Pintos threads and scheduling

Recap

- So far we know how to
  - run and debug pintos tests
  - add our own tests
  - make use of `lib/kernel/list.h`

Threads

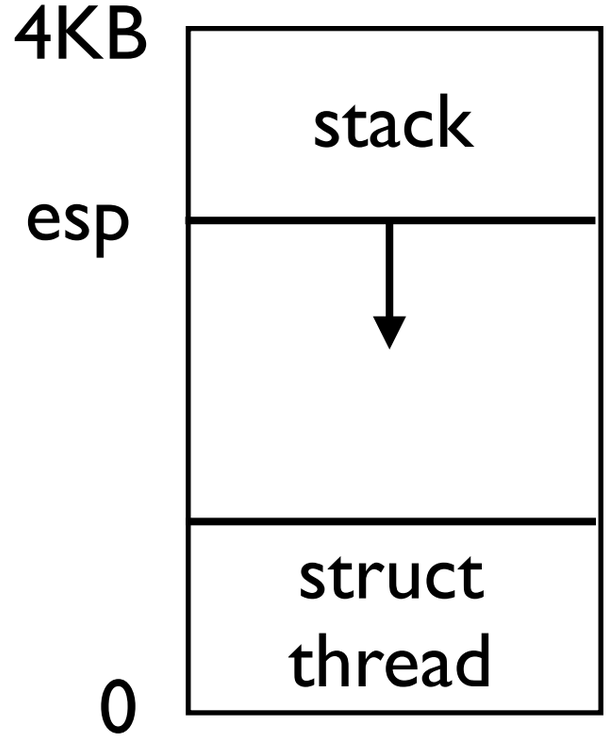
- Baseline kernel (`pintos/threads/`)
  - Multiple threads running in kernel mode
  - Threads are similar to processes
  - Share the same address space

Thread control block

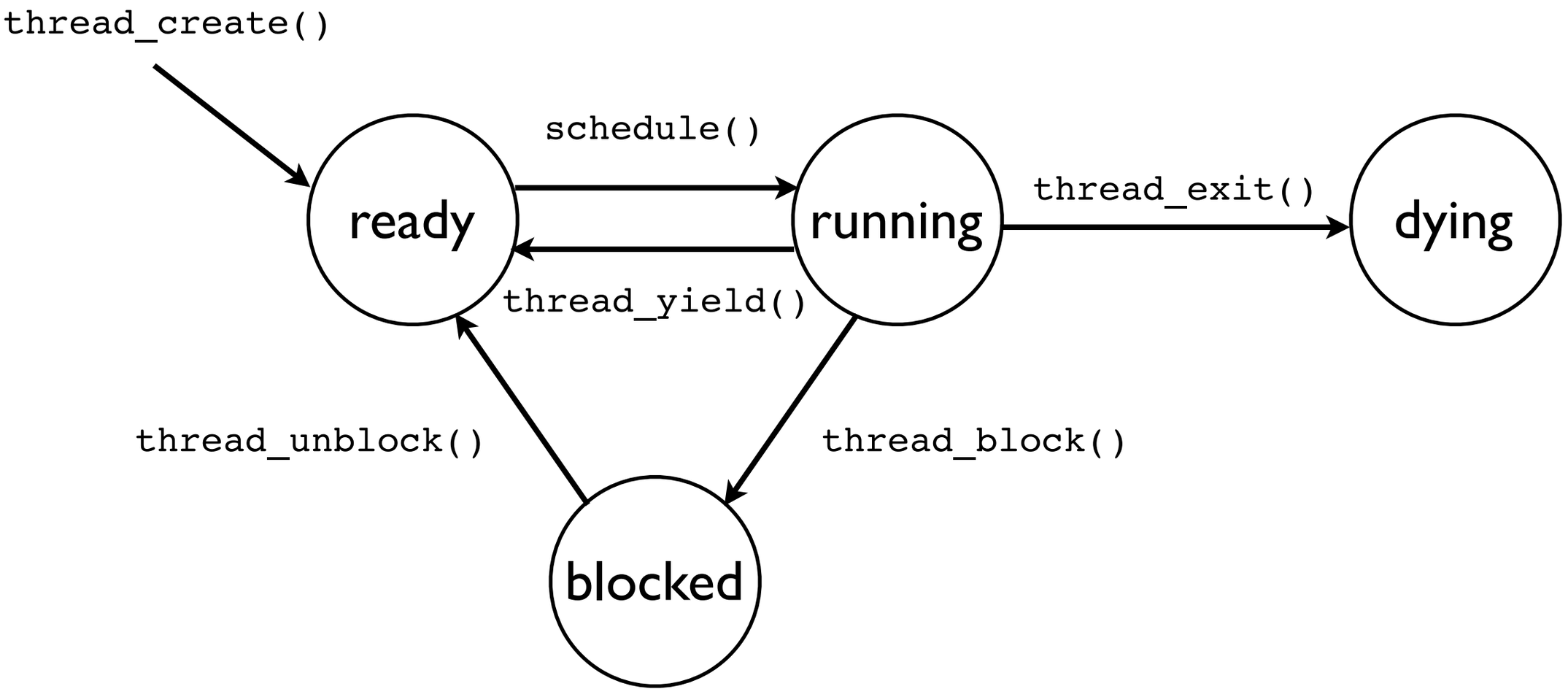
- Kernel maintains Thread Control Blocks (see `struct thread` in `threads.h`)
  - tid (thread id)
  - thread name (for debugging)
  - stack pointer
  - status (ready, running, blocked, dying)

