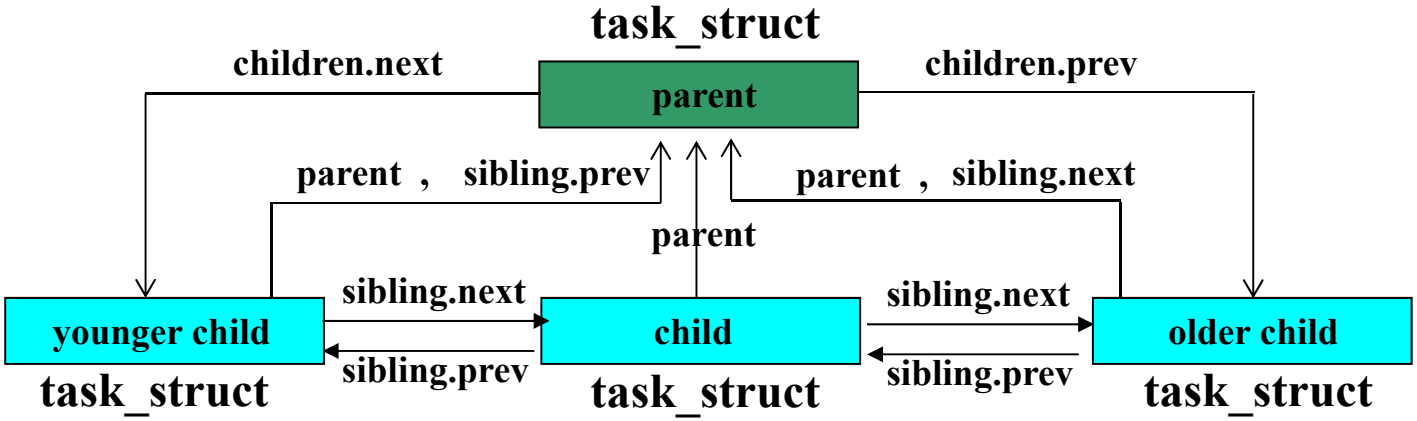
태스크 연결 자료 구조

- 태스크 연결 구조



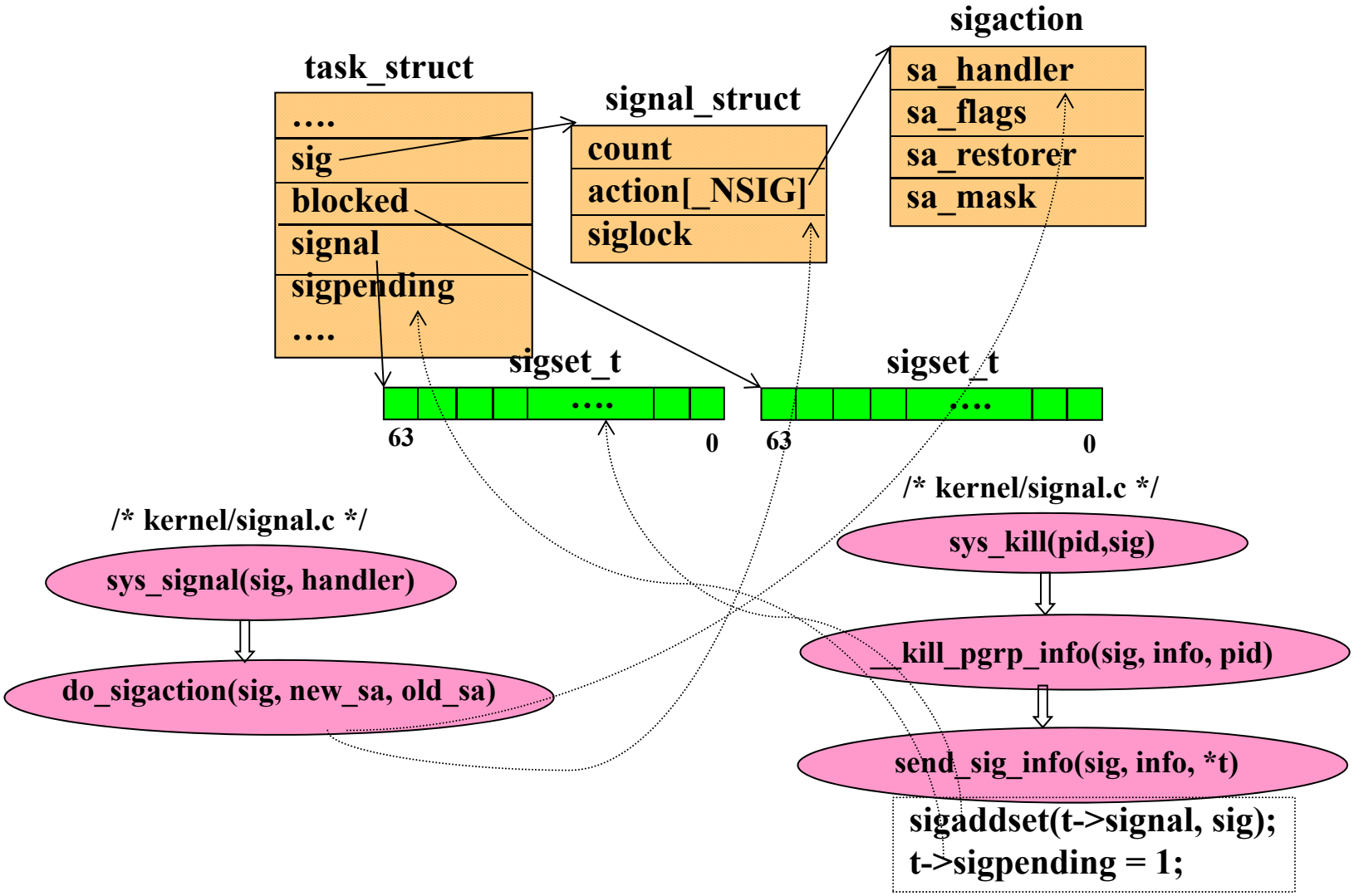
☞where is run_queue ? prev_task? next_task?

