IPC & Scheduling

Week 06 – Tutorial

Team

- Instructor
  - Giancarlo Succi
- Teaching Assistants
  - Nikita Lozhnikov (also Tutorial Instructor)
  - Manuel Rodriguez
  - Shokhista Ergasheva

Problem 1

- A shared variable *x*, initialized to zero, is operated on by four concurrent processes *W*, *X*, *Y*, *Z* as follows:
  - Processes *W* and *X* read *x* from memory, increment by one, store it to memory, and then terminate
  - Processes Y and Z read *x* from memory, decrement by two, store it to memory, and then terminate
  - Each process before reading *x* invokes the *P* operation (i.e., wait) on a counting semaphore *S* and invokes the *V* operation (i.e., signal) on the semaphore *S* after storing *x* to memory
  - Semaphore S is initialized to two
- What is the maximum possible value of *x* after all processes complete execution?

Problem 1 – Solution (1/2)

- Processes can run in many ways, below is one of the cases in which *x* attains max value:
  - Semaphore *S* is initialized to 2
  - Process *W* executes S=1, x=1 but it **doesn't update the x variable** because of a context switch
  - Process *Y* executes S=0, decrements *x* and invokes *V* operation; now x=-2 and semaphore S=1

Problem 1 – Solution (2/2)

- Process Z executes S=0, decrements x and invokes V operation; now x=-4 and semaphore S=1
- Now process *W* updates *x*; x=1, S=2
- Then process *X* executes increments x; x=2, S=2

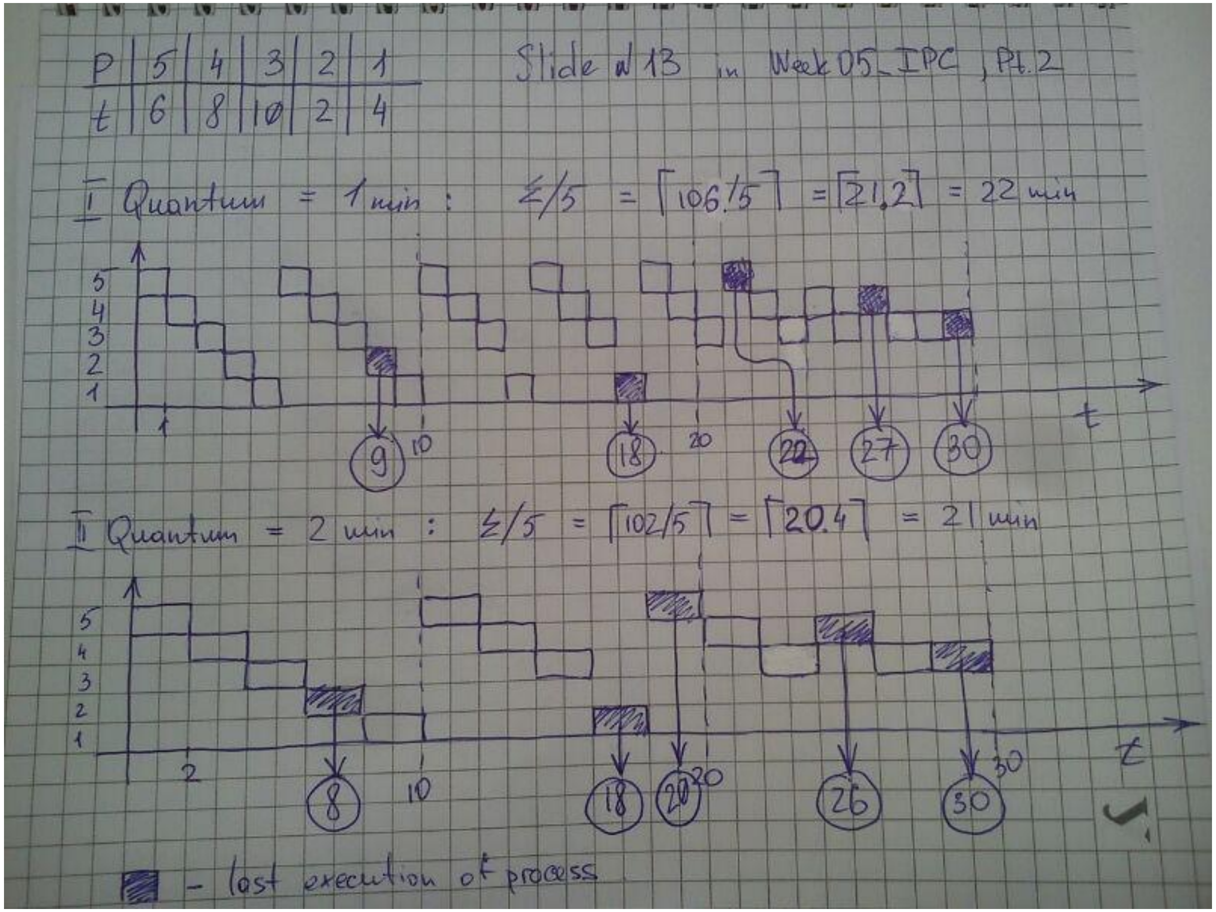
Problem 2.45 (1/2)

