CSE 306 Operating Systems

Linux Process Scheduling

YoungMin Kwon

Linux Scheduling Implementation

- Complete Fair Scheduling (CFS)
  - Time Accounting
  - Process Selection
  - Scheduler Entry Point

Time Accounting

- On each tick of the system
  - Timeslice of the current process is decreased by the tick period
  - When the timeslice reaches 0, the process is preempted

```
struct sched_entity {
    ...
    struct rb_node run_node;      // red-black tree node
    u64 exec_start;               // execution start time
    u64 vruntime;                 // virtual runtime
    ...
};

struct task_struct {
    ...
    struct sched_entity se;
    ...
};
```

Time Accounting

- Virtual runtime
  - Actual runtime normalized (weighted) by the number of runnable processes
  - CFS use vruntime to account for how long a process has run and how much longer it ought to run
  - update_curr is called periodically by the system timer, …

```
static void update_curr(struct cfs_rq *cfs_rq) {
    struct sched_entity *curr = cfs_rq->curr;
    u64 now = rq_of(cfs_rq)->clock;
    unsigned long delta_exec;
    ...
    delta_exec = (unsigned long)(now - curr->exec_start);
    curr->exec_start = now;
    curr->vruntime += calc_delta_fair(delta_exec, curr);
    ...
}
```

Process Selection

- CFS picks the process with the smallest vruntime
- CFS uses red-black tree to manage the runnable processes
  - Finding the process with smallest vruntime is easy

```
struct sched_entity *__pick_first_entity(struct cfs_rq *cfs_rq)
{
    struct rb_node *left = cfs_rq->rb_leftmost;

    if (!left)
        return NULL;

    return rb_entry(left, struct sched_entity, run_node);
}
```

Adding Processes to RB Tree

```
static void __enqueue_entity(struct cfs_rq *cfs_rq, struct sched_entity *se) {
    struct rb_node **link = &cfs_rq->tasks_timeline.rb_node;
    struct rb_node *parent = NULL;
    struct sched_entity *entry;
    int leftmost = 1; //assume that se is the smallest

    // Find the right place in the rbtree:
    while (*link) {
        parent = *link;
        entry = rb_entry(parent, struct sched_entity, run_node);

        if (entity_before(se, entry)) {
            link = &parent->rb_left;
        } else {
            link = &parent->rb_right;
            leftmost = 0; //se is not the smallest
        }
    }

    // Maintain a cache of leftmost tree entries (it is frequently used):
    if (leftmost)
        cfs_rq->rb_leftmost = &se->run_node;

    rb_link_node(&se->run_node, parent, link);
    rb_insert_color(&se->run_node, &cfs_rq->tasks_timeline);
}
```

Removing Processes from RB Tree

```
static void __dequeue_entity(struct cfs_rq *cfs_rq,
                             struct sched_entity *se)
{
    // update the leftmost, if it is removed
    if (cfs_rq->rb_leftmost == &se->run_node) {
        struct rb_node *next_node;
        next_node = rb_next(&se->run_node);

        cfs_rq->rb_leftmost = next_node;
    }

    rb_erase(&se->run_node, &cfs_rq->tasks_timeline);
}
```

Scheduler Entry Point

- schedule()
  - The main entry point into the process schedule
  - pick_next_task(): find the next process to dispatch
    - For each sched_class, check if it has a task to run next
  - context_siwtch(): dispatch the next process to run
    - switch_mm(): switch the virtual memory mapping
    - switch_to(): switch the processor state (stack pointer, registers, …)

schedule()

```
static void __sched notrace __schedule(bool preempt)
{
    struct task_struct *prev, *next;
    struct rq *rq;
    int cpu;
    ...

    cpu = smp_processor_id();
    rq = cpu_rq(cpu);
    prev = rq->curr;
    ...

    next = pick_next_task(rq, prev, cookie);
    ...

    rq = context_switch(rq, prev, next, cookie);
    ...
}
```

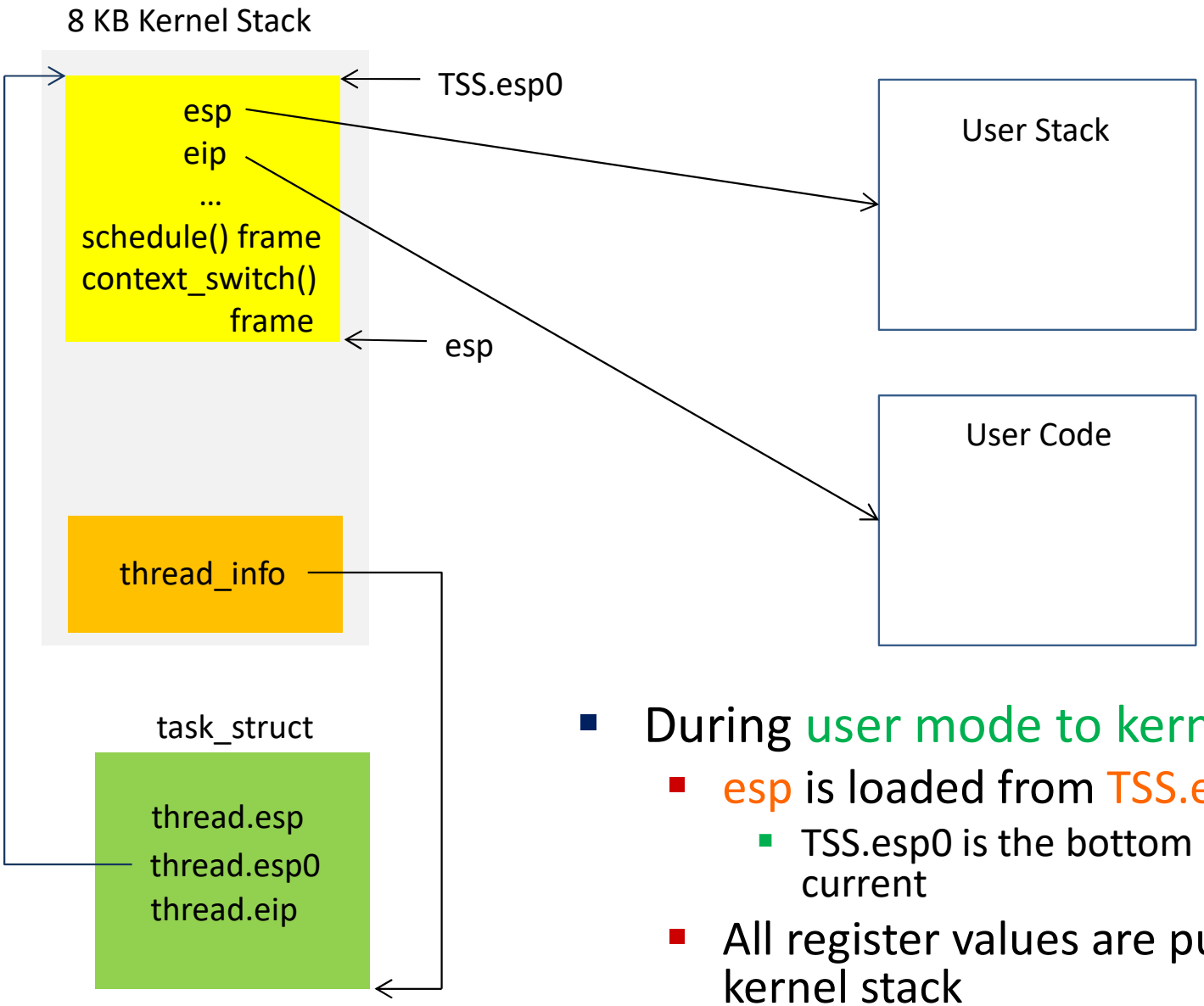
pick_next_task()

```
static inline struct task_struct *
pick_next_task(struct rq *rq, struct task_struct *prev,
                              struct pin_cookie cookie) {
    const struct sched_class *class = &fair_sched_class;
    struct task_struct *p;

    // Optimization: we know that if all tasks are in
    // the fair class we can call that function directly:
    if (likely(prev->sched_class == class &&
           rq->nr_running == rq->cfs.h_nr_running)) {
        p = fair_sched_class.pick_next_task(rq, prev, cookie);
        ...
        return p;
    }

    for_each_class(class) {
        p = class->pick_next_task(rq, prev, cookie);
        if (p) {
            ...
            return p;
        }
    }
}
```

context_switch()