- 태스크 가족 관계



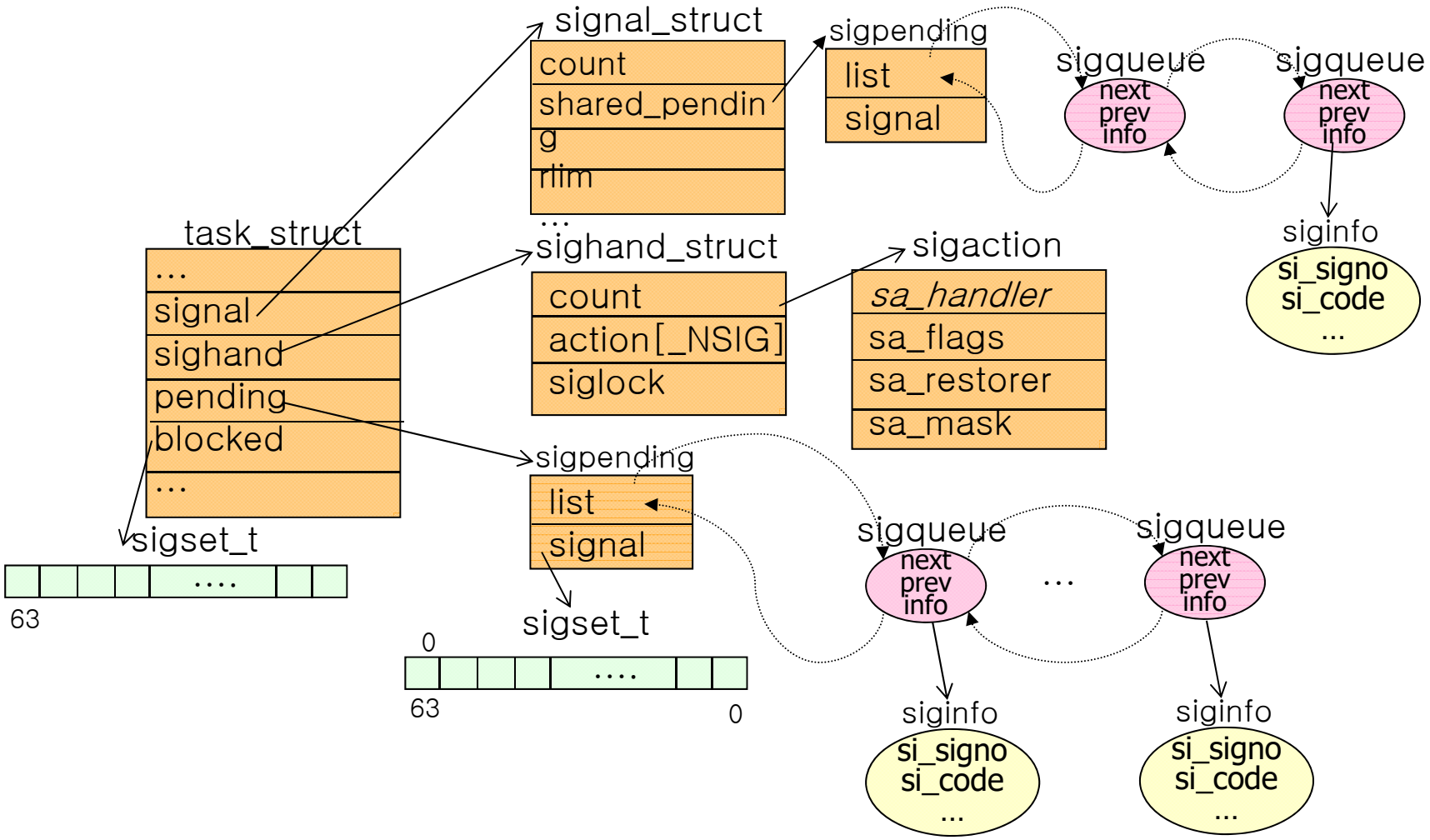
Linux Signal

- 시그널 처리 자료 구조



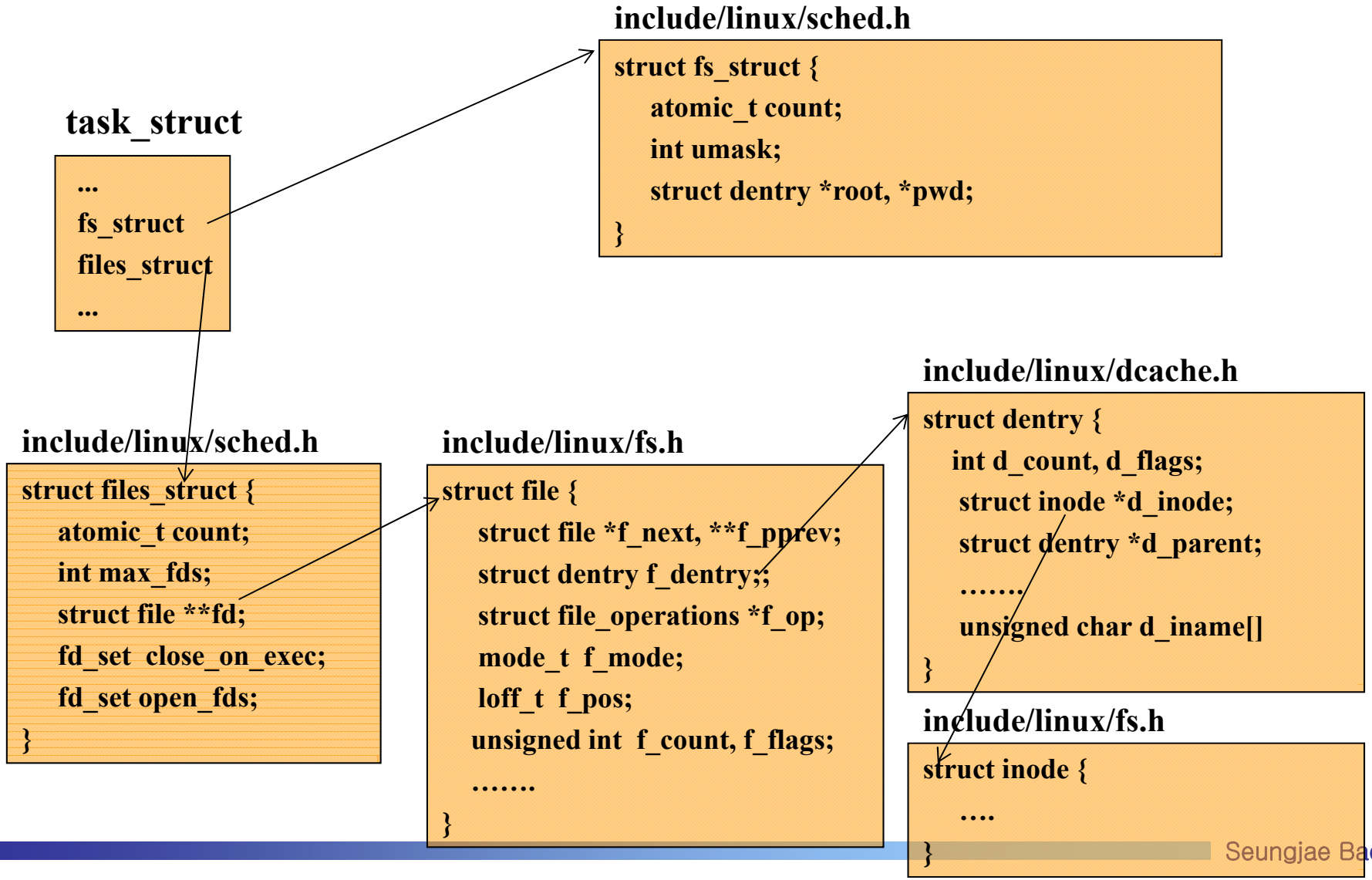
Linux Signal

- 시그널 처리 자료 구조



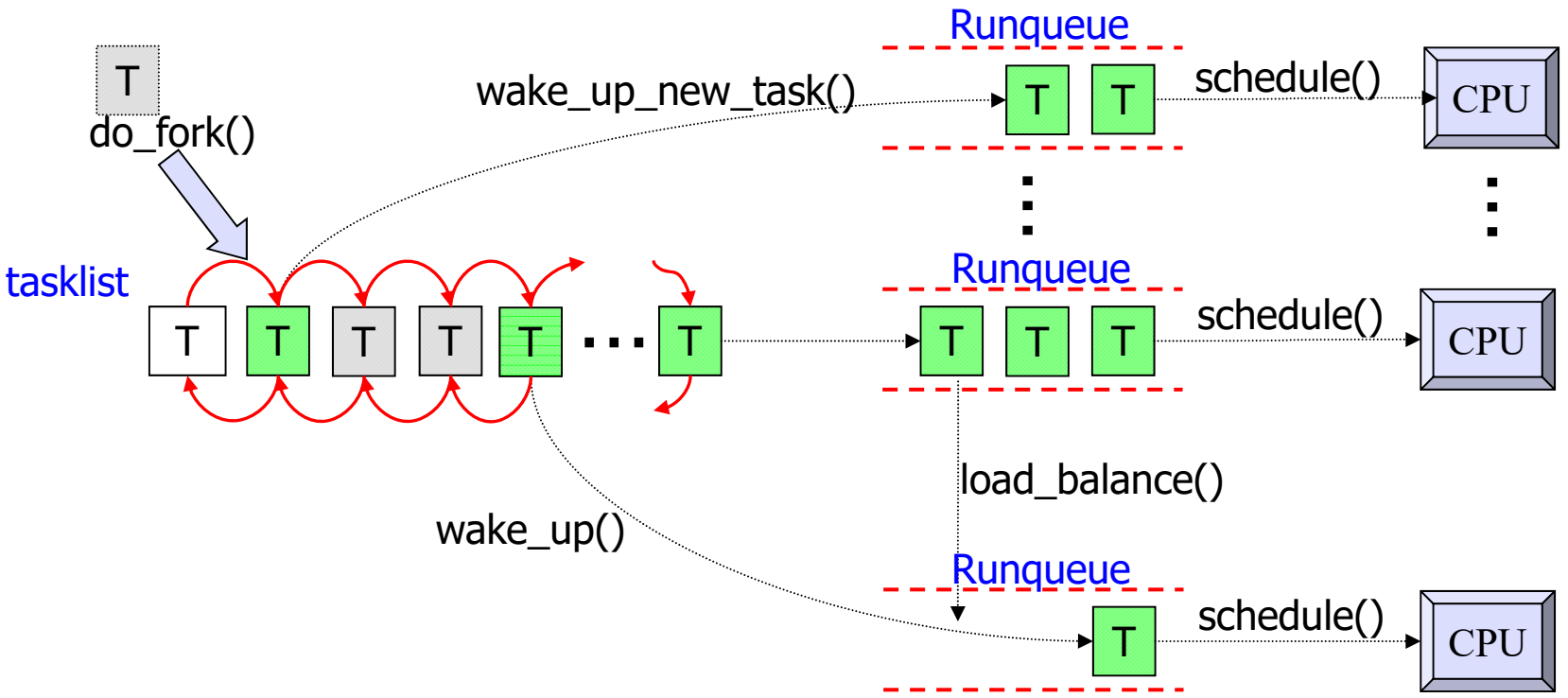
Linux File

- 파일 관리 자료 구조 : fd, inode

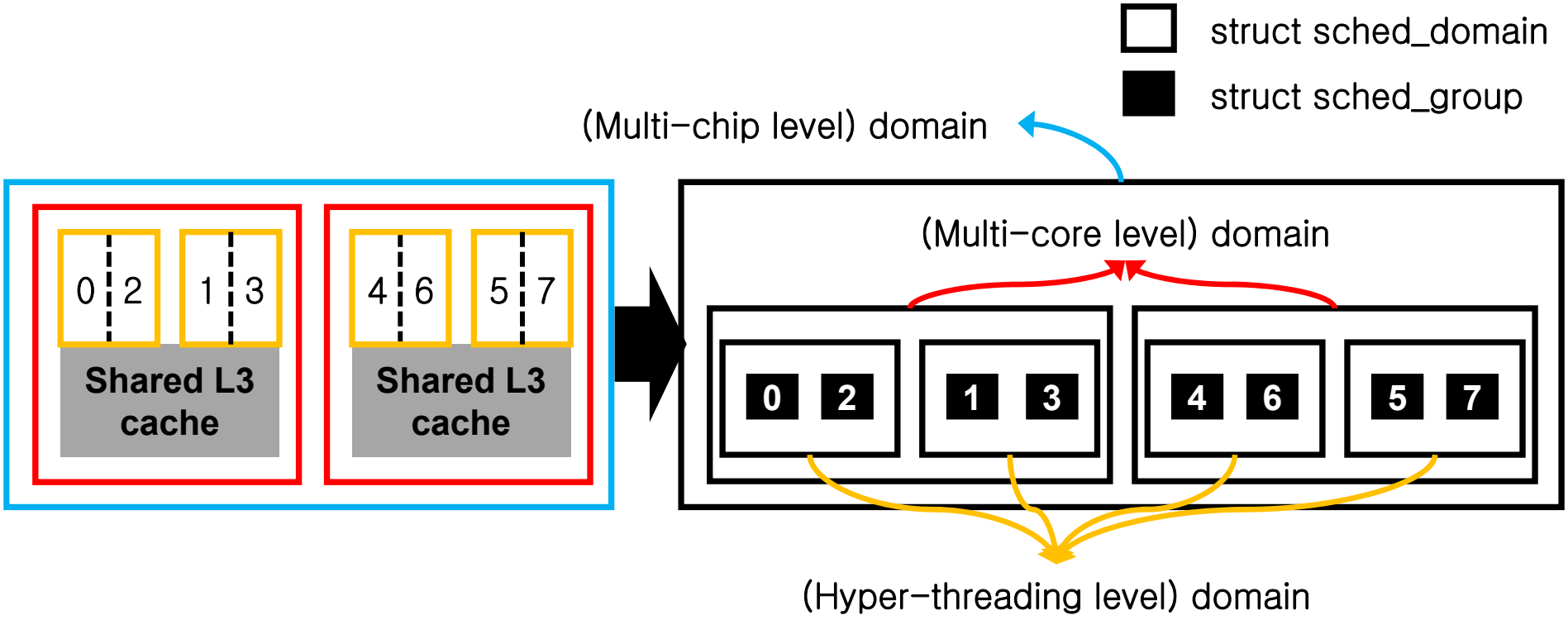


Multi-CPU를 위한 리눅스 특징

- 스케줄링
  - ✓ Run queue per each CPU
    - 일반 task : SCHED_NORMAL, SCHED_BATCH, SCHED_IDLE 100~139(-20~19)
    - 실시간 task : SCHED_FIFO, SCHED_RR, SCHED_DEADLINE, 0~99
  - ✓ Load balancing

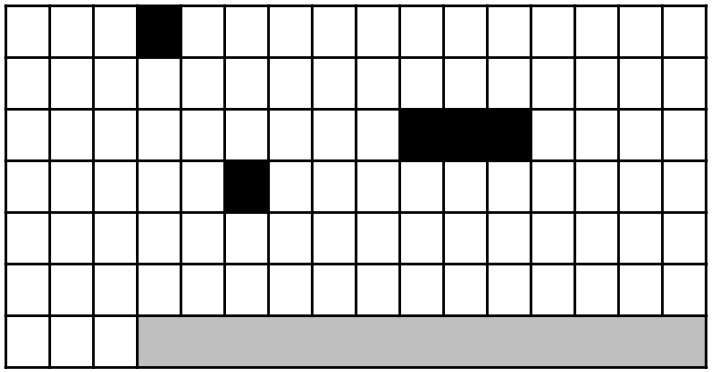


Scheduling Domain & Group

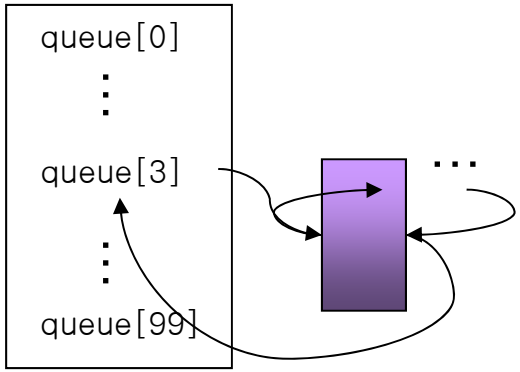


RT Task Scheduling

- FIFO & RR
  - ✓ O(1) scheduler



rt_prio_array.bitmap



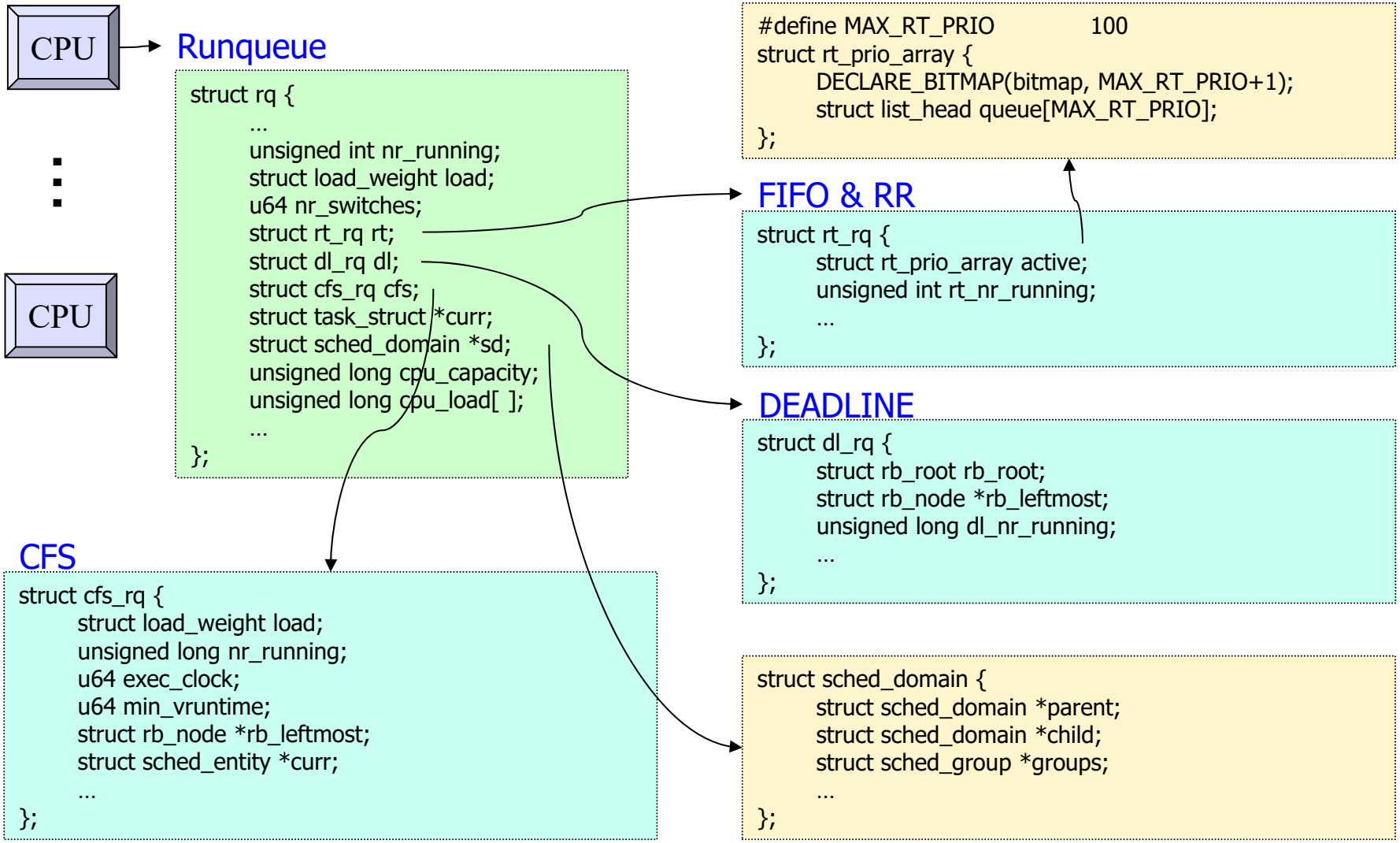
rt_prio_array.queue

- DEADLINE (a.k.a., EDF)
  - ✓ deadline, runtime, period
  - ✓ RB-tree (with deadline)
  - ✓ Provides deterministic scheduling

