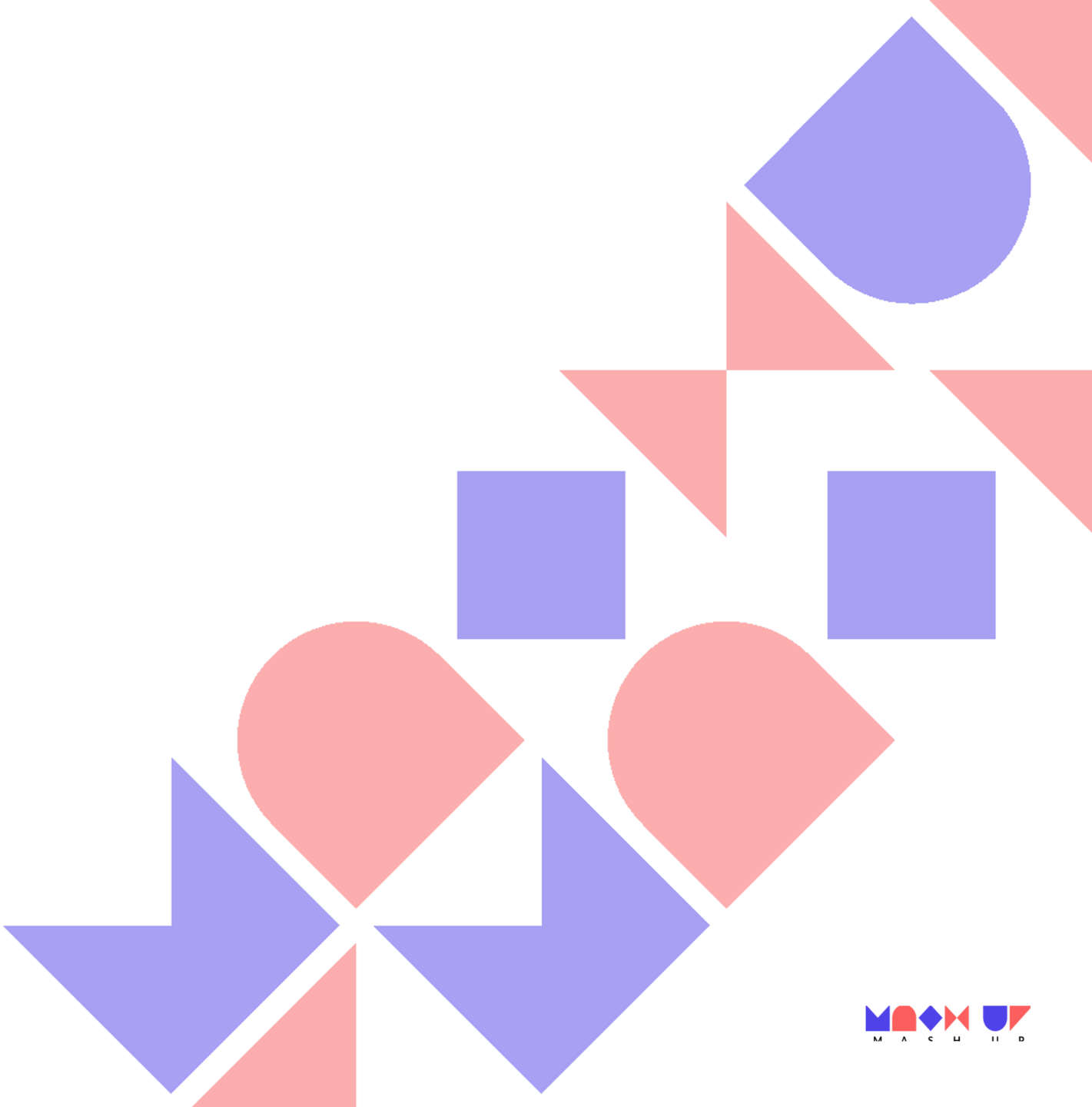
Backend Seminar

Interrupt & Context Switching

2020.09.19

김승현