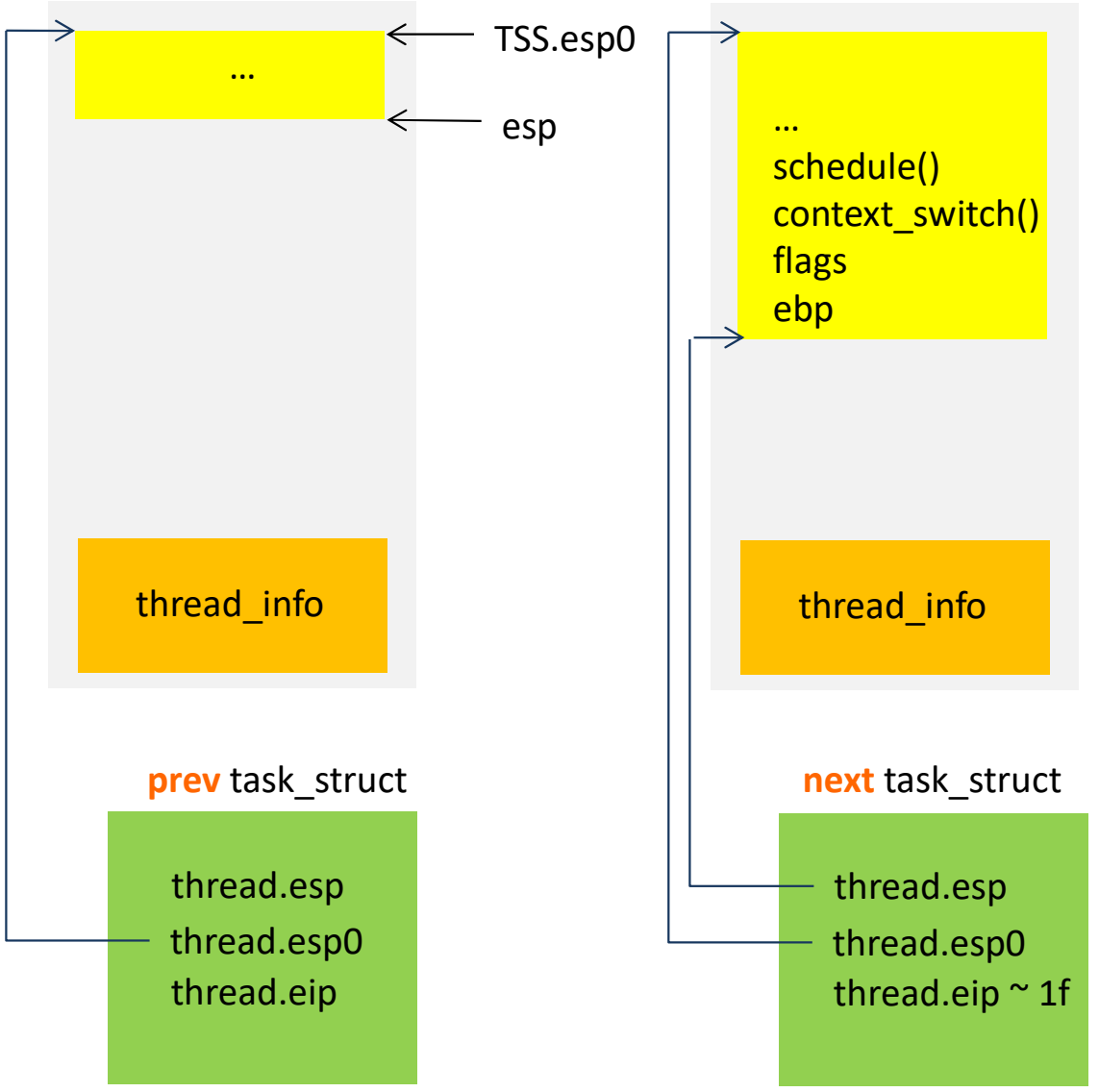
```
static __always_inline struct rq *
context_switch(struct rq *rq, struct task_struct *prev,
               struct task_struct *next, struct pin_cookie cookie)
{
    struct mm_struct *mm, *oldmm;
    mm = next->mm;
    oldmm = prev->active_mm;
    …
    /* replace the address space: CR3, TLB flush, …*/
    switch_mm_irqs_off(oldmm, mm, next);
    …

    /* switch the register state, the stack, etc. */
    switch_to(prev/*prev*/, next/*next*/, prev/*last*/);
    …
}
```

