Operating Systems

Introduction to Lab 6

Department of Computer Science & Technology
Tsinghua University

Outline

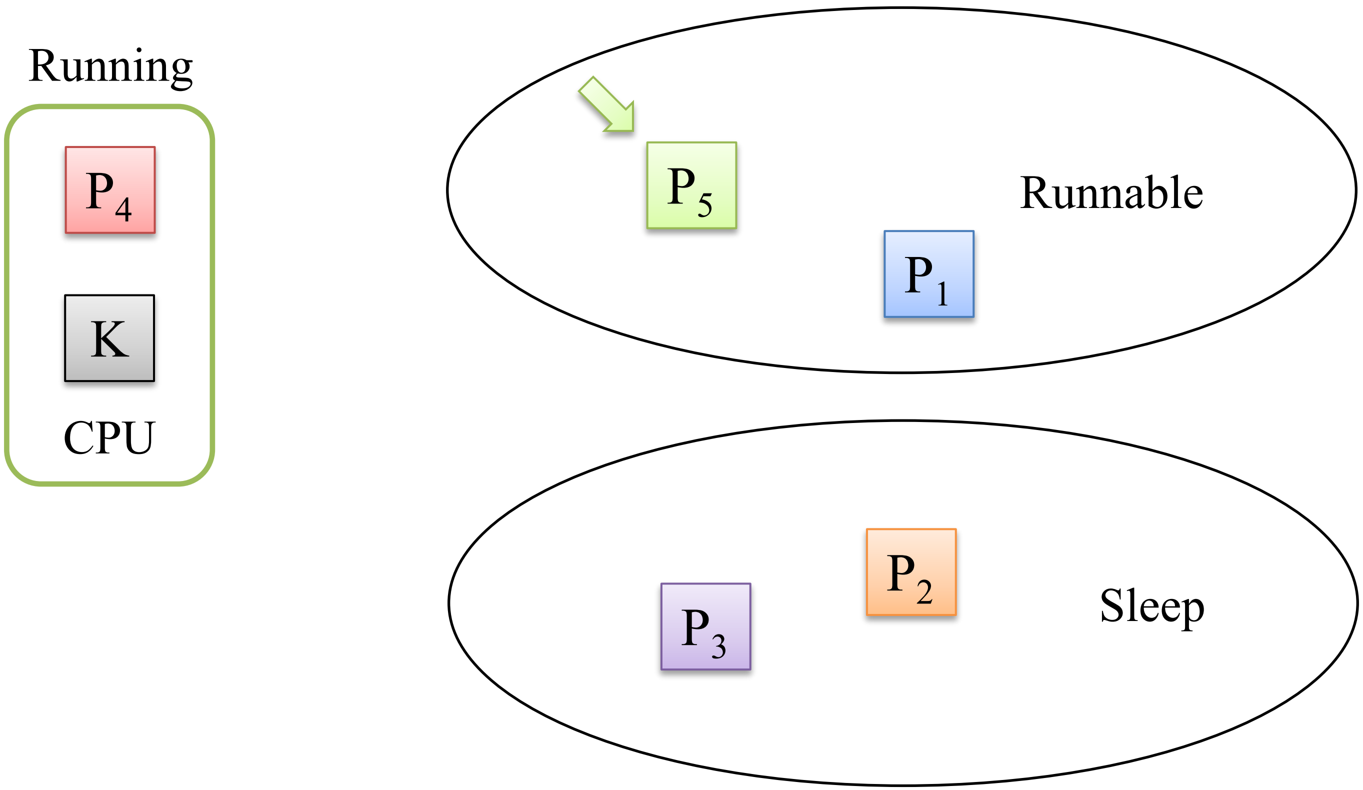
- Scheduling events
- Scheduling algorithm framework
- Round Robin & Stride scheduling

SCHEDULING EVENTS

Scheduling events - overview

1. trigger scheduling
2. pick up
3. ‘enqueue’
4. ‘dequeue’
5. process switch

HOWTO?



