User-level threads & Scheduler Trace

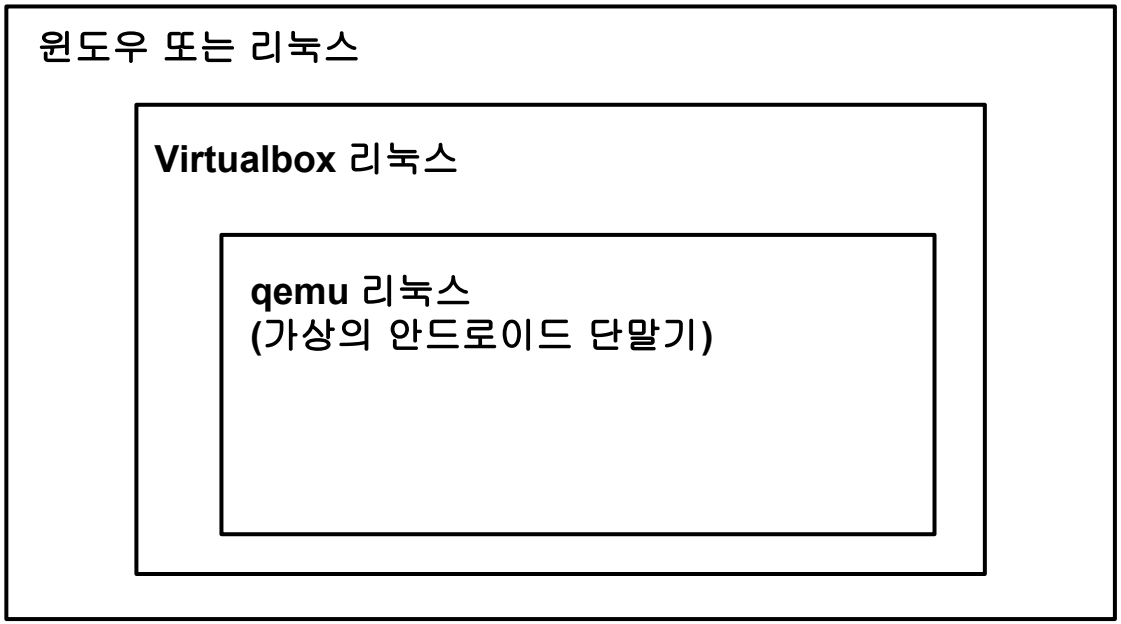
- 개발 환경 설정 및 실습 -

국민대학교 임베디드 연구실
경주현

Outline

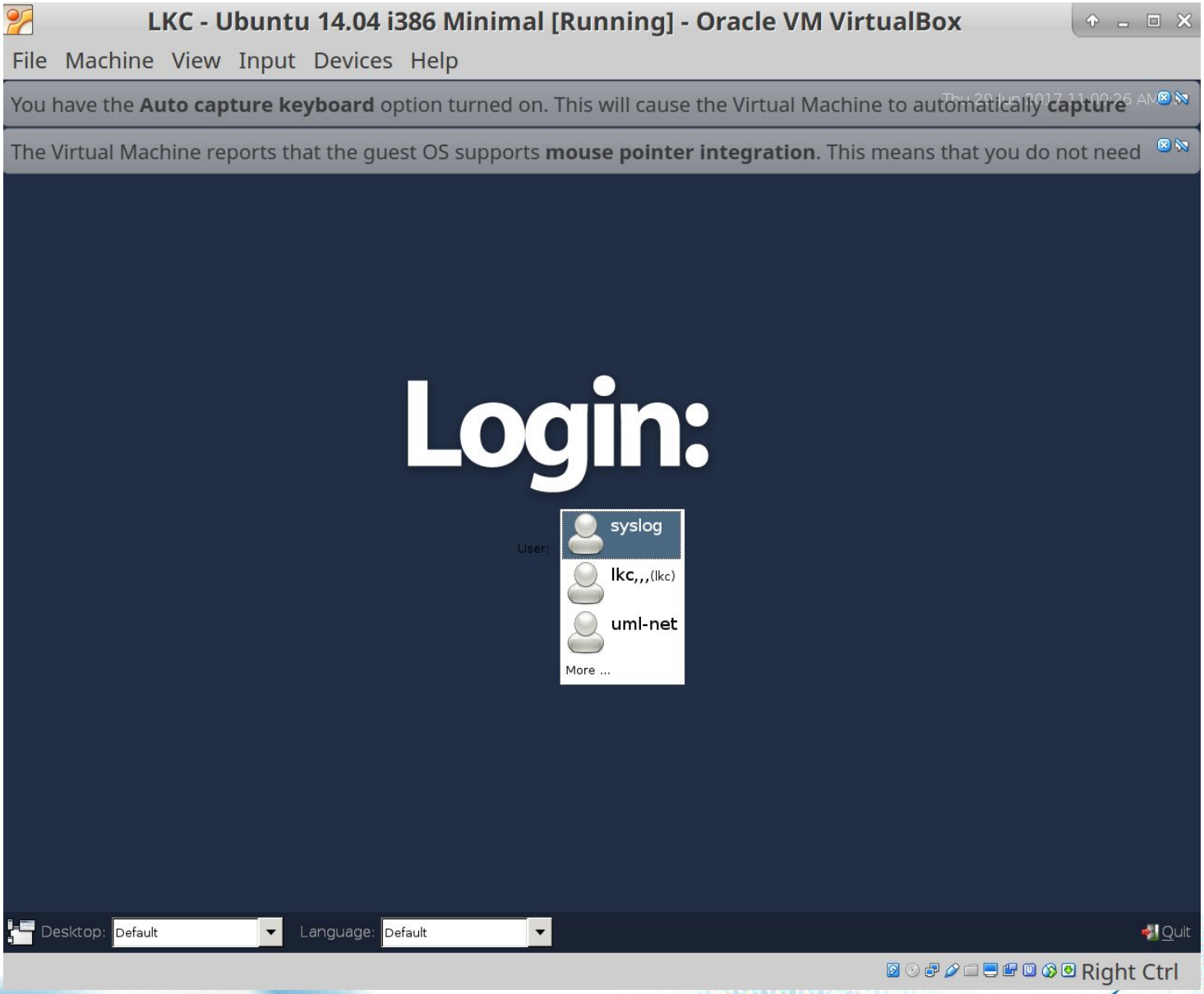
- 개발환경 설정
- 트레이스 실습
- **User-level threads**
- **User-level threads** 트레이스 실습

