Lecture11: Scheduling III, scheduling worksheet examples

Vikram Iyer

Adapted from material by Blake Hannaford

Reminders and announcements

- Lab 2 due Monday 5/6
  - Tip: Start by just trying to get the timer to output to a pin, and then adjust the timing. Might be easier to debug once you have an output
  - Tip: for scheduling, start with simple tasks
- Midterm on 5/10
  - During class here 90 min + separate room for DRS
  - Goal: make sure you know concepts, labs are worth more
  - Midterm resources posted

Plan for today

- More scheduling examples and worksheets

Scheduler types

Periodic scheduler- Infinite Loop

- Most primitive of all
- Also known as Non-preemptive Round Robin.

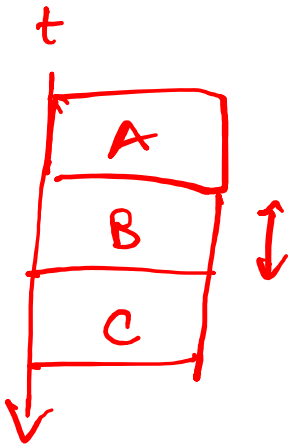
```
while(1) {
   task1_fcn();
   task2_fcn();
   task3_fcn();
   ...
}
```

Scheduler types

- Each task **must** voluntarily return to scheduler quickly
- C << P, P = T = 1 ms = 1000 µs (C: 100 ms)
- Sample task:

```
task1_fcn(){
  compute a little bit
  return;
}
```

100 ms

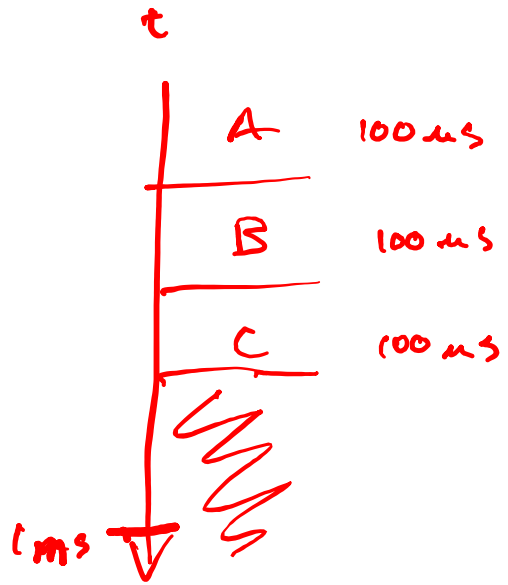


Scheduler types

Periodic scheduler- Synchronized Infinite Loop

- the top of the loop waits on a hardware clock.

```
while(1) {
  wait(CLOCK_PULSE);
  task1_fcn();
  task2_fcn();
  task3_fcn();
  ...
}
```



- Each task **must** voluntarily return to scheduler quickly, as above.
- $C << P$, $T = P$.

Scheduler function (pseudocode)

```
scheduler() {
  if(readytasks[task_index] == NULL && task_index != 0) {
    task_index=0;
  }
  if(readytasks[task_index] == NULL && task_index == 0){
    // figure out something to do because
    // there are no tasks to run!
  } else {
    start_function(readytasks[task_index]);
    task_index++;
  }
  //   Round Robin/we're taking turns
  return;
}
```

iterating through tasks

Alternative Data Structure for Scheduling

Let's set up a struct which contains everything we need to track about a Task:

```
typedef struct TCBstruct {
  void (*ftpr)(void *p);     // the function pointer
  void *arg_ptr;             // the argument pointer
  unsigned short int state; // the task state
  unsigned int delay;        // the task state
} ;
```

Scheduler types

Pre-emption

Each function in these schedulers must “voluntarily” return back to the main loop, and must make sure that it doesn’t compute for too much time. If a task gets stuck, it breaks the whole system.

The fix for this is **preemption** - the ability to break into a running task and stop it. In other words to force a context switch.

Scheduler Exercises

1. Our basic `while(1)` loop: Non-preemptive Round Robin (RR).
2. The same loop with synchronization timing loop or timer function: Synchronized Non-premptive RR wait (clock pulse)
3. Adding a `sleep(int d)` delay option for tasks: Non-premptive synchronized RR with delays.
4. Making the scheduler dynamic: Non-premptive synchronized RR with `halt me()` function.