$$current_{time} + runtime < deadline$$

$$CPU_{capability} \geq \sum_{i=0}^{n} Task_{(i,runtime)}$$

Runqueue and Scheduler Data Structure



Test Code

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

int main(void)
{
        struct sched_param param;
        int i, j;

        sched_getparam( 0, &param);

        printf(" ₩nBefore set₩n");
        printf(" Param.priority = %d₩n", param.sched_priority);
        printf(" Sched policy = %d₩n", sched_getscheduler(0));
                                ①
        param.sched_priority = 10;
        sched_setscheduler(0, SCHED_FIFO, &param);
        sched_getparam( 0, &param);

        printf(" ₩nFIFO set₩n");
        printf(" Param.priority = %d₩n", param.sched_priority);
        printf(" Sched policy = %d₩n", sched_getscheduler(0));
                                ②
        param.sched_priority = 20;
        sched_setscheduler(0, SCHED_RR, &param);
        sched_getparam( 0, &param);

        printf(" ₩nRR set₩n");
        printf(" Param.priority = %d₩n", param.sched_priority);
        printf(" Sched policy = %d₩n", sched_getscheduler(0));
                                ③
        return 0;
}
```

```
[root@embeddedDell root]# ./sched

Before set
 Param.priority = 0
 Sched policy = 0

FIFO set
 Param.priority = 10
 Sched policy = 1

RR set
 Param.priority = 20
 Sched policy = 2
[root@embeddedDell root]#
```

☞아래 코드를 화살표가 가리키고 있는 세 군데에 각각 넣고 실행해 보자.

☞ 실행 도중 키보드를 누르면 반응이 있는가?

☞언제 반응이 있고, 언제 없는가?

```
for(i=0; i<100000; i++)
        for(j=0; j<100000; j++);
```

Normal Task Scheduling

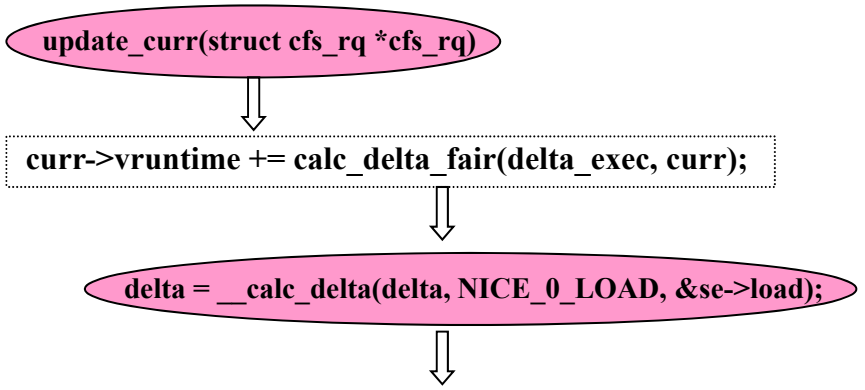
- CFS
  - ✓ Completely Fair Scheduler
  - ✓ RB-tree (with vruntime)
    - Timer tick에 의해 현재 수행 중인 태스크의 vruntime갱신

```
struct task_struct {
        ...
        const struct sched_class *sched_class;
        struct sched_entity se;
        ...
};
```

```
struct sched_entity {
        ...
        struct load_weight load;
        u64 vruntime;
        ...
};
```

```
struct load_weight {
        unsigned long weight;
        u32 inv_weight;
};
```

```
static const int prio_to_weight[40] = {
 /* -20 */     88761,     71755,     56483,     46273,     36291,
 /* -15 */     29154,     23254,     18705,     14949,     11916,
 /* -10 */      9548,      7620,      6100,      4904,      3906,
 /*  -5 */      3121,      2501,      1991,      1586,      1277,
 /*   0 */      1024,       820,       655,       526,       423,
 /*   5 */       335,       272,       215,       172,       137,
 /*  10 */       110,        87,        70,        56,        45,
 /*  15 */        36,        29,        23,        18,        15,
};
```



$$vruntime \mathrel{+}= physicalruntime \times \frac{weight_0}{weight_{curr}}$$

Normal Task Scheduling

- CFS
  - ✓ Completely Fair Scheduler
  - ✓ RB-tree (with vruntime)
    - Timer tick에 의해 현재 수행 중인 태스크의 vruntime갱신
    - To avoid frequent context switch, there are minimum runtime and

```
struct task_struct {
        ...
        const struct sched_class *sched_class;
        struct sched_entity se;
        ...
};
```

```
struct sched_entity {
        ...
        struct load_weight load;
        u64 vruntime;
        ...
};
```

```
struct load_weight {
        unsigned long weight;
        u32 inv_weight;
};
```

```
static const int prio_to_weight[40] = {
 /* -20 */     88761,     71755,     56483,     46273,     36291,
 /* -15 */     29154,     23254,     18705,     14949,     11916,
 /* -10 */      9548,      7620,      6100,      4904,      3906,
 /*  -5 */      3121,      2501,      1991,      1586,      1277,
 /*   0 */      1024,       820,       655,       526,       423,
 /*   5 */       335,       272,       215,       172,       137,
 /*  10 */       110,        87,        70,        56,        45,
 /*  15 */        36,        29,        23,        18,        15,
};
```

sched_slice(struct cfs_rq *cfs_rq, struct sched_entity *se)

⇩

slice = __sched_period(cfs_rq->nr_running + !se->on_rq);

⇩

