WE19.1

Bookmark this page

Video explanation of solution is provided below the problem.

Precedence Constraints with Semaphores

28 points possible (ungraded)
P1 and P2 are processes that run concurrently. P1 has two sections of code where section A is followed by section B. Similarly, P2 has two sections: C followed by D. Within each process execution proceeds sequentially, so we are guaranteed that A < B, i.e., A precedes B. Similarly we know that C < D. There is no looping; each process runs exactly once. You will be asked to add semaphores to the programs – you may need to use more than one semaphore. **Please give the initial values of any semaphores you use.** Your solution must use a minimum number of semaphores and it must not introduce any unnecessary precedence constraints.

1. Using a semaphore named S, please specify the initial value of semaphore S, and add wait(S) and signal(S) statements as needed in the spaces below so that the precedence constraint B < C is satisfied, i.e., execution of P1 finishes before execution of P2 begins. **For each drop down, select the missing line of code. If no command is missing, select None.**

   // Semaphore initial value

   semaphore S = Answer: 0 ;

// Process P1		// Process P2	
Select an option	Answer: None	Select an option	Answer: wait(S)
... Section A code ...		... Section C code ...	
Select an option	Answer: None	Select an option	Answer: None
... Section B code ...		... Section D code ...	
Select an option	Answer: signal(S)	Select an option	Answer: None

   Explanation
   In order to ensure that all of the code of P1 runs before P2 begins, we initialize our semaphore S to 0, and add a wait(S) at the beginning of process P2. We then ensure that this semaphore is only set to 1 when process P1 completes. So there is a signal(S) after the B code of P1.

2. Using a semaphore named S, please specify the initial value of semaphore S, and add wait(S) and signal(S) statements as needed in the spaces below so that that D < A or B < C, i.e., executions of P1 and P2 cannot overlap, but are allowed to occur in either order. For each drop down, select the missing line of code. If no command is missing, select None.

   // Semaphore initial value

   semaphore S = Answer: 1 ;

// Process P1		// Process P2	
Select an option	Answer: wait(S)	Select an option	Answer: wait(S)
... Section A code ...		... Section C code ...	
Select an option	Answer: None	Select an option	Answer: None

Calculator

... Section B code ... ... Section D code ...

Select an option Answer: signal(S) Select an option Answer: signal(S)

Explanation
In order to ensure that execution of P1 and P2 does not overlap, we need to initialize our semaphore S to 1 and ensure that only one of the processes grabs it at a time. So before beginning either process, the process must first wait(S) to grab the semaphore, then run to completion, and then only release the semaphore using a signal(S) after the end of the process code. This will ensure that only one of the processes can be running at any point in time, and that it will complete before the other processes begins.

3. Using two semaphores named S and T, please specify the initial values of semaphores S and T, and add wait(S), wait(T), signal(S), and signal(T) statements as needed in the spaces below so that A < D and C < B, i.e., the first section (A and C) of **both** processes completes execution before the second section (B or D) of **either** process begins execution.

   For each drop down, select the missing line of code. If a particular code region only requires one command, then select that command for the first drop down and select None for the second drop down. If no commands are needed in a region then select None for both answers. **Assume process P1 calls signal(S).**

   // Semaphore initial values

   semaphore S = Answer: 0 ;

   semaphore T = Answer: 0 ;

   // Process P1 // Process P2

   Select an option Answer: None Select an option Answer: None

   Select an option Answer: None Select an option Answer: None

   ... Section A code ... ... Section C code ...

   Select an option Answer: signal(S) Select an option Answer: signal(T)

   Select an option Answer: wait(T) Select an option Answer: wait(S)

   ... Section B code ... ... Section D code ...

