

# Lecture 9: Scheduling II, Scheduling worksheets

Vikram Iyer

# Announcement/Reminders

- Short office hour today
- Lab 2 is posted, due 5/5
  - 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
- Midterm on 5/12
  - During class here 90 min
  - Goal: make sure you know concepts, labs are worth more

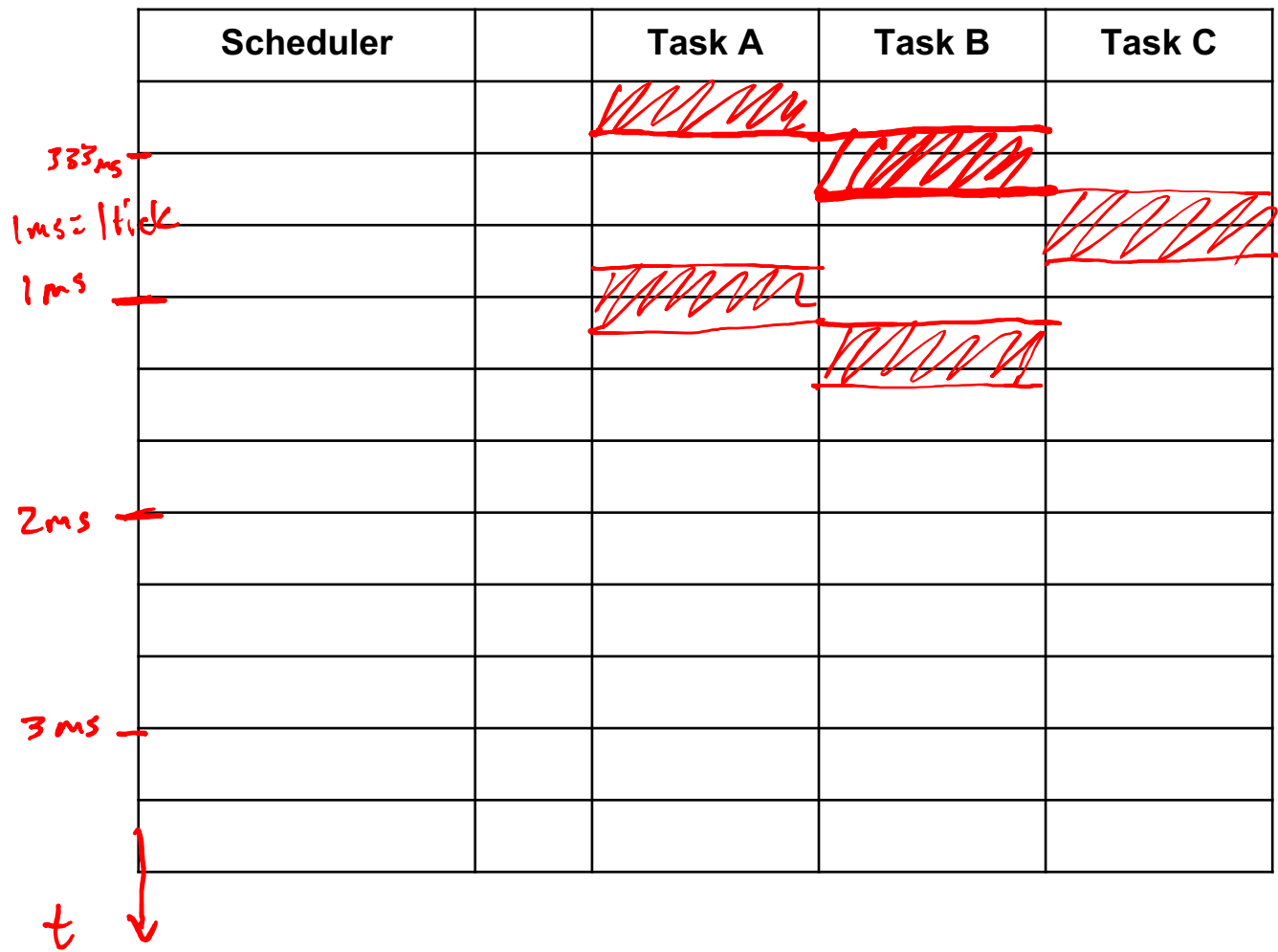


# Last time and plan for today

- Started with an intro to scheduling and concepts
- Plan for today is to continue with more advanced scheduling concepts
  - How to build a scheduler
  - Halting tasks, keeping track of functions
  - Worksheet examples

