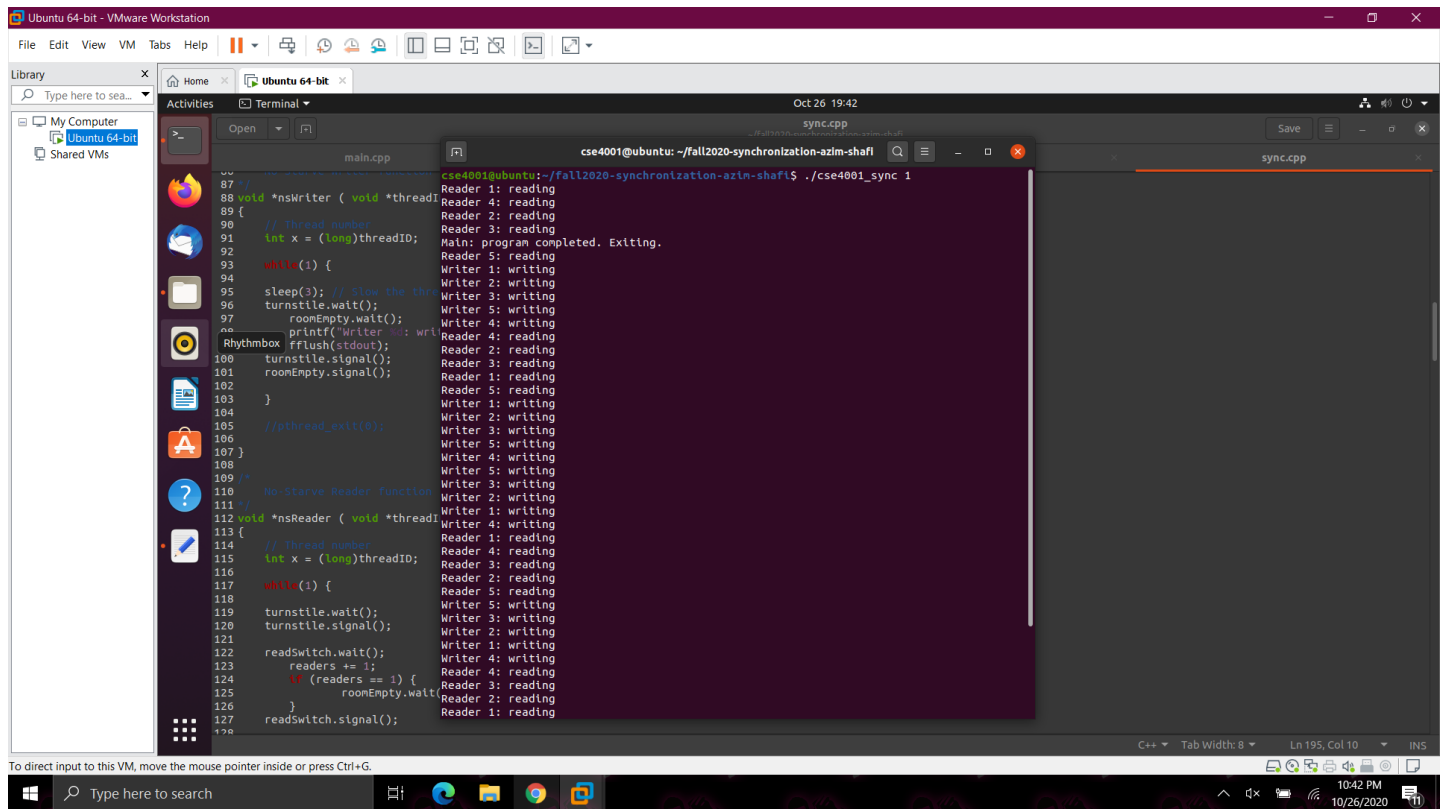


Solution #1



```
main.cpp
87 //
88 void *nsWriter ( void *thread) {
89 {
90 // Thread number
91 int x = (long)threadID;
92 while(1) {
93 sleep(3); // Slow the thread
94 turnstile.wait();
95 roomEmpty.wait();
96 printf("Writer %d: writing\n", x);
97 fflush(stdout);
98 turnstile.signal();
99 roomEmpty.signal();
100 }
101 //pthread_exit(0);
102 }
103 }
104 }
105 }
106 }
107 }
108 }
109 }
110 }
111 }
112 void *nsReader ( void *thread) {
113 {
114 // Thread number
115 int x = (long)threadID;
116 while(1) {
117 turnstile.wait();
118 readSwitch.wait();
119 readers += 1;
120 if (readers == 1) {
121 roomEmpty.wait();
122 readSwitch.signal();
123 }
124 }
125 }
126 }
127 }
128 }
```

```
Reader 1: reading
Reader 4: reading
Reader 2: reading
Reader 3: reading
Main: program completed. Exiting.
Writer 1: writing
Writer 2: writing
Writer 3: writing
Writer 5: writing
Writer 4: writing
Reader 4: reading
Reader 2: reading
Reader 3: reading
Reader 1: reading
Reader 5: reading
Writer 1: writing
Writer 2: writing
Writer 3: writing
Writer 5: writing
Writer 4: writing
Writer 5: writing
Writer 3: writing
Writer 2: writing
Writer 1: writing
Writer 4: writing
Reader 1: reading
Reader 4: reading
Reader 3: reading
Reader 2: reading
Reader 5: reading
Writer 5: writing
Writer 3: writing
Writer 2: writing
Writer 1: writing
Reader 4: reading
Reader 3: reading
Reader 2: reading
Reader 1: reading
```