개발환경



스케줄러 개발 환경

- **Virtual Box 설치**
- 리눅스 이미지 다운로드
  https://goo.gl/DMtLp8
- 실행


LKC - Ubuntu 14.04 i386 Minimal [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
You have the Auto capture keyboard option turned on. This will cause the Virtual Machine to automatically capture
The Virtual Machine reports that the guest OS supports mouse pointer integration. This means that you do not need
Login:
User:
syslog
lkc,,,(lkc)
uml-net
More ...
Desktop: Default
Language: Default
Quit
Right Ctrl

SSH를 이용하여 로그인

- ssh lkc@localhost -p 22022
- ID : lkc
- PW : lkc201^

커널 소스 위치

- **/home/lkc/Desktop/linux-linaro-stable**

- **Rootfs 위치**
  - /home/lkc/Desktop/linux-linaro-stable/rootfs/initrd

커널 컴파일

- **cd ~/Desktop/linux-linaro-stable**
- **make -j2 bzImage**

```
lkc@lkc-VM:~/Desktop/linux-linaro-stable$ make -j2 bzImage
  CHK     include/config/kernel.release
  CHK     include/generated/uapi/linux/version.h
  CHK     include/generated/utsrelease.h
  CHK     include/generated/timeconst.h
  CHK     include/generated/bounds.h
  CHK     include/generated/asm-offsets.h
  CALL    scripts/checksyscalls.sh
  CHK     include/generated/compile.h
Kernel: arch/x86/boot/bzImage is ready  (#1)
```

rootfs 컴파일

- cd ~/Desktop/linux-linaro-stable
- make rootfs

```
lkc@lkc-VM:~/Desktop/linux-linaro-stable$ make rootfs
11725 blocks
make: Nothing to be done for `rootfs'.
lkc@lkc-VM:~/Desktop/linux-linaro-stable$
```

커널 실행

- **cd ~/Desktop/linux-linaro-stable**
- **./prepare.sh**
- **make qemu**

```
lkc@lkc-VM:~/Desktop/linux-linaro-stable$ pwd
/home/lkc/Desktop/linux-linaro-stable
lkc@lkc-VM:~/Desktop/linux-linaro-stable$ make qemu
../qemu/i386-softmmu/qemu-system-i386 -smp 2 -kernel arch/x86/boot/bzImage  -initrd
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Initializing cgroup subsys cpuacct
[    0.000000] Linux version 4.4.71 (lkc@lkc-VM) (gcc version 4.8.4 (Ubuntu 4.8.4-2u
[    0.000000] x86/fpu: Legacy x87 FPU detected.
[    0.000000] x86/fpu: Using 'lazy' FPU context switches.
[    0.000000] e820: BIOS-provided physical RAM map:
[    0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000009fbff] usable
[    0.000000] BIOS-e820: [mem 0x000000000009fc00-0x000000000009ffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000000f0000-0x00000000000fffff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000000100000-0x0000000007fdffff] usable
[    0.000000] BIOS-e820: [mem 0x0000000007fe0000-0x0000000007ffffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000fffc0000-0x00000000ffffffff] reserved
[    0.000000] Notice: NX (Execute Disable) protection missing in CPU!
[    0.000000] SMBIOS 2.8 present.
[    0.000000] e820: last_pfn = 0x7fe0 max_arch_pfn = 0x100000
[    0.000000] MTRR: Disabled
```

Qemu key commands

- **Ctrl + a, x : qemu 종료.**
- **Ctrl + a, c : qemu mode와 os mode로 전환**
- **info : 실행되고 있는 환경에서의 정보를 볼 수 있음**
  - info pg : 실행되고 있는 page table 정보(실습을 위해 추가함)
  - info cpus : 실행되고 있는 cpu 정보

설정 확인

- 파일공유
  - VirtualBox linux <-> qemu linux
- VirtualBox linux **nfs** 디렉토리 위치
  - /nfs
- qemu linux **nfs** 디렉토리 위치
  - /nfs
- 파일 생성 후 타겟에서 확인
  - VirtualBox linux$ touch /nfs/test
  - qemu linux$ ls /nfs/