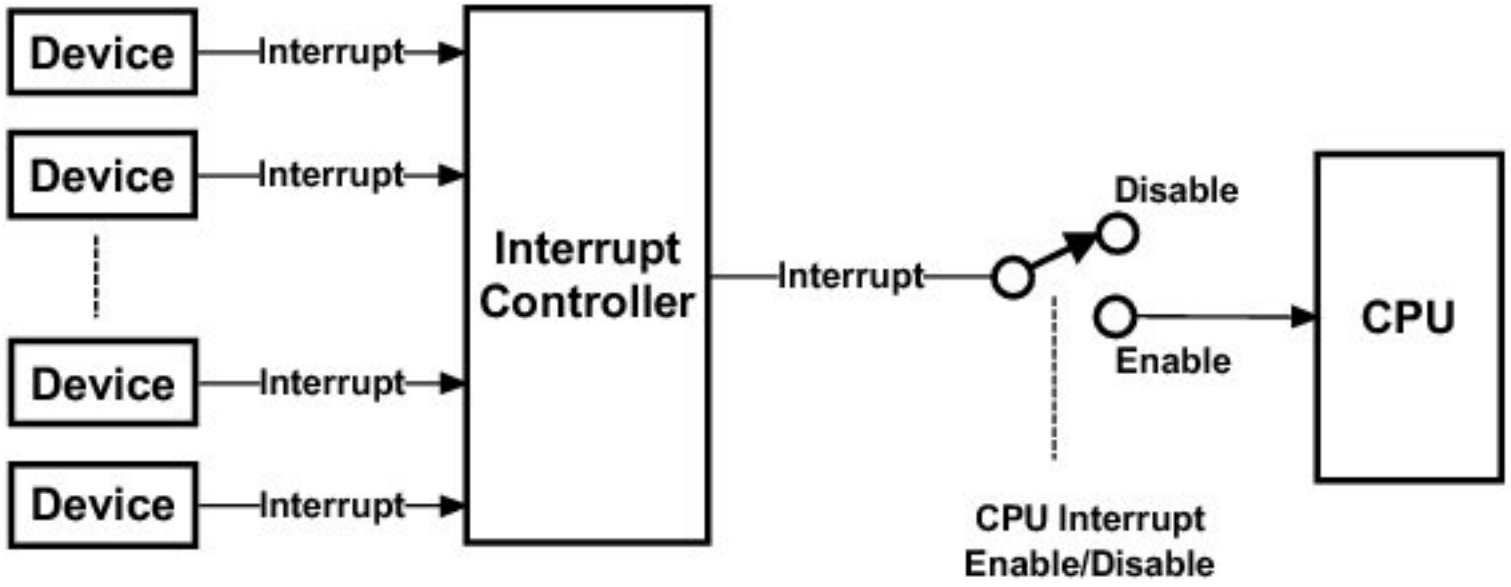
CONTENTS

1. Interrupt
2. Context Switching
3. Process & Thread

1.