Configuration Before switch_to

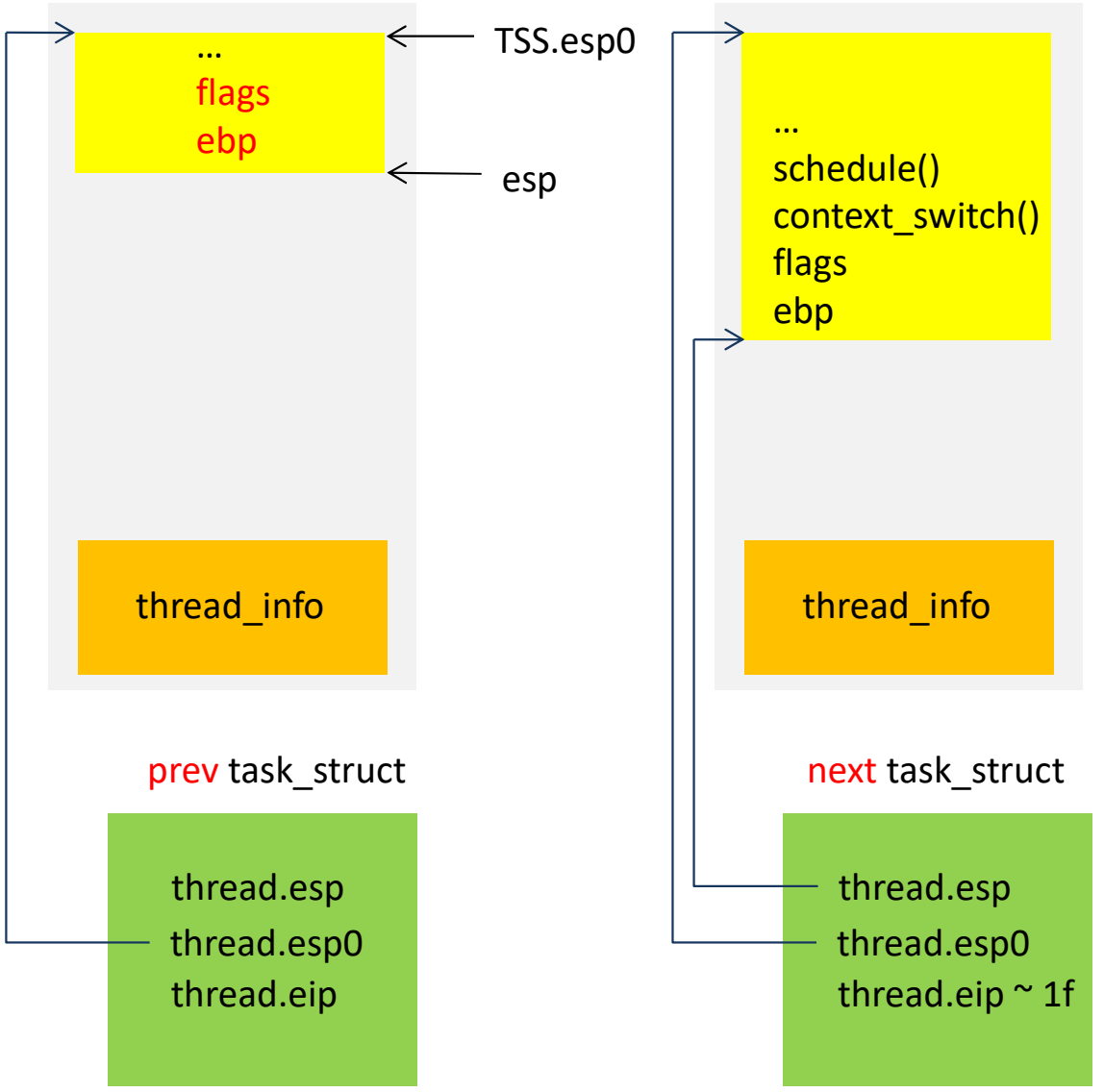


- During user mode to kernel mode transition
  - esp is loaded from TSS.esp0
    - TSS.esp0 is the bottom of the kernel stack for current
  - All register values are pushed onto the kernel stack

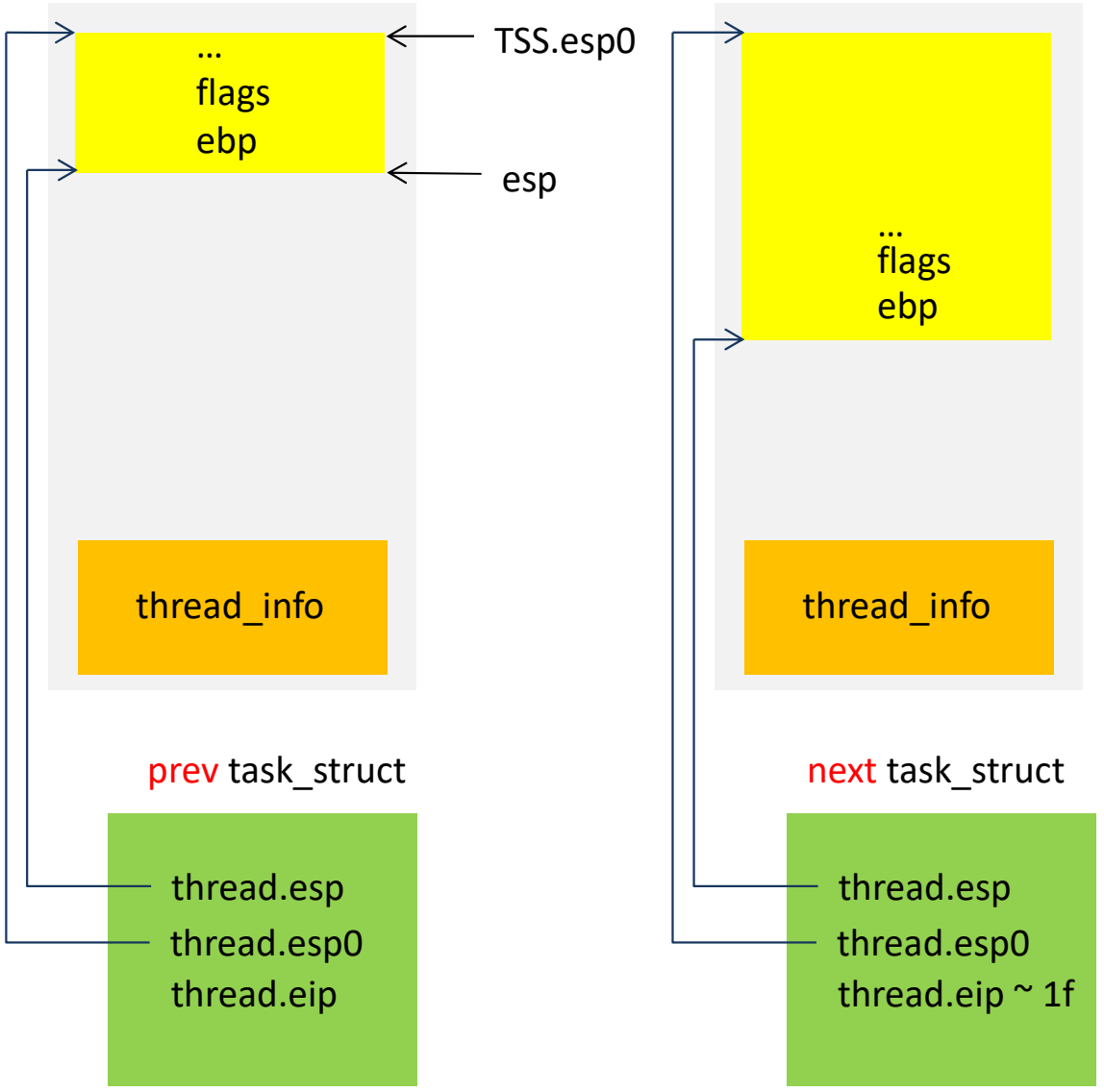
```
#define switch_to(prev, next, last)

//eax = prev, edx = next
movl prev, %eax
movl next, %edx

//save flags and ebp
pushfl
pushl %ebp

//prev->thread.esp = esp
movl %esp, 484(%eax)
//esp = next->thread.esp
movl 484(%edx), %esp

//prev->thread.eip = label 1
movl $1f, 480(%eax)
//push next->thread.eip
pushl 480(%edx)

//jump to __switch_to C func.
jmp __switch_to

//restore flags and ebp
1:
    popl %ebp
    popfl
    movl %eax, last
```



