Project 2: Preemption

CS 415

Alin Dobra
Department of Computer Science
Cornell University
dobra@cs.cornell.edu
February 12, 2002

What you need to do

- Add Preemption
- Add Alarms
- Add `minithread_sleep_with_timeout`

Interrupts

- In project 2 we have interrupts:
  - Much more realistic machine model
- For now only clock interrupts:
  - periodic interrupt
  - call `minithread_clock_init` to start
- Interrupts behave like they do on native hardware

How it works

- At startup interrupts are disabled
- You need to initialize the clock device:

```
#define PERIOD 50*MILLISECOND
typedef void (*interrupt_handler_t)(void* );
void minithread_clock_init(interrupt_handler_t clock_handler);
```

- Clock frequency defined in `interrupts.h`
  - Initially set it to something like 5 seconds
- Once the clock device is initialized, interrupts are still disabled

Enabling interrupts

- Turn interrupts **on** or **off** with:

```
interrupt_level_t set_interrupt_level(interrupt_level_t newlevel);
```

- Example:

```
set_interrupt_level(ENABLED);
```

- You will now start getting interrupts every 5 seconds
- The interrupts arrive on the stack of whatever thread happens to be currently executing

Interrupt stack

- Whatever the running thread was doing is interrupted
- Old state is saved onto its stack
- Your handler is called
- If your handler returns, the old state is resumed

Preemption

- Everything you run inside your interrupt handler executes in the context of the interrupted thread
- What happens if the interrupt handler calls `minithread_yield`?

Interrupt handlers

- You **cannot** do anything you want inside an interrupt handler
- You cannot take too long
  - It takes CPU time away from real computations
  - If you take far too long, you will get a second clock interrupt
- You cannot use spin locks
  - The interrupted thread may be holding the lock you are spinning on
  - Consequently, you will spin forever and the machine will hang
- You cannot block (i.e. call P)
  - It will block the thread the interrupt arrived on
- You **can** signal (i.e. V) other threads though

Disabling interrupts

- At critical points in your **system** code, you may not want to take interrupts (e.g. while manipulating the run queue)
- You can disable interrupts for short periods of time
  - Make sure you reenable them properly
  - On all paths, back to their prior value
  - Make sure you do not execute application code with interrupts disabled
  - Be aware of the fact that `minithread_switch` reenables interrupts when called

Preemption testing

- Once you have implemented preemption:
  - Reduce your quantum to 100ms
  - No `printfs`, no unnecessary tasks inside the clock interrupt handler
- Your FCFS scheduler became RR
  - Optional: Implement multi-level queue scheduling
- Run the idle thread only when the system is truly idle

Alarms

- Often, you need to schedule something to happen in the future:
  - E.g. Wake me up in 30ms.
- You need to implement an alarm facility where functions to be executed in the future can be registered
  - You need to keep track of time
  - You need to call the alarm functions when they expire

Alarm interface

- Two functions:

```
int register_alarm(int delay, void (*func)(void*), void *arg);
void deregister_alarm(int alarmid);
```

- Keep track of how many ticks have elapsed
- Execute the given function when enough ticks have gone by
- Assume that the functions are interrupt safe, i.e. you can call them from within the interrupt handler

Thread sleep

- Implement a call by which threads can sleep for a specified amount of time

```
minthread_sleep_with_timeout(int timeout);
```

- You need to take the calling thread out of the run queue, have it wait someplace until timeout
- Semaphores can help here
  - Make sure that if more than one thread calls `minithread_sleep_with_timeout` they sleep the right amount of time

Testing

- Test preemption with your *food services* implementation and other tests from Project 1 and 2
- `Makefile`: change the `MAIN` variable to indicate which `main()` function you are linking against