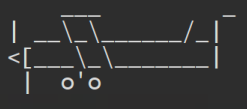
OS EXP
Design Review
#02

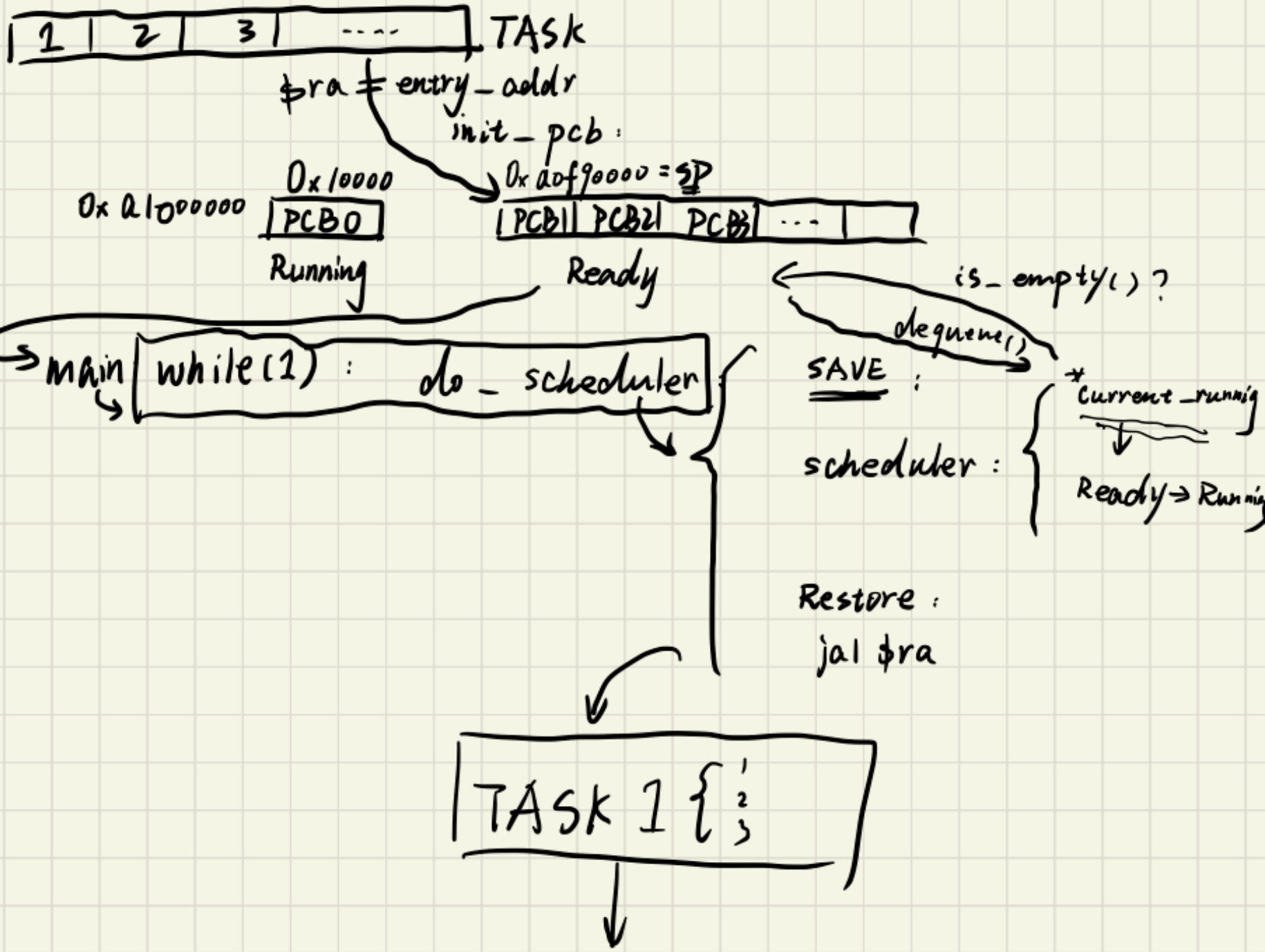
```
> [TASK] This task is to test scheduler. (17)
> [TASK] This task is to test scheduler. (17)
```