- Five batch jobs A through E, arrive at a computer center at almost the same time
- They have estimated running times of 10, 6, 2, 4, and 8 minutes
- Their (externally determined) priorities are 3, 5, 2, 1, and 4, respectively, with 5 being the highest priority
- For each of the following scheduling algorithms, determine the mean process turnaround time. Ignore process switching overhead

Problem 2.45 (2/2)

- Question (cont.):
  - (a) Round robin
  - (b) Priority scheduling
  - (c) First-come, first-served (run in order 10, 6, 2, 4, 8)
  - (d) Shortest job first
- For (a), assume that the system is **multiprogrammed**, and that **each job gets its fair share** of the CPU
- For (b) through (d), assume that only one job at a time runs, until it finishes. All jobs are completely CPU bound

Problem 2.45 – Solution (1/6)

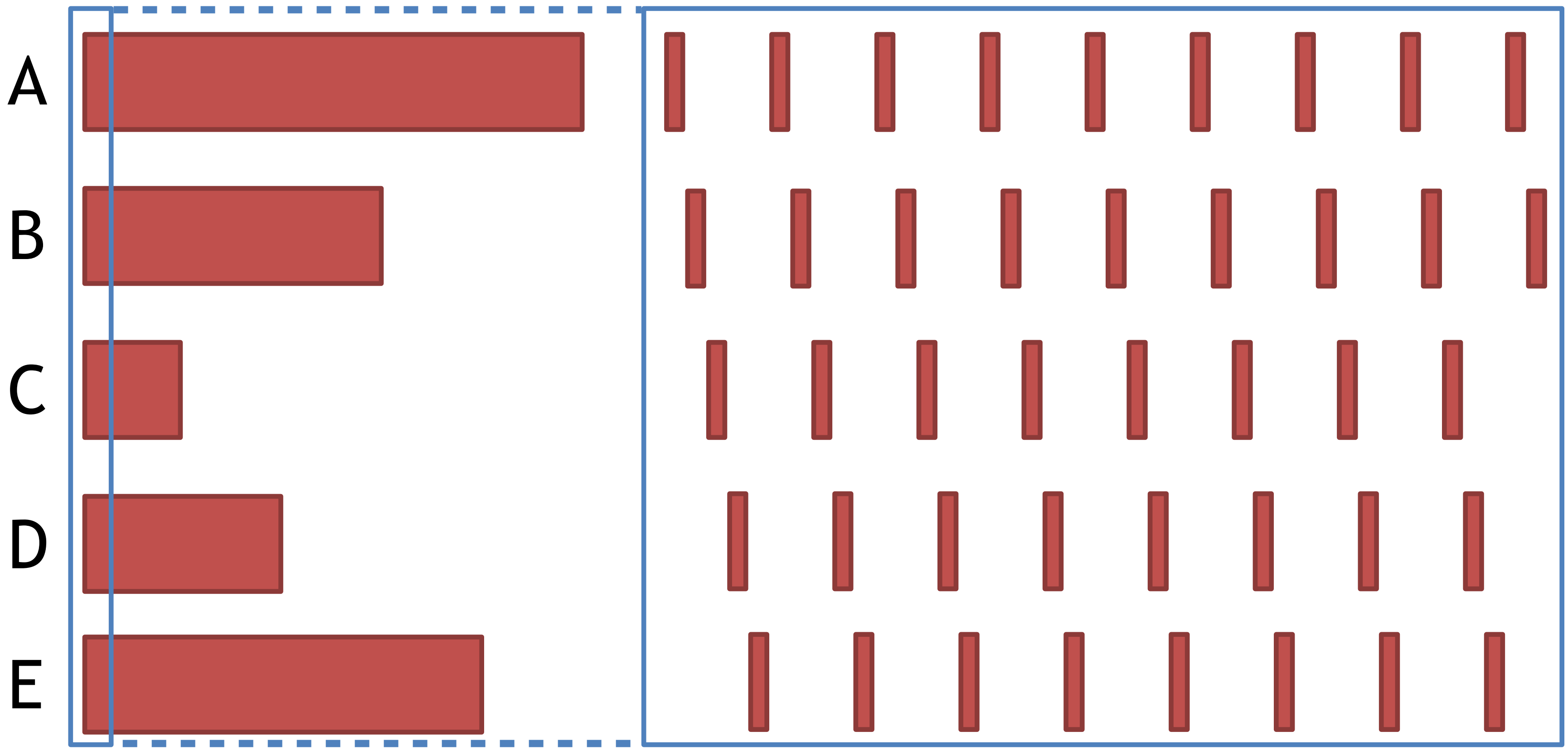


Wrong answer: (8 + 18 + 20 + 26 + 30) / 5

Problem 2.45 – Solution (2/6)