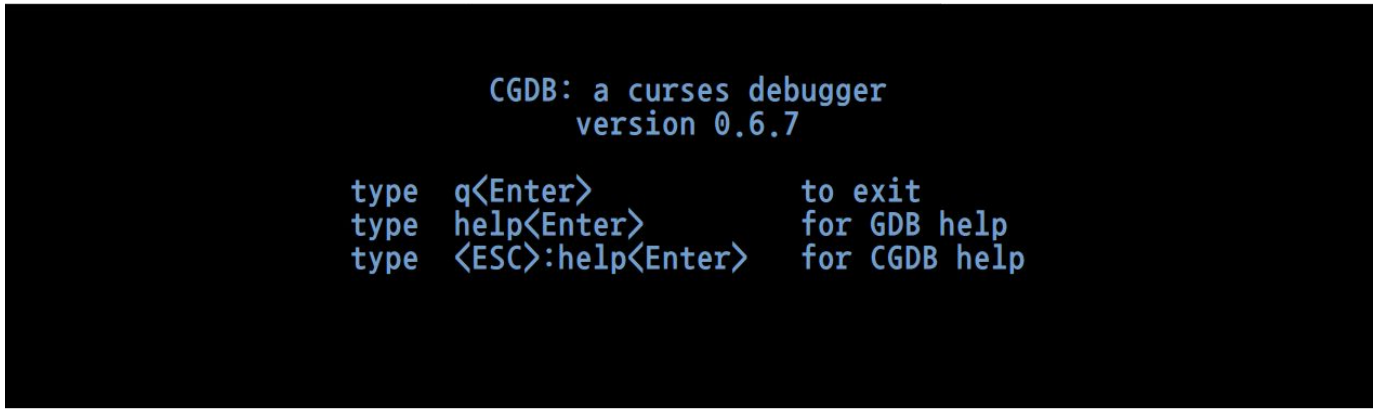
kernel + gdb stub 실행

- **make qemu-gdb**

```
lkc@lkc-VM:~/Desktop/linux-linaro-stable$ make qemu-gdb
*** Now run 'gdb'.
../qemu/i386-softmmu/qemu-system-i386 -smp 1 -kernel arch/x86/boot/bzImage  -ini
trd rootfs/bin/initrd.cpio -nographic -append "console=ttyS0 noapic ip=dhcp" -de
vice e1000,netdev=network0,mac=52:55:00:d1:55:01 -netdev tap,id=network0,ifname=
tap0,script=no,downscript=no  -s -S
```

gdb client 실행

- cgdb




```
Type "apropos word" to search for commands related to "word".
+ target remote localhost:1234
warning: A handler for the OS ABI "GNU/Linux" is not built into this configurat
on
of GDB.  Attempting to continue with the default i8086 settings.

The target architecture is assumed to be i8086
[f000:fff0]    0xffff0: ljmp   $0xf000,$0xe05b
0x0000fff0---Type <return> to continue, or q <return> to quit---
(gdb)
(gdb)
```

커널 디버깅

- **GDB Key Commands**
  - b + simbol_name : breakpoint
  - n : next step
  - si : next instruction step
  - ctrl + c : stop
  - c : continue

- 커널 디버깅을 위한 **GDB Commands** 실습

  1. make qemu--gdb
  2. break 포인트 설정 (start_kernel)
  3. single step (s)
  4. backtrace (bt)

b start_kernel

- 커널 디버깅 실습
- **b start_kenrel**
- **c**

```
489         pgtable_init();
490         vmalloc_init();
491         ioremap_huge_init();
492 }
493
494 asmlinkage __visible void __init start_kernel(void)
495 {
496         char *command_line;
497         char *after_dashes;
498
499         /*
500          * Need to run as early as possible, to initialize the
/home/lkc/Desktop/linux-linaro-stable/init/main.c
(gdb) b start_knerel
Function "start_knerel" not defined.
Make breakpoint pending on future shared library load? (y or [n]) n
(gdb) b start_kernel
Breakpoint 1 at 0xc1b317fb: file init/main.c, line 495.
(gdb) c
Continuing.
[  60:c1b317fb]    0xc1b31dfb <rootwait_setup+27>:      pop    %bp

Breakpoint 1, start_kernel () at init/main.c:495
(gdb)
```

