Thatcher Rickertsen (tor0002)

Comp 3500 – Introduction to Operating Systems

Homework 01

1. Number both P1 and P2 as follows (to provide clarity for the answers):

```
1       P1: {
2               shared int x;
3               x = 10;
4               while(1) {
5                       x = x - 1;
6                       x = x + 1;
7                       if (x != 10) {
8                               printf("x is %d", x);
9                       }
10              }
11      }
```

```
12      P2: {
13              shared int x;
14              x = 10;
15              while (1) {
16                      x = x - 1;
17                      x = x + 1;
18                      if (x != 10) {
19                              printf("x is %d", x);
20                      }
21              }
22      }
```

   1. The following combination would give an output of **“x is 10”**. Note that 0s are included as the line number when it is simply a bracket or otherwise is irrelevant towards the answer. *Code is only provided until the desired behavior is achieved (There would be clutter at the end).*

```
3       x = 10;
4       while (1) {
14              x = 10;
15              while (1) {
5                       x = x - 1
6                       x = x + 1
16                      x = x - 1
7                       if (x != 10) {
17                              x = x + 1;
8                               printf("x is %d", x);
```

   2. The following combination would give an output of **“x is 8”**. Note that 0s are included as the line number when it is simply a bracket or otherwise is irrelevant towards the answer. *Code is only provided until the desired behavior is achieved.*

```
3       x = 10;
4       while (1) {
14              x = 10;
15              while (1) {
5                       x = x - 1
6                       x = x + 1
16                      x = x - 1
7                       if  (x != 10) {
8                               printf("x is %d", x);
0                       }
0               }
```

   **This will produce “x is 9” the first time but the second time the inner loop operates it will produce “x is 8”.*

2. A binary semaphore (lock) can only keep a single resource locked or unlocked because it only has values of 0 and 1. A general semaphore (counting semaphore) can use an arbitrary resource count, therefore allowing them to manage multiple resources all at once.

3. A monitor, as it relates to synchronization, will check whether a resource is once again available. If it is, a monitor also often can signal the next process in line that it can now be run. This allows semaphores to put processes to sleep, rather than making them actively wait.
4. A semaphore usually has two operations associated with it: wait() and signal() (a.k.a. P() and V()). The wait() operation will check whether a process is clear to proceed or if it needs to wait, decrementing the semaphore's value as necessary. The signal() operation does the opposite, incrementing the semaphore's value as well as doing anything necessary to signal the next available process that it is its turn to proceed.