- The right answer is highly dependent on order of the jobs and the time quanta given to each of the jobs
- It will be safe to suppose that in a multiprogrammed system each job runs for milliseconds, not minutes (as shown on the next slide)

Problem 2.45 – Solution (3/6)



Round Robin job scheduling

Problem 2.45 – Solution (4/6)

- **Right answer (cont.):**
  In this case all the five jobs will run simultaneously for ~10 minutes until job C finishes
- Then remaining four jobs will run for 8 minutes until job D finishes which makes turnaround time for D equal 18 minutes
- Using the same technique we obtain turnaround times for B, E and A which are 24, 28 and 30 minutes respectively

Problem 2.45 – Solution (5/6)

- **Right answer (cont.):**
  - $T_C \approx 5 * 2$ min ≈ 10 min
  - $T_D \approx T_C + 4 * 2$ min ≈ 18 min
  - $T_B \approx T_C + T_D + 3 * 2$ min ≈ 24 min
  - $T_E \approx T_C + T_D + T_B + 2 * 2$ min ≈ 28 min
  - $T_A \approx T_C + T_D + T_B + T_E + 2$ min ≈ 30 min
- Average turnaround time is ~22 minutes ((10 + 18 + 24 + 28 + 30) / 5)

Problem 2.45 – Solution (6/6)

- Priority scheduling:
  - The processes will run in the following order: B(6), E(8), A(10), C(2), D(4)
  - $T_B \approx 6$ min
  - $T_E \approx T_B + 8$ min ≈ 14 min
  - $T_A \approx T_B + T_E + 10$ min ≈ 24 min
  - $T_C \approx T_B + T_E + T_A + 2$ min ≈ 26 min
  - $T_D \approx T_B + T_E + T_A + T_C + 4$ min ≈ 30 min
- Average turnaround time is ~20 minutes
  ((6 + 14 + 24 + 26 + 30) / 5)

Problems 2.47 - 2.48

- Consider a real-time system with two voice calls of periodicity 5 msec each with CPU time per call of 1 msec, and one video stream of periodicity 33 ms with CPU time per call of 11 msec. Is this system schedulable?
- For the above problem, can another video stream be added and have the system still be schedulable?

Problems 2.47 - 2.48 – Solution (1/2)

- Each voice call consumes 1 ms of CPU time each 5 msec which means that total CPU usage is 200 msec per second.
- Two voice calls consume 400 msec per second
- Video stream consumes about 367 msec (11 msec every 33.3 msec) of CPU time per second

Problems 2.47 - 2.48 – Solution (2/2)

- The sum is roughly 767 ms which makes the system schedulable
- Another video stream will require additional 367 msec
- Total amount of CPU time per second required to perform all the operations will exceed one second which makes the system to be not schedulable

Problem 2.49

- The ageing algorithm with $a = 1/2$ is being used to predict run times. The previous four runs, from oldest to most recent, are 40, 20, 40, and 15 msec. What is the prediction of the next time?

Problem 2.49 – Solution

- Let's denote measured runtime as *T* and predicted runtime as *t*
- $t_1 = T_0$ [40 msec]
- $t_2 = T_1/2 + t_1/2 = T_1/2 + T_0/2$ [30 msec]
- $t_3 = T_2/2 + t_2/2 = T_2/2 + T_1/4 + T_0/4$ [35 msec]
- $t_4 = T_3/2 + t_3/2 = T_3/2 + T_2/4 + T_1/8 + T_0/8$ [25 msec]
- The answer is 25 sec

Problem 2.50

- A soft real-time system has four periodic events with periods of 50, 100, 200, and 250 msec each. Suppose that the four events require 35, 20, 10, and *x* msec of CPU time, respectively. What is the largest value of *x* for which the system is schedulable?

Problem 2.50 – Solution

- The periodicity of each event is:
  - Cp1 = 50 msec; Cp2 = 100 msec; Cp3 = 200 msec; Cp4 = 250 msec
- The CPU time required by each event is:
  - Ct1 = 35 msec; Ct2 = 20 msec; Ct3 = 10 msec; Ct4 = *x* msec
- For the system to be schedulable the next inequality must hold:
  - $Ct1/Cp1 + Ct2/Cp2 + Ct3/Cp3 + Ct4/Cp4 \leq 1$, or
  - $35/50 + 20/100 + 10/200 + x/250 \leq 1$, or
  - $0.7 + 0.2 + 0.05 + x/250 \leq 1$, or
  - $x/250 \leq 0.05$, or
  - **$x \leq 12.5$ msec**

End

Week 06 – Tutorial