Homework Assignment #5

Due: Friday, October 8, 2020

1 What is the meaning of the term busy waiting? What other kinds of waiting are there in an operating system? Can busy waiting be avoided altogether? Explain your answer.

2 Illustrate how a binary semaphore can be used to implement mutual exclusion among *n* processes.

3 Describe how the compare_and_swap() instruction can be used to provide mutual exclusion that satisfies the bounded-waiting requirement.

4 Show how to implement the wait() and signal() semaphore operations in multiprocessor environments using the test_and_set() instruction. The solution should exhibit minimal busy waiting.

5 A semaphore puts a thread to sleep:

(a) if it tries to decrement the semaphore's value below 0.

(b) if it increments the semaphore's value above 0.

(c) until another thread issues a notify on the semaphore.

(d) until the semaphore's value reaches a specific number.

6 What is the difference between Mesa and Hoare scheduling for monitors?

7 The first known correct software solution to the critical-section problem for two processes was developed by Dekker. The two processes, P0 and P1, share the following variables:

```
boolean flag[2];    /* initially false */
int turn;
```

The structure of process Pi (i == 0 or 1) is shown in Figure 1. The other process is Pj (j == 1 or 0). Prove that the algorithm satisfies all three requirements for the critical-section problem.

```
do {
flag[i] = true;
while (flag[j]) {
        if (turn == j) {
                flag[i] = false;
                while (turn == j); /* do nothing */
        flag[i] = true;
        }
}
/* critical section */
turn = j;
flag[i] = false;
        /* remainder section */
} while (true);
```

Figure 1 The structure of process P_i in Dekker's algorithm

8 Recall that semaphores can be used to implement mutual exclusion or thread scheduling dependencies. Show in pseudocode how a semaphore can be used to implement the join operation on a thread. Be sure to indicate the initial value of the semaphore. What would be the fill in the program at LINE A, LINE B and LINE C?

```
Thread::Thread() {
 …
        if (joinable) {
        _____ //Line A：Your code goes here
        }
 …
}
void Thread::Join() {
        ASSERT(joinable);
         _____ ///LineB：Your code goes here
        delete this;
}
void Thread::Finish() {
        kernel->interrupt->SetLevel(IntOff);
         _____ ///Line C：Your code goes here
 if (joinable)
         Sleep(FALSE);
else
        Sleep(TRUE);
}
```

9 The readers-writers problem relates to an object such as a file that is shared between multiple threads. Some of these threads are readers i.e. they only want to read the data from the object and some of the threads are writers i.e. they want to write into the object. Try to use *Pthread* semaphore API to implement the reader-writer problem.