No-Starve Reader-Writer: This solution works by allowing the readers to lock the room using a lightswitch, but aren't allow to stay indefinitely since the writes control a turnstile semaphore that limits how many readers can enter. Locks were implemented using semaphores and counters rather than creating a class.

Solution #2

[illegible]

Writer Priority Reader-Writer: This solution works by allowing both readers and writers to lock the room, but writers are given priority since writers must enter and leave one-by-one, but readers enter one-by-one then exit enmasse. The readswitch implementation used in the first solution was mirrored to create the writeswitch here.

Solution #3

The image shows a VMware Workstation interface with an Ubuntu 64-bit virtual machine. The terminal window is open, displaying the execution of a C++ program named 'main.cpp'. The program implements a dining philosophers problem with five philosophers (1-5) and a turnstile. The output shows the philosophers alternating between thinking and eating, synchronized by the turnstile and read/write switches. The terminal output is as follows: Main: program completed. Exiting. Philosopher 1: thinking, Philosopher 1: eating, Philosopher 2: thinking, Philosopher 3: thinking, Philosopher 3: eating, Philosopher 4: thinking, Philosopher 5: thinking, Philosopher 4: eating, Philosopher 2: eating, Philosopher 3: thinking, Philosopher 1: thinking, Philosopher 1: eating, Philosopher 3: eating, Philosopher 2: thinking, Philosopher 5: eating, Philosopher 4: thinking, Philosopher 2: eating, Philosopher 5: eating, Philosopher 4: thinking, Philosopher 1: thinking, Philosopher 1: eating, Philosopher 3: eating, Philosopher 4: thinking, Philosopher 2: thinking, Philosopher 5: eating, Philosopher 3: thinking, Philosopher 1: thinking. The VM interface includes a sidebar with 'My Computer' and 'Shared VMs', a top menu bar with 'File', 'Edit', 'View', 'VM', 'Tabs', 'Help', and a bottom status bar showing 'C++', 'Tab Width: 8', 'Ln 195, Col 10', and 'INS'.

Dining Philosophers #1: This solution solves deadlock by ensuring that only four can be seated at the table to ensure a situation in which all aren't waiting for their second fork. An array of semaphores and modulo statements were used as suggested by the book, but due to trouble implementing the provided semaphore class as an array, five semaphores and a switch statement were used originally, but a discussion in class cleared up that issue.

Solution #4

The screenshot shows a VMware Workstation window titled "Ubuntu 64-bit - VMware Workstation". The interface includes a menu bar (File, Edit, View, VM, Tabs, Help), a toolbar with icons for file operations and VM management, and a sidebar on the left with icons for applications like Rhythmbox and a file manager. The main window displays a terminal window titled "cse4001@ubuntu: ~/fall2020-synchronization-azim-shafi". The terminal shows the execution of a C++ program that simulates a dining philosophers problem. The program output is as follows:

```

Main: program completed. Exiting.
Philosopher 1: thinking
Philosopher 1: eating
Philosopher 2: thinking
Philosopher 3: thinking
Philosopher 3: eating
Philosopher 4: thinking
Philosopher 5: thinking
Philosopher 5: eating
Philosopher 2: eating
Philosopher 3: thinking
Philosopher 1: thinking
Philosopher 3: eating
Philosopher 5: eating
Philosopher 4: thinking
Philosopher 4: eating
Philosopher 3: thinking
Philosopher 5: thinking
Philosopher 5: eating
Philosopher 4: thinking
Philosopher 1: thinking
Philosopher 3: eating
Philosopher 1: eating
Philosopher 2: thinking
Philosopher 5: thinking
Philosopher 4: eating
Philosopher 2: eating
Philosopher 1: thinking
Philosopher 3: thinking
Philosopher 3: eating
Philosopher 5: eating
Philosopher 2: thinking
Philosopher 4: thinking
Philosopher 1: eating
Philosopher 4: eating
Philosopher 3: thinking
Philosopher 5: thinking

```

The VMware interface also shows a "Library" pane on the left with "My Computer" and "Shared VMs" sections. The bottom status bar indicates "C++", "Tab Width: 8", "Ln 268, Col 5", and "INS".

Dining Philosophers #2: This solution solves deadlock by ensuring at least one leftie and rightie are at the table. This was done by assigning odd philosophers to grab the left fork first, while even philosophers grab the right fork first.