ISCAS NIST

循序渐进，学习开发一个 RISC-V 上的操作系统

第 15 章 软件定时器

汪辰

2021/4

本章内容

- 软件定时器的分类
- 软件定时器的设计和实现
- 软件定时器的优化

本章内容

- 软件定时器的分类
- 软件定时器的设计和实现
- 软件定时器的优化

定时器分类

- **硬件定时器**：芯片本身提供的定时器，一般由外部晶振提供，提供寄存器设置超时时间，并采用外部中断方式通知 CPU，参考 第 12 章介绍。优点是精度高，但定时器个数受硬件芯片的设计限制。

- **软件定时器**：操作系统中基于硬件定时器提供的功能，采用软件方式实现。扩展了硬件定时器的限制，可以提供数目更多（几乎不受限制）的定时器；缺点是精度较低，必须是 Tick 的整数倍。

软件定时器的分类

➢ 按照定时器设定方式分：

- 单次触发定时器：创建后只会触发一次定时器通知事件，触发后该定时器自动停止（销毁）
- 周期触发定时器：创建后按照设定的周期无限循环触发定时器通知事件，直到用户手动停止。

➢ 按照定时器超时后执行处理函数的上下文环境分：

- 超时函数运行在中断上下文环境中，要求执行函数的执行时间尽可能短，不可以执行等待其他事件等可能导致中断控制路径挂起的操作。优点是响应比较迅速，实时性较高。
- 超时函数运行在任务上下文环境中，即创建一个任务来执行这个函数，函数中可以等待或者挂起，但实时性较差。

本章内容

- 软件定时器的分类
- **软件定时器的设计和实现**
- 软件定时器的优化

软件定时器的设计

code/os/10-swtimer/os.h

```
/* software timer */
struct timer {
        void (*func)(void *arg);
        void *arg;
        uint32_t timeout_tick;
};

extern struct timer *timer_create(
        void (*handler)(void *arg),
        void *arg,
        uint32_t timeout);

extern void timer_delete(struct timer *timer);
```

code/os/10-swtimer/timer.c

```
#define MAX_TIMER 10
static struct timer timer_list[MAX_TIMER];
```

code/os/10-swtimer/timer.c

```
void timer_init()
{
        struct timer *t = &(timer_list[0]);
        for (int i = 0; i < MAX_TIMER; i++) {
                t->func = NULL;
                t->arg = NULL;
                t++;
        }
        ......
}
```

软件定时器的设计

code/os/10-swtimer/os.h

```
/* software timer */
struct timer {
        void (*func)(void *arg);
        void *arg;
        uint32_t timeout_tick;
};

extern struct timer *timer_create(
        void (*handler)(void *arg),
        void *arg,
        uint32_t timeout);

extern void timer_delete(struct timer *timer);
```

code/os/10-swtimer/timer.c

```
#define MAX_TIMER 10
static struct timer timer_list[MAX_TIMER];
```

code/os/10-swtimer/timer.c

```
struct timer *timer_create(
        void (*handler)(void *arg),
        void *arg,
        uint32_t timeout)
{
        ......
        struct timer *t = &(timer_list[0]);
        for (int i = 0; i < MAX_TIMER; i++) {
                if (NULL == t->func) {
                        break;
                }
                t++;
        }
        ......
        t->func = handler;
        t->arg = arg;
        t->timeout_tick = _tick + timeout;
        ......
        return t;
}
```

软件定时器的设计

code/os/10-swtimer/os.h

```
/* software timer */
struct timer {
        void (*func)(void *arg);
        void *arg;
        uint32_t timeout_tick;
};

extern struct timer *timer_create(
        void (*handler)(void *arg),
        void *arg,
        uint32_t timeout);

extern void timer_delete(struct timer *timer);
```

code/os/10-swtimer/timer.c

```
#define MAX_TIMER 10
static struct timer timer_list[MAX_TIMER];
```

code/os/10-swtimer/timer.c

```
void timer_delete(struct timer *timer)
{
        ......
        struct timer *t = &(timer_list[0]);
        for (int i = 0; i < MAX_TIMER; i++) {
                if (t == timer) {
                        t->func = NULL;
                        t->arg = NULL;
                        break;
                }
                t++;
        }
        ......
}
```

