Lab 2 Introduction

Operating Systems, EDA093 - DIT400

Pintos

- Time to explore an operating system!
- Pintos already implements a simple threading system
  - Thread creation and termination
  - Synchronization primitives (semaphores, locks, condition variables)
- But this system has problems:
  - Wait is based on a spinlock (i.e. it just wastes CPU)

Pintos Threading System

```
struct thread
  {
    tid_t tid;                /* Thread identifier. */
    enum thread_status status;     /* Thread state. */
    char name[16];   /* Name (for debugging purposes).
    */
    uint8_t *stack;    /* Saved stack pointer. */
    int priority;          /* Priority. */
    struct list_elem allelem;  /* List element for
    all-threads list.*/
    /* Shared between thread.c and synch.c. */
    struct list_elem elem;       /* List element. */

You add more fields here as you need them.

#ifdef USERPROG
    /* Owned by userprog/process.c. */
    uint32_t *pagedir;          /* Page directory. */
#endif
    /* Owned by thread.c. */
    unsigned magic;  /* Detects stack overflow. */
  };
```

~/pintos/src/threads/threads.h

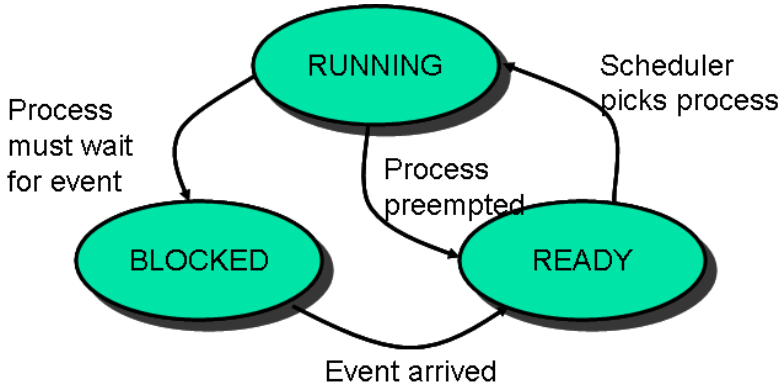
Pintos Threading System

```
struct thread
  {
    tid_t tid;                  /* Thread identifier. */
    enum thread_status status;     /* Thread state. */
    char name[16];   /* Name (for debugging purposes).
    */
    uint8_t *stack;    /* Saved stack pointer. */
    int priority;          /* Priority. */
    struct list_elem allelem;  /* List element for
    all-threads list.*/
    /* Shared between thread.c and synch.c. */
    struct list_elem elem;       /* List element. */

You add more fields here as you need them.

#ifdef USERPROG
    /* Owned by userprog/process.c. */
    uint32_t *pagedir;          /* Page directory. */
#endif
    /* Owned by thread.c. */
    unsigned magic;  /* Detects stack overflow. */
  };
```

~/pintos/src/threads/threads.h



Threads continued….

- Look at:
  - threads/thread.h
  - threads/thread.c
  - threads/synch.h
  - threads/synch.c

to understand

  - How threads are created and executed
  - How the provided scheduler works
  - How the various synchronizations primitives (locks, semaphores, condition variables) are implemented

Implementation Suggestions

- You may modify functions or add your own code in
  - threads.h
  - timer.h
  - timer.c
- There are several tests which test the sleep function in different ways.
- Run make check from the *~/pintos/src/threads* directory

Implementation Suggestions

- The tests `alarm-*` are for lab2
- Ignore the test result of “batch-scheduler” for now. That is for Lab3!
- A correct solution should “pass” all these tests.

```
pass tests/threads/alarm-single
pass tests/threads/alarm-multiple
pass tests/threads/alarm-simultaneous
pass tests/threads/alarm-zero
pass tests/threads/alarm-negative
pass tests/threads/batch-scheduler
All 6 tests passed.
```

Submission

- Test the code
- Write the report
- Prepare the archive