SCHEDULING ALGORITHM FRAMEWORK

Scheduling algorithm framework

1. trigger scheduling
2. pick up
3. ‘enqueue’
4. ‘dequeue’
5. process switch

- We need to find scheduling-algorithm-specific operations in these events…

Scheduling algorithm framework

1. trigger scheduling
2. pick up
3. ‘enqueue’
4. ‘dequeue’
5. process switch

- A process exits: *do_exit()* @ proc.c:480
- A parent process waits for its child to exit: *do_wait()* @ proc.c:709
- The ancestor process waits for all children to exit: *init_main()* @ proc.c:807
- The idle loop: *cpu_idle()* @ proc.c:861
- Failed to acquire locks: *lock()* @ sync.h:45
- A process yields its time slice: *trap()* @ trap.c:292
- A process uses up its time slice: *trap()* @ trap.c:292

This is sched-algorithm specific…

Scheduling algorithm framework

1. trigger scheduling *proc_tick*
2. pick up
3. ‘enqueue’
4. ‘dequeue’
5. process switch

- Q: How can scheduling algorithms track time usage of processes?
- A: Make the algorithm aware of timer interrupts!

Scheduling algorithm framework

1. trigger scheduling *proc_tick*
2. pick up *pick_next*
3. ‘enqueue’
4. ‘dequeue’
5. process switch

- This is the key work of scheduling algorithms…

Scheduling algorithm framework

1. trigger scheduling *proc_tick*
2. pick up *pick_next*
3. ‘enqueue’ *enqueue*
4. ‘dequeue’ *dequeue*
5. process switch

- Put a process into a ‘run queue’
- We may not know how the queue is implemented, or how a process should be inserted…

Scheduling algorithm framework

1.	trigger scheduling	*proc_tick*
2.	pick up	*pick_next*
3.	‘enqueue’	*enqueue*
4.	‘dequeue’	*dequeue*
5.	process switch	

- There is nothing scheduling algorithms should care here…

Scheduling algorithm framework

1. trigger scheduling *proc_tick*
2. pick up *pick_next*
3. ‘enqueue’ *enqueue*
4. ‘dequeue’ *dequeue*
5. process switch

```
struct sched_class {
    const char *name;
    void (*init)(struct run_queue *rq);
    void (*enqueue)(struct run_queue *rq, struct proc_struct *proc);
    void (*dequeue)(struct run_queue *rq, struct proc_struct *proc);
    struct proc_struct *(*pick_next)(struct run_queue *rq);
    void (*proc_tick)(struct run_queue *rq, struct proc_struct *proc);
}
```

Scheduling algorithm framework

```
void schedule(void) {
    bool intr_flag;
    struct proc_struct *next;
    local_intr_save(intr_flag);
    {
        current->need_resched = 0;
        if (current->state == PROC_RUNNABLE) {
            sched_class_enqueue(current);
        }
        if ((next = sched_class_pick_next()) != NULL) {
            sched_class_dequeue(next);
        }
        if (next == NULL) {
            next = idleproc;
        }
        next->runs ++;
        if (next != current) {
            proc_run(next);
        }
    }
    local_intr_restore(intr_flag);
}
```

ROUND ROBIN & STRIDE SCHEDULING

Round Robin scheduling – initialization (default_sched.c)

```
static void
RR_init(struct run_queue *rq) {
    list_init(&(rq->run_list));
    rq->proc_num = 0;
}
```

```
struct run_queue {
    list_entry_t run_list;
    unsigned int proc_num;
    int max_time_slice;
    // For LAB6 ONLY
    skew_heap_entry_t *lab6_run_pool;
};
```

Round Robin scheduling – proc_tick (default_sched.c)

```
static void
RR_proc_tick(struct run_queue *rq, struct proc_struct *proc) {
    if (proc->time_slice > 0) {
        proc->time_slice --;
    }
    if (proc->time_slice == 0) {
        proc->need_resched = 1;
    }
}
```

current process

```
struct run_queue {
    list_entry_t run_list;
    unsigned int proc_num;
    int max_time_slice;
    // For LAB6 ONLY
    skew_heap_entry_t *lab6_run_pool;
};
```

Round Robin scheduling – pick_next (default_sched.c)

```
static struct proc_struct *
RR_pick_next(struct run_queue *rq) {
    list_entry_t *le = list_next(&(rq->run_list));
    if (le != &(rq->run_list)) {
        return le2proc(le, run_link);
    }
    return NULL;
}
```

Q: NULL?!