Threads

- Each thread gets a page of memory



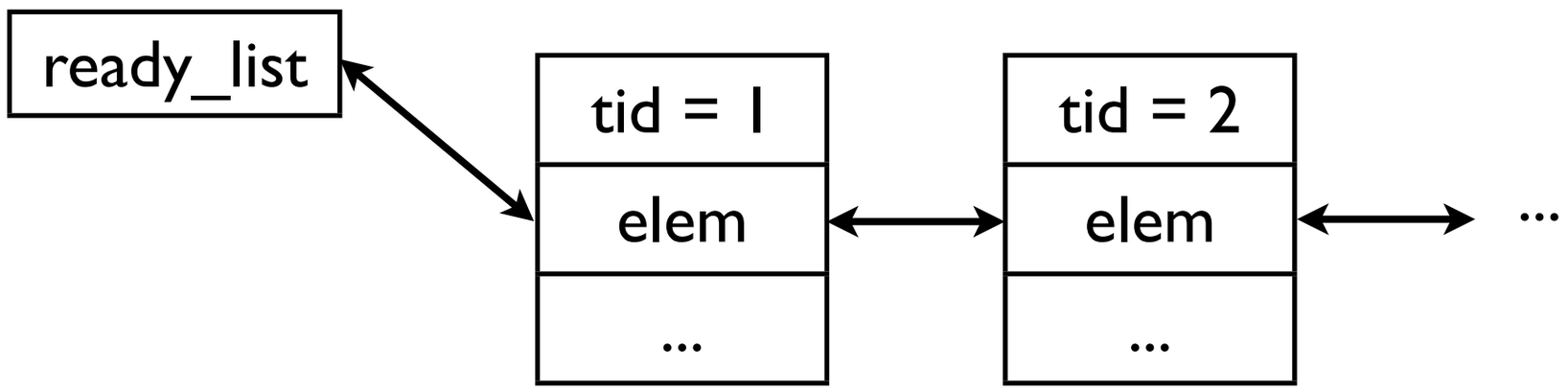
Thread states



Scheduling

- Preemptive round-robin scheduler
  - Executes one thread at a time
    (only one thread is in the running state)
  - Each thread is given a time slice
  - Switch threads when time slice expires

Ready list



- Push current thread to tail of the list
- Head of the list is the next thread to run

Scheduler

```
static void
schedule (void)
{
  struct thread *cur = running_thread ();
  struct thread *next = next_thread_to_run ();
  ...
  if (cur != next)
    prev = switch_threads (cur, next);
  ...
}

static struct thread *
next_thread_to_run (void)
{
  if (list_empty (&ready_list))
    return idle_thread;
  else
    return list_entry (list_pop_front (&ready_list), struct thread, elem);
}
```

