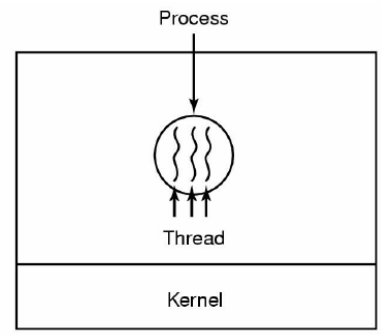
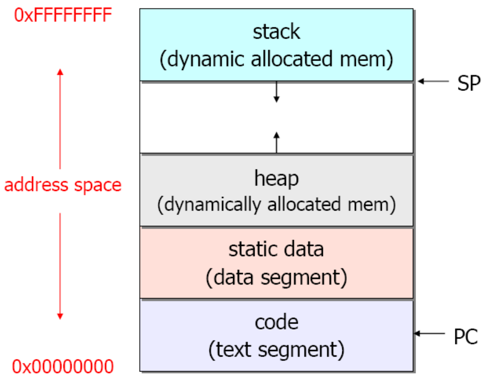
Pintos Project 1: Threads

Hoonsung Chwa
Minsup Lee
Juyoung Lee
Division of Computer Science, KAIST

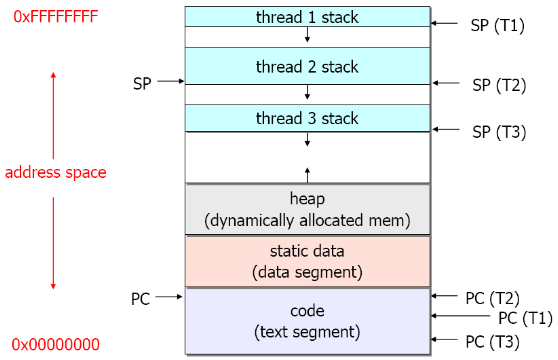
What is a Thread?

- **Thread**
  - A flow of control in a process
  - A sequence of instructions being executed in a program.
  - Threads share the process instructions and most of its data.
    - A change in shared data by one thread can be seen by the other threads in the process





1 process



threads

From Prof. Jin-Soo Kim's slides

To do

- **Alarm Clock**
- **Priority Scheduler**
- **Advanced Scheduler (Optional)**
- **Design Document**

Alarm Clock (1)

- **To suspend threads for some timer ticks**
  - Threads call timer_sleep(TICKS) in device/timer.c
  - Ex) thread_create(thread_id, NULL, sleeper, ticks)

```
/* Sleeper thread. */
static void
sleeper (void *t_)
{
  struct sleep_thread *t = t_;
  struct sleep_test *test = t->test;
  int i;

  for (i = 1; i <= test->iterations; i++)
    {
      int64_t sleep_until = test->start + i * t->duration;
      timer_sleep (sleep_until - timer_ticks ());

      lock_acquire (&test->output_lock);
      *test->output_pos++ = t->id;
      lock_release (&test->output_lock);
    }
}
```

Alarm Clock (2)

- **Requirement**
  - Must not be "busy waiting"
- **Timer_sleep() in device/timer.c**
  - Now, implemented by 'busy waiting'

```
/* Suspends execution for approximately TICKS timer ticks. */
void
timer_sleep (int64_t ticks)
{
  int64_t start = timer_ticks ();
  while (timer_elapsed (start) < ticks)
    thread_yield ();
}
```

  - Reimplement timer.c to avoid 'busy waiting'
    - Make a 'thread_list' for waiting threads.
    - When timer interrupt occurs, wake the threads up
      - » Timer_interrupt() in device/timer.c
      - » Ticks++

Priority Scheduling (1)

- **Now, Round-Robin scheduling**
  - ready_list in thread.c
  - Schedule(), next_thread_to_run() in thread.c

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

```c
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);
}
```

- **Implement priority scheduling**
  - Thread which has the highest priority in the ready_list gets CPU.