   Select an option Answer: None Select an option Answer: None

   Select an option Answer: None Select an option Answer: None

   Explanation
   To ensure that code sections A and C run before B and D, we begin by just letting the code in section A and C run without waiting or signaling anything. As soon as those sections are done, they need to signal the semaphore that the other process is waiting for. This ensures that if the semaphores are both initialized to 0, then A or C can run possibly overlapping. Section D won't run before A and C have completed because D comes after C in the code and because after completing C, it waits for semaph

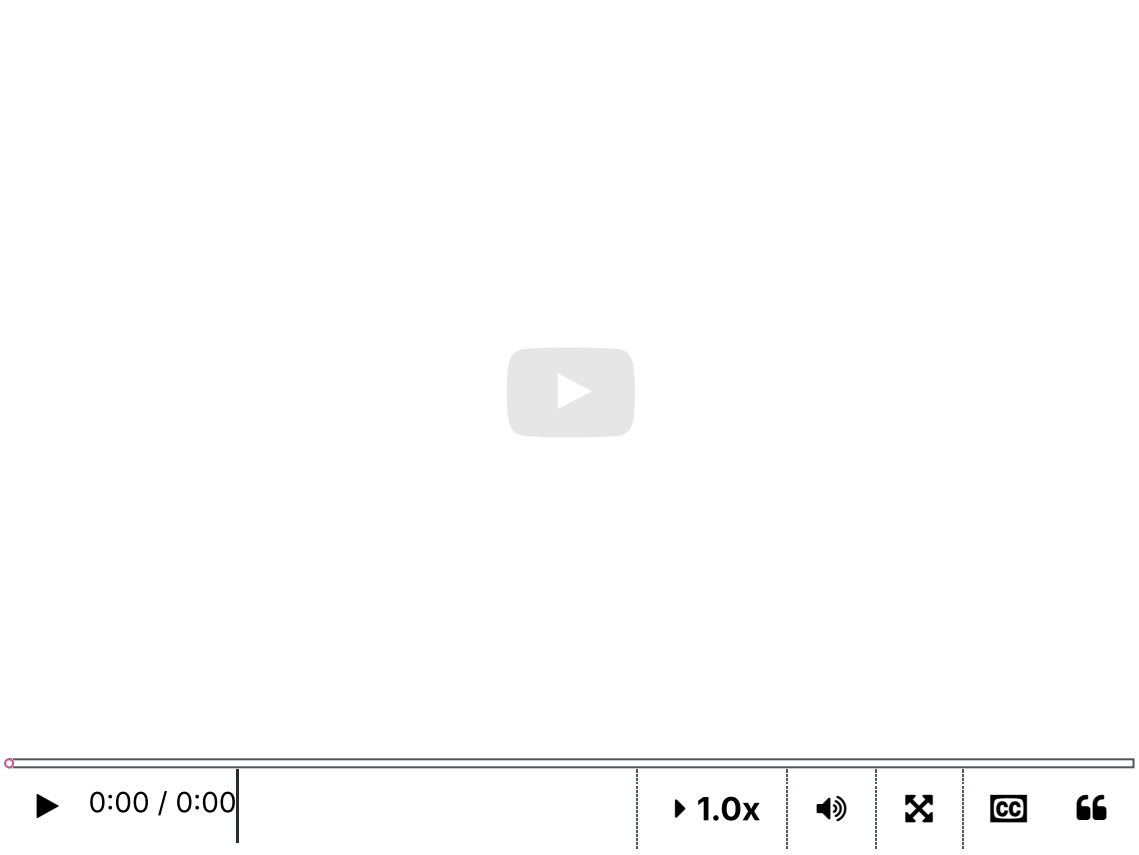
Calculator

which is only signaled after A completes. Similarly, section B won't run until sections A and C haves completed because B comes after A in the code and because after completing A, it waits for semaphore T which is only signaled after C completes.

Submit

Answers are displayed within the problem

Semaphores



Start of transcript. Skip to the end.

In this exercise, we will learn how semaphores can be used to ensure that different precedence constraints in our programs can be satisfied.

Before diving into the use of semaphores to enforce our precedence requirements, let's review what tools we have available to us.

You can think of a semaphore as a shared resource that is limited in quantity.

Video
Download video file

Transcripts
Download SubRip (.srt) file
Download Text (.txt) file

Discussion

Topic: 19. Concurrency and Synchronization / WE19.1

Hide Discussion

Add a Post

Show all posts by recent activity

[WE19.1.C] Isolating constraints, and then ORing them. 3

Previous Next

edX

About
Affiliates
edX for Business

Calculator