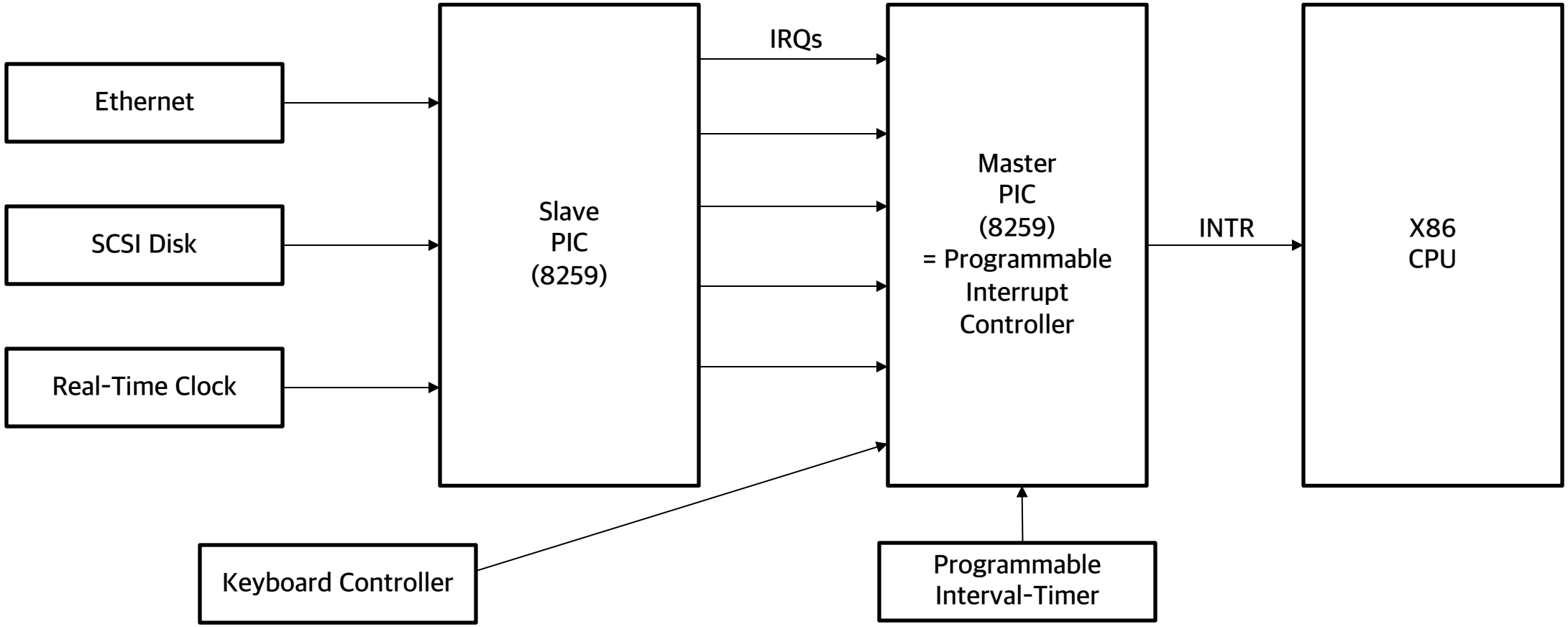
Interrupt

Interrupt

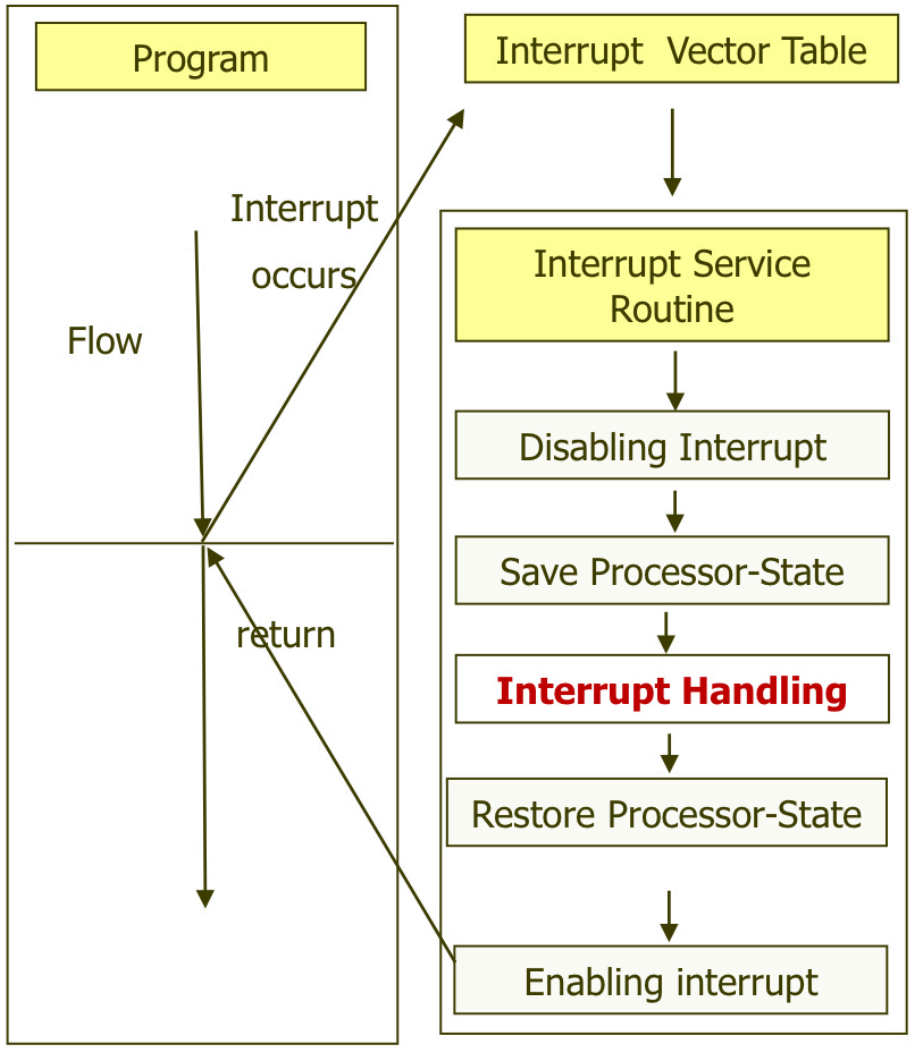


"Device Controller가 CPU에게 작업이 끝났음을 알리는 행위"

