Real-Time Recap

Jonas Flodin
What do we mean with RT?

- Systems with strict timing requirements
  - Automotive
  - Avionics
  - Robotics
  - Telecommunication
  - etc.

- Late computations are as bad as if they were incorrect
How do we approach RT?

- Many RT systems host safety critical applications
  - Timeliness must be guaranteed
- All sources of delay must be identified
Delay – Hardware

- Hardware is very complex
- Typically modelled as worst case constants
- Ex: Memory fetch, pipeline, speculative execution
Delay – Operating System

- Needs to be adapted for RT performance
- Predictible timing of system functions
  - System calls
  - Context switches
  - Scheduling
  - Interrupt/exception handling

![Operating System Stack Diagram]
Delay - Applications

- Programming constructs might be hard to bound
- Timeliness of one application depends on execution of other applications
  - Preemption by higher priority
  - Shared critical sections
  - Communication
Periodic Task Model

- Each application is described by tasks
- Tasks are defined by
  - P – Period
  - C – Worst case execution time (WCET)
  - D – Deadline
- User requirements, HW and OS effects are included in these parameters
Periodic Task Implementation

- Typically implemented as infinite loops with sleep statements

```c
int my_task(int argc, char ** argv){
    // Local declarations
    time_t next_time;

    // Work loop
    for(;;){
        next_time = current_time() + my_period;
        if(!do_work(argc, argv)){
            printf("ERROR: Task failed!\n");
            break;
        }
        sleep_until(next_time);
    }
    return 0;
}
```
Scheduling

- Protocol for deciding which task to run
- Many scheduling policies exist
  - RM – Rate Monotonic
  - EDF – Earliest Deadline First
- Provably optimal
- Schedulability test ensures feasibility of the system
FP Scheduling

- Priority assigned according to period (RM scheduling)
- Lower period = higher priority
- Scheduler keeps a queue of ready tasks
- Head of the queue is the highest priority and is selected to run

1 2 5 7

Running

Waiting
FP Scheduling

- Task activation means inserting a task into the queue
FP Scheduling

- When a task finishes it is removed from the queue and the next task is selected to run.
FP Scheduling

- If a higher priority task arrives, the currently running task is put back in the queue and the new task starts executing.

- This is called preemptive scheduling.
Implementation Problem

- Concurrency might cause incorrect behaviour

```c
int my_task(int argc, char ** argv){
    // Local declarations
    time_t next_time;

    // Work loop
    for(;;){
        next_time = current_time() + my_period;
        if(!do_work(argc, argv)){
            printf("ERROR: Task failed!\n");
            break;
        }
        sleep_until(next_time);
    }
    return 0;
}
```

Problem!
Protection Against Preemption Problems

- Disabling interrupts
  - Suitable for small sections
  - Mostly for use in OS
- Use a mutex (or semaphore)
  - For locking data structures
  - Might cause priority inversion or deadlock
Priority Inversion

- A situation where a lower priority task can indefinitely block a higher priority task
Priority Inversion Solutions

- Priority inheritance
  - When trying to take a lock, raise the priority of the task holding the lock

- Priority Ceiling
  - Priority is raised when lock is held
Deadlock

- Tasks are waiting for each other in a circular fashion

```cpp
int task1() {
    // Acquire locks
    mutex_lock(mutexA);
    mutex_lock(mutexB);

    // Work with global data
    update_datastructure();

    // Release locks
    mutex_unlock(mutexB);
    mutex_unlock(mutexA);

    return 0;
}

int task2() {
    // Acquire locks
    mutex_lock(mutexB);
    mutex_lock(mutexA);

    // Work with global data
    update_datastructure();

    // Release locks
    mutex_unlock(mutexA);
    mutex_unlock(mutexB);

    return 0;
}
```