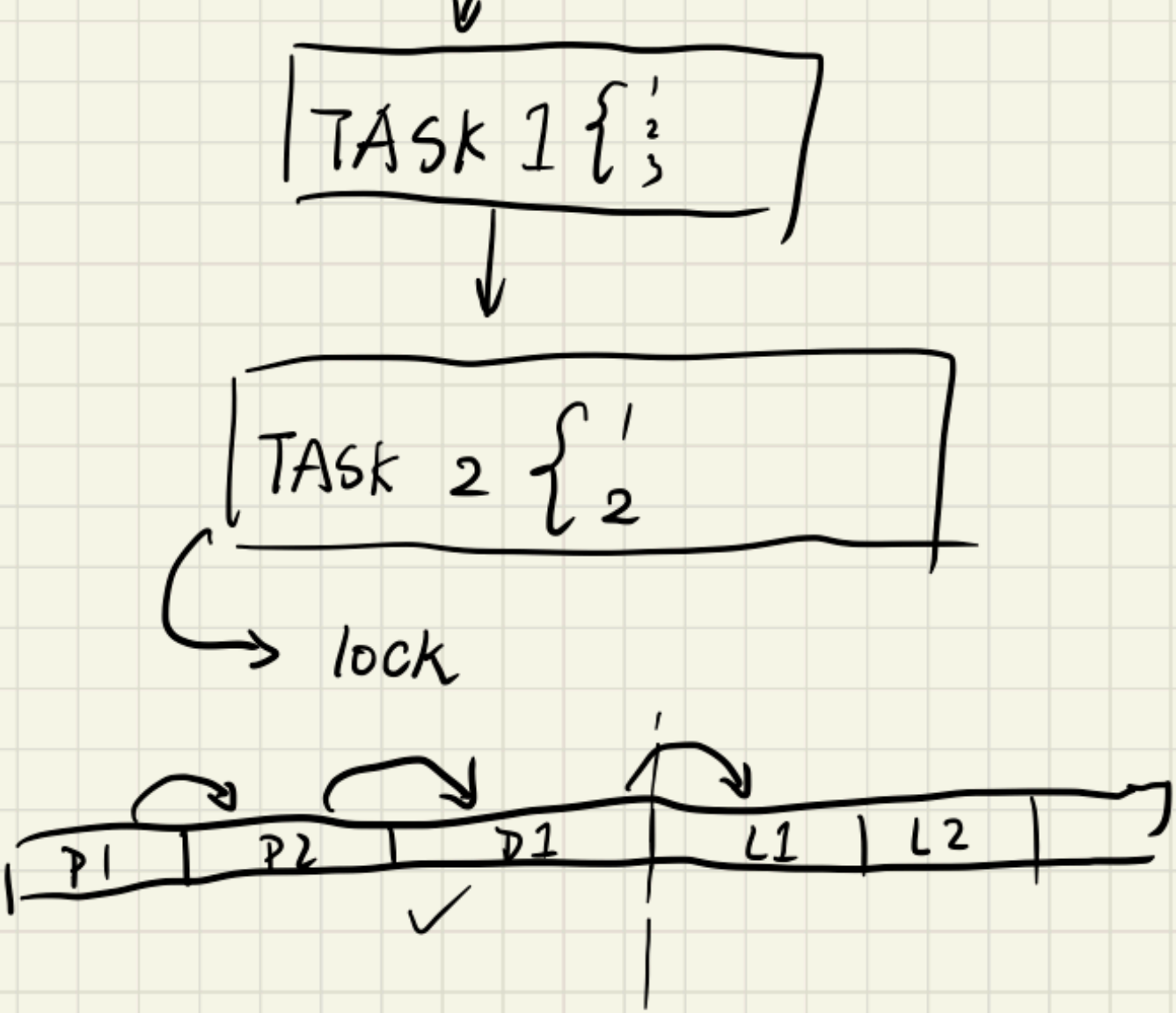


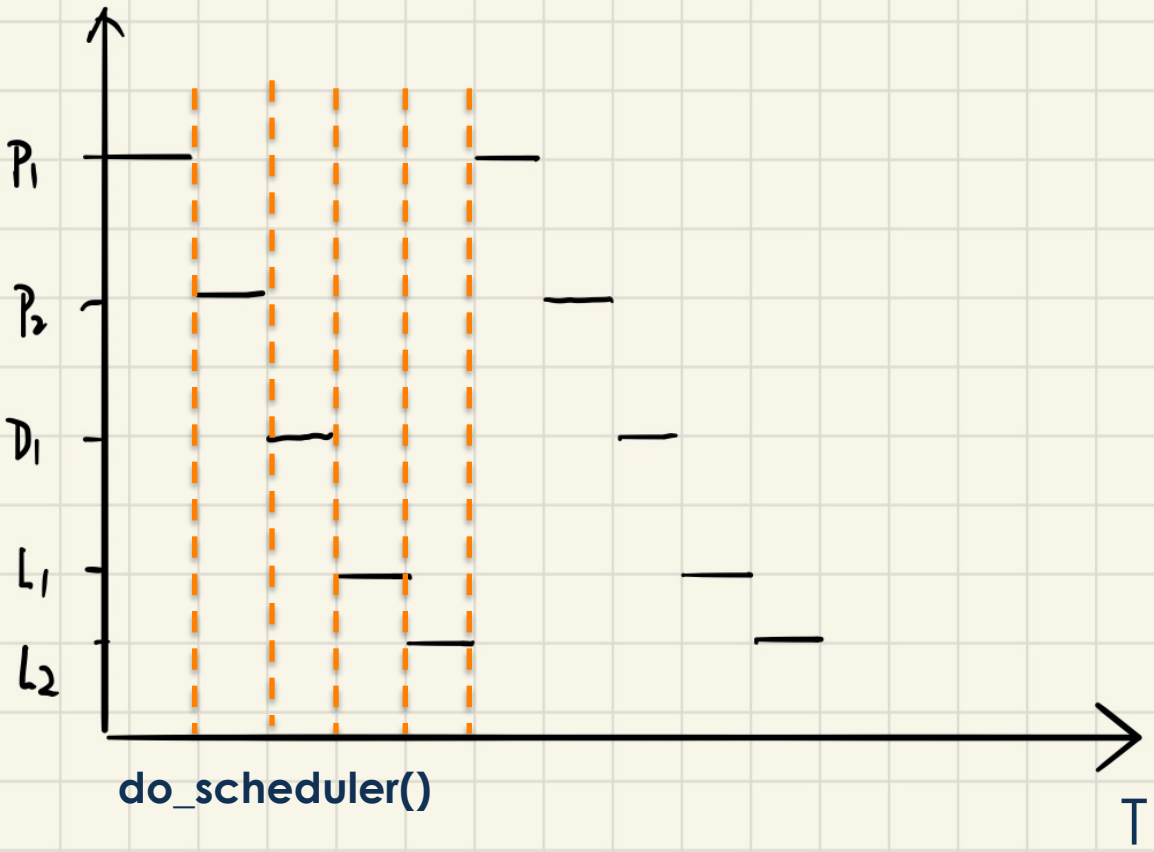
蔡润泽&张博源

01

Overview on Design


1
2
3
TASK
$ra = entry_addr
init_pcb:
0x10000
0x a0f90000 = SP
0x a1000000
PCB0
PCB1
PCB2
PCB3
Running
Ready
is_empty()?
dequeue()
main
while(1):
do_scheduler
SAVE:
scheduler:
*current_running
Ready → Running
Restore:
jal $ra
TASK 1 { ; ; ;


TASK 1 { ; ;
TASK 2 { ;
2
lock
P1
P2
D1
L1
L2


P1
P2
D1
L1
L2
do_scheduler()
T

01

Q&A

PCB

```c
typedef struct pcb
{
    /* register context */
    regs_context_t kernel_context;
    regs_context_t user_context;

    uint32_t kernel_stack_top;
    uint32_t user_stack_top;

    /* previous, next pointer */
    void *prev;
    void *next;

    /* process id */
    pid_t pid;

    /* kernel/user thread/process */
    task_type_t type;

    /* BLOCK | READY | RUNNING */
    task_status_t status;

    /* cursor position */
    int cursor_x;
    int cursor_y;

} pcb_t;
```

Task Initialization

```
static void init_pcb()
{
    int i,j;
    pcb[0].pid=process_id++;
    pcb[0].status=TASK_RUNNING;
    int stack_top=STACK_MAX;
    queue_id=1;

    queue_init(&ready_queue);
    for(i=0;i<num_sched1_tasks;i++,queue_id++){
        for(j=0;j<32;j++){
            pcb[queue_id].kernel_context.regs[j]=0;
        }
        pcb[queue_id].pid=process_id++;
        pcb[queue_id].type=sched1_tasks[i]->type;
        pcb[queue_id].status=TASK_READY;

        pcb[queue_id].kernel_stack_top=stack_top;
        pcb[queue_id].kernel_context.regs[29]=stack_top;
        stack_top-=STACK_SIZE;

        pcb[queue_id].kernel_context.regs[31]=sched1_tasks[i]->entry_point;
        queue_push(&ready_queue,(void *)&pcb[queue_id]);
    }

    for(i=0;i<num_lock_tasks;i++,queue_id++){