Interrupt


Ethernet
SCSI Disk
Real-Time Clock
Slave PIC (8259)
IRQs
Master PIC (8259) = Programmable Interrupt Controller
INTR
X86 CPU
Keyboard Controller
Programmable Interval-Timer

Interrupt



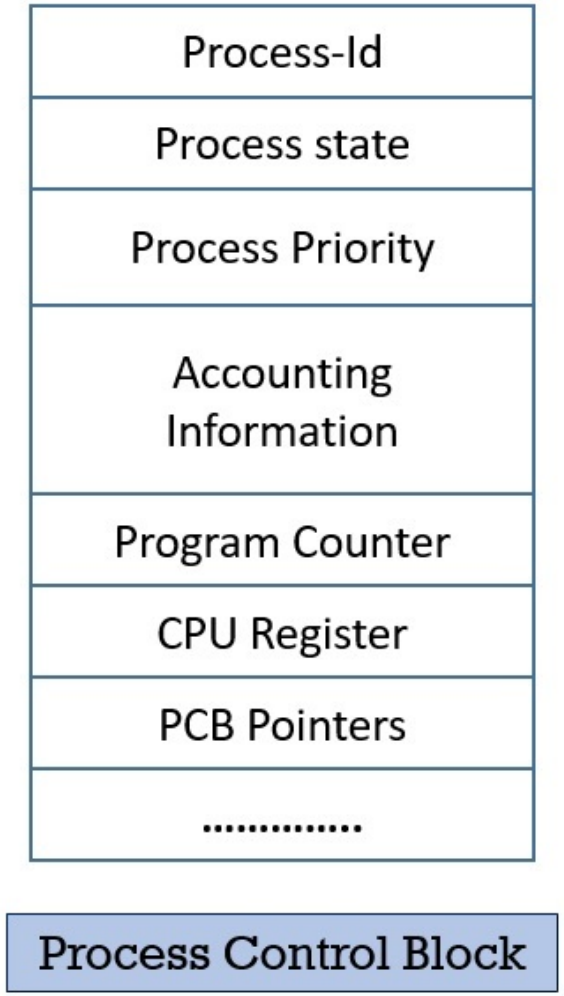
1. Interrupt는 제어를 Interrupt service routine으로 이동
2. Interrupt의 종류에 따라 다른 루틴 수행
3. Trap 혹은 exception은 에러나 사용자 요청에 의해 발생되는 interrupt

* 이를 SGI(Software Generated Interrupt)라고 하며,
CPU내에 있는 interrupt line을 세팅해서 발생시킨다.

2.