Diagrams

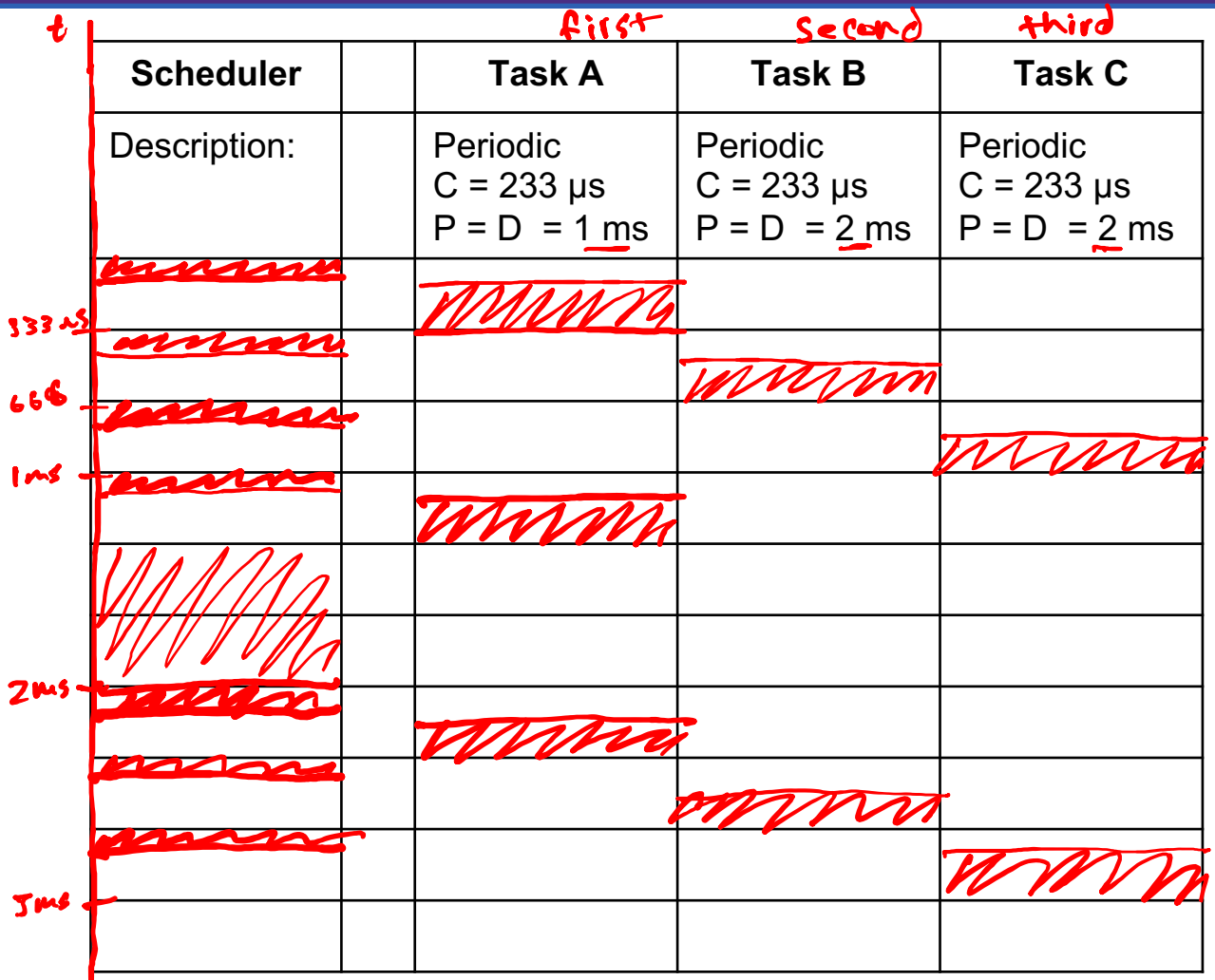
Each Line = 333 µsec.

Scheduler time ticks every 1,000 µsec. 1ms

Mark ticks at left edge of every third line. Each time scheduler runs it requires 100µsec (approximately 1/3 of a line).

```
while(1) {
    scheduler();
    time_delay(); // 1 ms timer
}
```

100 µs



Each Line = 333 µsec.

Scheduler time ticks every 1,000 µsec.

Mark ticks at left edge of every third line. Each time scheduler runs it requires 100µsec (approximately 1/3 line).

```
while(1) {
     scheduler();
time_delay();
// 1 ms timer
}
```

time ticks

Delay functions and scheduling

Sometimes tasks will need to wait. Examples:

- To create time-based events (such as Flash an LED for 1 sec)
- To wait for something external to happen such as for a device
- While waiting we want to free up CPU time for other processes.
- To determine for themselves how often they run:

```
void taskA(){
  compute(); toggle LED
  sleep(20ms);
  return;// Run every ~20ms
}
```

This task sets its own period, P , to 20ms. How accurate?

Delay functions and scheduling

Assume a time delay function:

`sleep(int d)`

```
sleep(int d) {
  TaskList[t_curr].delay = d;
  TaskList[t_curr].state =
    STATE_WAITING;
}
```

Task List and Characteristics:

- The three tasks: A, B, C

Task A: **P** = 2.0 ms, **C** = 233 µs, **D** = 1.0ms

Task B: **P** = 4.0 ms, **C** = 233 µs, **D** = 1.0ms

task pseudocode:

```
taskB(void) {
   compute for 233 µs;
   sleep(3);
   return;
}
```

Task C: **P** = 1.0 ms, **C** = 100 µs, **D** = 1.0ms

Delay functions and scheduling

A bad delay function:

```
#define DELAY      22000
for (i=0; i< DELAY ; i++) ; // just loop to use up time
```

The scheduler can help implement delays `sleep(int delay)`:

- returns to the scheduler
- sets a software counter to `delay`.
- schedules the next task.
- the task which called `OS TimeDelay()` will be set to Waiting (not started by the scheduler)
- Each tick, decrement `delay` value for each sleeping task. when the `delay` is over, set task to `Ready`

Scheduler		Task A	Task B	Task C
Description:		C = 233 µs P = 2 ms, D = 1 ms	C = 233 µs P = 4 ms, D = 2 ms	C = 100 µs P = D = 1 ms

```
taskB(void) {
  compute for 233 us;
  sleep(3);
  return;
}
```