# Scheduler types

1. Non-preemptive Round Robin (RR)- Basic `while(1)` loop
2. Synchronized non-preemptive RR- The same loop with a timing function for synchronization
3. Modifications to Synchronized RR to enable different task periods
4. Adding a `sleep(int d)` delay option for tasks  
*nonblocking delay*
5. Making the scheduler dynamic with a `halt_me()` function



```
while(1) {
    taskA();
    taskB();
    taskC();
}
```

Task	P	C	D
A	1	0.2	P
B	2	0.2	P
C	2	0.2	P

↑  
1 tick = 1ms

# How to Build a Scheduler

So far we've had very simple programs

```
while(1) {  
    taskA(); ← IO operation  
    taskB();  
    taskC();  
    time_delay(); ← synchronize  
} ↑ to 1ms
```

SW counter  
or delay  
problem is the value might  
vary

Another approach: HW timer/counter

# How to Build a Scheduler

New features we want to add

1. Ability to keep track of a function's state and start it
2. Ability for a task to halt itself
3. Ability of a task to sleep and let other tasks run

# Part I: Using function pointers to start a task

*task A()*

We learned that the syntax:

```
type (*name) (type arg1, type arg2, ...)
```

indicates a function pointer called name which can point to any function having the same prototype. For example

```
int (*fp) (char x, int y, double z)
```

*void function pointer*

```
void start_function(void (*functionPTR) () ) {  
    functionPTR();  
}
```

# Part I: Using function pointers to start a task

Now our basic scheduler could be written:

```
while(1) {  
    start_function( taskA);  
    start_function( taskB);  
    start_function( taskC);  
    time_delay();  
}
```

✓ Function pointers

**This version is a trivial change but this is an important step**

## Part II: Keeping lists of functions

We can create an array of function pointers as follows:

```
#define NUMBER_OF_FUNCTIONS 10
```

```
void (*list_of_functions[NUMBER_OF_FUNCTIONS])(void *p)
```

**Example:**

```
#define NT_RUNNING 10
```

```
void (*runningTasks[NT_RUNNING])(void *p)
```

... a list of all the tasks which are currently running.

```
loop() {
  scheduler()
}
```

↑  
our function  
that controls what  
task to run

```
void scheduler() {
  for (int i=0; i<NT_RUNNING; i++)
    start-function(runningTasks
                 runningTasks[i]);
}
```

## Part II: Keeping lists of functions

We can create an array of function pointers as follows:

```
#define NUMBER_OF_FUNCTIONS 10
```

```
void (*list_of_functions[NUMBER_OF_FUNCTIONS])(void *p)
```

**Example:**

```
#define NT_RUNNING 10
```

```
void (*runningTasks[NT_RUNNING])(void *p)
```

*[taskA, taskB, NULL...  
..... ]*

... a list of all the tasks which are currently running.

We can use a NULL pointer at the end of the list if there are less than NT RUNNING active tasks (similar to the zero byte at the end of a string).

```
//function prototypes for tasks
void taskA(void *p);
void taskB(void *p);
...

// function prototype for
scheduler void scheduler();

#define NTASKS 4

// lets make the task list global
// an array of function pointers,
// each one has a void*
// parameter.

void (*readytasks[NTASKS])
(void *p);
```

```
//continued...
main() {
    // initialize array of pointers
    // to tasks

    readytasks[0] = taskA;
    readytasks[1] = taskB;
    ...

    // NULL signals the last task
    readytasks[2]= NULL;

    // now start scheduler
    while(1) {
        scheduler(); for loop
        time_delay(); // 1 ms timer
    }
} // end of main
```

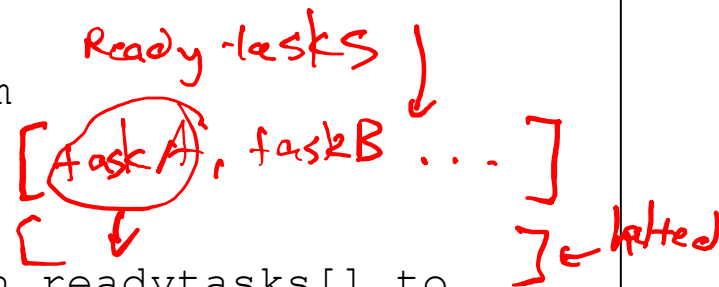
# Scheduler function (pseudocode)

```
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;
}
```