커널 트레이스

- **trace-cmd + kernelshark**

- **http://elinux.org/images/6/64/Elc2011_rostedt.pdf**

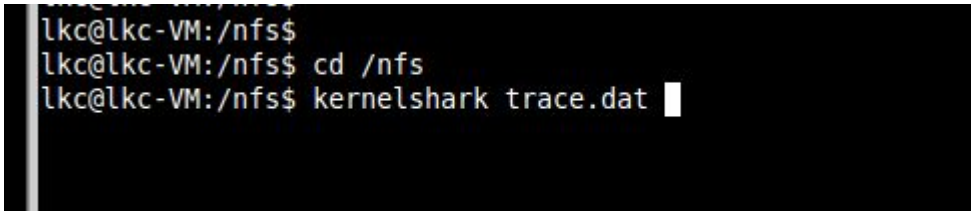
Scheduler 트레이스 - qemu linux

- **#>trace-cmd record -e sched <프로그램>**

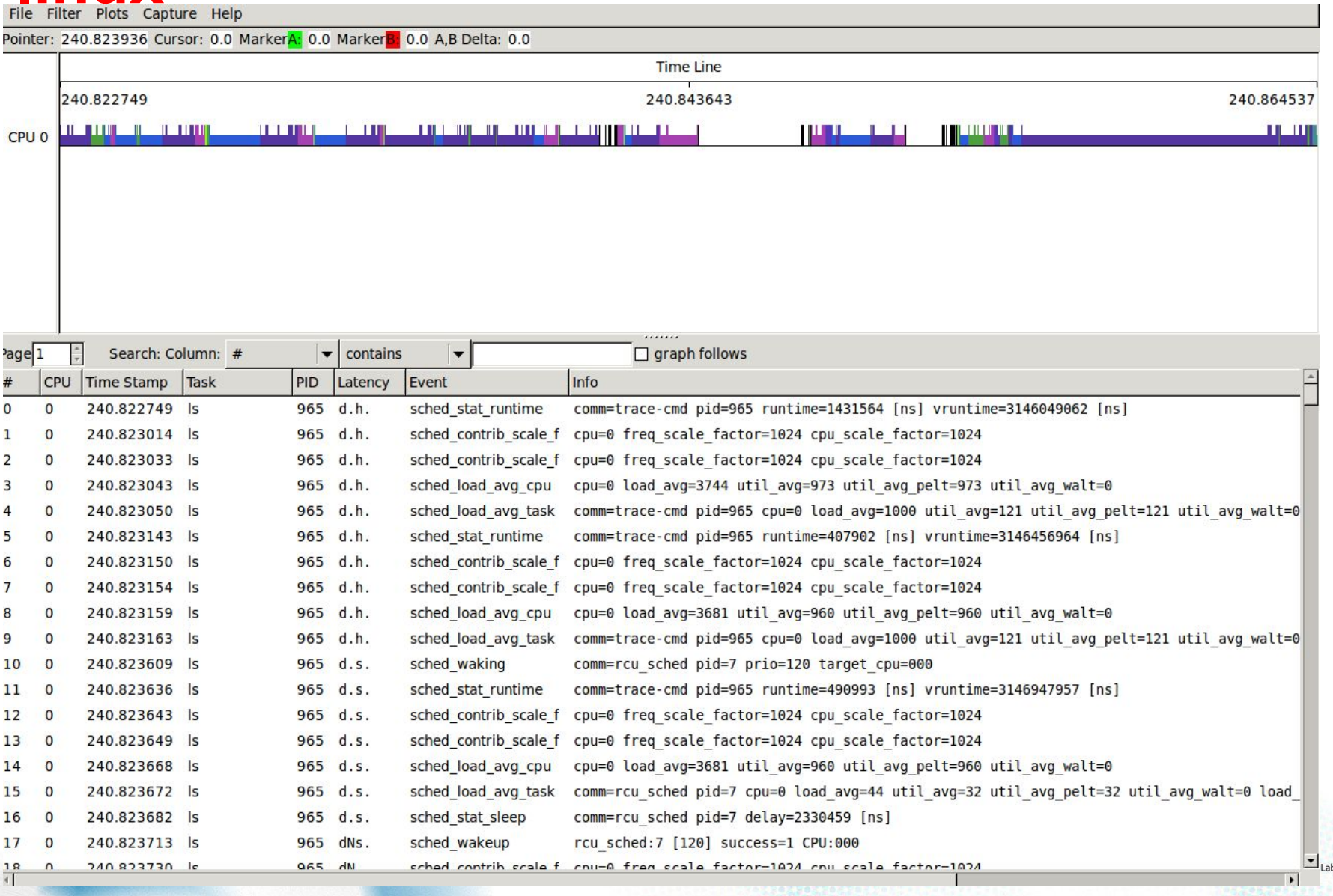
```
/ # ls
bin      init     nfs      root     sys
dev      linuxrc  proc     sbin     usr
/ # cd nfs
/nfs # ls
dsafas
/nfs # trace-cmd record -e sched ls
dsafas              trace.dat.cpu0  trace.dat.cpu1
```