```
if(nr_running > sched_nr_latency)
    return sysctl_sched_min_granularity * nr_running
else
    return sysctl_sched_latency
```

⇩

$$task's\ slice = slice \times \frac{weight_{curr}}{\sum_{i=0}^{n} weight_i}$$

Scheduler Activation

- How
  - ✓ Directly call schedule()
  - ✓ Indirectly, mark *need_resched* flag
- When

Check check_preempt_tick() function for more details!

  - ✓ Timer tick
  - ✓ 수행 중이던 태스크가 자신의 타임 슬라이스를 다 썼거나 대기 상태로 전환
  - ✓ 현재 태스크보다 높은 우선순위를 가진 태스크가 깨어난 경우
  - ✓ 스케줄링 관련 시스템 콜 호출

함수 명	목적
set_tsk_need_resched(task)	주어진 프로세스의 need_resched 플래그를 설정
clear_tsk_need_resched(task)	주어진 프로세스의 need_resched 플래그를 해제
need_resched()	need_resched 플래그를 검사. Set이면 true, clear이면 false를 리턴

Linux 스케줄링 인터페이스

- 스케줄링 관련 시스템 콜
  - ✓ nice() (just for backward compatibility)
    - sys_nice()
    - 절대값 40 넘으면 40으로 맞춤
    - 음수는 관리자 권한 필요 ➔ capable()함수 호출해 CAP_SYS_NICE 특성이 있는지 확인
    - 권한이 있다면 current의 nice필드에 increment값 더함
    - 단, -20~19사이 유지하게 함

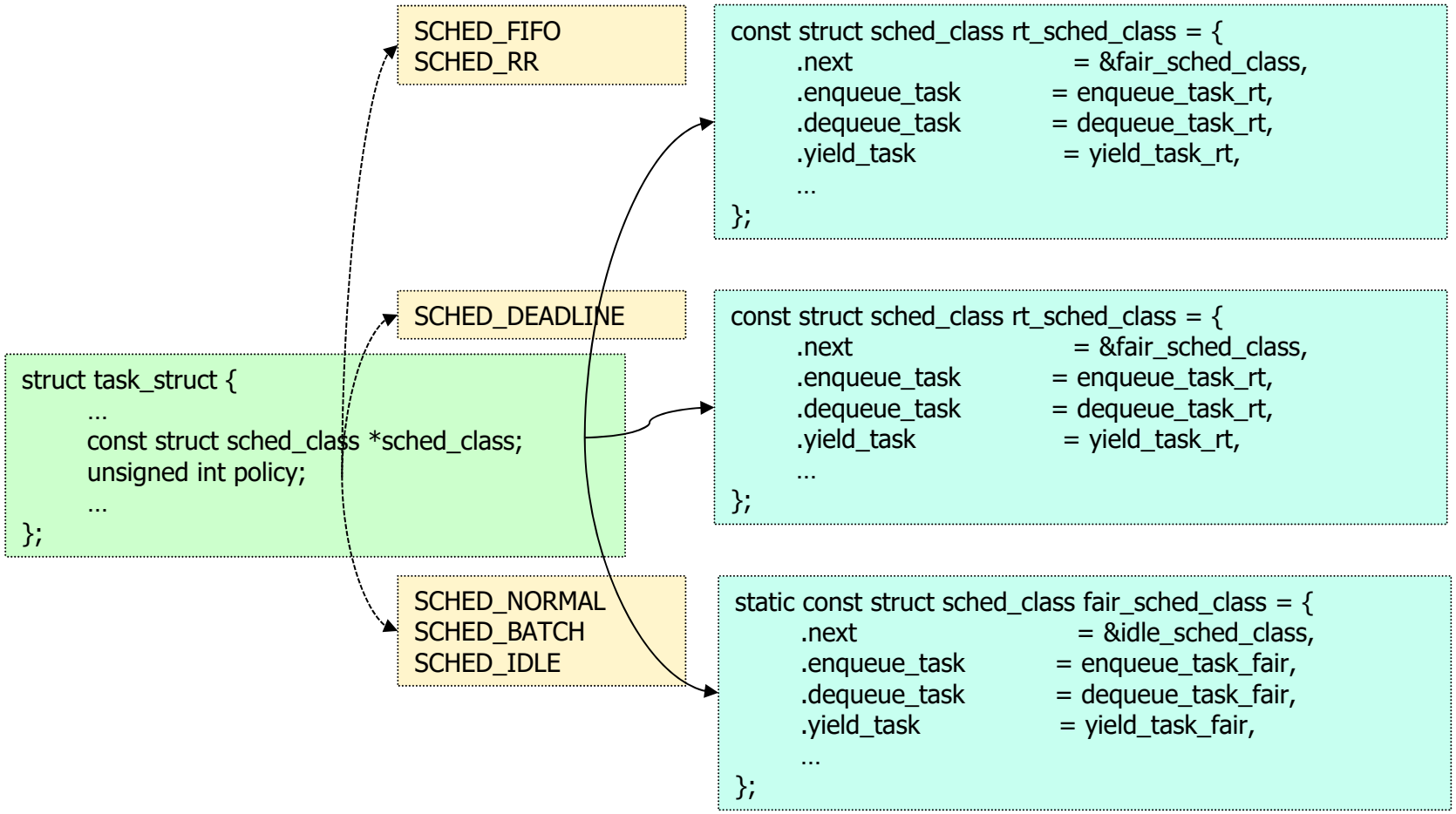
Linux 스케줄링 인터페이스

- ✓ getpriority(), setpriority()
  - 지정한 process그룹에 있는 모든 process의 기본 우선 순위에 영향을 미침
  - getpriority()는 그룹 내 모든 process중 가장 낮은 nice필드 값 밴 값 반환
  - setpriority()는 지정 그룹 내 모든 process의 기본우선순위 설정
  - sys_getpriority(), sys_setpriority()
  - 매개변수
    - which : 프로세스 그룹을 식별
      - PRIO_PROCCESS