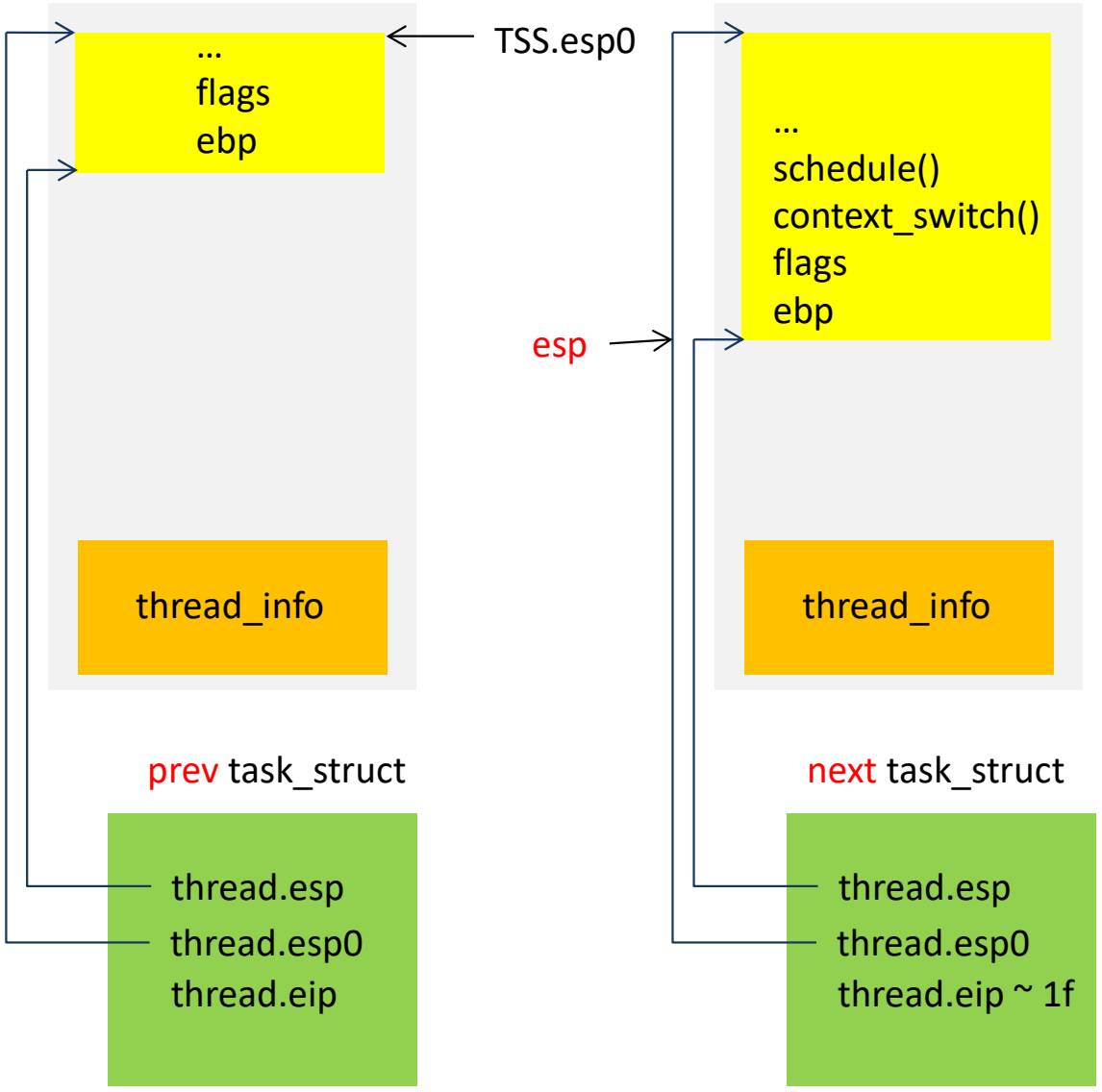
```
#define switch_to(prev, next, last)

//eax = prev, edx = next
movl prev, %eax
movl next, %edx

//save flags and ebp
pushfl
pushl %ebp

//prev->thread.esp = esp
movl %esp, 484(%eax)
//esp = next->thread.esp
movl 484(%edx), %esp

//prev->thread.eip = label 1
movl $1f, 480(%eax)
//push next->thread.eip
pushl 480(%edx)

//jump to __switch_to C func.
jmp __switch_to

//restore flags and ebp
1:
    popl %ebp
    popfl
    movl %eax, last
```



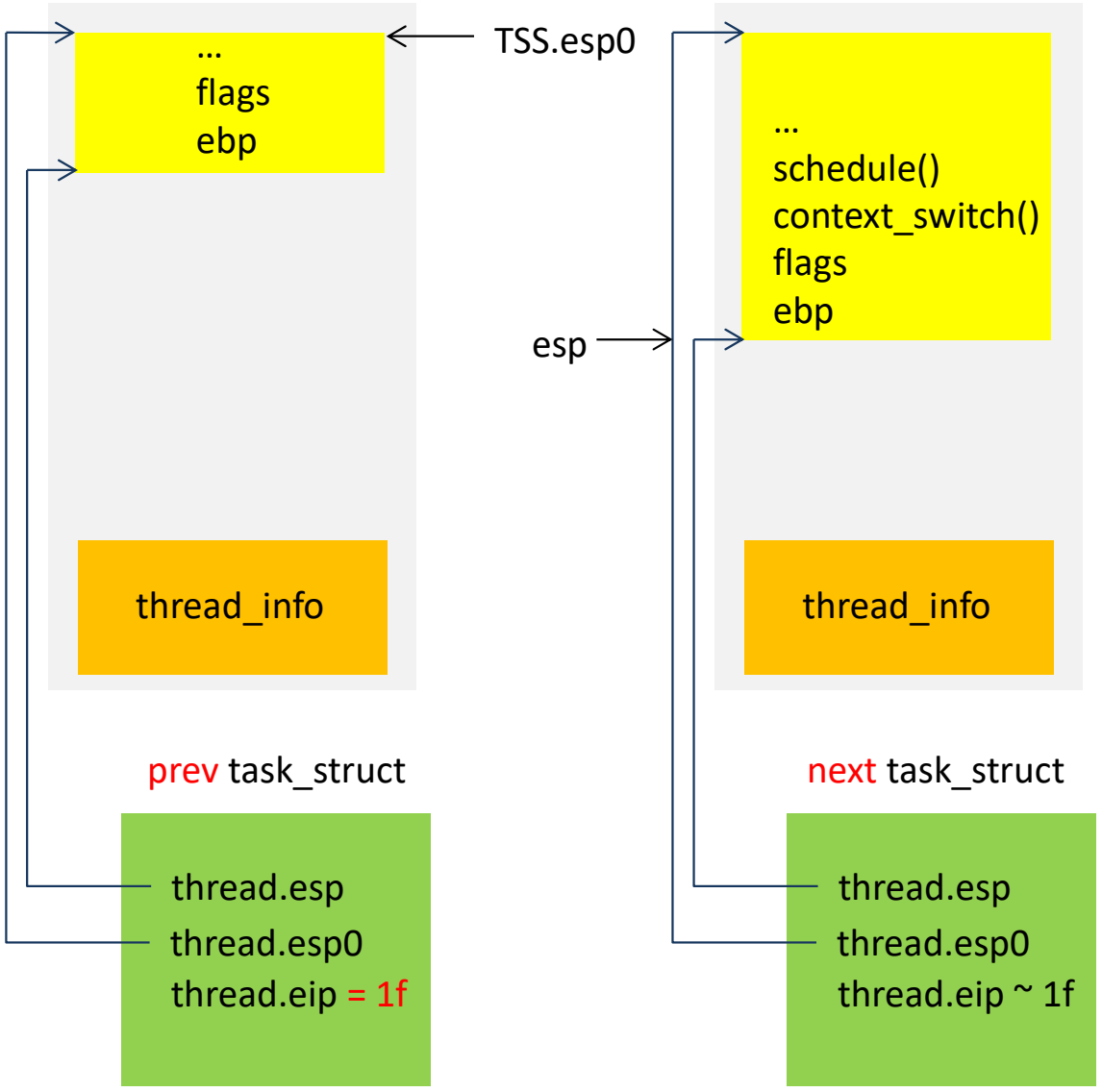
```
#define switch_to(prev, next, last)

//eax = prev, edx = next
movl prev, %eax
movl next, %edx

//save flags and ebp
pushfl
pushl %ebp

//prev->thread.esp = esp
movl %esp, 484(%eax)
//esp = next->thread.esp
movl 484(%edx), %esp

//prev->thread.eip = label 1
movl $1f, 480(%eax)
//push next->thread.eip
pushl 480(%edx)

//jump to __switch_to C func.
jmp __switch_to

//restore flags and ebp
1:
    popl %ebp
    popfl
    movl %eax, last
```