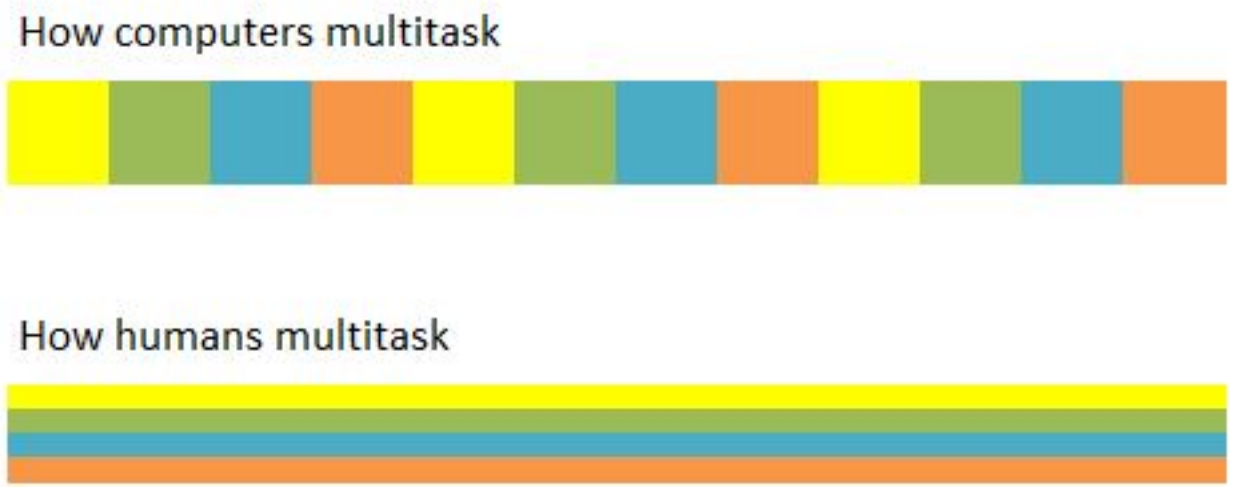
Context Switching

Context Switching



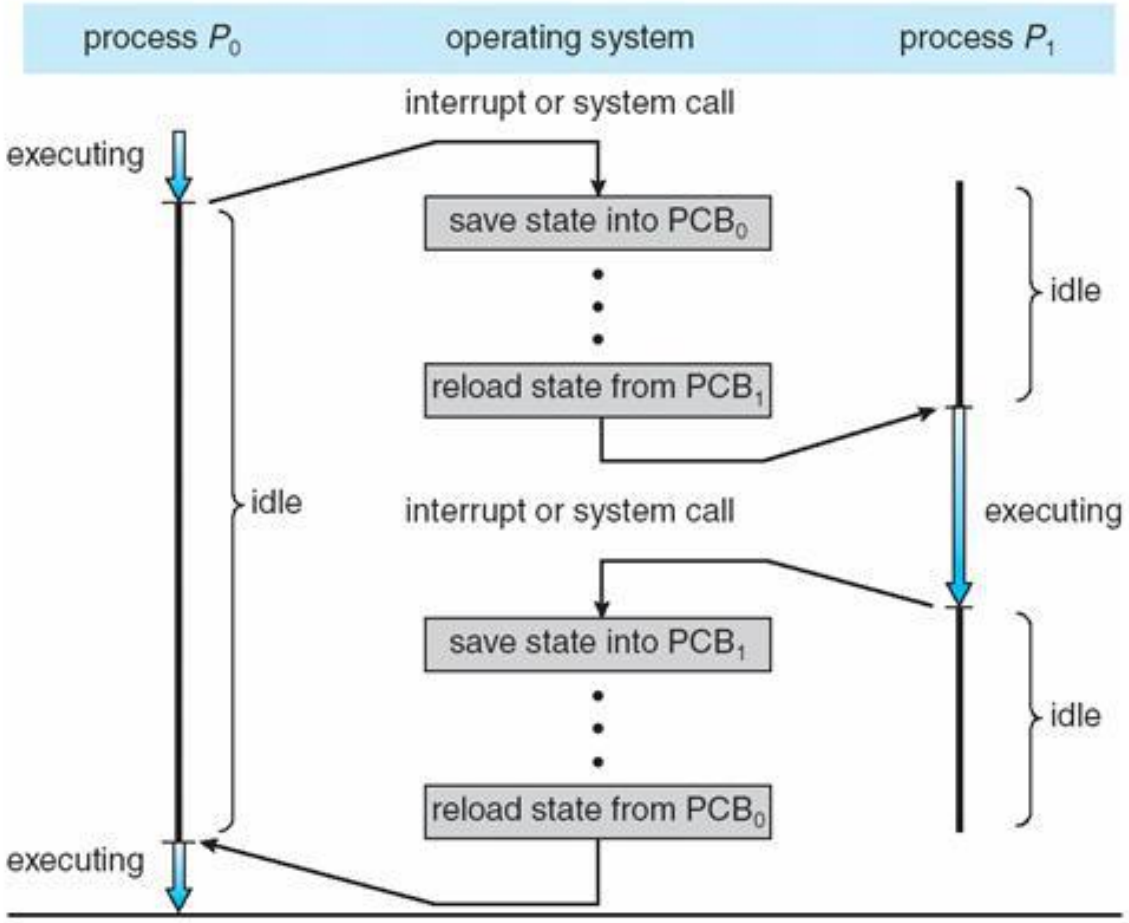
“Context”