Scheduler 트레이스 확인 - VirtualBox linux

- cd /nfs
- kernelshark trace.dat

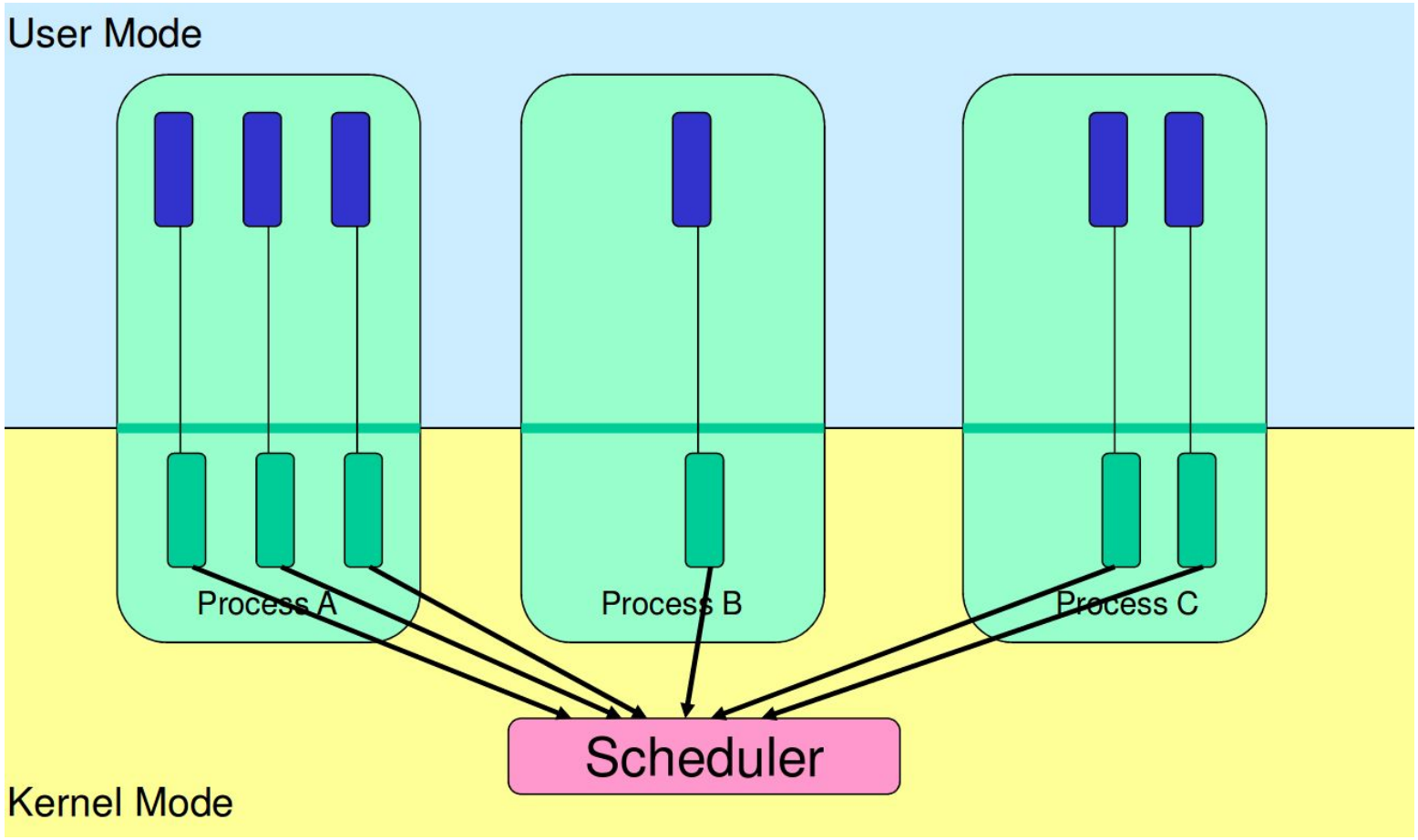


Scheduler 트레이스 확인 - VirtualBox linux



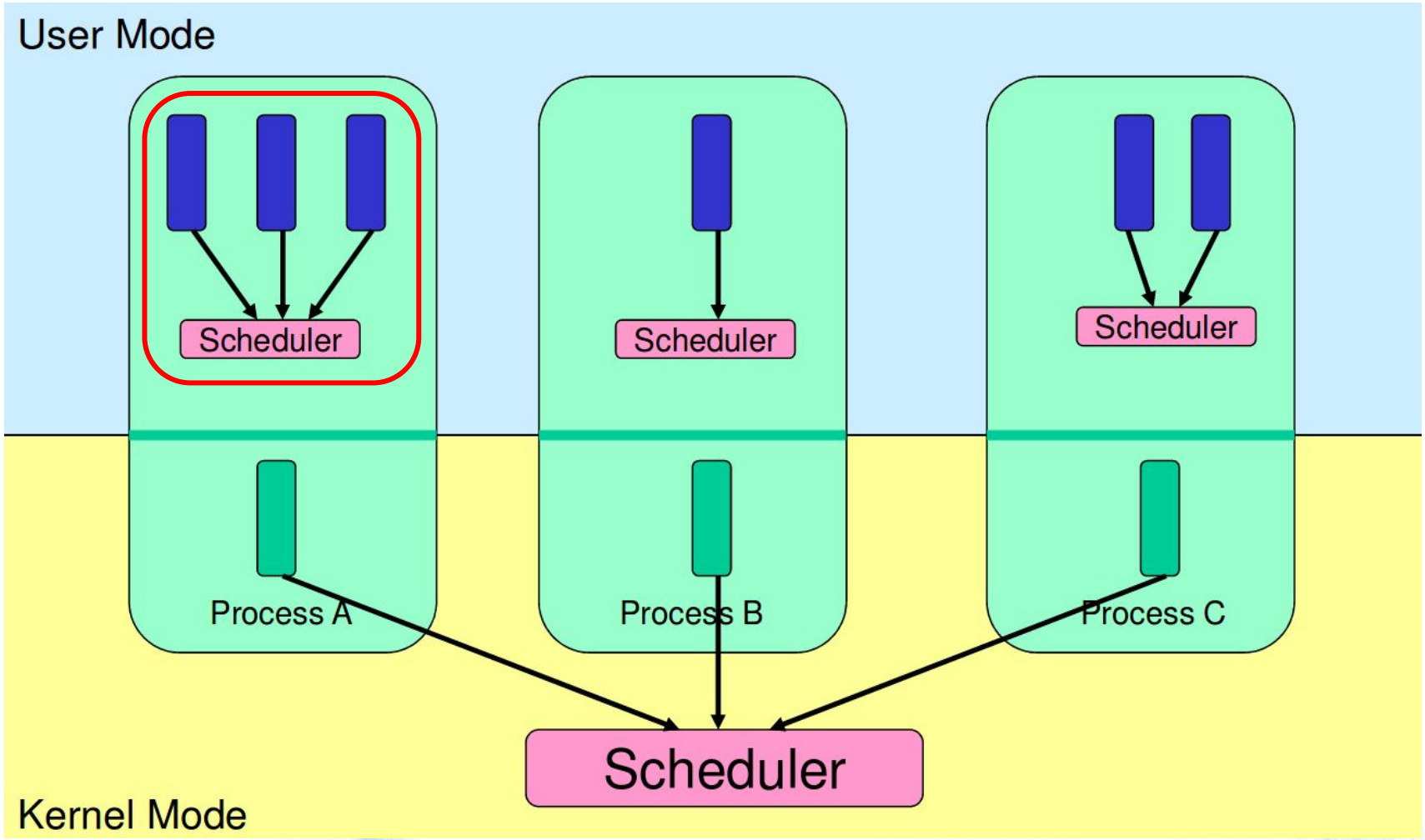
#	CPU	Time Stamp	Task	PID	Latency	Event	Info
0	0	240.822749	ls	965	d.h.	sched_stat_runtime	comm=trace-cmd pid=965 runtime=1431564 [ns] vruntime=3146049062 [ns]
1	0	240.823014	ls	965	d.h.	sched_contrib_scale_f	cpu=0 freq_scale_factor=1024 cpu_scale_factor=1024
2	0	240.823033	ls	965	d.h.	sched_contrib_scale_f	cpu=0 freq_scale_factor=1024 cpu_scale_factor=1024
3	0	240.823043	ls	965	d.h.	sched_load_avg_cpu	cpu=0 load_avg=3744 util_avg=973 util_avg_pelt=973 util_avg_walt=0
4	0	240.823050	ls	965	d.h.	sched_load_avg_task	comm=trace-cmd pid=965 cpu=0 load_avg=1000 util_avg=121 util_avg_pelt=121 util_avg_walt=0
5	0	240.823143	ls	965	d.h.	sched_stat_runtime	comm=trace-cmd pid=965 runtime=407902 [ns] vruntime=3146456964 [ns]
6	0	240.823150	ls	965	d.h.	sched_contrib_scale_f	cpu=0 freq_scale_factor=1024 cpu_scale_factor=1024
7	0	240.823154	ls	965	d.h.	sched_contrib_scale_f	cpu=0 freq_scale_factor=1024 cpu_scale_factor=1024
