Lab 3 Introduction

Operating Systems, EDA093 - DIT400

Lab Overview

- Pintos
- Main challenge: 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.

Bridge Problem

A two way east-west road contains a narrow bridge with only one lane. An eastbound (or westbound) car can pass over the bridge only if there is no oncoming car on the bridge. Traffic may only cross the bridge in one direction at a time, and if there are ever more than 3 vehicles on the bridge at one time, it will collapse under their weight. In this system, each car is represented by one thread, which executes the procedure OneVehicle when it arrives at the bridge.

```
OneVehicle(Direction direc) {
    ArriveBridge(direc);
    CrossBridge(direc);
    ExitBridge(direc); }
```

direc gives the direction in which the vehicle will cross the bridge

Bridge Problem - Solution

a) Correctness Constraints

   I. At most 3 cars are on the bridge at a time

   II. All cars on the bridge go in the same direction

   III. Whenever the bridge is empty and a car is waiting, that car should get on the bridge

   IV. Whenever the bridge is not full and a car is waiting to go the same direction as the cars on the bridge, that car should get on the bridge

b) Cars will be waiting to get on the bridge, but in two directions. Use an array of two condition variables, waitingToGo[2].

c) It will be necessary to know the number of cars on the bridge (cars, initialized to 0), and the direction of these cars if there are any (call it current-direction). It will also be useful to know the number of cars waiting to go in each direction; use an array waiters[2].

Bridge Problem - Solution

```
ArriveBridge(int direction) {
    lock.acquire();

    // while can't get on the bridge, wait
    while ((cars == 3) || (cars > 0 && currentdirection != direction)) {
        waiters[direction]++;
        waitingToGo[direction].wait();
        waiters[direction]--;
    }

    // get on the bridge
    cars++;
    currentdirection = direction;
    lock.release();
}
```

Bridge Problem - Solution

```
ExitBridge() {
    lock.acquire();
    cars--;

    // if anybody wants to go the same direction, wake them
    if (waiters[currentdirection] > 0)
        waitingToGo[currentdirection].signal();

    // else if empty, try to wake somebody going the other way
    else if (cars == 0)
        waitingToGo[1-currentdirection].broadcast();

    lock.release();
}
```

Lab Task

- Classical IPC Problem
- Implement a Shared bus system
  - Up to 3 threads of the same direction can use bus concurrently
  - High priority threads ahead of low priority
  - No need to consider fairness!
  - Prototype functions already implemented in:
    - src/devices/batch-scheduler.c

Lab Task

```
typedef struct {
    int direction;
    int priority;
} task_t;
```

{SENDER, RECEIVER}

{NORMAL, HIGH}

Lab Task

```
void batchScheduler(unsigned int num_tasks_send, unsigned int num_task_receive,
        unsigned int num_priority_send, unsigned int num_priority_receive)
{
    unsigned int i;
    /* create sender threads */
    for(i = 0; i < num_tasks_send; i++)
        thread_create("sender_task", 1, senderTask, NULL);

    /* create receiver threads */
    for(i = 0; i < num_task_receive; i++)
        thread_create("receiver_task", 1, receiverTask, NULL);

    /* create high priority sender threads */
    for(i = 0; i < num_priority_send; i++)
       thread_create("prio_sender_task", 1, senderPriorityTask, NULL);

    /* create high priority receiver threads */
    for(i = 0; i < num_priority_receive; i++)
       thread_create("prio_receiver_task", 1, receiverPriorityTask, NULL);
}
```

Lab Task

DO NOT CHANGE/ADD ANY CODE HERE

```
void batchScheduler(unsigned int [illegible]send, unsigned int num_task_receive,
        unsigned int num_priority_sen[illegible] int num_priority_receive)
{
    unsigned int i;
    /* create sender threads */
    for(i = 0; i < num_tasks_send; i++)
        thread_create("sender_task", 1, senderTask, NULL);

    /* create receiver threads */
    for(i = 0; i < num_task_receive; i++)
        thread_create("receiver_task", 1, receiverTask, NULL);

    /* create high priority sender threads */
    for(i = 0; i < num_priority_send; i++)
       thread_create("prio_sender_task", 1, senderPriorityTask, NULL);

    /* create high priority receiver threads */
    for(i = 0; i < num_priority_receive; i++)
       thread_create("prio_receiver_task", 1, receiverPriorityTask, NULL);
}
```

Lab Task

You need to implement the following three functions:

```
/* abstract task execution*/
void oneTask(task_t task) {
  getSlot(task);
  transferData(task);
  leaveSlot(task);
}
```

If you print in any of these functions, you might get a timeout and fail the test.

Assignment Overview

- The lab assignment will involve 2 objectives:
  1. Modifying the Pintos code
  2. Writing a report that explains your solution
- Execute command “make check” in the *~/pintos/src/threads* directory to run the test.

```
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.
```

! The automated test only determines that the execution terminates

Submission

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