---- TEAM ----

**>> Team name.**

**>> Fill in the names, email addresses and contributions of your team members.**

FirstName LastName <email@domain.example> (contribution1)

FirstName LastName <email@domain.example> (contribution2)

contribution1 + contribution2 = 100

**>> Specify how many tokens your team will use.**

---- PRELIMINARIES ----

**>> If you have any preliminary comments on your submission, notes for the TAs, or extra credit, please give them here.**

**>> Please cite any offline or online sources you consulted while preparing your submission, other than the Pintos documentation, course text, lecture notes, and course staff.**

ALARM CLOCK

===========

---- DATA STRUCTURES ----

**>> A1: Copy here the declaration of each new or changed `struct' or `struct' member, global or static variable, `typedef', or enumeration. Identify the purpose of each in 25 words or less.**

---- ALGORITHMS ----

**>> A2: Briefly describe what happens in a call to timer\_sleep(), including the effects of the timer interrupt handler.**

**>> A3: What steps are taken to minimize the amount of time spent in the timer interrupt handler?**

---- SYNCHRONIZATION ----

**>> A4: How are race conditions avoided when multiple threads call timer\_sleep() simultaneously?**

**>> A5: How are race conditions avoided when a timer interrupt occurs during a call to timer\_sleep()?**

---- RATIONALE ----

**>> A6: Why did you choose this design? In what ways is it superior to another design you considered?**

PRIORITY SCHEDULING

===================

---- DATA STRUCTURES ----

**>> B1: Copy here the declaration of each new or changed `struct' or `struct' member, global or static variable, `typedef', or enumeration. Identify the purpose of each in 25 words or less.**

**>> B2: Explain the data structure used to track priority donation. Draw a diagram in a case of nested donation.**

---- ALGORITHMS ----

**>> B3: How do you ensure that the highest priority thread waiting for a lock, semaphore, or condition variable wakes up first?**

**>> B4: Describe the sequence of events when a call to lock\_acquire() causes a priority donation. How is nested donation handled?**

**>> B5: Describe the sequence of events when lock\_release() is called on a lock that a higher-priority thread is waiting for.**

---- SYNCHRONIZATION ----

**>> B6: Describe a potential race in thread\_set\_priority() and explain how your implementation avoids it. Can you use a lock to avoid this race?**

---- RATIONALE ----

**>> B7: Why did you choose this design? In what ways is it superior to another design you considered?**

SURVEY QUESTIONS

================

Answering these questions is optional, but it will help us improve the course in future quarters. Feel free to tell us anything you want--these questions are just to spur your thoughts. You may also choose to respond anonymously in the course evaluations at the end of the quarter.

**>> In your opinion, was this assignment or any one of the two problems in it, too easy or too hard? Did it take too long or too little time?**

**>> Did you find that working on a particular part of the assignment gave you greater insight into some aspect of OS design?**

**>> Is there some particular fact or hint we should give students in future quarters to help them solve the problems? Conversely, did you find any of our guidance to be misleading?**

**>> Do you have any suggestions for the TAs to more effectively assist students, either for future quarters or the remaining projects?**

**>> Any other comments?**