8	0	240.823159	ls	965	d.h.	sched_load_avg_cpu	cpu=0 load_avg=3681 util_avg=960 util_avg_pelt=960 util_avg_walt=0
9	0	240.823163	ls	965	d.h.	sched_load_avg_task	comm=trace-cmd pid=965 cpu=0 load_avg=1000 util_avg=121 util_avg_pelt=121 util_avg_walt=0
10	0	240.823609	ls	965	d.s.	sched_waking	comm=rcu_sched pid=7 prio=120 target_cpu=000
11	0	240.823636	ls	965	d.s.	sched_stat_runtime	comm=trace-cmd pid=965 runtime=490993 [ns] vruntime=3146947957 [ns]
12	0	240.823643	ls	965	d.s.	sched_contrib_scale_f	cpu=0 freq_scale_factor=1024 cpu_scale_factor=1024
13	0	240.823649	ls	965	d.s.	sched_contrib_scale_f	cpu=0 freq_scale_factor=1024 cpu_scale_factor=1024
14	0	240.823668	ls	965	d.s.	sched_load_avg_cpu	cpu=0 load_avg=3681 util_avg=960 util_avg_pelt=960 util_avg_walt=0
15	0	240.823672	ls	965	d.s.	sched_load_avg_task	comm=rcu_sched pid=7 cpu=0 load_avg=44 util_avg=32 util_avg_pelt=32 util_avg_walt=0 load_
16	0	240.823682	ls	965	d.s.	sched_stat_sleep	comm=rcu_sched pid=7 delay=2330459 [ns]
17	0	240.823713	ls	965	dNs.	sched_wakeup	rcu_sched:7 [120] success=1 CPU:000
18	0	240.823730	ls	965	dN	sched_contrib_scale_f	cpu=0 freq_scale_factor=1024 cpu_scale_factor=1024

