Programming Assignment 2 - Pintos

Operating Systems, EDA092 - DIT400

Slides by Ivan Walulya, Yiannis Nikolakopoulos

Lab Overview

- Time to explore an operating system!
- Main challenge: synchronization.
- This assignment is divided into 2 tasks:
  - Implement a sleep function without busy-waiting.
  - Synchronize access to a shared resource:
    Schedule jobs for an external hardware accelerator (e.g. GPU, co-processor) that send and receive data through a common bus.

Outline

- **Introduction to Pintos (Lab 0)**
- Pintos Thread System
- Assignment Overview
  - Task 1
  - Task 2

Outline

- Introduction to Pintos (Lab 0)
- **Pintos Thread System**
- Assignment Overview
  - Task 1
  - Task 2

Threads in Pintos

- Pintos already implements a simple threading system
  - Thread creation and termination
  - Simple scheduler based on timer preemption
  - Synchronization primitives (semaphores, locks, condition variables)
- But this system has problems:
  - Wait is based on a spinlock (i.e. it just wastes CPU)

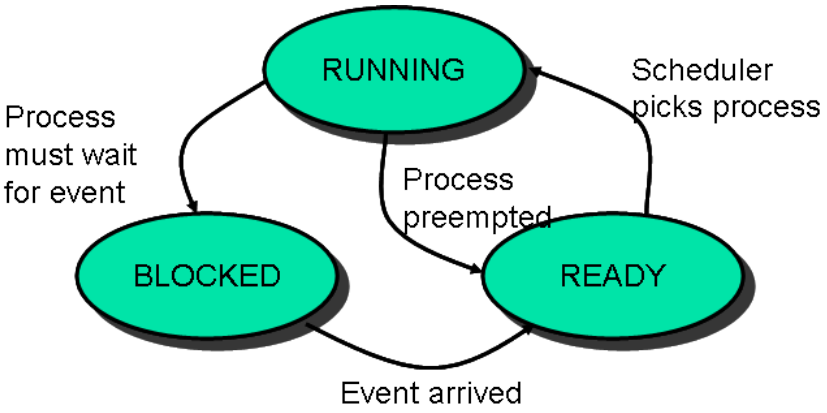
Pintos Threading System

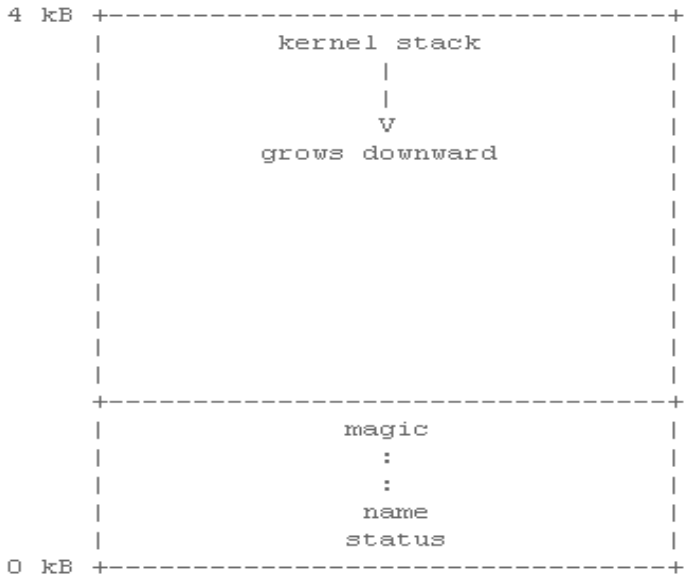
```
struct thread
  {
    tid_t tid;                  /* Thread identifier. */
    enum thread_status status;      /* Thread state. */
    char name[16];    /* Name (for debugging purposes).
    */
    uint8_t *stack;     /* Saved stack pointer. */
    int priority;           /* Priority. */
    struct 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





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 are implemented (condition variables, locks, semaphores and optimization barriers)

Outline

- Introduction to Pintos
  - Features
  - Documentation
  - Setup, building and running
  - Testing
- Pintos Thread System
- **Assignment Overview**
  - Task 1
  - Task 2

Assignment Overview

- The lab assignment will involve 2 objectives:
  1. Modifying the Pintos OS
  2. Producing a `DESIGNDOC` that explains your modifications
- The automated tests only determine that the execution terminates
- We will evaluate the correctness of your solution based on the `code and DESIGNDOC`
  - Template for DESIGNDOC is `project.tmpl`

Tasks

1. Fix the `timer_sleep()` function to use better synchronization
   - No busy waiting
2. Implement a Shared bus system
   - Up to 3 threads of same category can use bus concurrently
   - High priority threads ahead of low priority
   - No need to consider fairness!

Task 1: Alarm Clock

Fixing `timer_sleep()`

- Problem: Pintos' implementation of sleep is *wasteful*
  - Busy waiting
- `src/devices/timer.c`

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

Implementation Suggestions

- You may modify other functions or add your own code in `threads.h`, `timer.c` and `timer.h` files.

Test Your Implementation

- Run `make check` from the `~/pintos/src/threads` directory
- There is a number of tests `alarm-*` which will test `timer_sleep()` function in different ways
- You may run (and debug if necessary) one test at a time

```
$ pintos -q run alarm-simultaneous
```

Task 2: Batch Scheduler

- Classical IPC problem
- Synchronization on a half-duplex communication bus.
- Bus maximum capacity is 3 connections.
- High priority tasks have precedence over other tasks.
- Prototype functions already implemented in:

  `src/devices/batch-scheduler.c`

Implementation Suggestions

- Every task is represented by its own thread.
  1. Task requires and gets slot on bus system.
  2. Transfer data.
  3. Leave the bus.
- Implement *`batchScheduler`* to create the appropriate threads
  - `void batchScheduler(num_tasks_send, num_task_receive, num_priority_send, num_priority_receive)`

Implementation Suggestions

- Create semaphores to monitor:
  1. Bus capacity
  2. High priority tasks waiting to send
  3. High priority tasks waiting to receive
- Utilize mutex locks where necessary

Overall File Modifications

- What files will you be modifying?
  - `src/devices/timer.c`
  - `src/devices/batch-scheduler.c` ← Most edits will be here…
  - `src/threads/thread.h`
  - `DESIGNDOC` ← Text file that you will submit

Demo and Submission Deadlines

- Demo: week 41, according to schedule
- Submission:
  1. Compress your pintos directory, including DESIGNDOC

```
$ tar -zcvf pintos.tar.gz ~/pintos
```

  2. Submit compressed file to PingPong

- DUE: OCTOBER18, 2017

11:59:59PM CET