A: NULL will be replaced by ‘idle’ in the framework

```
struct run_queue {
    list_entry_t run_list;
    unsigned int proc_num;
    int max_time_slice;
    // For LAB6 ONLY
    skew_heap_entry_t *lab6_run_pool;
};
```

Round Robin scheduling – enqueue (default_sched.c)

```
static void
RR_enqueue(struct run_queue *rq, struct proc_struct *proc) {
    list_add_before(&(rq->run_list), &(proc->run_link));
    if (proc->time_slice == 0 ||
        proc->time_slice > rq->max_time_slice) {
        proc->time_slice = rq->max_time_slice;
    }
    proc->rq = rq;
    rq->proc_num ++;
}
```

```
struct run_queue {
    list_entry_t run_list;
    unsigned int proc_num;
    int max_time_slice;
    // For LAB6 ONLY
    skew_heap_entry_t *lab6_run_pool;
};
```

Round Robin scheduling – dequeue (default_sched.c)

```
static void
RR_dequeue(struct run_queue *rq, struct proc_struct *proc) {
    list_del_init(&(proc->run_link));
    rq->proc_num --;
}
```

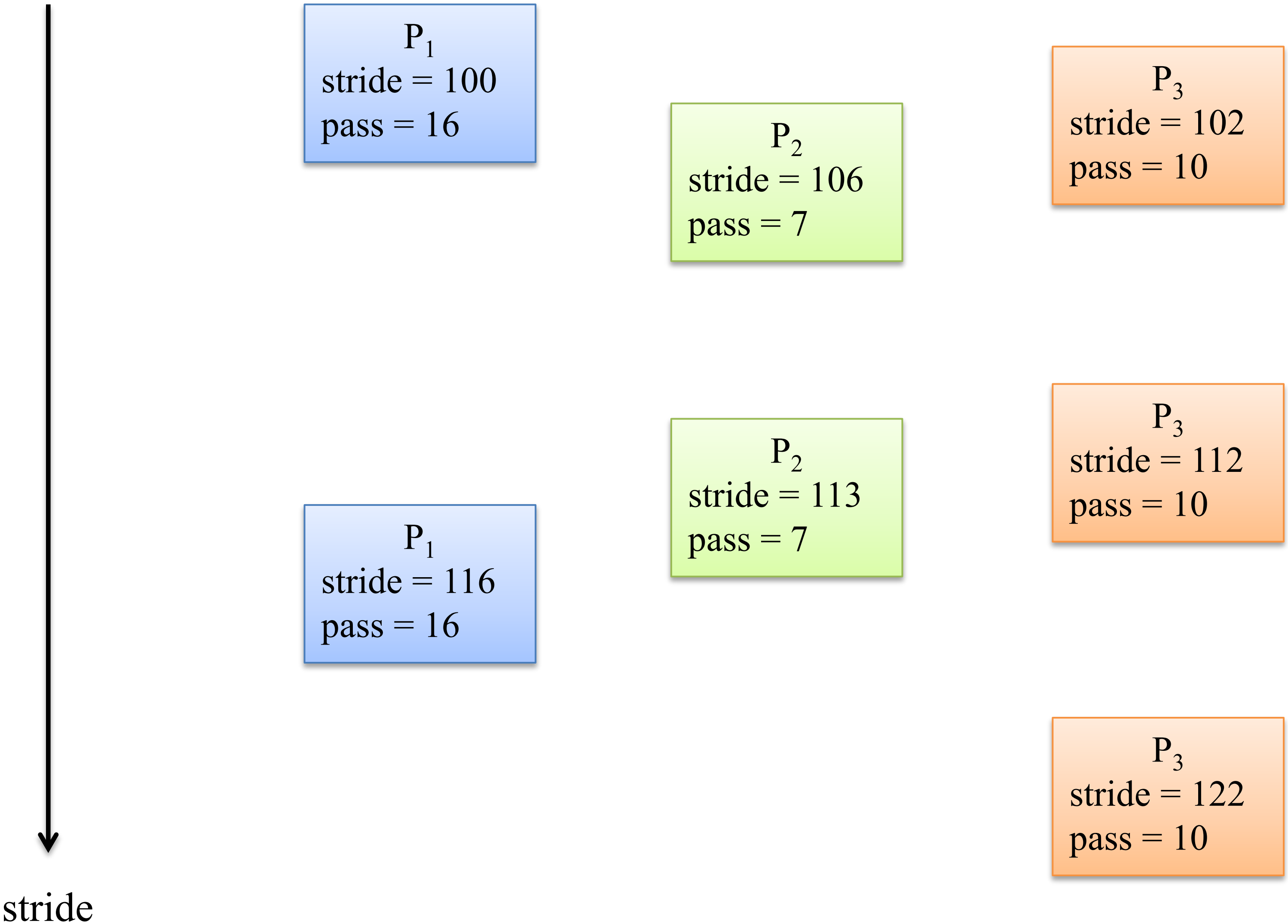
```
struct run_queue {
    list_entry_t run_list;
    unsigned int proc_num;
    int max_time_slice;
    // For LAB6 ONLY
    skew_heap_entry_t *lab6_run_pool;
};
```

