Problem 5.1.

(1) A multiprogramming system uses only one processor, so the resources are split differently than they would be on a multiprocessing system.
(2) Because a multiprogramming system has only one processor, current items run semi-concurrent, meaning they must run mostly offset of one another, unless strictly running the same command.

Problem 5.3(a)

(1) P1: shared int x;
(2) P2: shared int x;
(3) P1: x = 10;
(4) P2: x = 10;
(5) P1: while ... The P1 while loop will execute "infinitely", not letting the P2 ever execute. This is where "x is 10" is printed

Question from the chapter:

The exchange instruction will only let one of two options control the processor, but not both. Whichever option is picked, the other is blocked.

Problem 5.5.

Busy waiting will always be less efficient than blocking wait because the processor is continually spinning during a busy wait.

Problem 5.7

(1) part a:
The program initializes two arrays, a boolean choosing of size n and integer number of size n. We then enter an infinite loop; index i of choosing is set to true, then index i of number is set equal to one more than either the length of the number array or n.

(2) part b:

Problem 5.8.

This program does not violate mutual exclusion.

Self-study 5.5

(1) The system accomplishes synchronization in this way: an instruction may not execute until it is received and cannot be sent until the proper resources are available to execute it.

(2) A message may only be passed if there is no blocking occurring. If there is no mutual exclusion, then there is blocking.

Problem 5.19.

Draw a similar figure as in Table 5.4 to show, with the revised code, the problem as discussed has been solved.