

# Lab 1: Scheduling

## Information

吴悦天

[22000131\\_2](#)

徐陈皓

## Alarm Clock

### Data Structures

```
1      declaration
      30

pub struct Alarm(Mutex<Vec<(Arc<Thread>, i64)>>>);
```

### Algorithms

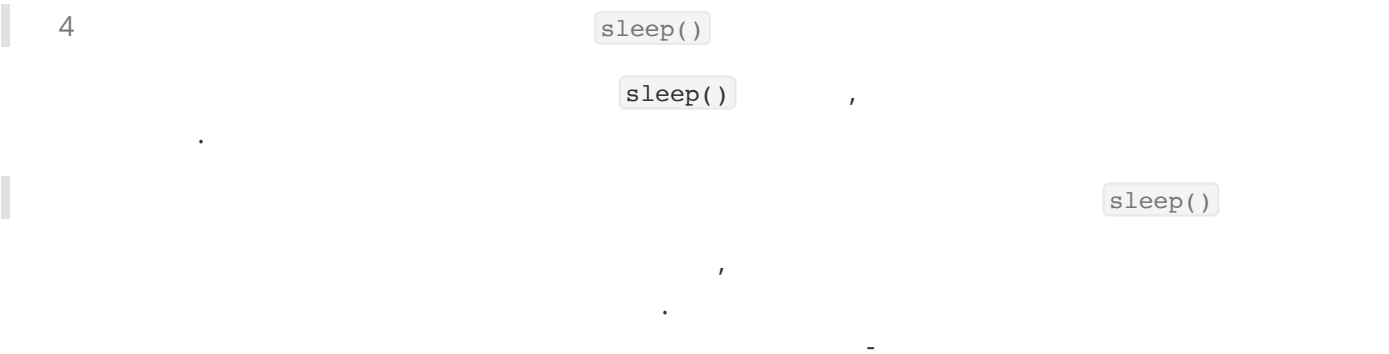
```
2      sleep()

      sleep()

      sleep()

3
```

# Synchronization



## Priority Scheduling

### Data Structures

```
1      declaration
      30

pub struct Thread {
    tid: isize,
    name: &'static str,
    stack: usize,
    status: Mutex<Status>,
    context: Mutex<Context>,
    pub priority: AtomicU32,

    #[cfg(feature = "thread-scheduler-priority")]
    pub effective_priority: AtomicU32,

    #[cfg(feature = "thread-scheduler-priority")]
    pub donee: Mutex<Option<Arc<Thread>>>,

    #[cfg(feature = "thread-scheduler-priority")]
    pub donors: Mutex<Vec<Arc<Thread>>>,

    pub userproc: Option<UserProc>,
    pub pagetable: Option<Mutex<PageTable>>,
}

pub struct PriorityScheduler;

pub struct Queue(Mutex<Vec<Arc<Thread>>>, Intr>);
```

```
pub struct Donate;
```

```
2
```