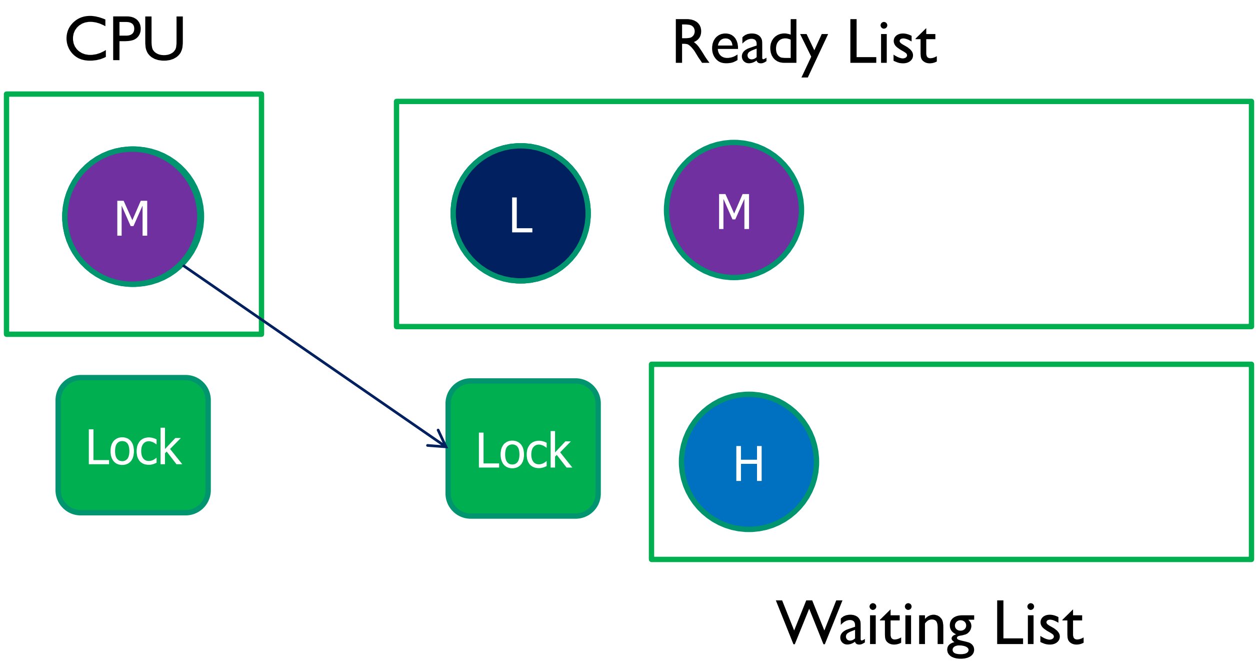
Priority Scheduling (2)

- **Requirements**
  - If a thread is added to the ready list which has a higher priority than the currently running thread, immediately yield the processor to the new thread.
  - Also, when threads are waiting for a lock, semaphore, or condition variable, the highest priority waiting thread should be woken up first.
  - Solve the 'Priority Inversion' problem of 'Lock'.
    - Implement priority inheritance (effective priority).
- **References**
  - Thread_create(), next_thread_to_run(), … in thread.c
  - Sema_up(), lock_acquire(), cond_signal(), … in synch.c

Priority Scheduling (3)

- **Priority inversion problem**
  - A situation where a higher-priority job is unable to run because a lower-priority job is holding a resource it needs, such as a lock.
- **Example**
  - There are three threads, H, M and L
    - Priority: H > M > L
  - H and L need same resource (e.g. lock) and L holds the resource
    - Expected execution order is L, H, M
  - In real, H is blocked and M is firstly executed. Then L and H.
    - P(M) > P(L)
  - Consequently, the execution order is M, L, H.

Priority Scheduling (4)



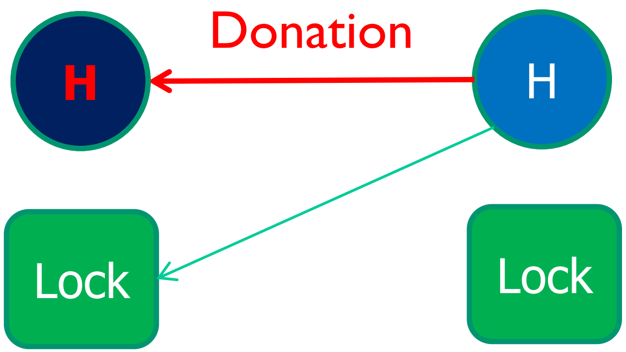
- **Input order : L, H, M**
- **Execution order : M, L, H**

From 2007 CS330's slides

Priority Scheduling (5)

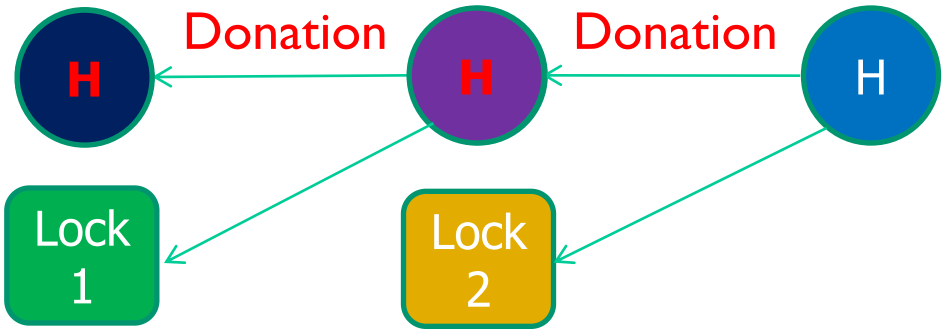
- **Priority inversion problem**
  - A situation where a higher-priority job is unable to run because a lower-priority job is holding a resource it needs, such as a lock.
- **Solution: Priority Inheritance**
  - The higher-priority job donate its priority to the lower-priority job holding the resource it requires (effective priority).
  - The beneficiary of the inheritance will now have a higher scheduling priority.
    - Get scheduled to run sooner.
  - It can finish its work and release the resource, at which point to the original priority is returned to the job.