Context Switching



“기존 프로세스의 상태 또는 레지스터 값을 저장하고
CPU가 다음 프로세스를 수행하도록 새로운 프로세스의 상태 또는 레지스터 값을 교체하는 작업”

Context Switching



Context Switching

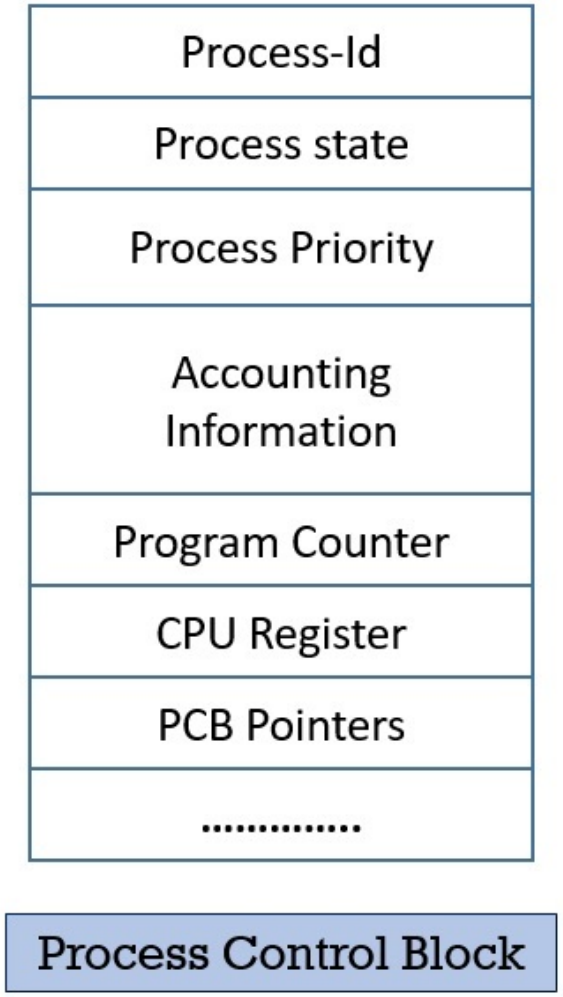
```
 1 /*
 2  * Register switch for ARMv3 and ARMv4 processors
 3  * r0 = previous task_struct, r1 = previous thread_info, r2 = next thread_info
 4  * previous and next are guaranteed not to be the same.
 5  */
 6 ENTRY(__switch_to)
 7  UNWIND(.fnstart        )
 8  UNWIND(.cantunwind     )
 9         add     ip, r1, #TI_CPU_SAVE
10  ARM(   stmia   ip!, {r4 - sl, fp, sp, lr} )    @ Store most regs on stack
11  THUMB( stmia   ip!, {r4 - sl, fp}         )    @ Store most regs on stack
12  THUMB( str     sp, [ip], #4               )
13  THUMB( str     lr, [ip], #4               )
14         ldr     r4, [r2, #TI_TP_VALUE]
15         ldr     r5, [r2, #TI_TP_VALUE + 4]
16 #ifdef CONFIG_CPU_USE_DOMAINS
17         ldr     r6, [r2, #TI_CPU_DOMAIN]
18 #endif
19         switch_tls r1, r4, r5, r3, r7
20 #if defined(CONFIG_CC_STACKPROTECTOR) && !defined(CONFIG_SMP)
21         ldr     r7, [r2, #TI_TASK]
22         ldr     r8, =__stack_chk_guard
23         ldr     r7, [r7, #TSK_STACK_CANARY]
24 #endif
25 #ifdef CONFIG_CPU_USE_DOMAINS
26         mcr     p15, 0, r6, c3, c0, 0           @ Set domain register
27 #endif
28         mov     r5, r0
29         add     r4, r2, #TI_CPU_SAVE
30         ldr     r0, =thread_notify_head
31         mov     r1, #THREAD_NOTIFY_SWITCH
32         bl      atomic_notifier_call_chain
33 #if defined(CONFIG_CC_STACKPROTECTOR) && !defined(CONFIG_SMP)
34         str     r7, [r8]
35 #endif
36  THUMB( mov     ip, r4                     )
37         mov     r0, r5
38  ARM(   ldmia   r4, {r4 - sl, fp, sp, pc}  )    @ Load all regs saved previously
39  THUMB( ldmia   ip!, {r4 - sl, fp}         )    @ Load all regs saved previously
40  THUMB( ldr     sp, [ip], #4               )
41  THUMB( ldr     pc, [ip]                   )
42  UNWIND(.fnend          )
43 ENDPROC(__switch_to)
```