软件定时器的设计

code/os/10-swtimer/trap.c

```
reg_t trap_handler(reg_t epc, reg_t cause)
{
        ......
        if (cause & 0x80000000) {
                switch (cause_code) {
                ......
                case 7:
                        timer_handler();
                        break;
                ......
}
```

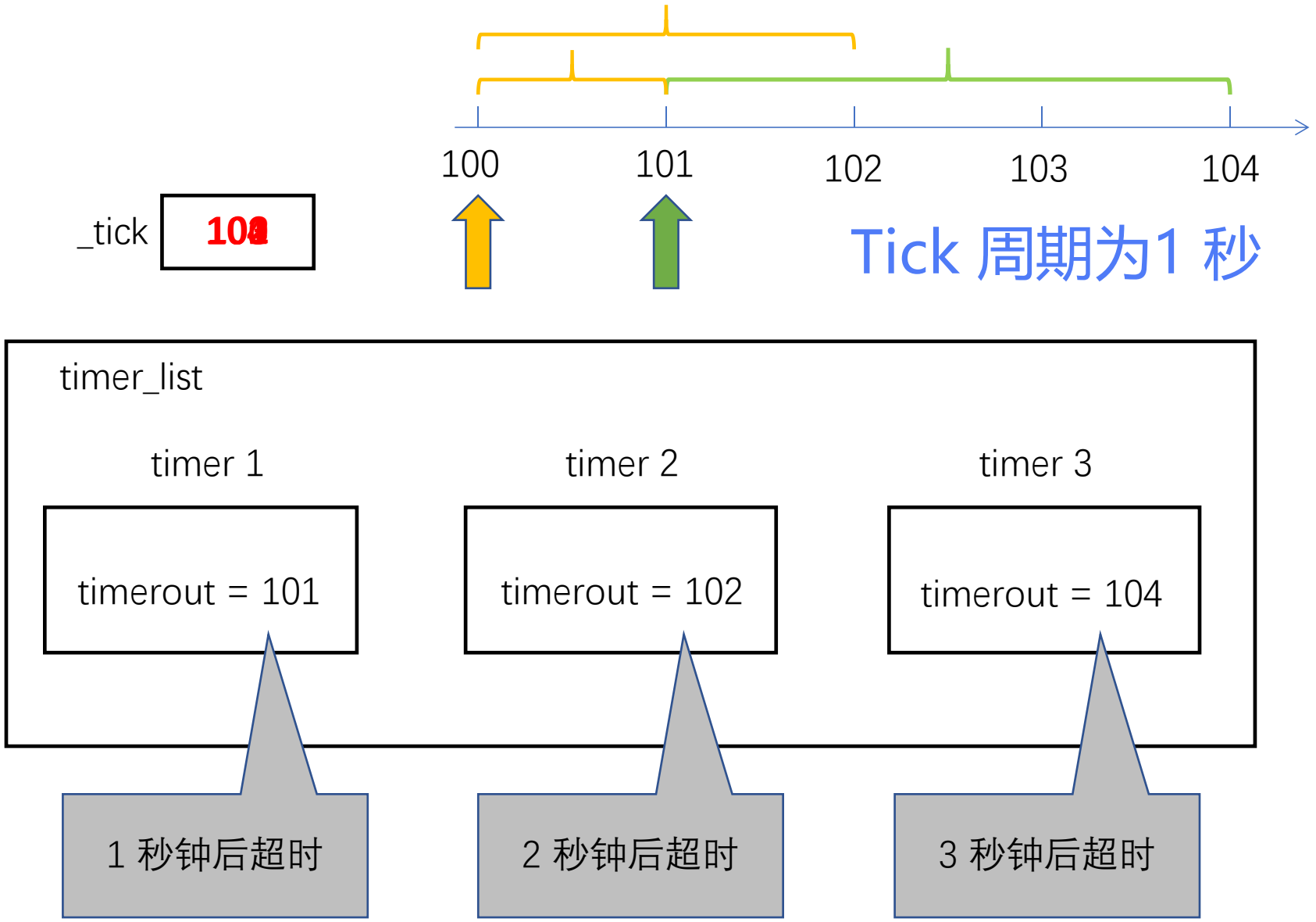
code/os/10-swtimer/timer.c

```
void timer_handler()
{
        _tick++;
        timer_check();
        timer_load(TIMER_INTERVAL);
        schedule();
}
```

code/os/10-swtimer/timer.c

```
static inline void timer_check()
{
        struct timer *t = &(timer_list[0]);
        for (int i = 0; i < MAX_TIMER; i++) {
                if (NULL != t->func) {
                        if (_tick >= t->timeout_tick) {
                                t->func(t->arg);
                                t->func = NULL;
                                t->arg = NULL;
                                break;
                        }
                }
                t++;
        }
}
```

