**Scheduler implementation:**

**15-323/623 Spring 2018**

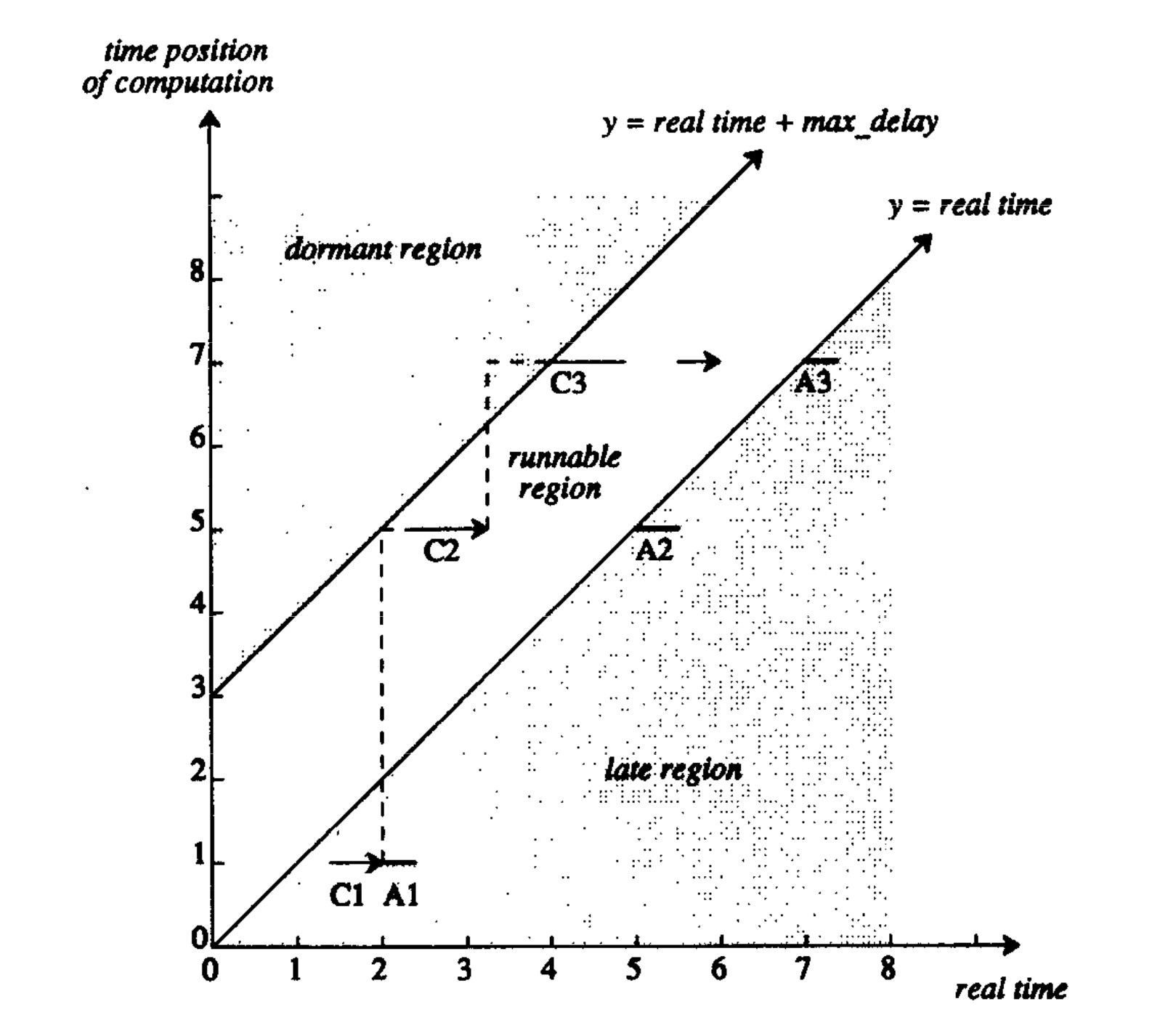
**Homework 2**

**Due Feb 15**

1. Compare an ordered linked list, a heap, and a timing wheel implementation of a scheduler.

Fill in a table like the following with characteristics of these approaches. Assume N means the number of currently scheduled events. – 15 points, 1 for each square

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Algorithm | Expected Insertion Time per event | Expected dispatch time per event | Worst case insertion time | Worst case dispatch time | Clarify what would cause the worst case situation and assumptions |
| Ordered link list | O(N) | O(1) | O(N) | O(N) or O(1) | Insert: have to insert at very end  Dispatch: remove is N in serpent or have to take first event and dispatch |
| Heap priority queue | O(logN) | O(logN) | O(logN) | O(logN) | Element to replace removed root needs to be bubbled up |
| Timing wheel | O(1) | O(1) | O(N) or O(1) | O(N) | Table length is 1 🡪 linked list |