Priority Scheduling (6)

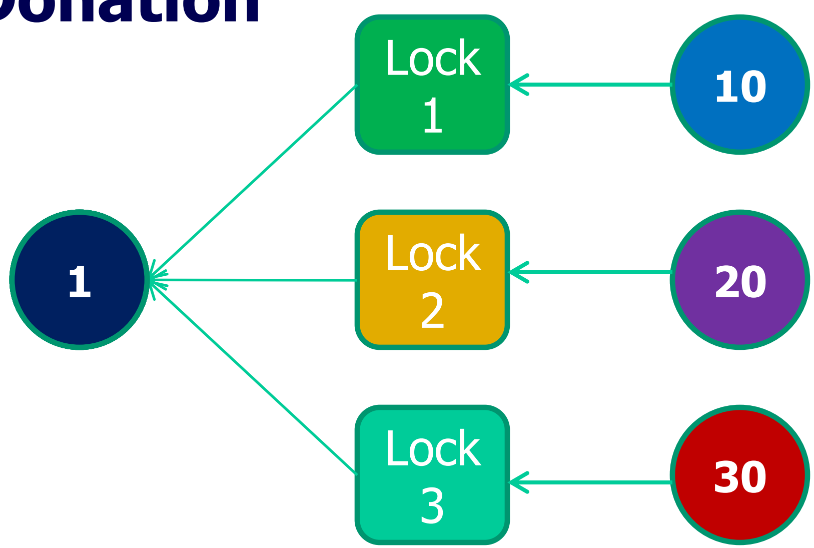


Priority Scheduling (7)

- **Nested Donation**



- **Multiple Donation**



Semaphore

- **synch.c and synch.h**

```c
/* A counting semaphore. */
struct semaphore
  {
    unsigned value;             /* Current value. */
    struct list waiters;        /* List of waiting threads. */
  };

void sema_init (struct semaphore *, unsigned value);
void sema_down (struct semaphore *);
bool sema_try_down (struct semaphore *);
void sema_up (struct semaphore *);
void sema_self_test (void);
```

- **Usage**
  - Sema_down();
  - critical section
  - Sema_up();

Lock

- **synch.c and synch.h**

```
/* Lock. */
struct lock
  {
    struct thread *holder;      /* Thread holding lock (for debugging). */
    struct semaphore semaphore; /* Binary semaphore controlling access. */
  };

void lock_init (struct lock *);
void lock_acquire (struct lock *);
bool lock_try_acquire (struct lock *);
void lock_release (struct lock *);
bool lock_held_by_current_thread (const struct lock *);
```

```
static void
a_thread_func (void *lock_)
{
  struct lock *lock = lock_;

  lock_acquire (lock);
  msg ("Thread a acquired lock a.");
  lock_release (lock);
  msg ("Thread a finished.");
}
```

→ **critical section**

Condition variable

- **synch.c and synch.h**

```
/* Condition variable. */
struct condition
  {
    struct list waiters;        /* List of waiting threads. */
  };

void cond_init (struct condition *);
void cond_wait (struct condition *, struct lock *);
void cond_signal (struct condition *, struct lock *);
void cond_broadcast (struct condition *, struct lock *);
```

- **Usage**
  - Thread1
    - Cond_wait(&cv, &lock_cv);
  - Thread2
    - Cond_signal(&cv, &lock_cv); or cond_broadcast(&cv, &lock_cv);
  - A lock is required to prevent race condition.

Advanced Scheduler

- **It is optional.**
- **If you implement *advanced scheduler(4.4BSD scheduler)*, you can get extra score.**
- **Read the below link carefully**
  - http://www.scs.stanford.edu/07au-cs140/pintos/pintos_7.html#SEC128

Submission

- **Due**
  - 10/4 (Sun) 24:00
- **Help desk**
  - 9/24 (Thur) 19:00~22:00 at Eve room(1st floor)
- **E-mail to 'cs330_submit AT calab.kaist.ac.kr'**
  - Title: [cs330] project 1
  - Contents
    - Source code (archive of 'pintos/src' directory)
    - Design Documentation
      - » http://cps.kaist.ac.kr/courses/2009_fall_cs330/project/project1.tmpl
- **Cheating will not be forgiven.**

Tips

- **Strongly RECOMMEND to use ‘ctags’**
- **Don’t make list/hash structure.**
  - Use list.h list.c, hash.h hash.c at lib/kernel
- **Check progress yourself by using ‘make check’**
  - alarm-single, alarm-multiple, alarm-simultaneous, alarm-priority, alarm-zero, alarm-negative,
  - priority-change, priority-donate-one, priority-donate-multiple, priority-donate-multiple2, priority-donate-nest, priority-donate-sema, priority-donate-lower, priority-fifo, priority-preempt, priority-sema, priority-condvar, priority-donate-chain
  - mlfqs-load-1, mlfqs-load-60, mlfqs-load-avg, mlfqs-recent-1, mlfqs-fair-2, mlfqs-fair20, mlfqs-nice-2, mlfqs-nice10, mlfqs-block
- **TAs**
  - Noah course board
  - 좌훈승, chwahs_at_gmail.com
  - 이민섭, ssangbuja_at_cps.kaist.ac.kr
  - 이주영, ljy8904_at_kaist.ac.kr