软件定时器的设计


100
101
102
103
104
_tick
100
Tick 周期为1 秒
timer_list
timer 1
timer 2
timer 3
timerout = 101
timerout = 102
timerout = 104
1 秒钟后超时
2 秒钟后超时
3 秒钟后超时

本章内容

- 软件定时器的分类
- 软件定时器的设计和实现
- 软件定时器的优化

软件定时器的优化（1）

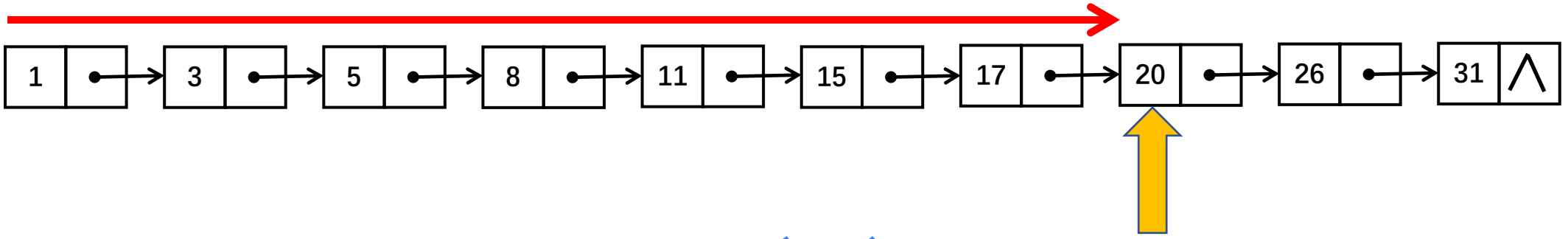
code/os/10-swtimer/timer.c

```
struct timer *timer_create(
        void (*handler)(void *arg),
        void *arg,
        uint32_t timeout)
{
        ......
        struct timer *t = &(timer_list[0]);
        for (int i = 0; i < MAX_TIMER; i++) {
                if (NULL == t->func) {
                        break;
                }
                t++;
        }
        ......
        t->func = handler;
        t->arg = arg;
        t->timeout_tick = _tick + timeout;
        ......
        return t;
}
```