      - PRIO_PGRP
      - PRIO_USER
    - who : process선택에 사용하는 pid나, pgrp, uid필드의 값, 0이면 current
    - niceval : 새로운 우선순위 값

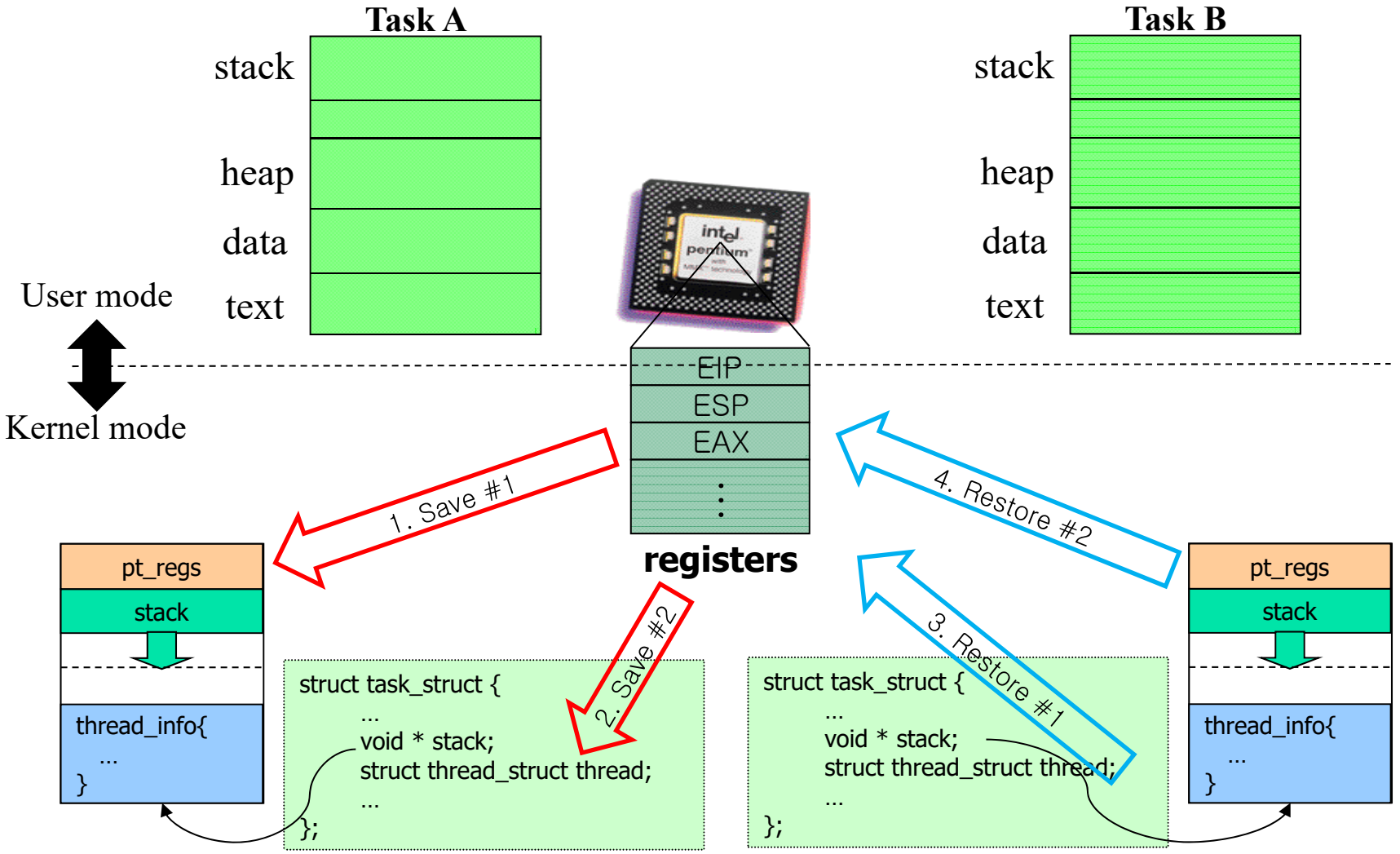
Linux 스케줄링 인터페이스

- 실시간 process관련 시스템 콜
  - ✓ sched_getscheduler(), sched_setscheduler()
    - 매개변수 pid로 지정한 process에 현재 적용중인 스케줄링 정책 질의/설정
    - Pid를 가지고 find_process_by_pid()호출해, policy값을 반환
  - ✓ sched_getparam(), sched_setparam()
    - pid에 해당하는 process의 스케줄링 인자 얻어오거나/설정
    - rt_priority 필드를 sched_param 타입 지역변수로 저장 후
    - copy_to_user()호출해 복사해줌
  - ✓ sched_yield()
    - Process 를 보류상태로 만들지 않고 자발적으로 CPU반납
  - ✓ sched_get_priority_min(), sched_get_priority_max()