Round Robin scheduling – exporting

```
===== default_sched.c =====
struct sched_class default_sched_class = {
    .name = "RR_scheduler",
    .init = RR_init,
    .enqueue = RR_enqueue,
    .dequeue = RR_dequeue,
    .pick_next = RR_pick_next,
    .proc_tick = RR_proc_tick,
};

===== sched.c =====
void sched_init(void) {
    ......
    sched_class = &default_sched_class;
    ......
}
```

Stride scheduling – overview



Stride scheduling – characteristics

- Priority-based
- Deterministic

Stride scheduling – Implementation (YOUR WORK!)

- Choose a proper data structure (list, priority queue, etc.)
  - Initialize your structure in *init()*
  - Update your structure in *enqueue()* and *dequeue()*
- Implement the algorithm for choosing next task in *pick_next()*
- Handle timer ticks in *proc_tick()*
  - Set *proc->need_resched* if you think this process has used up its time slice
- Construct a *sched_class* for your scheduling algorithm and replace *default_sched_class* with it in *sched_init()*
- Test your algorithm with '**make run-priority**' to see if it works as expected

Stride scheduling – Skew heap

```
struct skew_heap_entry {
     struct skew_heap_entry *parent, *left, *right;
};
typedef int(*compare_f)(void *a, void *b);

void skew_heap_init(skew_heap_entry_t *a);
skew_heap_entry_t *skew_heap_insert(
     skew_heap_entry_t *a, skew_heap_entry_t *b,
     compare_f comp);
skew_heap_entry_t *skew_heap_remove(
     skew_heap_entry_t *a, skew_heap_entry_t *b,
     compare_f comp);
```

Stride scheduling – Specific fields in structures

```
struct proc_struct {
    ……
    // For constructing skew heap
    // Use le2proc(proc, lab6_run_pool) to get the PCB
    skew_heap_entry_t lab6_run_pool;
    uint32_t lab6_stride;                   // For your algorithm
    uint32_t lab6_priority                  // Set by syscall;
};

struct run_queue {
    list_entry_t run_list;
    unsigned int proc_num;
    int max_time_slice;
    // For LAB6 ONLY
    skew_heap_entry_t *lab6_run_pool; // The queue you use
};
```

Stride scheduling – more notes

- Relationship between *pass* and priority?
  - $pass = \frac{BIG\_VALUE}{priority}$
- How to handle stride overflow?
  - Though x or y may overflow, we can still tell which is bigger according to $(x - y)$ as long as the modulus of the result is not too big

Stride scheduling – references

- **C. A. Waldspurger and E. Weihl. W.** *Stride Scheduling: Deterministic Proportional- Share Resource Management*, 1995 URL: http://dl.acm.org/citation.cfm?id=889650

That's all. Thanks!