- 定时器按照超时时间排序
- 链表方式实现对定时器的管理

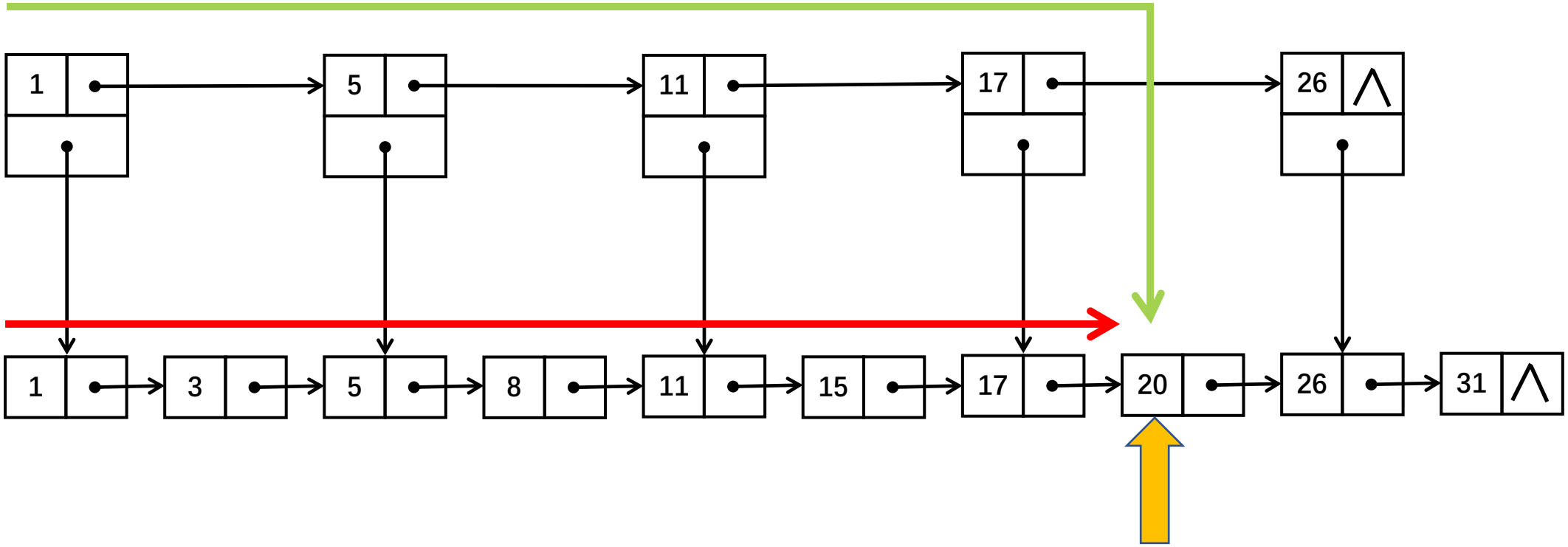
练习 15-1

软件定时器的优化（2）

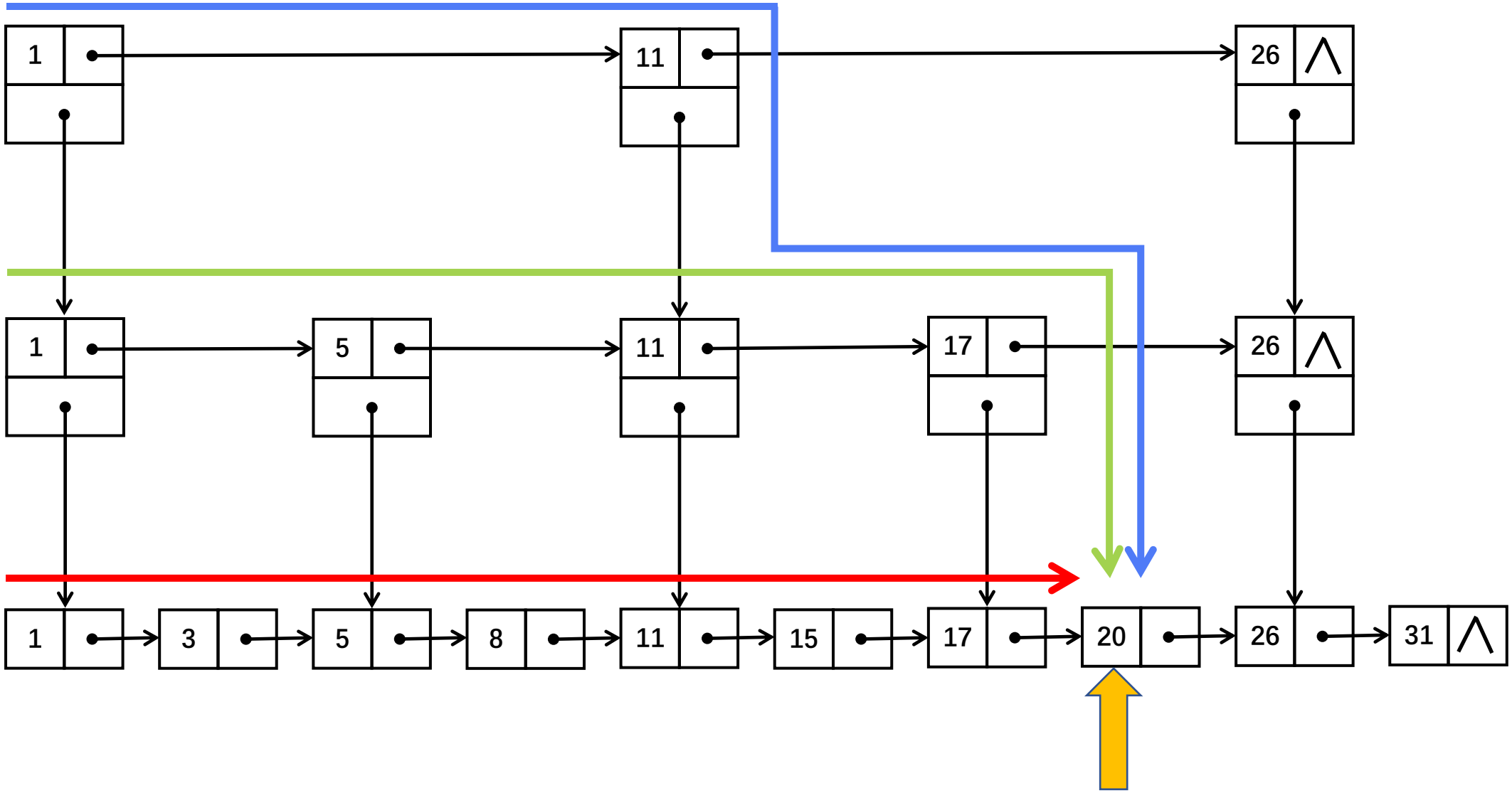


$O(n)$

软件定时器的优化（2）