    - policy매개 변수로 지정한 스케줄링 정책에서 사용할 수 있는 실시간 정적 우선 순위의 최소, 최대 값을 반환
  - ✓ sched_rr_get_interval()
    - pid 매개 변수로 지정한 실시간 P의 RR타임 퀀텀을 사용자 모드 주소 공간에 있는 구조체에 기록 / 가져옴

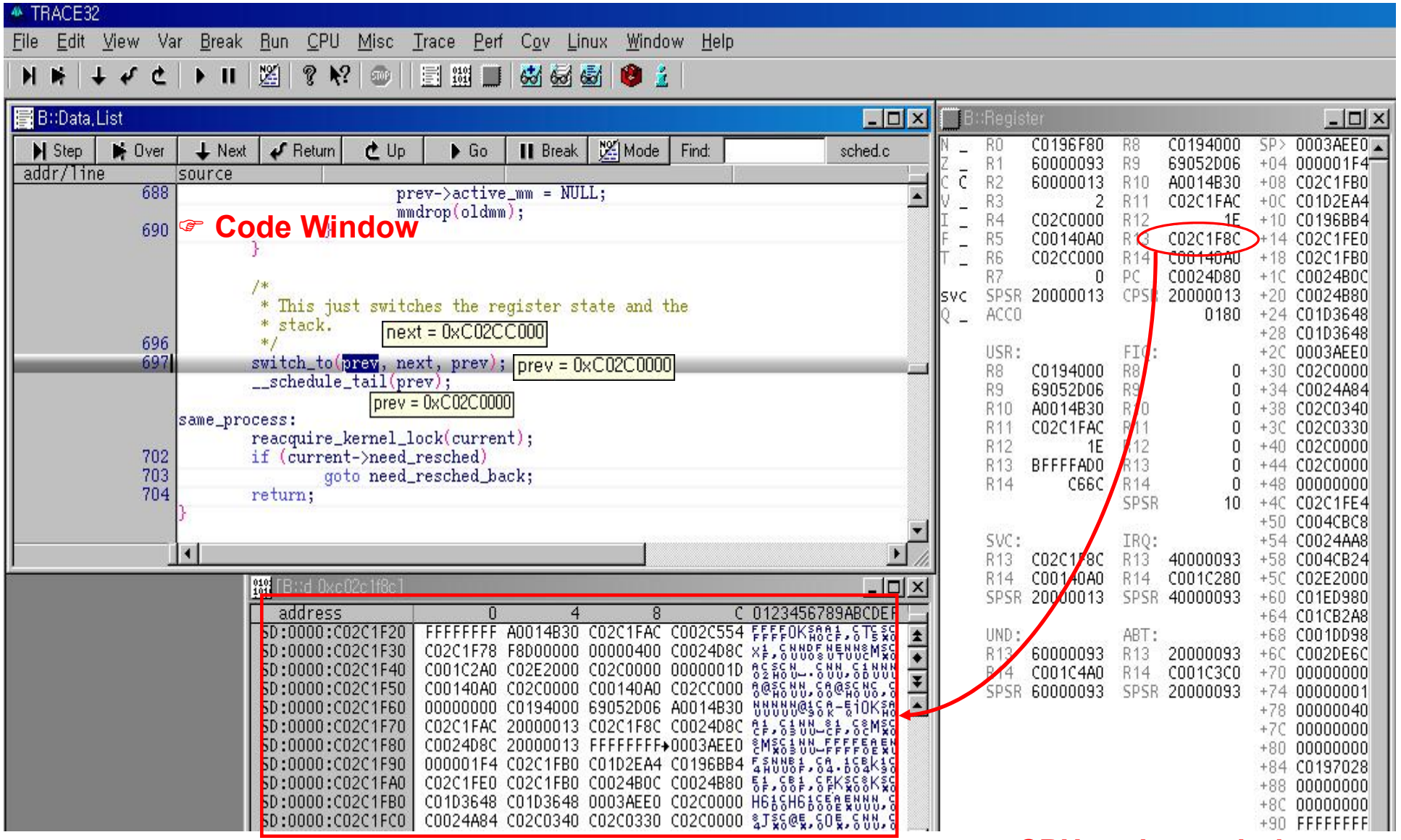
Scheduling Policy and Class



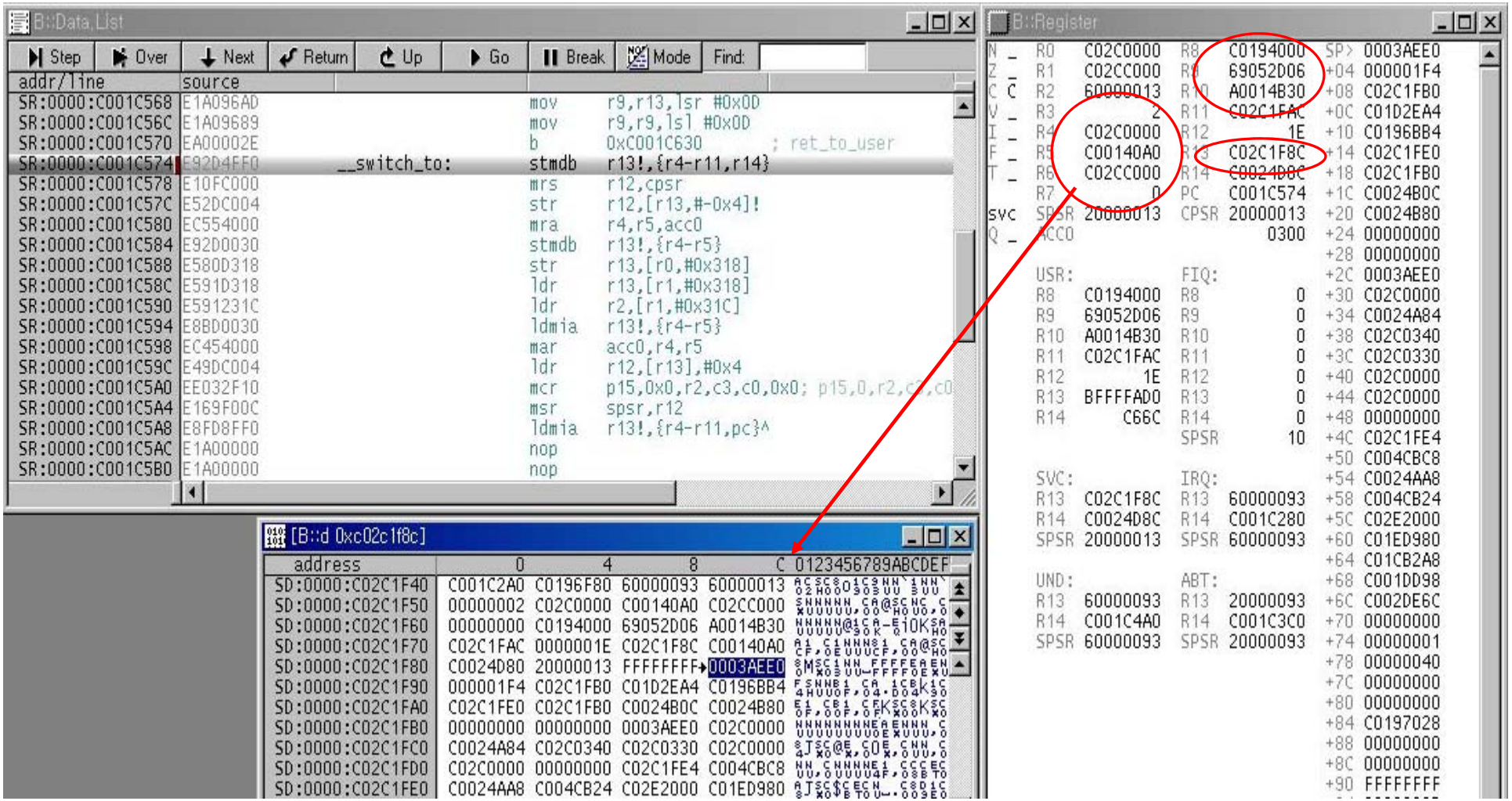
Context Switching



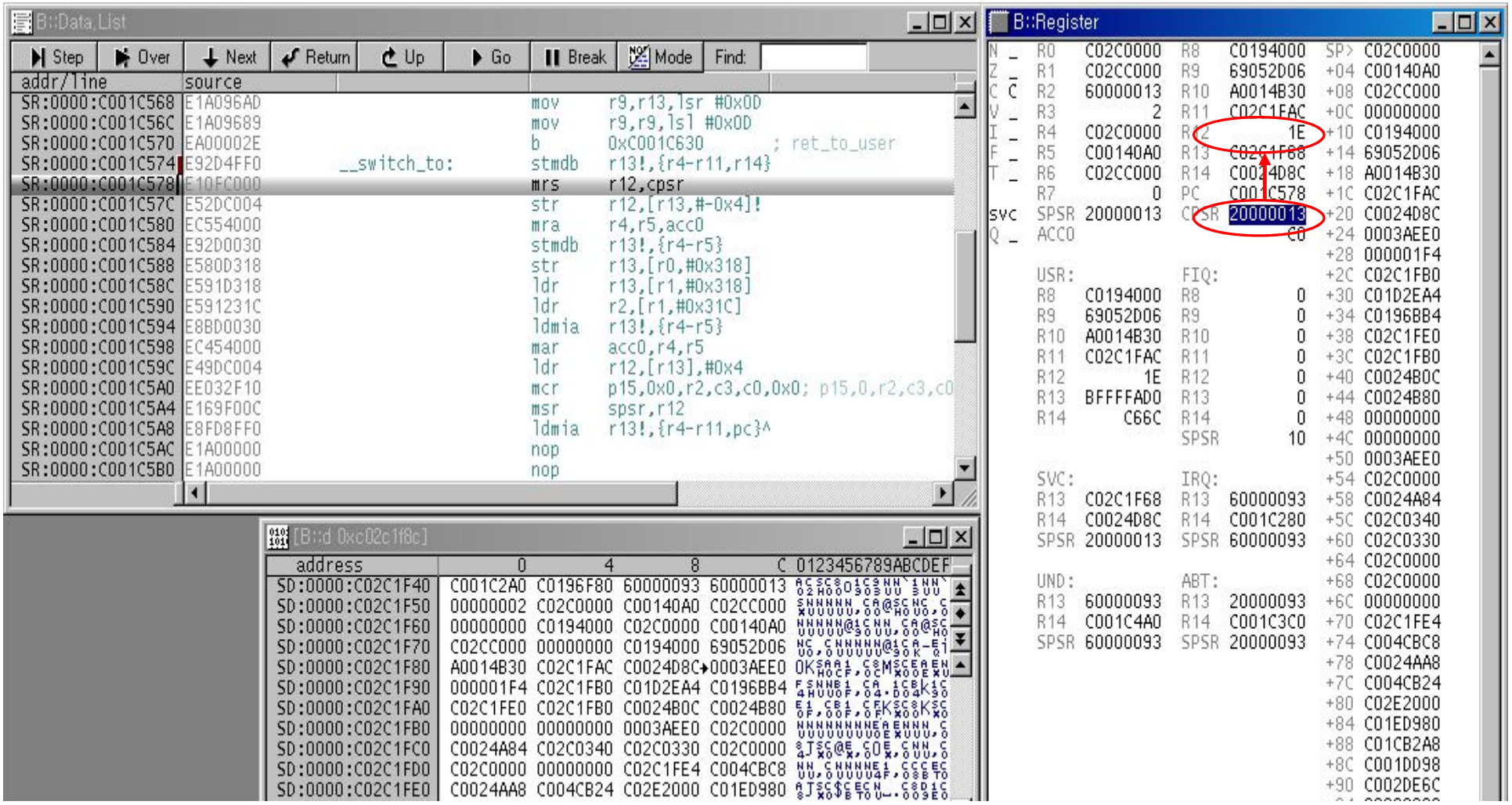
__switch_to() in ARM (1/16)



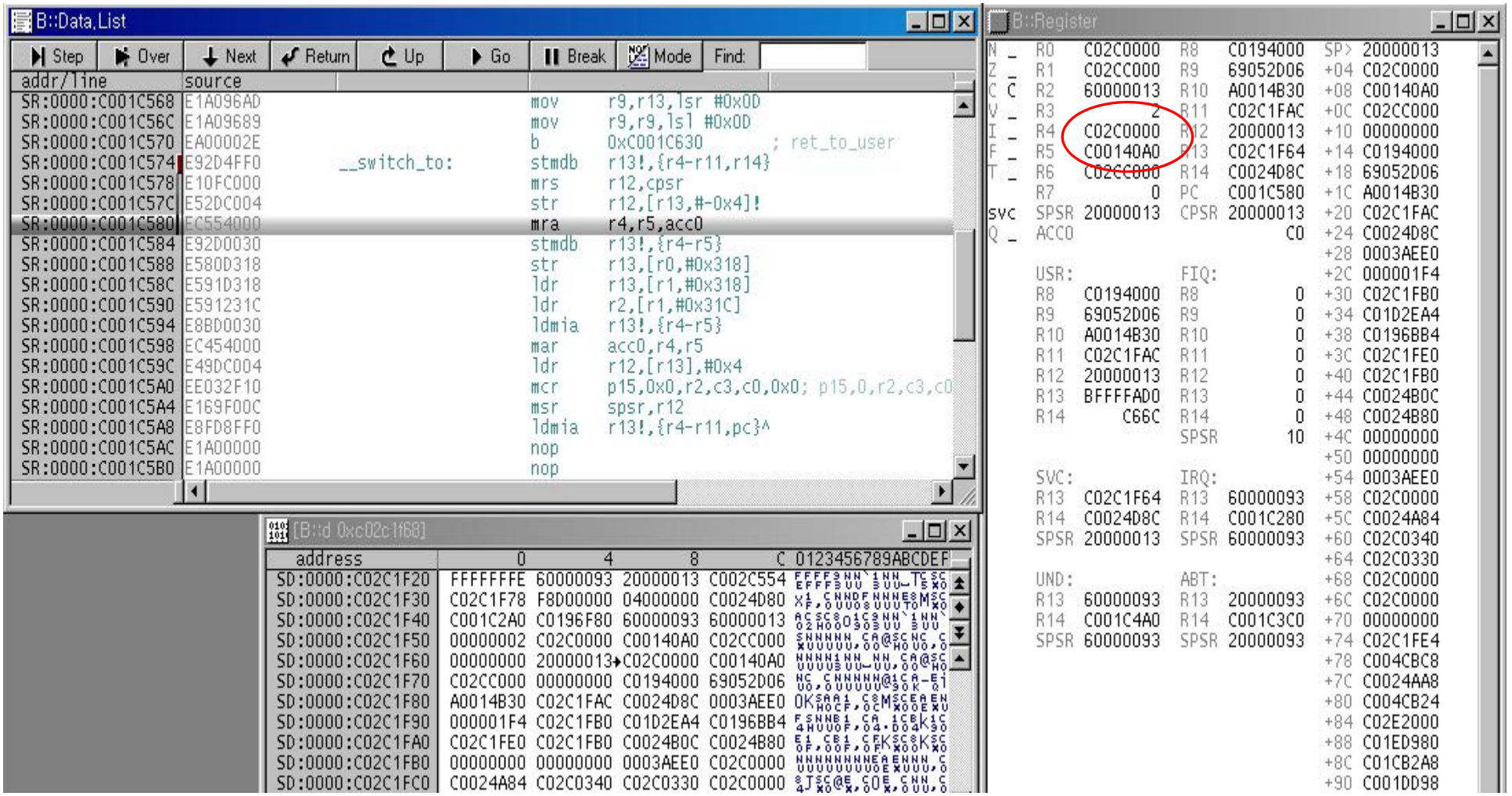
__switch_to() in ARM (2/16)



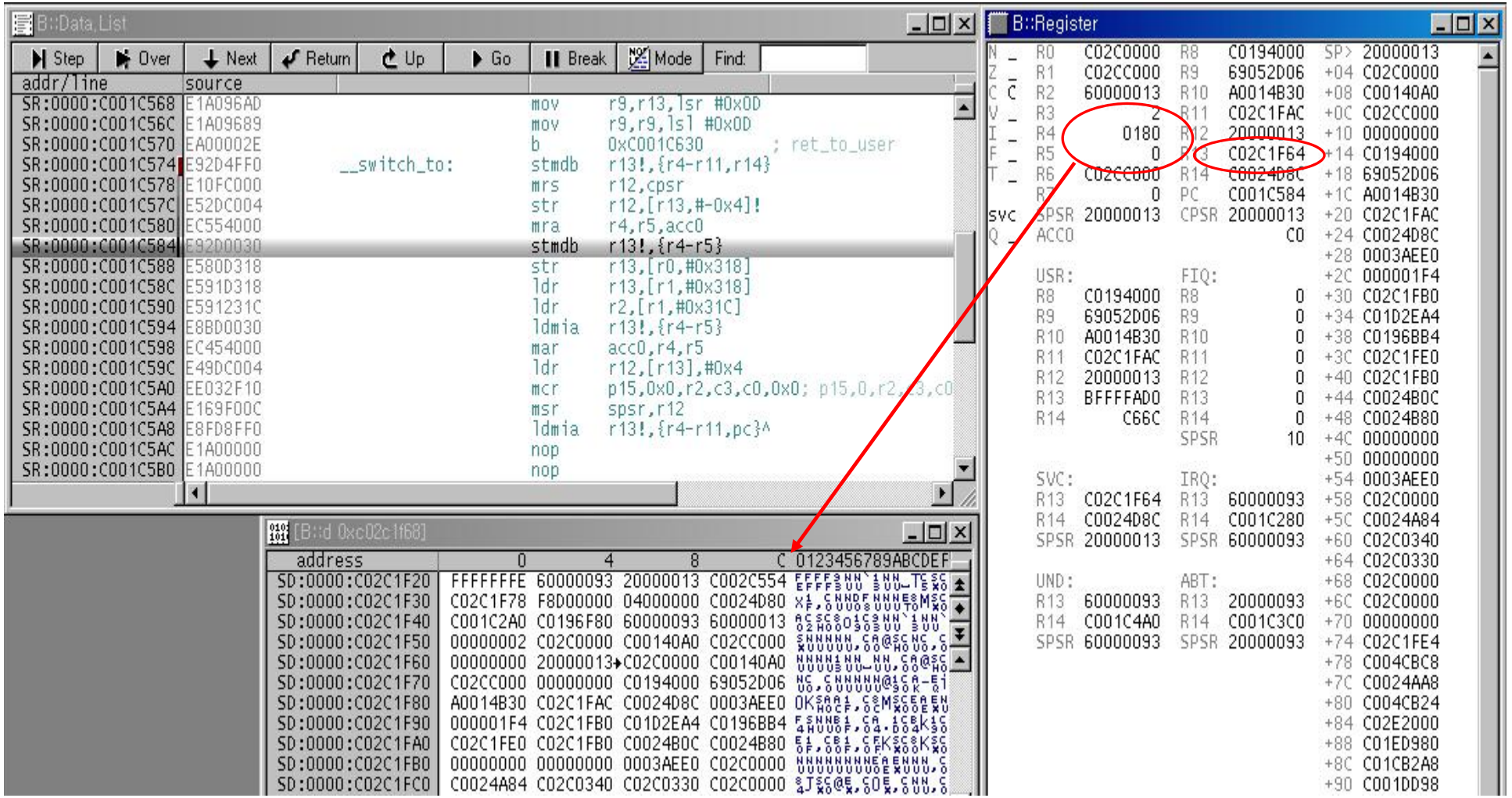
__switch_to() in ARM (3/16)



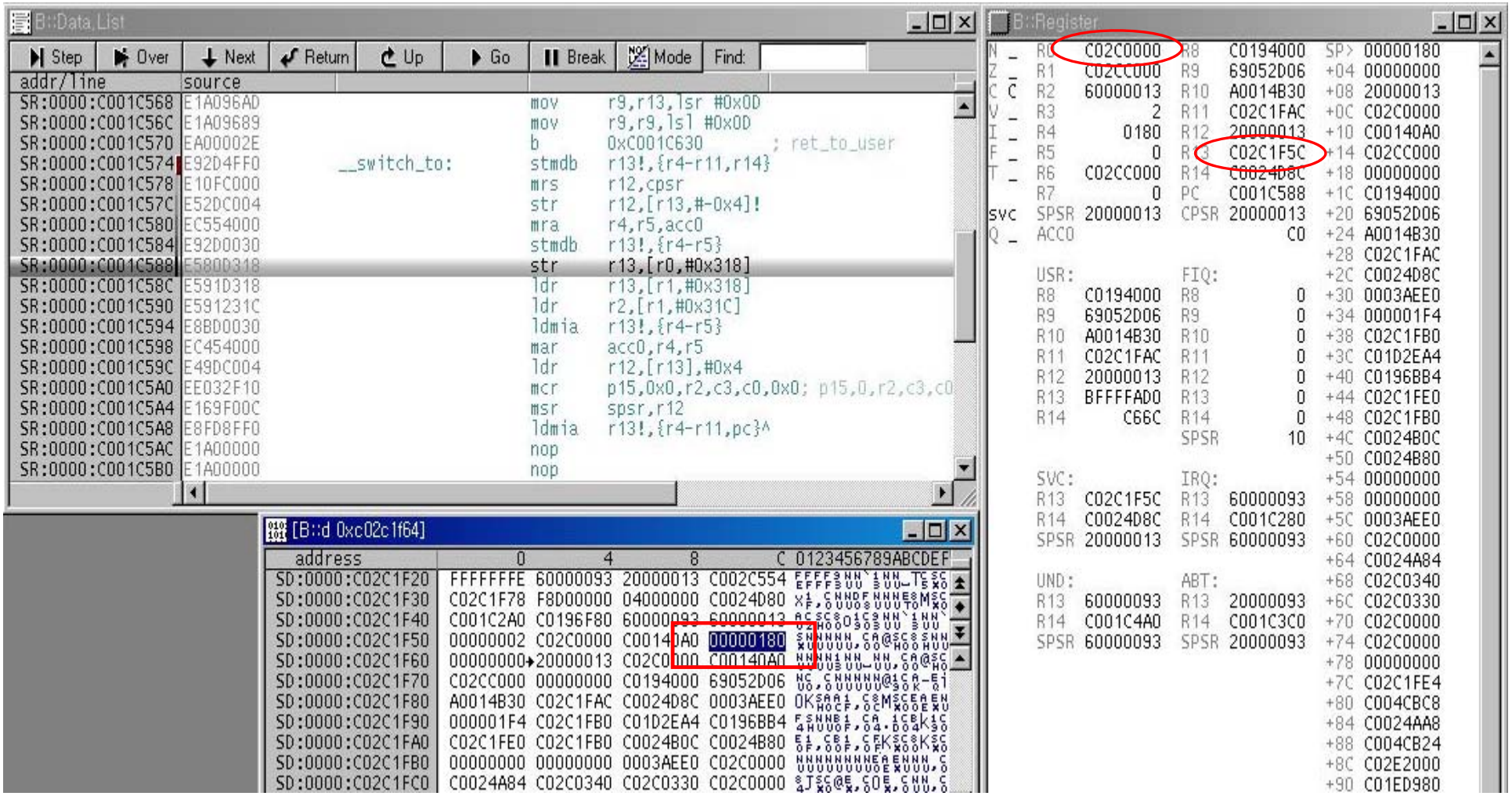
__switch_to() in ARM (4/16)



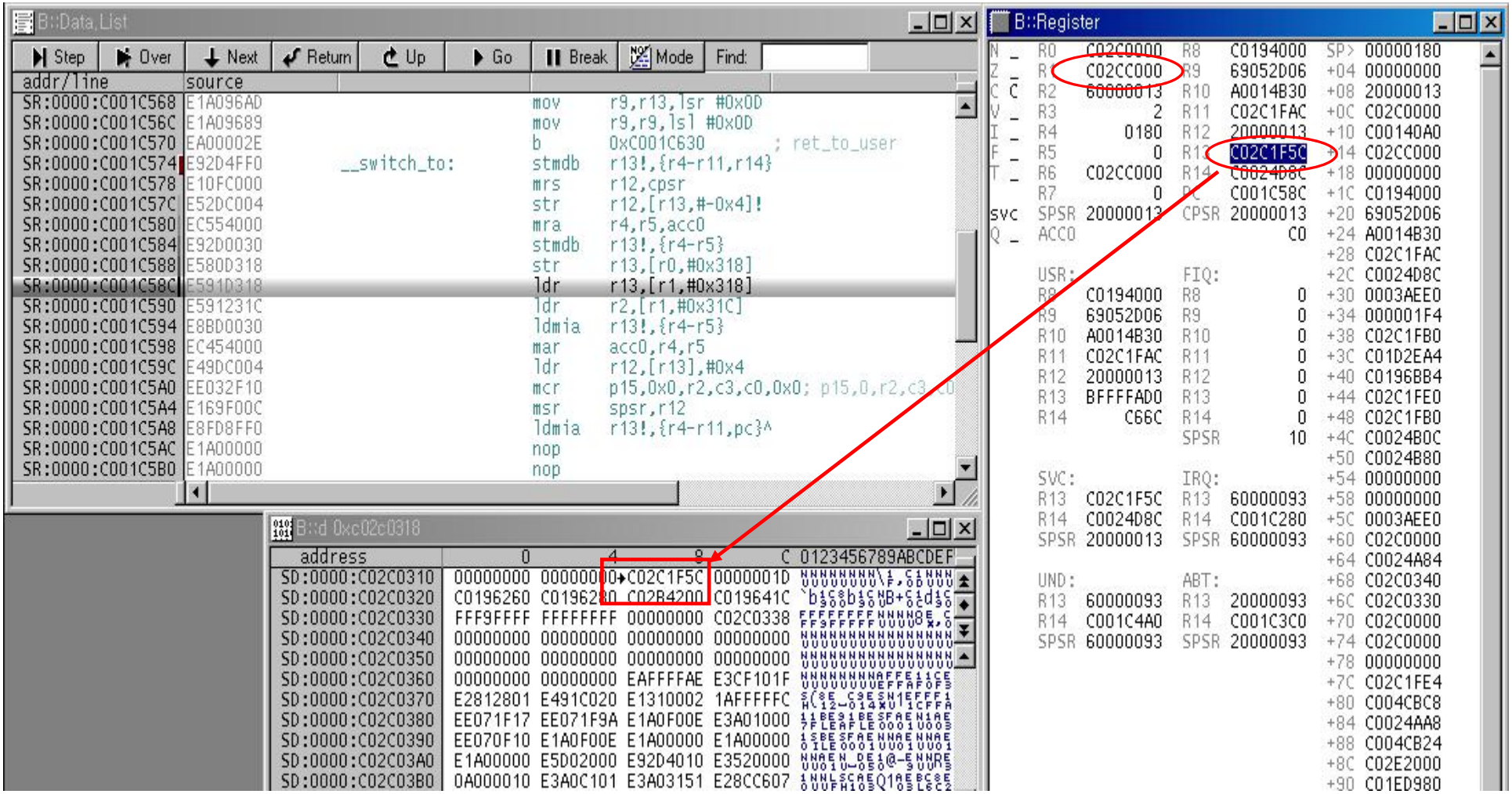
__switch_to() in ARM (5/16)



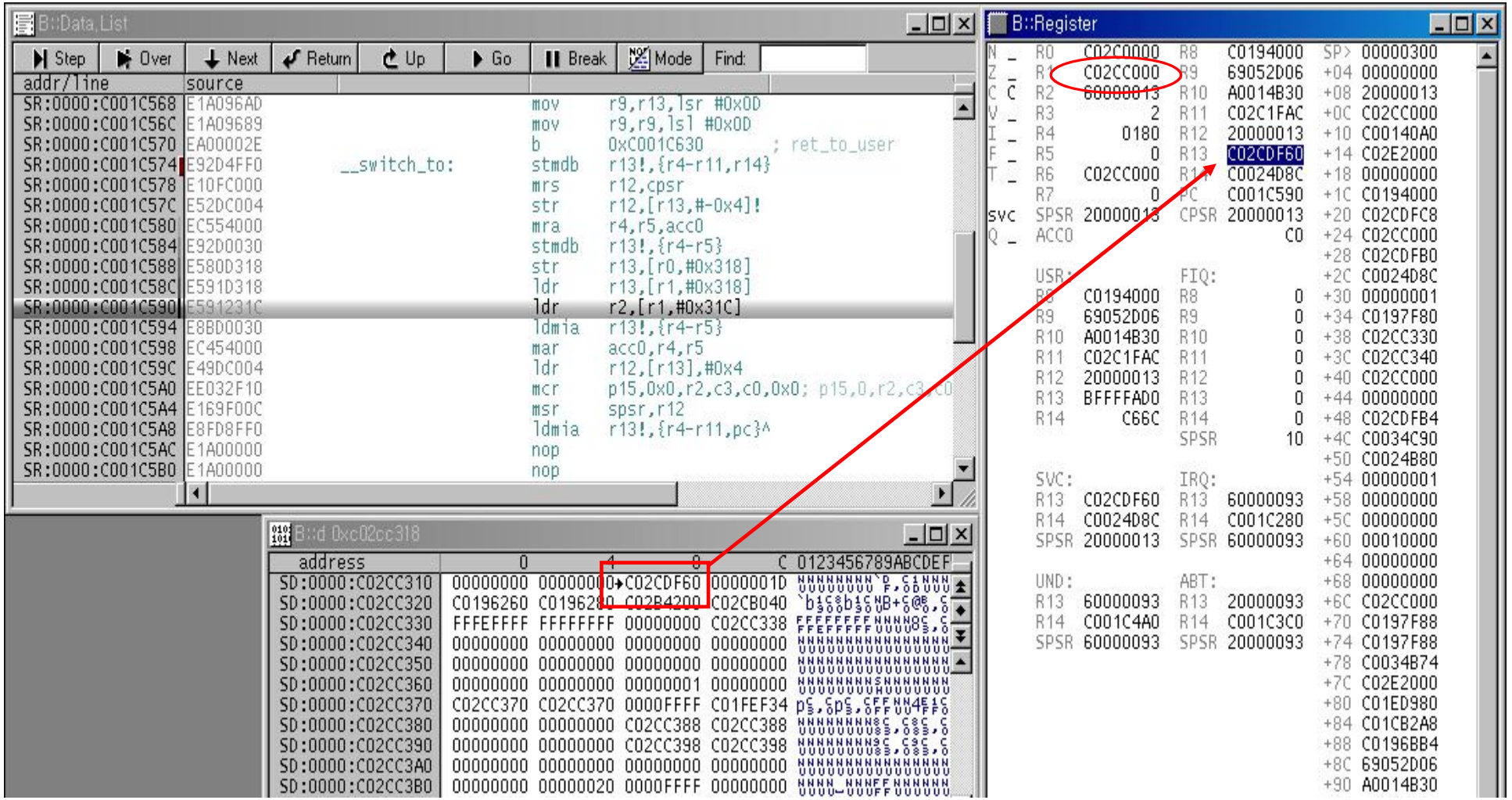