In this **Timing in FORMULA** graph (Figure 6 of Anderson and Kuivila from the readings; also week 3, slide 26), we see three scheduled and executed events. Show your understanding of this important graph by answering the following questions: - #2-6 is 6 each, #7 is 10

2. Assume that C1 takes 0.5 time units. What time would C1 have to begin in order to compute results on time?

Between times 0 to 0.5

3. Assuming C1 started early enough and A1 also takes 0.5 time units, what time would A1 produce output?

Time 1 to time 1.5

4. There is separation in time between C2 and A2. Why is there no time separation between C1 and A1?

There is no separation between C1 and A1 because action 1 is late so it is computed and

performed immediately.

5. C2 becomes runnable at time 2, but it starts later at about 2.5. Why?

The process is still performing A1, which takes priority over C2.

6. C3 follows C2, but we do *not* immediately start computation C3 after C2. Why not?

The time that C2 finishes exceeds the maximum delay allowed, so the process is dormant until it is within the runnable region.

7. Suppose that the CPU is fast enough so that all computations are meeting deadlines with time to spare, but the system is not very responsive because of pre-computation and event buffering. How would you change the parameters and how would the graph change to make the system more responsive?

Decrease max delay 🡪 runnable region becomes narrower. If max delay is too large, events are precomputed very early. If you want to change later events (ex. transpose), the events that are already computed cannot be changed 🡪 system is unresponsive

Graph change: narrow the runnable region.

**More questions…**

8. Suppose you had Serpent (or another language) with preemptive threads. To get more computation done, you propose to:

o use a thread for each task (e.g. each drum in a drum machine could be on a different thread),

o use a single scheduler: threads will sleep by waiting on an synchronization event, but first the thread schedules a function call that signals the event. Thus, the thread

blocks on the synchronization event until the correct logical time, after which the thread will run again,

o you use locks around the scheduler to avoid concurrent scheduling by multiple

threads (which would undoubtedly have race conditions), but since scheduling is fast,

you can assume that the scheduler is not a bottleneck.

10 points for concurrency, 10 for timing

What other problems would you have to deal with? There are at least 2 problems. Try to answer with one problem having to do with concurrency and one having to do with timing.

FROM PIAZZA: Concurrency problem: The problem states that "you use locks around the scheduler to avoid concurrent scheduling by multiple threads (which would undoubtedly have race conditions)" so we're assuming the scheduler is not going to get into concurrency problems. Where else could there be concurrent access to shared variables causing concurrency problems?

ANSWER: threads could access shared variables almost anywhere: in shared library functions, in output functions, in graphic update functions, in updating application variables such as a counter of MIDI messages.

Timing problem: Show how two events could be executed in reverse order of their logical times.

If a thread has a long computation for time t and falls behind, another thread could get scheduled for time t+1 and compute it before the time t computation. OR, if there are several events for time t, t+ε, t+2ε, then they could all wake up at the same time and it would be a race to see who finishes first, whereas a single-thread system would process these late but in time order.

9. The “Global Drum Circle” aims to allow drummers across the globe to play together. But internet delays are high (over 100 ms for intercontinental connections) and there is considerable variability (jitter) in transmission times. Describe briefly what you would do to transmit drumming eventss to a remote location over the Internet. Assume that network messages are delivered reliably but with a latency that varies from 25 to 500 ms. Assume further that delays of up to 1000 ms are acceptable (i.e. the receiver can hear each drumming event exactly 1000 ms after the performer plays), but you need to reproduce drum timing to within 5ms to achieve musically acceptable results. 10 points

For each message

Timestamp each message to 500ms later

Send the message.

Play the message

10. Here is a naïve Serpent program for playing a sequence of notes (don’t worry about the details of play\_note()):- 15 points

for i = 0 to 10: play\_note() time\_sleep(0.5)

Assuming that your program will have a scheduler rtschedinitialized and running, finish the program below by defining play\_sequenceto use the scheduler to achieve the same general effect (i.e. rewrite the naïve version to make it correct). Since this program is not concerned with the details of play\_note(), you should not worry about timestamps or forward-synchronous scheduling:

sched\_select(rtsched) *// prepare to use rtsched*

play\_sequence(0)

def play\_sequence(i):

if i< 10:

play\_note()

sched\_cause(0.5, nil, ‘play\_sequence’, i+1)