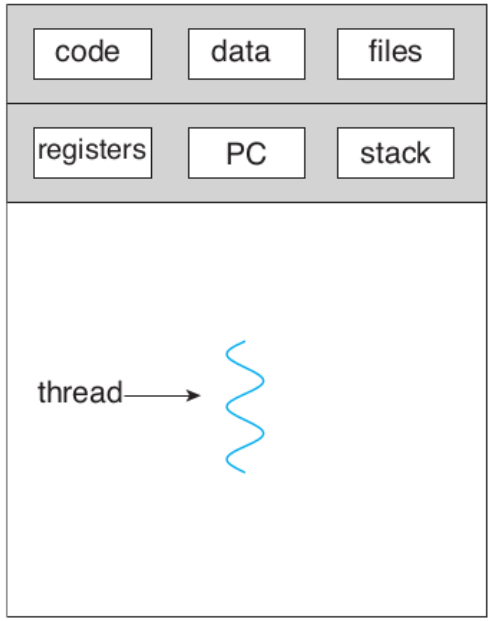
1. 9~10: 기존 thread_info->cpu_context에 r4 레지스터부터 대부분의 레지스터 백업한다.
2. 14~18: 레지스터 r4, r5, r6에 순서대로 thread_info의 tp_value[0], tp_value[1], cpu_domain 값을 가져온다.
3. 19: process context 스위칭을 위해 TLS 스위칭을 한다.
4. 20~24: SMP가 아닌 시스템에서 CONFIG_CC_STACKPROTECTOR 커널 옵션을 사용하는 경우 다음 thread_info->task->stack_canary 값을 r7 레지스터에 읽어오고 r8 레지스터에는 __stack_chk_guard 주소 값을 읽어 온다.
5. 25~27: 도메인 레지스터에 cpu_domain 값을 설정한다.
6. 28: r5 레지스터에 이전 태스크를 가리키는 r0 레지스터를 잠시 백업한다.
7. 29: r4 레지스터에 다음 thread_info->cpu_context 값을 대입한다.
8. 30~32: thread_notify_head 리스트에 등록된 notify 블럭의 모든 함수들을 호출한다. 호출 시 THREAD_NOTIFY_SWITCH 명령과 다음 thread_info 포인터 값을 전달한다.
9. 33~35: 다음 태스크의 stack_canary 값을 __stack_chk_guard 전역 변수에 저장한다.
10. 37: r5에 보관해둔 이전 태스크 포인터 값을 다시 r0에 대입한다.
11. 38: 다음 thread_info->cpu_context 값을 가리키는 r4를 통해서 r4 레지스터 부터 대부분의 레지스터들을 회복한다.
    11.1 마지막에 읽은 pc 레지스터로 점프하게 된다.

3.