        for(j=0;j<32;j++){
            pcb[queue_id].kernel_context.regs[j]=0;
        }
        pcb[queue_id].pid=process_id++;
        pcb[queue_id].type=lock_tasks[i]->type;
        pcb[queue_id].status=TASK_READY;

        pcb[queue_id].kernel_stack_top=stack_top;
        pcb[queue_id].kernel_context.regs[29]=stack_top;
        stack_top-=STACK_SIZE;

        pcb[queue_id].kernel_context.regs[31]=lock_tasks[i]->entry_point;
        queue_push(&ready_queue,(void *)&pcb[queue_id]);
    }
     current_running=&pcb[0];
}
```

When is context switching in this project?
Provide the workflow or pseudo code of the context switching

```
while (1)
        do_scheduler();
```

do_scheduler(); in 2 lock tasks

```
save regs(except k0,k1)
```

```
if(!queue_is_empty(&ready_queue))
        next_running=(pcb_t *)queue_dequeue(&ready_queue);
else
        next_running=current_running;

if(current_running->status!=TASK_BLOCKED){
        current_running->status=TASK_READY;
        if(current_running->pid!=1){
                queue_push(&ready_queue,current_running);
        }
}
current_running=next_running;
current_running->status=TASK_RUNNING;
```

```
restore regs(except k0,k1)
```

When a task is blocked?

acquire lock failed

How does the kernel handle the blocked task?

lock release

```
void do_mutex_lock_acquire(mutex_lock_t *lock)
{
        if(lock->status==LOCKED){
                do_block(&block_queue);
        }
        else
                lock->status=LOCKED;
}

void do_mutex_lock_release(mutex_lock_t *lock)
{
        if(!queue_is_empty(&block_queue)){
                do_unblock_one(&block_queue);
                lock->status=LOCKED;
        }
        else
                lock->status=UNLOCKED;

}
```

Thanks