*wait until next clock tick*  
*buffer*

## Part III: Manipulating the Task Lists

```
halt_me() {  
    // 1. Identify which task is currently running (i.e.  
    //    look at task_index)  
  
    // 2. Copy the function pointer from  
    //    readytasks[task_index] to the  
    //    haltedtasks array  
  
    // 3. Move the remaining tasks up in readytasks[] to  
    //    fill the empty hole and copy NULL into the  
    //    last element.  
  
    return;  
}
```



## Part III: Manipulating the Task Lists

Watch out for potential logic bug

*task A*

```
if (time_to_halt) {  
    halt_me();  
  
}
```

## Part III: Manipulating the Task Lists

Watch out for potential logic bug

```
if (time_to_halt) {  
    halt_me(); ← halt me returns here  
    return; // Need to make sure the function returns  
}
```

Need to make sure the code actually returns after calling `halt_me()`

# Typical Task using sleep()

```
task_c(3) {
    do_computation(); // compute for a little while
    if(time_to_sleep){
        sleep(10); ← non blocking delay
        return;
    }
    else {
        do_computation(); // compute for a little while
    }
    return;
}
```

## Sleep function (pseudocode)

```
sleep(int d) {  
    // 1. Copy function pointer from  
    //    readytasks[task_index] to the  
    //    waitingtasks[] array.  
    // 2. Clean up readytasks[] as in halt_me();  
    //    copy d into the delays array with the same  
    //    index as the function pointer has in  
    //    waitingtasks[]  
}
```

## Keeping track of sleep delays

```
// continued...
// 5. for each element of waitingtasks which is
//     not NULL: decrement the delay value d.
//     if (d==0) move the function pointer to the
//     end of the readytasks[] array and remove
//     it from waitingtasks.
// end of scheduler
return;
}
```

ready task [ ]  
waiting [ ]  
halted [ ]

# 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 ready  
    unsigned int delay;      // the task state hold  
                                Sleep  
                                Sleep  
                                Count  
};
```

```
#define STATE_RUNNING    0
#define STATE_READY     1
#define STATE_WAITING   2
#define STATE_INACTIVE  3
TCBStruct TaskList[N_MAX_TASKS];

// Then we can set up the task // list as follows:
int j=0;
int task_B_Arg;

TaskList[j].ftpr = task_A();
TaskList[j].arg_ptr = NULL;
TaskList[j].state = STATE_INACTIVE;
TaskList[j].delay = 0;
j++;
```

```
// continued... start task B
TaskList[j].fptr = task_B();

// (example: let's say we need an arg value of 56)
task_B_Arg = 56;
int *ip = &task_B_Arg;
TaskList[j].arg_ptr = (void*)ip;
TaskList[j].state = STATE_READY;
TaskList[j].delay = 0;
j++;

TaskList[j].fptr = NULL; // marks end of list

... maybe other tasks ...
```

## Examples of `halt_me()`, `start_task()` and `delay()`

```
halt_me() {  
  
}  
  
start_task(int task_id) {  
  
}  
  
delay(int d) {  
  
}
```

## Examples of `halt_me()`, `start_task()` and `delay()`

```
halt_me() {
    TaskList[t_curr].state = STATE_INACTIVE;
}

start_task(int task_id) {

}

delay(int d) {

}
```

## Examples of `halt_me()`, `start_task()` and `delay()`

```
halt_me() {  
    TaskList[t_curr].state = STATE_INACTIVE;  
}  
  
start_task(int task_id) {  
    TaskList[task_id].state = STATE_READY;  
}  
  
sleep  
delay(int d) {  
  
}
```

## Examples of `halt_me()`, `start_task()` and `delay()`

```
halt_me() {
    TaskList[t_curr].state = STATE_INACTIVE;
}

start_task(int task_id) {
    TaskList[task_id].state = STATE_READY;
}

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