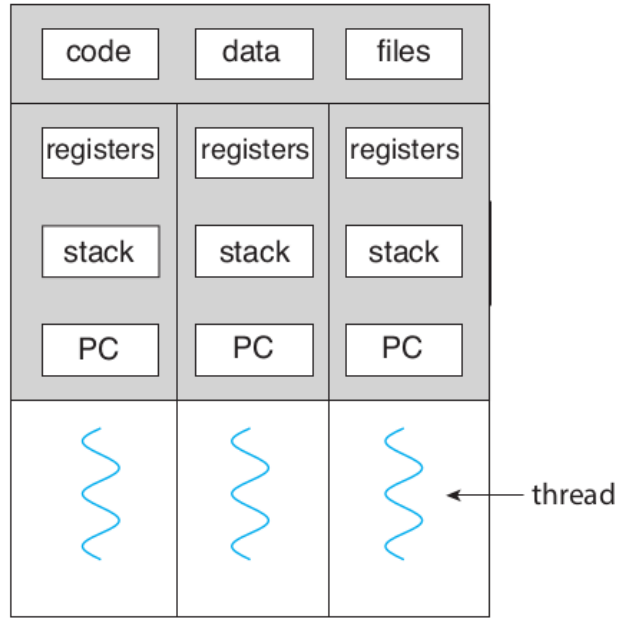
Process & Thread

Process & Thread



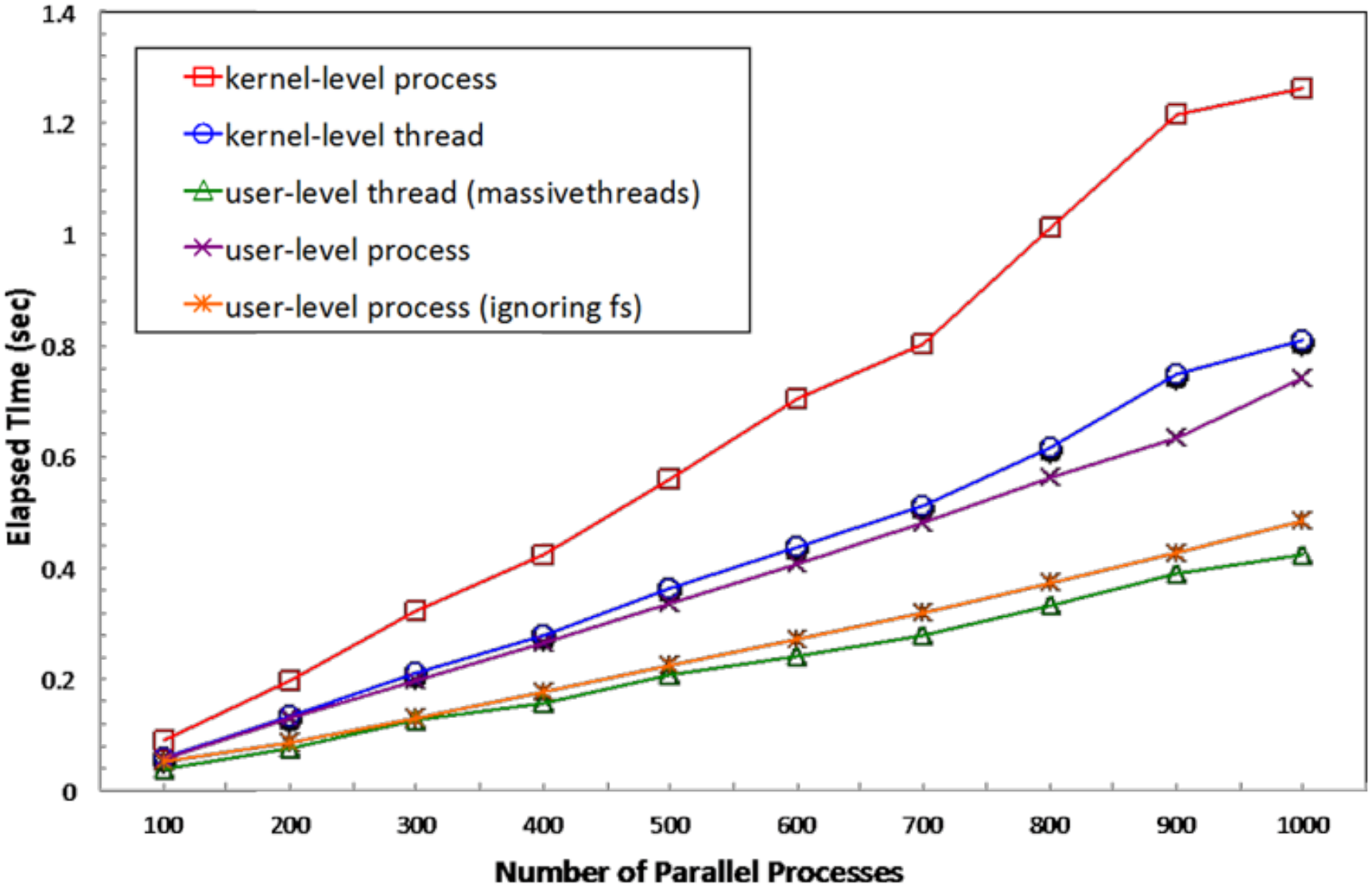


single-threaded process



multithreaded process

Process & Thread



THANKS FOR WATCHING

감사합니다