Kernel-level thread

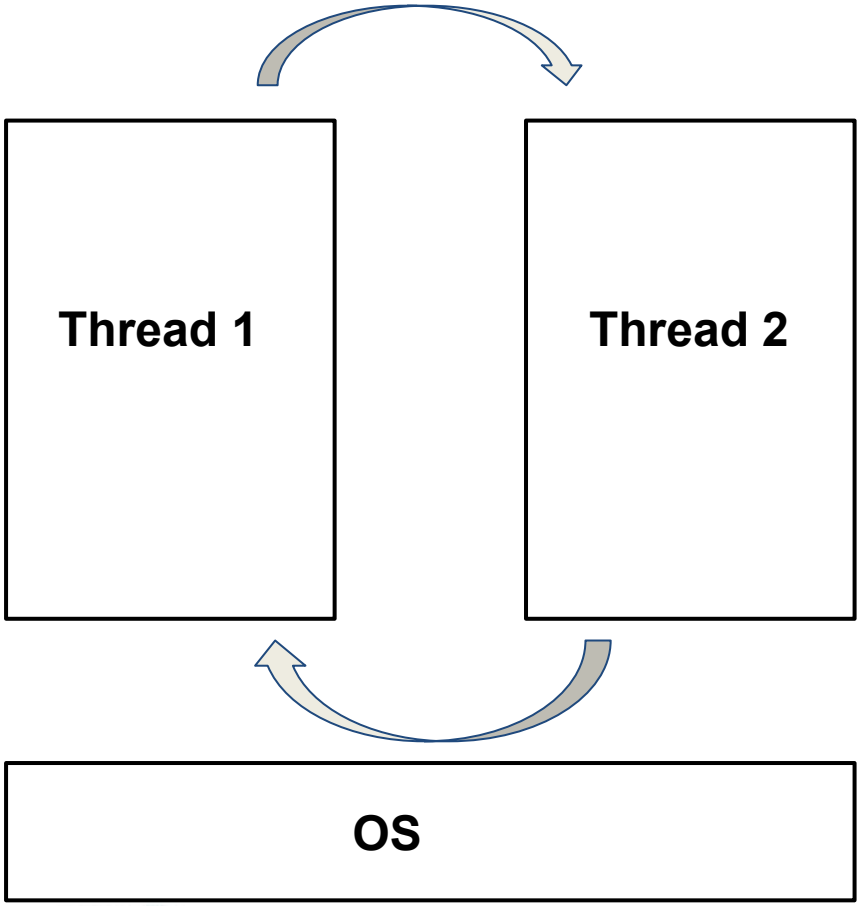


https://cgi.cse.unsw.edu.au/~cs3231/12s1/lectures/SchedulerActivations.pdf

User-level thread



User-level threads



https://cgi.cse.unsw.edu.au/~cs3231/12s1/lectures/SchedulerActivations.pdf
https://pdos.csail.mit.edu/6.828/2016/homework/xv6-uthread.html

Why talk about user-level threads?

User-level threads

- **C/W in User-level threads**
  - mimics modern kernel level C/W.

Source

- **/home/lkc/Desktop/linux-linaro-stable/rootfs/initrd/root/userlevel**
  - Makefile thread_switch.S uthread uthread.c

- **Compile**
  - make

- **run**
  - ./uthread

```
lkc@lkc-VM:~/Desktop/linux-linaro-stable/rootfs/initrd/root/userlevel$ ./uthread
my thread running
my thread 0x80ecfa8
my thread running
my thread 0x80eefb0
my thread 0x80ecfa8
my thread 0x80eefb0
my thread 0x80ecfa8
my thread 0x80eefb0
my thread 0x80ecfa8
my thread 0x80eefb0
my thread 0x80ecfa8
my thread 0x80eefb0
my thread 0x80ecfa8
my thread 0x80eefb0
my thread 0x80ecfa8
my thread 0x80eefb0
my thread 0x80ecfa8
```

How to implement an O/S?

Task structure & Task State

```
/* Possible states of a thread; */
#define FREE        0x0
#define RUNNING     0x1
#define RUNNABLE    0x2

#define STACK_SIZE  8192
#define MAX_THREAD  4

typedef struct thread thread_t, *thread_p;
typedef struct mutex mutex_t, *mutex_p;

struct thread {
  int         sp;                 /* curent stack pointer */
  char stack[STACK_SIZE];         /* the thread's stack */
  int         state;              /* FREE, RUNNING, RUNNABLE */
};
static thread_t all_thread[MAX_THREAD];
thread_p  current_thread;
thread_p  next_thread;
extern void thread_switch(void);
```

Schedule and Context switch

```
static void
thread_schedule(void)
{
  thread_p t;

  /* Find another runnable thread. */
  next_thread = 0;
  for (t = all_thread; t < all_thread + MAX_THREAD; t++) {
    if (t->state == RUNNABLE && t != current_thread) {
      next_thread = t;
      break;
    }
  }

  if (t >= all_thread + MAX_THREAD && current_thread->state == RUNNABLE) {
    /* The current thread is the only runnable thread; run it. */
    next_thread = current_thread;
  }

  if (next_thread == 0) {
    printf(2, "thread_schedule: no runnable threads\n");
    exit();
  }

  if (current_thread != next_thread) {         /* switch threads?  */
    next_thread->state = RUNNING;
    thread_switch();
  } else
    next_thread = 0;
}
```