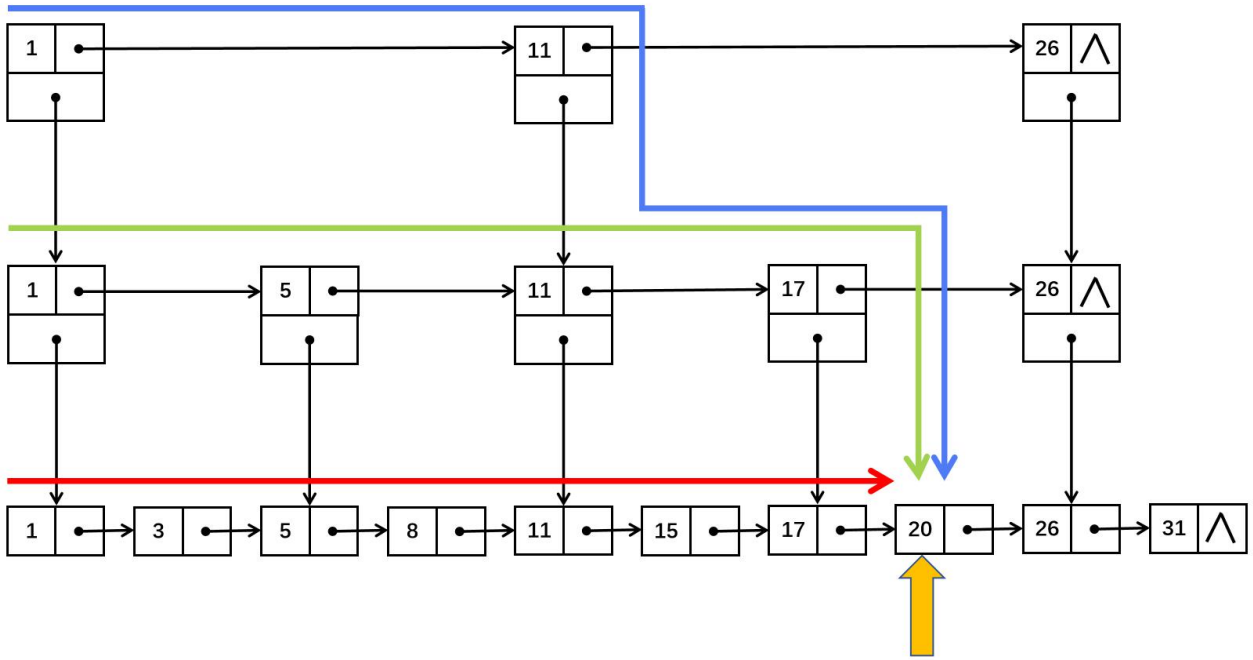


软件定时器的优化（2）



软件定时器的优化（2）

➢ 跳表（Skip List）算法



$O(log(n))$

练习 15-2

练习 15-1

练习 15-2

练习 15-3

谢 谢

欢迎交流合作