DEPARTMENT OF COMPUTER SCIENCE
CSC130 OPERATING SYSTEMS

CLASS EXERCISES 6 – PROCESS SYNCHRONIZATION
SUGGESTED SOLUTION

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.

Answer: *Busy waiting* means that a process is waiting for a condition to be satisfied in a tight loop without relinquish the processor. Alternatively, a process could wait by relinquishing the processor, and block on a condition and wait to be awakened at some appropriate time in the future. Busy waiting can be avoided but incurs the overhead associated with putting a process to sleep and having to wake it up when the appropriate program state is reached.

2. Explain why implementing synchronization primitives by disabling interrupts is not appropriate in a single-processor system if the synchronization primitives are to be used in user-level programs.

Answer: If a user-level program is given the ability to disable interrupts, then it candisable the timer interrupt and prevent context switching from taking place, thereby allowing it to use the processor without letting other processes to execute.

3. Explain why interrupts are not appropriate for implementing synchronization primitives in multiprocessor systems.

Answer: Interrupts are not sufficient in multiprocessor systems since disabling interrupts only prevents other processes from executing on the processor in which interrupts were disabled; there are no limitations on what processes could be executing on other processors and therefore the process disabling interrupts cannot guarantee mutually exclusive access to program state.

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

Answer:

```
do {
    waiting[i] = TRUE;
    key = TRUE;
    while (waiting[i] && key)
        key = Swap(&lock, &key);
    waiting[i] = FALSE;
    /* critical section */
```

```
        j = (i+1) % n;
        while ((j != i) && !waiting[j])
        j = (j+1) % n;
        if (j == i)
            lock = FALSE;
        else
            waiting[j] = FALSE;
        /* remainder section */
} while (TRUE);
```

6. Servers can be designed to limit the number of open connections. For example, a server may wish to have only *N* socket connections at any point in time. As soon as *N* connections are made, the server will not accept another incoming connection until an existing connection is released. Explain how semaphores can be used by a server to limit the number of concurrent connections.

Answer: A semaphore is initialized to the number of allowable open socket connections.When a connection is accepted, the acquire()method is called, when a connection is released, the release() method is called. If the system reaches the number of allowable socket connections, subsequent calls to acquire() will block until an existing connection is terminated and the release method is invoked.

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

Answer: Here is the pseudocode for implementing the operations:

```
int guard = 0;
int semaphore value = 0;
wait()
{ while (TestAndSet(&guard) == 1);
        if (semaphore value == 0) {   atomically add process to a queue of processes
                                      waiting for the semaphore and set guard to 0;
                                      }
        else {    semaphore value--;
                  guard = 0;
            }
}
signal()
{ while (TestAndSet(&guard) == 1);
if (semaphore value == 0 && there is a process on the wait queue)
        wake up the first process in the queue of waiting processes
else
        semaphore value++;
guard = 0;
}
```

8. Briefly describe the characteristics of a complete solution to the critical section problem.

- **Mutual Exclusion - If process Pi is executing in its critical section, then no other processes can be executing in their critical sections**
- **Progress - If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely**
- **Bounded Waiting - A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted**
- **Assume that each process executes at a nonzero speed**
- **No assumption concerning relative speed of the N processes**

9. Briefly describe how the semaphore provides mutual exclusion of 2 processes to access their shared common data.

Binary semaphore – integer value can range only between 0
and 1; can be simpler to implement
Also known as mutex locks
Can implement a counting semaphore S as a binary semaphore
Provides mutual exclusion

```
Semaphore S;      //   initialized to 1
wait (S);
Critical Section
signal (S);
```

10. Briefly describe the solution of signal(S) and wait(S) implementation without busy waiting.

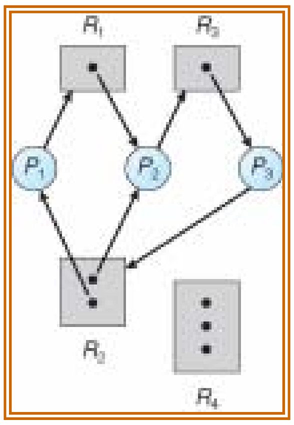
```
wait (S){     value--;
              if (value < 0) {     add this process to waiting queue
                                   block();   }
       }
```

```
Signal (S)   {    value++;
                  if (value <= 0) {   remove a process P from the waiting queue
                                      wakeup(P);   }
             }
```

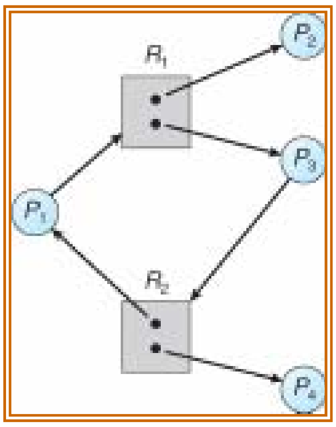
11. For a resource allocation graph, briefly describe the basic facts of resource allocation graph for deadlock condition, gives example to illustrate your answers.

- **If graph contains no cycles ⇒ no deadlock.**
- **If graph contains a cycle ⇒**
- **if only one instance per resource type, then deadlock.**
- **if several instances per resource type, possibility of deadlock.**

DEADLOCK



NO DEADLOCK



12. Deadlock is defined as two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes. Give example of deadlock for 2 processes sharing some common semaphore.

Let S and Q be two semaphores initialized to 1

```
P0                  P1
wait (S);           wait (Q);
wait (Q);           wait (S);
.                   .
.                   .
.                   .
signal   (S);       signal (Q);
signal (Q);         signal (S);
```

Starvation – indefinite blocking. A process may never be removed from the semaphore queue in which it is suspended.