Context Switch

- **struct thread**

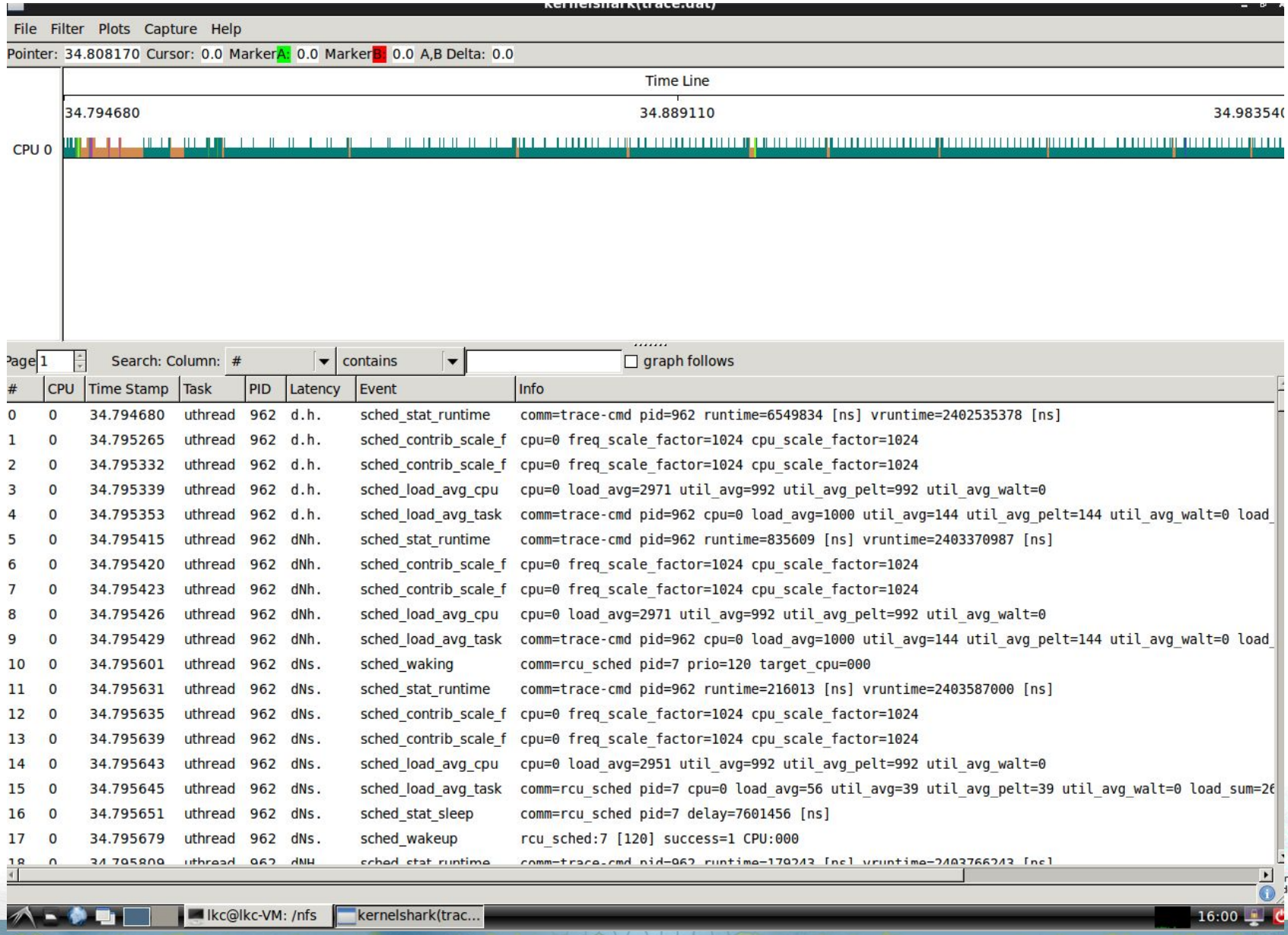
```
--------------------
| 4 bytes for state|
--------------------
| stack size bytes |
| for stack        |
--------------------
| 4 bytes for sp   |
--------------------  <--- current_thread
     ......

     ......
--------------------
| 4 bytes for state|
--------------------
| stack size bytes |
| for stack        |
--------------------
| 4 bytes for sp   |
--------------------  <--- next_thread
```

```
        .globl thread_switch
thread_switch:
        pushal
        movl current_thread, %eax
        movl %esp, (%eax)
        movl next_thread, %eax
        movl %eax, current_thread
        movl $0, next_thread
        movl current_thread, %eax
        movl (%eax), %esp
        popal

        ret
```

User-level threads trace 실습



User-level threads

- **Pro :**

  Fast thread management (creation, deletion, switching, synchronisation...)
- **Cons :**

  Blocking blocks all threads in a process
  - page fault
  - system call

  No thread-level parallelism on multiprocessor
  - Solution: Scheduler Activations
- **Real world**
  - M:N model, today google go language
  - https://intl.aliyun.com/forum/read-916
  - co-routines
    - https://www.youtube.com/watch?v=YYtzQ355_Co
    - http://www.gamedevforever.com/291

Next Step.

Energy-aware scheduling: EAS

1. **CFS scheduler - Kernel level**
2. **Load Balancer(Group Scheduling, Bandwidth Control, PELT)**
3. **EAS features**

Reference

- https://pdos.csail.mit.edu/6.828/2016/schedule.html
- http://web.mit.edu/6.033
- http://www.rdrop.com/~paulmck/
- "**Is Parallel Programming Hard, And If So, What Can You Do About It?"**
- Davidlohr Bueso. 2014. Scalability techniques for practical synchronization primitives. *Commun. ACM* 58

http://queue.acm.org/detail.cfm?id=2698990

- **"CPUFreq and The Scheduler Revolution in CPU Power Management", Rafael J. Wysocki**
- **https://sites.google.com/site/embedwiki/oses/linux/pm/pm-qos**
- **https://intl.aliyun.com/forum/read-916**
- **User-level threads : co-routines**

  http://www.gamedevforever.com/291

  https://www.youtube.com/watch?v=YYtzQ355_Co

- **Scheduler Activations**
  - https://cgi.cse.unsw.edu.au/~cs3231/12s1/lectures/SchedulerActivations.pdf
- **https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics)**
- **http://jake.dothome.co.kr/**
- **http://www.linuxjournal.com/magazine/completely-fair-scheduler?page=0,0**
- **https://www2.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/6_CPU_Scheduling.html**
- **"Energy Aware Scheduling", Byungchul Park, LG Electronic**
- **"Update on big.LITTLE scheduling experiments", ARM**
- **"EAS Update" 2015 september ARM**
- **"EAS Overview and Integration Guide", ARM TR**
- **"Drowsy Power Management", Matthew Lentz, SOSP 2015**
- **https://www.slideshare.net/nanik/learning-aosp-android-hardware-abstraction-layer-hal**
- https://www.youtube.com/watch?v=oTGQXqD3CNI
- https://www.youtube.com/watch?v=P80NcKUKpuo
- https://lwn.net/Articles/398470/
- "SCHED_DEADLINE: It's Alive!", ARM, 2017