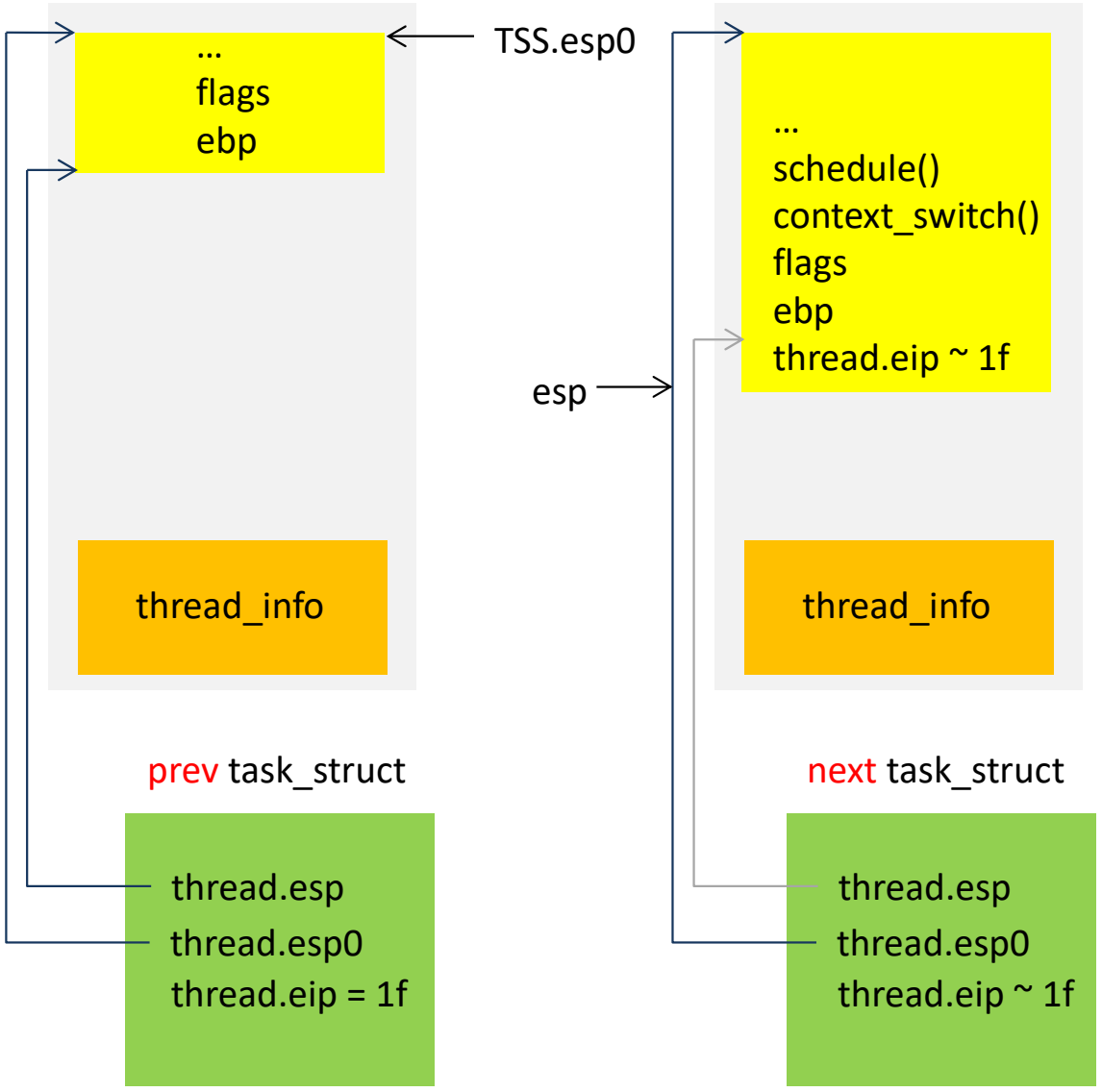
```
#define switch_to(prev, next, last)

//eax = prev, edx = next
movl prev, %eax
movl next, %edx

//save flags and ebp
pushfl
pushl %ebp

//prev->thread.esp = esp
movl %esp, 484(%eax)
//esp = next->thread.esp
movl 484(%edx), %esp

//prev->thread.eip = label 1
movl $1f, 480(%eax)
//push next->thread.eip
pushl 480(%edx)

//jump to __switch_to C func.
jmp __switch_to

//restore flags and ebp
1:
    popl %ebp
    popfl
    movl %eax, last
```



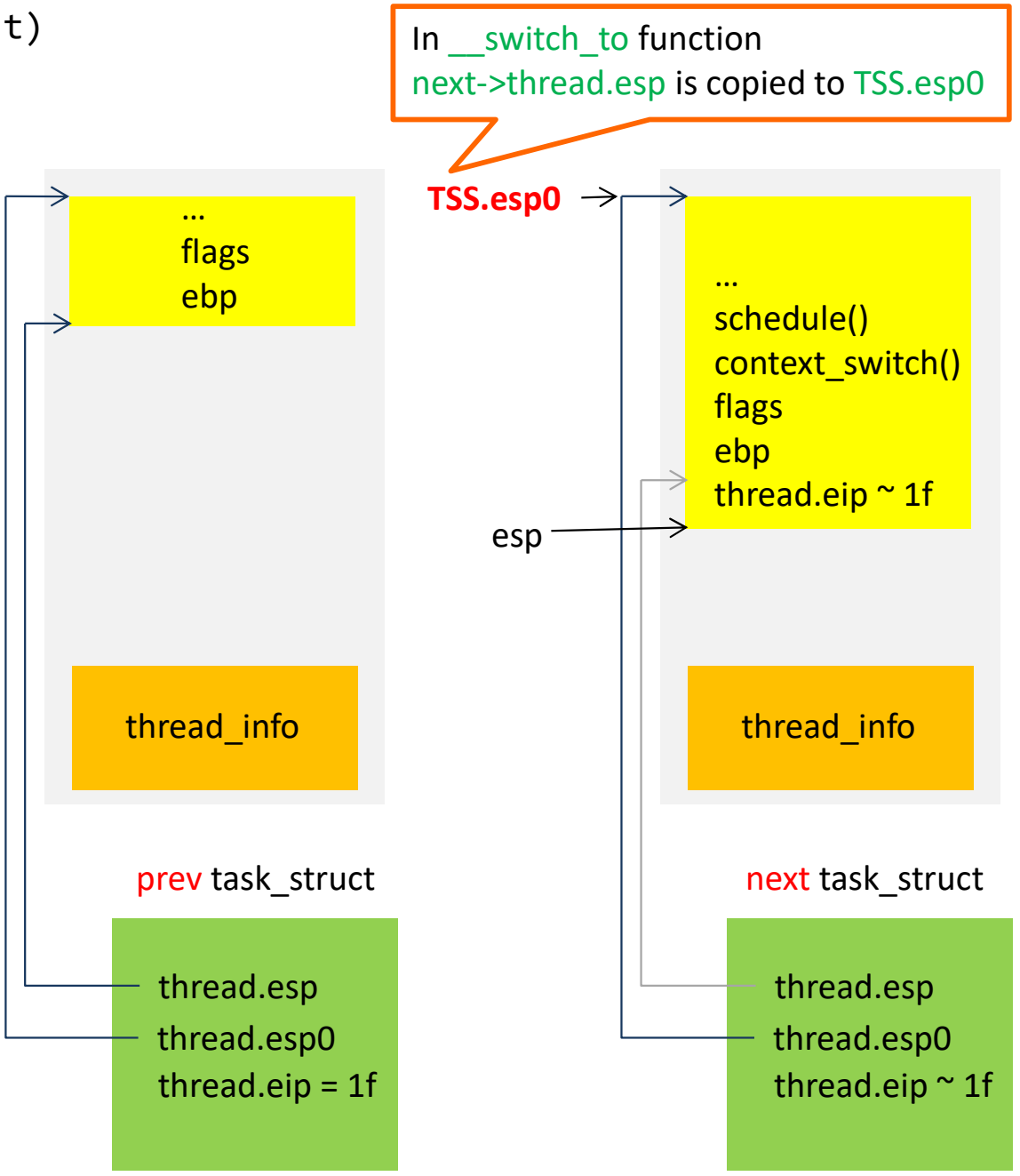
```
#define switch_to(prev, next, last)

//eax = prev, edx = next
movl prev, %eax
movl next, %edx

//save flags and ebp
pushfl
pushl %ebp

//prev->thread.esp = esp
movl %esp, 484(%eax)
//esp = next->thread.esp
movl 484(%edx), %esp

//prev->thread.eip = label 1
movl $1f, 480(%eax)
//push next->thread.eip
pushl 480(%edx)

//jump to __switch_to C func.
jmp __switch_to

//restore flags and ebp
1:
    popl %ebp
    popfl
    movl %eax, last
```