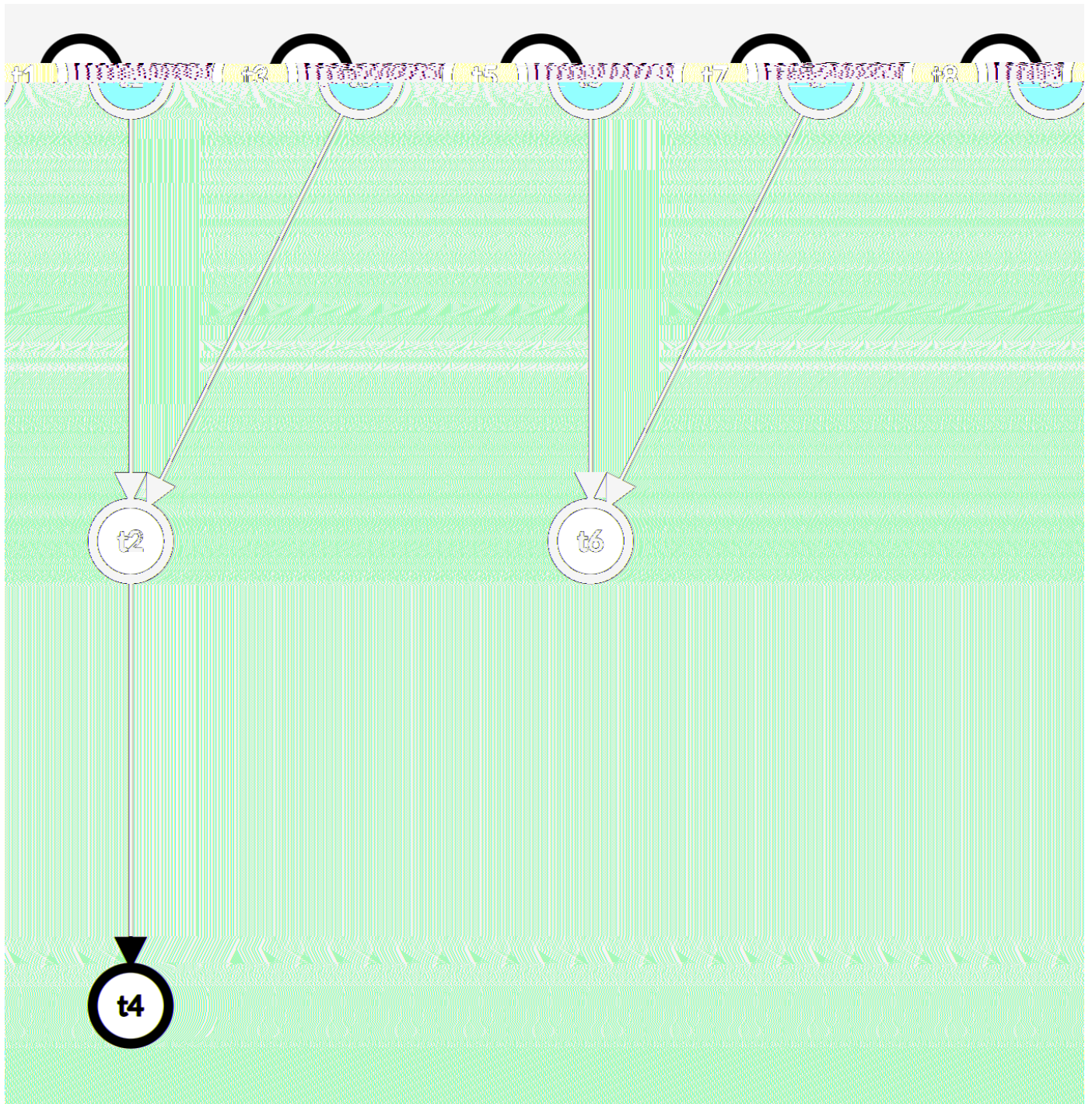
```
( .. , ).
```

```
, effective_priority, donee, donors.
```

```
( .. , ( .. , donee
```

```
-- , donors
```

```
( .. ).
```



## Algorithms

3

Intr

Sleep

schedule

4

1. `lock` is a `std::lock_guard` object that is created when the thread enters the critical section and is destroyed when the thread exits the critical section.
2. `down` is a `std::atomic<int>` object that is used to keep track of the number of threads that are currently in the critical section.
3. `lock` is a `std::lock_guard` object that is created when the thread enters the critical section and is destroyed when the thread exits the critical section.

(`lock`, `down`, `lock`.)

1. `lock` is a `std::lock_guard` object that is created when the thread enters the critical section and is destroyed when the thread exits the critical section.
2. `down` is a `std::atomic<int>` object that is used to keep track of the number of threads that are currently in the critical section.
3. `up` is a `std::atomic<int>` object that is used to keep track of the number of threads that are currently in the critical section.

## Synchronization

`thread::set_priority()`

`thread::set_priority()`

*t*

`thread::get_priority()`,

`set_priority()`,

## Rationale

1

1. `lock` is a `std::lock_guard` object that is created when the thread enters the critical section and is destroyed when the thread exits the critical section.
2. `down` is a `std::atomic<int>` object that is used to keep track of the number of threads that are currently in the critical section.