__switch_to() in ARM (6/16)



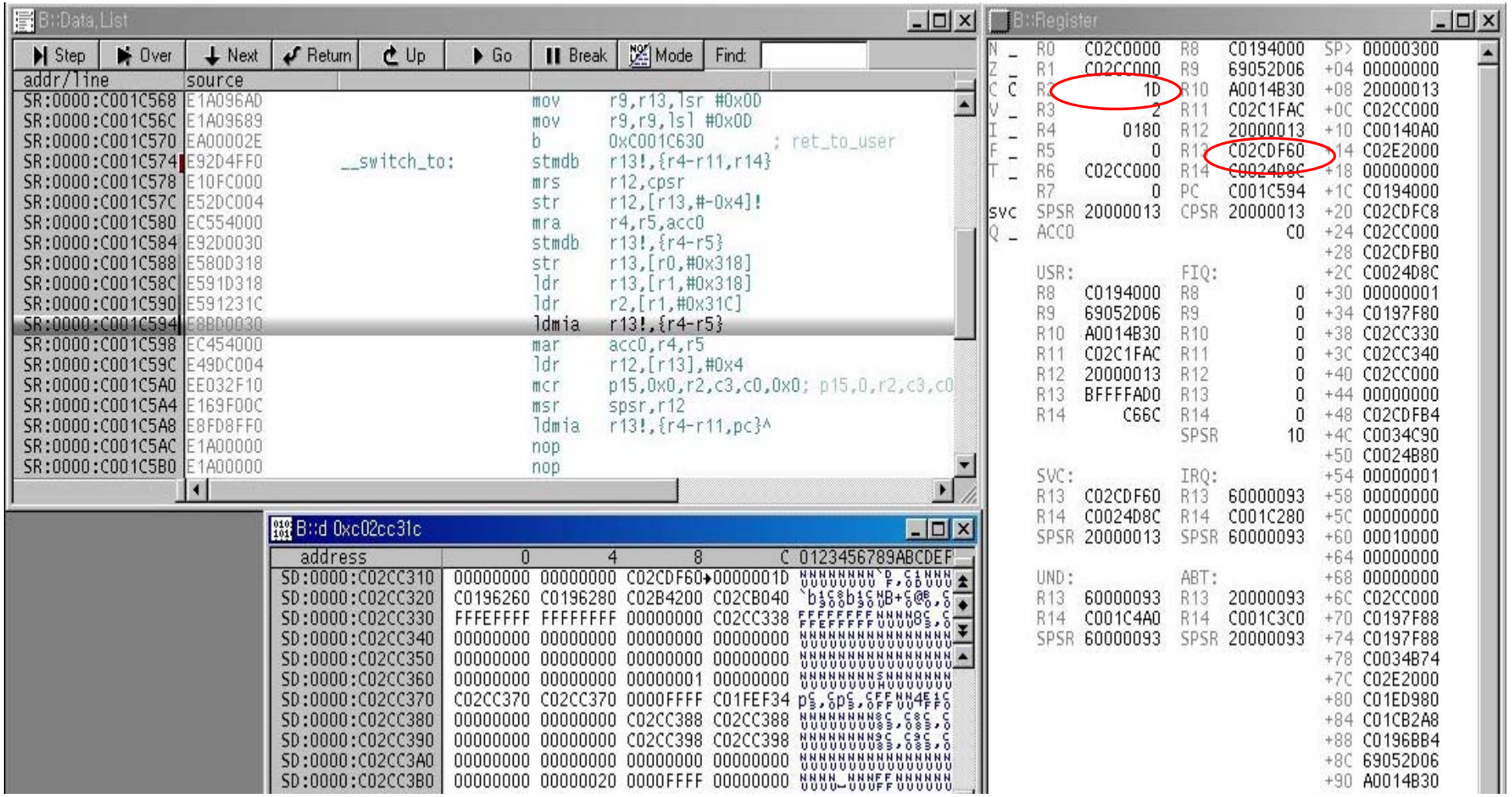
__switch_to() in ARM (7/16)



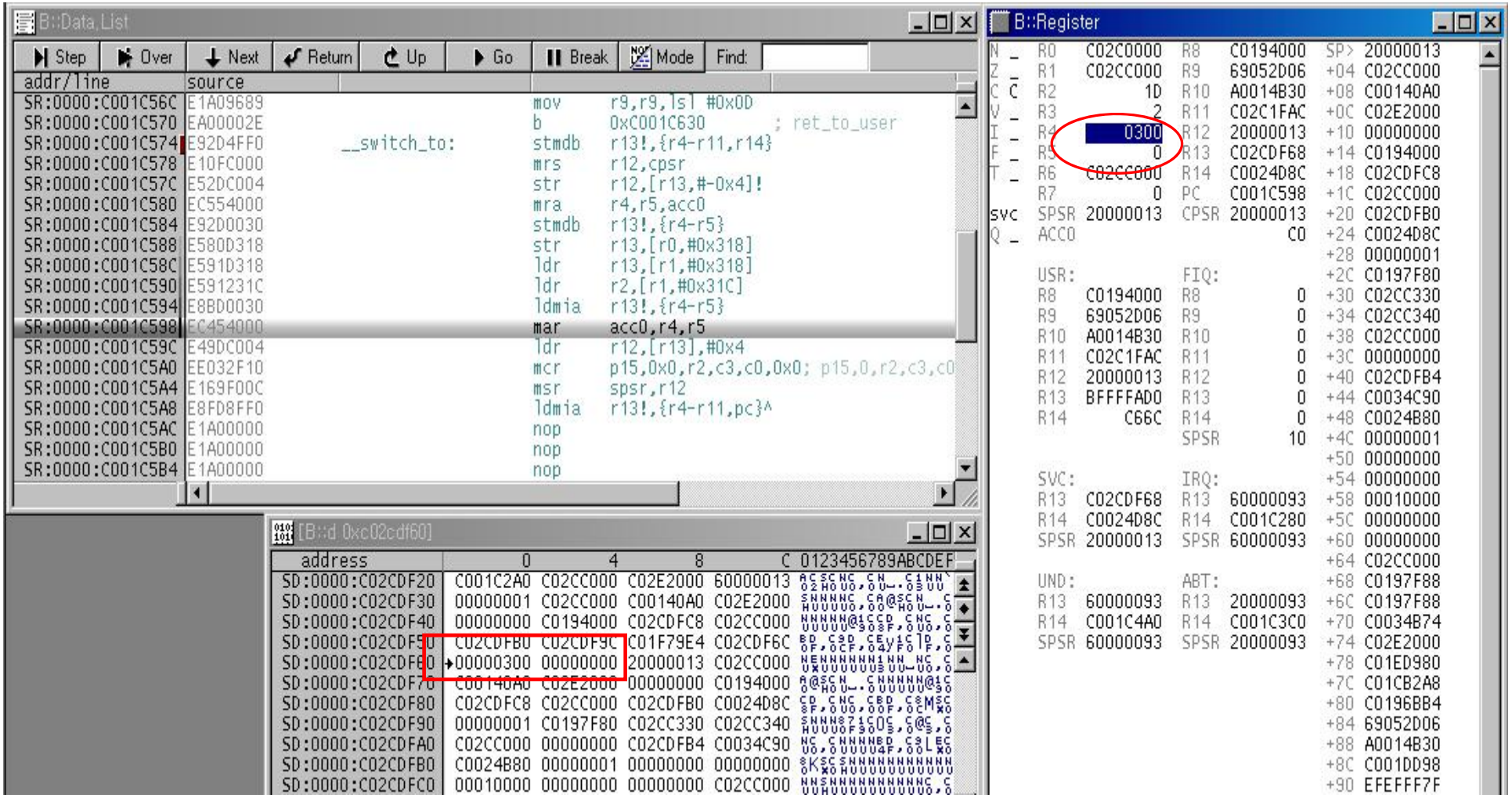
__switch_to() in ARM (8/16)



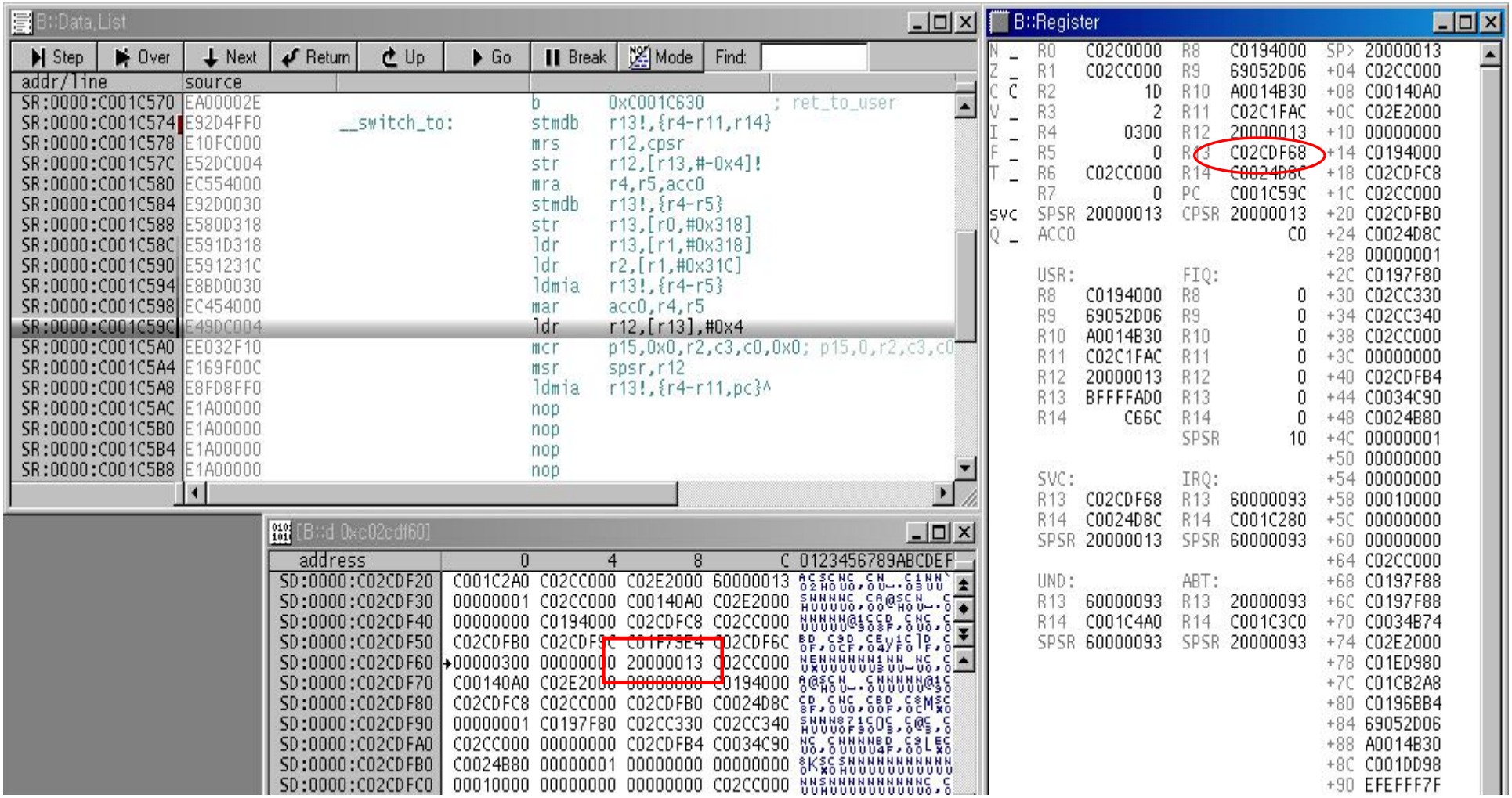
__switch_to() in ARM (9/16)



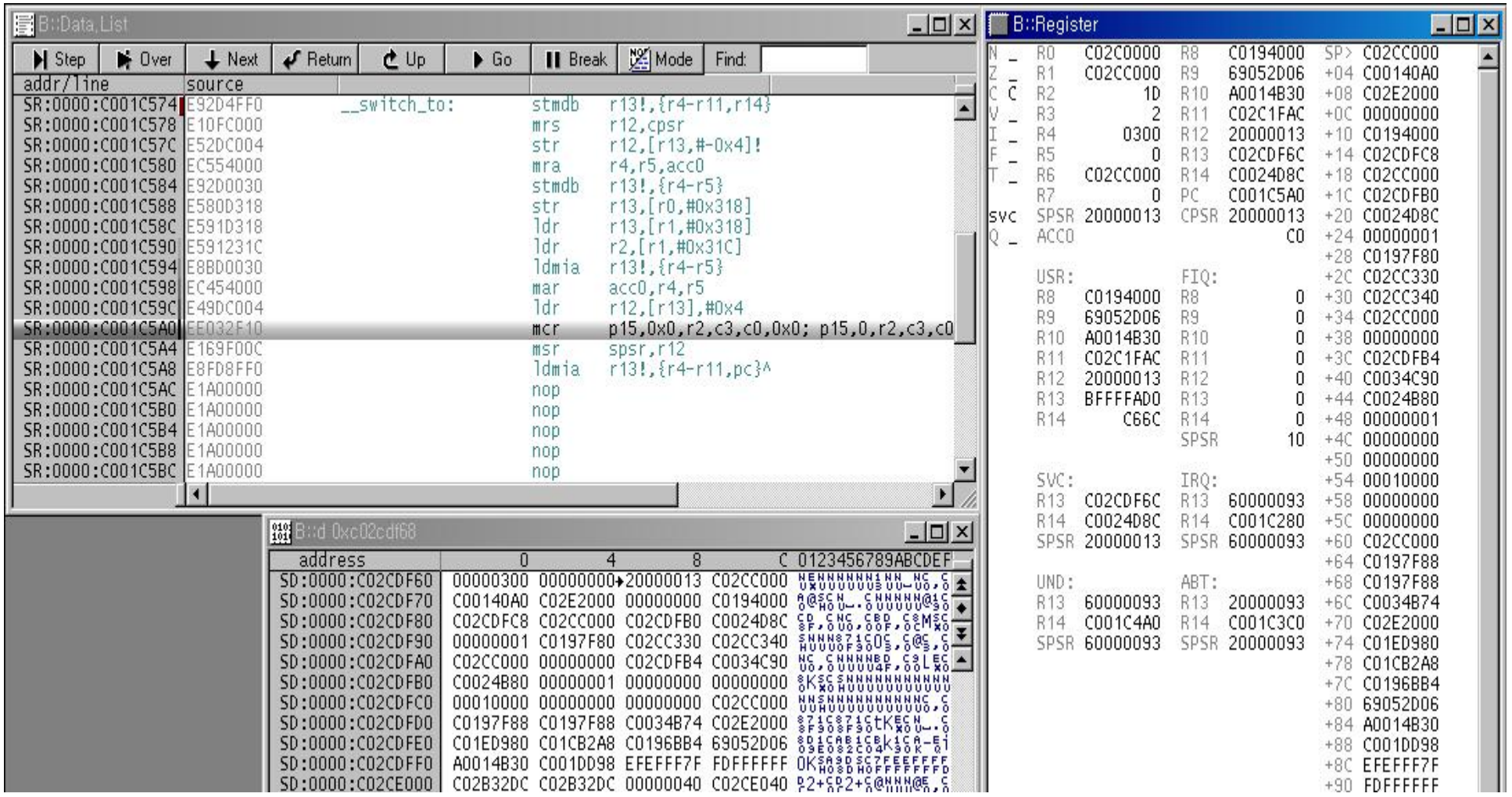
__switch_to() in ARM (10/16)



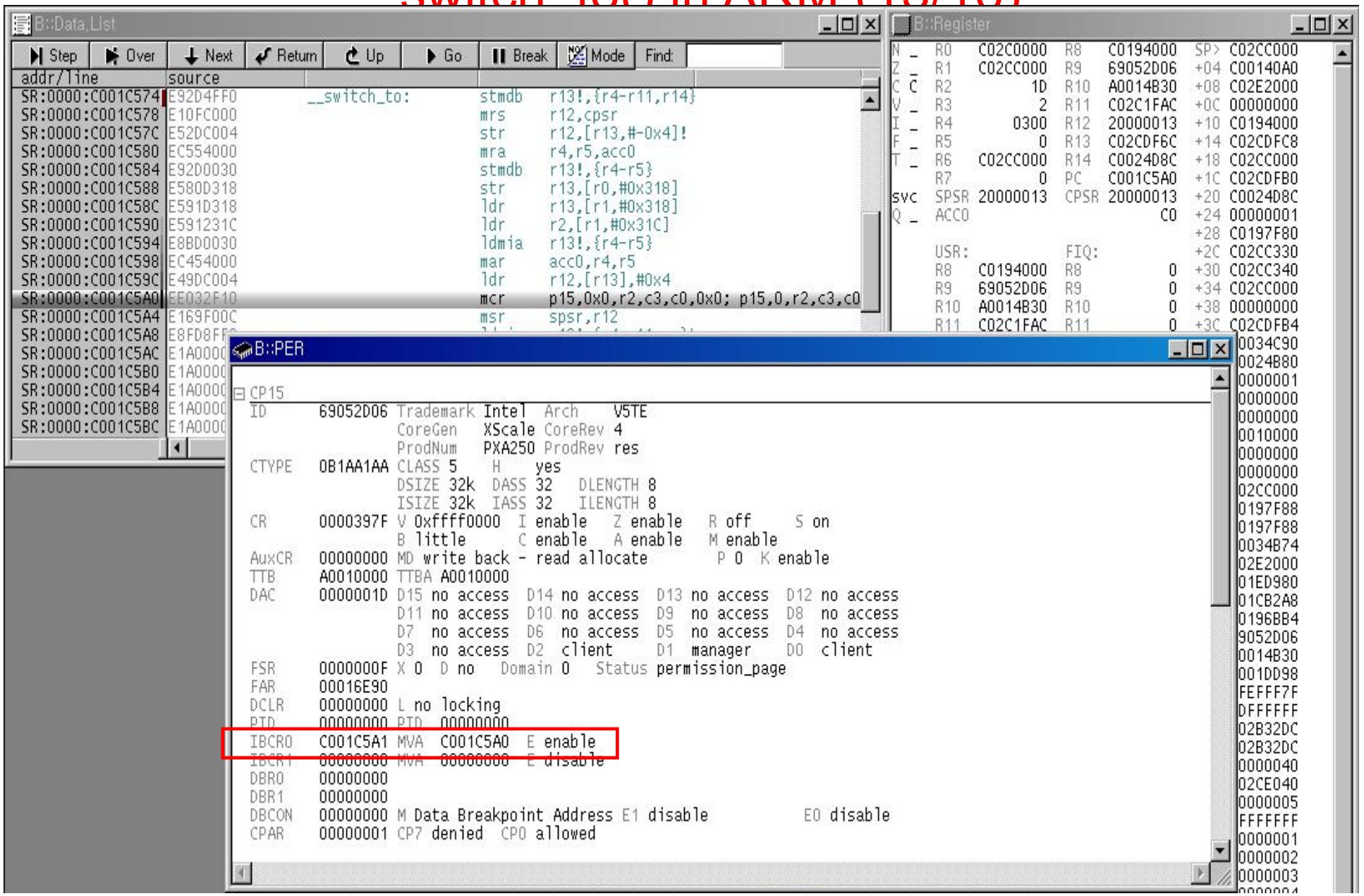
__switch_to() in ARM (11/16)



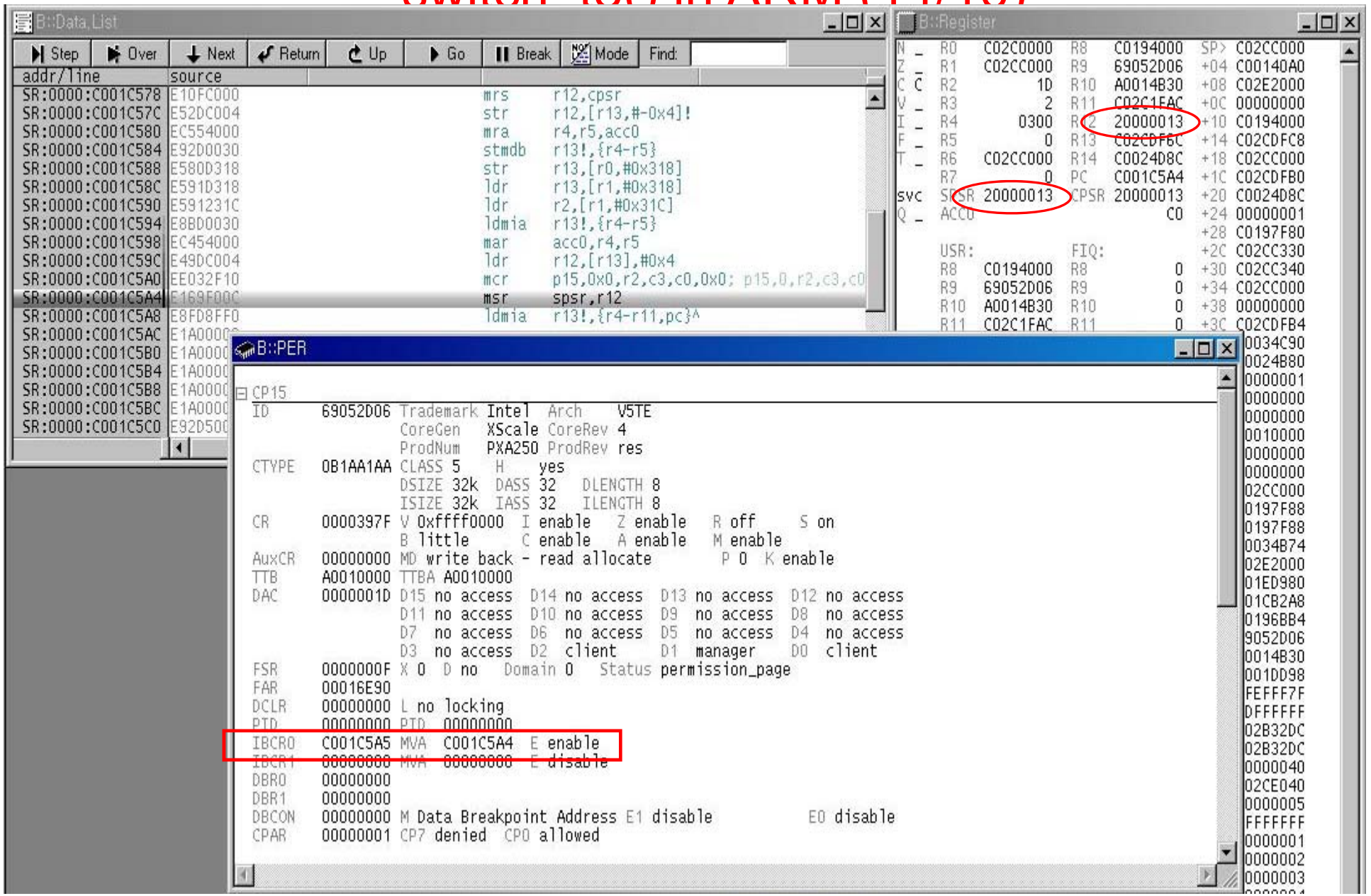
__switch_to() in ARM (12/16)



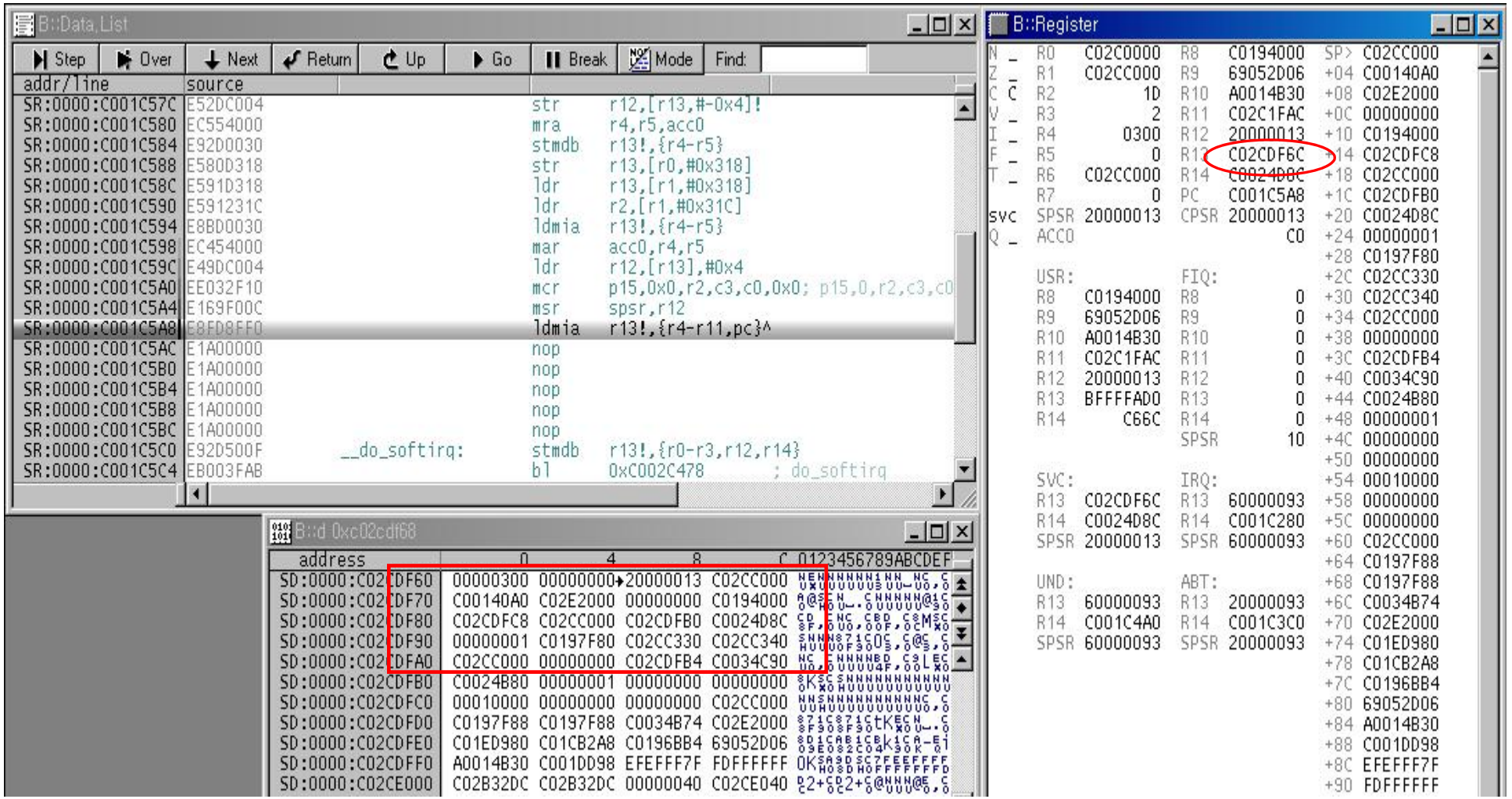
switch_to() in ARM (13/16)



switch_to() in ARM (14/16)



__switch_to() in ARM (15/16)



__switch_to() in ARM (16/16)