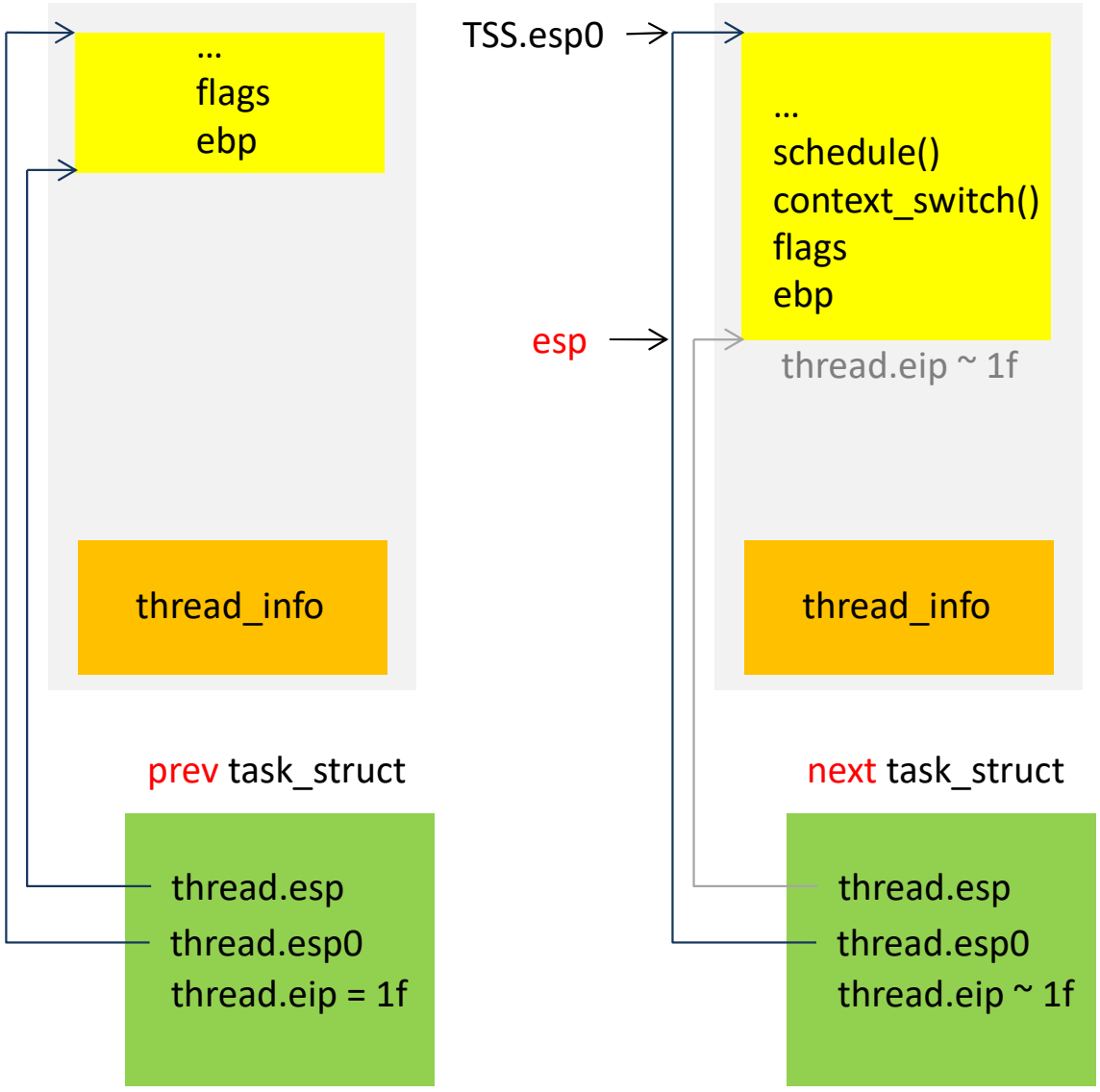
```
#define switch_to(prev, next, last)

//eax = prev, edx = next
movl prev, %eax
movl next, %edx

//save flags and ebp
pushfl
pushl %ebp

//prev->thread.esp = esp
movl %esp, 484(%eax)
//esp = next->thread.esp
movl 484(%edx), %esp

//prev->thread.eip = label 1
movl $1f, 480(%eax)
//push next->thread.eip
pushl 480(%edx)

//jump to __switch_to C func.
jmp __switch_to

//restore flags and ebp
1:
    popl %ebp
    popfl
    movl %eax, last
```



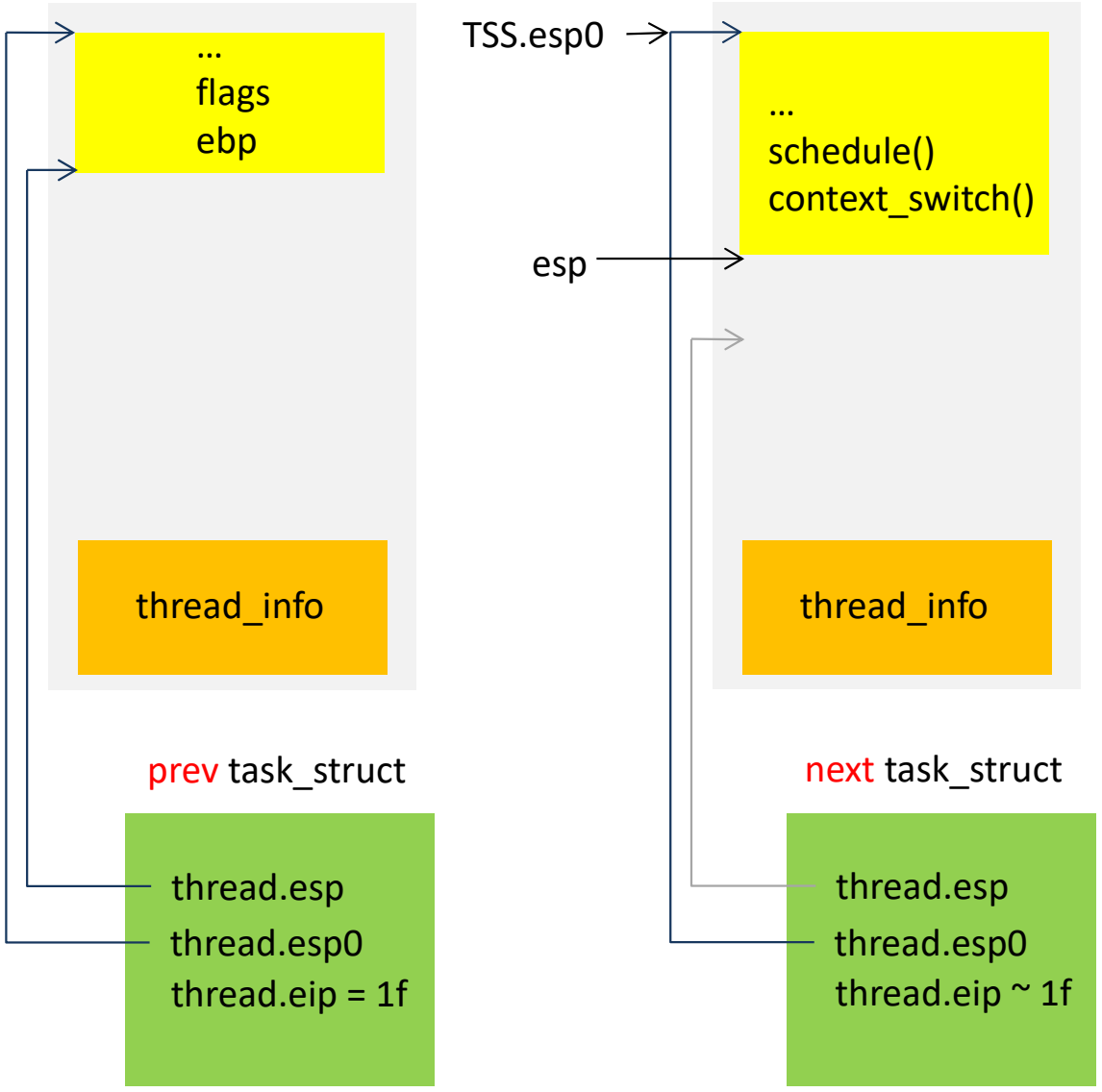
```
#define switch_to(prev, next, last)

//eax = prev, edx = next
movl prev, %eax
movl next, %edx

//save flags and ebp
pushfl
pushl %ebp

//prev->thread.esp = esp
movl %esp, 484(%eax)
//esp = next->thread.esp
movl 484(%edx), %esp

//prev->thread.eip = label 1
movl $1f, 480(%eax)
//push next->thread.eip
pushl 480(%edx)

//jump to __switch_to C func.
jmp __switch_to

//restore flags and ebp
1:
    popl %ebp
    popfl
    movl %eax, last
```