Thread Switching

- The magic happens in `switch_threads(cur, next)`
  - Save registers on the stack
  - Save CPU's stack pointer in `cur->stack`
  - CPU stack pointer gets `next->stack`
  - Restore registers from stack

Timer interrupts

- Timer interrupts (see `devices/timer.c`)
  - Generates `TIMER_FREQ` "ticks" per second
  - Threads preempted every `TIME_SLICE` ticks

Timer interrupts

```c
/* timer.c */
static void
timer_interrupt (struct intr_frame *args UNUSED)
{
  ticks++;
  thread_tick ();
}
```

```c
/* thread.c */
void
thread_tick (void)
{
  struct thread *t = thread_current ();
  /* Update statistics. */
  ...
  /* Enforce preemption. */
  if (++thread_ticks >= TIME_SLICE)
    intr_yield_on_return ();
}
```

Timer interrupts

```c
/* timer.c */
static void
timer_interrupt (struct intr_frame *args UNUSED)
{
  ticks++;
  thread_tick ();
}
```

```c
/* thread.c */
void
thread_tick (void)
{
  struct thread *t = thread_current ();
  /* Update statistics. */
  ...
  /* Enforce preemption. */
  if (++thread_ticks >= TIME_SLICE)
    intr_yield_on_return ();
}
```

`intr_yield_on_return()` causes the interrupt handler to call `thread_yield()` before returning

Synchronization

- Pintos provides several synchronization primitives
  - Semaphores
  - Locks
  - Condition variables
- We will study these mechanisms later on...

Synchronization

- For now: access to shared data should be protected by disabling interrupts

```
enum intr_level old_level;
old_level = intr_disable ();

/* critical section */

intr_set_level (old_level);
```

Assignment

- Reimplement `timer_sleep()`
- Current implementation

```
void
timer_sleep (int64_t ticks)
{
  int64_t start = timer_ticks ();
  while (timer_elapsed (start) < ticks)
    thread_yield ();
}
```

- What is the problem???

Assignment

- Reimplement `timer_sleep()`
- Current implementation

```
void
timer_sleep (int64_t ticks)
{
  int64_t start = timer_ticks ();
  while (timer_elapsed (start) < ticks)
    thread_yield ();
}
```

- What is the problem??? Busy wait!

Assignment

- Your submission should
  - avoid busy waiting
  - not brake tests that already pass!
  - ideally, be efficient

Assignment

```
$ make check
....
pass tests/threads/alarm-single
pass tests/threads/alarm-multiple
pass tests/threads/alarm-simultaneous
FAIL tests/threads/alarm-priority
pass tests/threads/alarm-zero
pass tests/threads/alarm-negative
FAIL tests/threads/priority-change
FAIL tests/threads/priority-donate-one
FAIL tests/threads/priority-donate-multiple
FAIL tests/threads/priority-donate-multiple2
FAIL tests/threads/priority-donate-nest
FAIL tests/threads/priority-donate-sema
FAIL tests/threads/priority-donate-lower
FAIL tests/threads/priority-fifo
FAIL tests/threads/priority-preempt
FAIL tests/threads/priority-sema
FAIL tests/threads/priority-condvar
FAIL tests/threads/priority-donate-chain
FAIL tests/threads/mlfqs-load-1
FAIL tests/threads/mlfqs-load-60
FAIL tests/threads/mlfqs-load-avg
FAIL tests/threads/mlfqs-recent-1
pass tests/threads/mlfqs-fair-2
pass tests/threads/mlfqs-fair-20
FAIL tests/threads/mlfqs-nice-2
FAIL tests/threads/mlfqs-nice-10
FAIL tests/threads/mlfqs-block
20 of 27 tests failed.
```

Hints

- Use `thread_block()` and keep track of sleeping threads
- During timer interrupts, `thread_unblock()` sleeping threads if their timer expired
- Disable interrupts to protect critical sections

Readings

- Read pintos docs!
- Chapter 2 up to 2.2.2
- Appendix A.2
- Appendix A.3.1
- Skim through Appendix A.1