Project 2: Preemption and Alarms

CS 414 / CS 415

Niranjan Nagarajan
Department of Computer Science
Cornell University
niranjan@cs.cornell.edu

What's the agenda?

- Add Preemption
- Add Alarms
- Implement `minithread_sleep_with_timeout`
- Replace FCFS with multilevel feedback scheduler

How to add Preemption

- Write a clock interrupt handler
  - Updates time
  - Schedules the next thread
- Install clock interrupt handler
- Start scheduling threads

How to add Preemption: what to use

- `Interrupts.h`

```
#define PERIOD 100*MILLISECOND
extern long ticks;
typedef void (*interrupt_handler_t) (void*);

typedef int interrupt_level_t;
#define ENABLED 1
#define DISABLED 0

interrupt_level_t set_interrupt_level(interrupt_level_t newlevel);

void minithread_clock_init(interrupt_handler_t clock_handler);
```

- Define `void clock_handler(void*)` in `minithread.c`

How to add Preemption: how it would work

- Initially interrupts are disabled
- Interrupt handler is installed during system initialization
- Clock interrupts are enabled by `minithread_switch`
- Interrupt Processing
  - Arrive on the stack of the running thread
  - State saved on the current stack and handler is called
  - On return the state is restored

Interrupt Handling Care

- Shouldn’t take too long (no printf’s)
- Extra precaution if using
  - Spin Locks (Why?)
  - Blocking (for example through a P)

Protecting Critical Sections

- May need to use `set_interrupt_level(DISABLED)` to disable interrupts before modifying system data
- Interrupts should be disabled for as short a period of time as possible
- Interrupts should be re-enabled before returning control to application code

Alarms

- Need to implement

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

- Keep track of time using `ticks`
- The function `func` needs to be called with `arg` as argument sometime after `delay` milliseconds have gone by
- `register_alarm` returns an alarm id that can be used with `deregister_alarm` to deregister the alarm

Thread Sleep

- On calling `minithread_sleep_with_timeout(delay)` thread should sleep for `delay` milliseconds and become available for scheduling sometime after `delay` milliseconds have passed
- Use the alarm functions to implement this
- Advice: Use semaphores instead of explicit stops and starts

Multilevel Feedback Scheduler

- Implement multilevel queues using queues

```
typedef void* multilevel_queue_t;
multilevel_queue_t multilevel_queue_new(int number_of_levels);
int multilevel_queue_enqueue(multilevel_queue_t q, int level, any_t item);
int multilevel_queue_dequeue(multilevel_queue_t q, int level, any_t *item);
int multilevel_queue_free(multilevel_queue_t q);
```

- Add priorities to minithreads and enqueue them based on priority.
- Ageing policy: Time-slice between the various levels for the search for the next thread to schedule