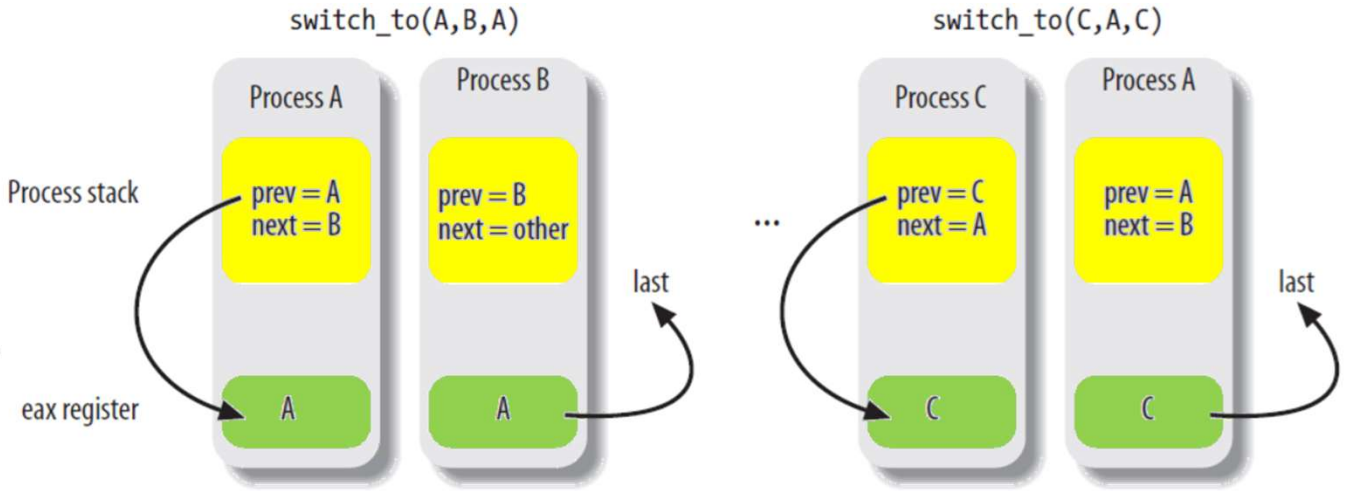
```
#define switch_to(prev, next, last)

//eax = prev, edx = next
movl prev, %eax
movl next, %edx

//save flags and ebp
pushfl
pushl %ebp

//prev->thread.esp = esp
movl %esp, 484(%eax)
//esp = next->thread.esp
movl 484(%edx), %esp

//prev->thread.eip = label 1
movl $1f, 480(%eax)
//push next->thread.eip
pushl 480(%edx)

//jump to __switch_to C func.
jmp __switch_to

//restore flags and ebp
1:
    popl %ebp
    popfl
    movl %eax, last
```



```
#define switch_to(prev, next, last)

//eax = prev, edx = next
movl prev, %eax
movl next, %edx

//save flags and ebp
pushfl
pushl %ebp

//prev->thread.esp = esp
movl %esp, 484(%eax)
//esp = next->thread.esp
movl 484(%edx), %esp

//prev->thread.eip = label 1
movl $1f, 480(%eax)
//push next->thread.eip
pushl 480(%edx)

//jump to __switch_to C func.
jmp __switch_to

//restore flags and ebp
1:
    popl %ebp
    popfl
    movl %eax, last
```



```
#define switch_to(prev, next, last)

//eax = prev, edx = next
movl prev, %eax
movl next, %edx

//save flags and ebp
pushfl
pushl %ebp

//prev->thread.esp = esp
movl %esp, 484(%eax)
//esp = next->thread.esp
movl 484(%edx), %esp

//prev->thread.eip = label 1
movl $1f, 480(%eax)
//push next->thread.eip
pushl 480(%edx)

//jump to __switch_to C func.
jmp __switch_to

//restore flags and ebp
1:
    popl %ebp
    popfl
    movl %eax, last //last is an output parameter
```



Assignment 4

- In this assignment, we'll implement a simple scheduler class called my_sched_class
  - Modify sched.h, rt.c, and core.c as in the following pages
    - my_sched_class will be inserted in between rt_sched_class and fair_sched_class
  - Create mysched.c and implement the TODOs

- Due date TBD

```c
// User space program
// myschedtest.c
//
#include <sched.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/wait.h>
int main() {
    const struct sched_param param = {
        .sched_priority = 0,
    };
    sched_setscheduler(getpid(), 7 /*policy*/, &param);

    int i;
    for (i = 0; i < 10; i++)
        if (fork() == 0) {
            sleep(0);
            printf("child pid: %d\n", getpid());
            return 0;
        }

    printf("parent pid: %d\n", getpid());
    for (i = 0; i < 10; i++)
        wait(NULL);

    return 0;
}
```

```c
// my-task scheduling class
// mysched.c
//
#include <linux/list.h>
#include <linux/slab.h>
#include <linux/sched.h>
#include "../kernel/sched/sched.h"

#define PENTER \
    printk(KERN_INFO "entering %s\n", __FUNCTION__)
#define PENTERID(id) \
    printk(KERN_INFO "entering %s, %d\n", __FUNCTION__, (id));

struct my_rq {
    struct sched_entity *se;
    struct list_head list;
} my_rq[NR_CPUS];

static int my_rq_initialized = 0;
```

```
void init_my_sched_class(void)
{
    int i;
    PENTER;

    for(i = 0; i < NR_CPUS; i++)
        // TODO: initialize my_rq[i]
        //       set se to NULL, initialize list head

    my_rq_initialized = 1;
}

static struct task_struct *
my_pick_next_task(struct rq *rq, struct task_struct *prev,
                  struct rq_flags *rf)
{
    if (!my_rq_initialized)  return NULL;

    //TODO: if my_rq[rq->cpu].list is empty, return NULL
    //      otherwise, get the sched_entity of the first list element
    //      let next be the task_struct containing the sched_entity
    //      print next's pid and prev's pid if prev is valid
    //      return next
}
```

```
static void
my_enqueue_task(struct rq *rq, struct task_struct *p, int flags)
{
    PENTERID(p->pid);
    //TODO: allocate my_rq, initialize it with p->se
    //      add the new my_rq to my_rq[rq->cpu].list
    //      increase rq's running count by add_nr_running(rq, 1);
}


static void
my_dequeue_task(struct rq *rq, struct task_struct *p, int flags)
{
    PENTERID(p->pid);
    //TODO: find my_rq with p->se from my_rq[rq->cpu].list
    //      remove it from the list, deallocate it
    //      decrease rq's running count by sub_nr_running(rq, 1);
}
```

```c
int my_prio(void) { return 1000; }

int is_my_prio(int prio) { return prio == my_prio() ? 1 : 0; }

int my_policy(void) { return 7; } //use 7 for sched_setscheduler

#ifdef CONFIG_SMP
static int
my_select_task_rq(struct task_struct *p, int cpu, int sd_flag, int flags) {
    PENTER;
    return task_cpu(p); /* don't migrate */
}
static int
my_balance(struct rq *rq, struct task_struct *p, struct rq_flags *rf) {
    PENTER;
    return 0;
}
#endif

static void
my_check_preempt_curr(struct rq *rq, struct task_struct *p, int flags)
{ PENTER; }

static void
my_yield_task(struct rq *rq)
{ PENTER; }
```

```
static void
my_put_prev_task(struct rq *rq, struct task_struct *prev)
{ PENTERID(prev->pid); }

static void
my_set_next_task(struct rq *rq, struct task_struct *p, bool first)
{ PENTER; }

static void
my_task_tick(struct rq *rq, struct task_struct *curr, int queued)
{ PENTER; }

static void
my_switched_to(struct rq *rq, struct task_struct *p)
{ PENTERID(p->pid); }

static void
my_prio_changed(struct rq *rq, struct task_struct *p, int oldprio)
{ PENTER; }

static unsigned int
my_get_rr_interval(struct rq *rq, struct task_struct *task)
{ PENTER; return 0; }

static void
my_update_curr(struct rq *rq) { PENTER; }
```

```
const struct sched_class my_sched_class = {
    .next                = &fair_sched_class,
    /* called when task enters/leaves a ready sate */
    .enqueue_task        = my_enqueue_task,
    .dequeue_task        = my_dequeue_task,
    /* called as a result of sched_yield system call */
    .yield_task          = my_yield_task,
    /* should check if the current task should be preempted by
       a new ready task -> call resched_task if so */
    .check_preempt_curr = my_check_preempt_curr,
    /* pick the next task to executed, called from schdule() */
    .pick_next_task      = my_pick_next_task,
    /* called when running task is rescheduled,
       called from schedule() before pick_next_task */
    .put_prev_task       = my_put_prev_task,
    /* account for a task changing its policy or group */
    .set_next_task       = my_set_next_task,
#ifdef CONFIG_SMP
    .select_task_rq      = my_select_task_rq,
    .set_cpus_allowed    = set_cpus_allowed_common,
    .balance             = my_balance,
#endif
    .task_tick           = my_task_tick,     //on timer (Hz freq)
    .switched_to         = my_switched_to,   //on scheduling class change
    .prio_changed        = my_prio_changed, //on priority change
    .get_rr_interval     = my_get_rr_interval,
    .update_curr         = my_update_curr,
};
```

Modify linux-5.4.49/kernel/sched/sched.h

```
#define TASK_ON_RQ_QUEUED       1
#define TASK_ON_RQ_MIGRATING    2

#define CSE306_HW4  1       //CSE306: add these 4 lines
#if CSE306_HW4
extern int my_policy(void);
#endif
...

//CSE306: update this function
static inline bool valid_policy(int policy)
{
#if CSE306_HW4
    return idle_policy(policy) || fair_policy(policy) ||
           rt_policy(policy)   || dl_policy(policy)   ||
           policy == my_policy();
#else
    return idle_policy(policy) || fair_policy(policy) ||
           rt_policy(policy)   || dl_policy(policy);
#endif
}
```

Modify linux-5.4.49/kernel/sched/sched.h

```
...
extern const struct sched_class stop_sched_class;
extern const struct sched_class dl_sched_class;
extern const struct sched_class rt_sched_class;
extern const struct sched_class fair_sched_class;
extern const struct sched_class idle_sched_class;
#if CSE306_HW4  //CSE306: add these 3 lines
extern const struct sched_class my_sched_class;
#endif
```

Modify linux-5.4.49/kernel/sched/rt.c

```
const struct sched_class rt_sched_class = {
#if CSE306_HW4 //CSE306: add these 5 lines
    .next           = &my_sched_class,
#else
    .next           = &fair_sched_class,
#endif
    .enqueue_task       = enqueue_task_rt,
    .dequeue_task       = dequeue_task_rt,
...
```

Modify linux-5.4.49/kernel/sched/core.c

```
#include <trace/events/sched.h>
#if CSE306_HW4 // CSE306: add these 5 lines
extern int my_prio(void);
extern int is_my_prio(int prio);
extern void init_my_sched_class(void);
#endif

DEFINE_PER_CPU_SHARED_ALIGNED(struct rq, runqueues);

...
static inline int normal_prio(struct task_struct *p)
{
...
    else if (task_has_rt_policy(p))
        prio = MAX_RT_PRIO-1 - p->rt_priority;
#if CSE306_HW4 //CSE306: add these 4 lines
    else if(p->policy == my_policy())
        prio = my_prio();
#endif
    else
        prio = __normal_prio(p);
    return prio;
}
```

Modify linux-5.4.49/kernel/sched/core.c

```
int sched_fork(unsigned long clone_flags, struct task_struct *p)
{
...
    if (dl_prio(p->prio)) {
        put_cpu();
        return -EAGAIN;
    } else if (rt_prio(p->prio)) {
        p->sched_class = &rt_sched_class;
#if CSE306_HW4 //CSE306: add these 4 lines
    } else if (is_my_prio(p->prio)) {
        p->sched_class = &my_sched_class;
#endif
    } else {
        p->sched_class = &fair_sched_class;
    }
...
}
```

Modify linux-5.4.49/kernel/sched/core.c

```
void rt_mutex_setprio(struct task_struct *p, int prio)
{
...
    } else if (rt_prio(prio)) {
        if (dl_prio(oldprio))
            p->dl.dl_boosted = 0;
        if (oldprio < prio)
            queue_flag |= ENQUEUE_HEAD;
        p->sched_class = &rt_sched_class;
#if CSE306_HW4 //CSE306: add these 4 lines
    } else if (is_my_prio(prio)) {
        p->sched_class = &my_sched_class;
#endif
    } else {
        if (dl_prio(oldprio))
...
}
```

Modify linux-5.4.49/kernel/sched/core.c

```
static void __setscheduler(struct rq *rq, struct task_struct *p,
                const struct sched_attr *attr, bool keep_boost)
{
...
    else if (rt_prio(p->prio))
        p->sched_class = &rt_sched_class;
#if CSE306_HW4 //CSE306: add these 4 lines
    else if (is_my_prio(p->prio))
        p->sched_class = &my_sched_class;
#endif
    else
        p->sched_class = &fair_sched_class;
...
}
```

Modify linux-5.4.49/kernel/sched/core.c

```
void __init sched_init_smp(void)
{
...
    init_sched_rt_class();
    init_sched_dl_class();
#if CSE306_HW4 //CSE306: add these 3 lines
    init_my_sched_class();
#endif
    sched_smp_initialized = true;
}
```