Operating System 2021
Quiz #4

- The following solution for the bounded buffer problem was discussed in class:

```
const int N; shared int in = 0, out = 0, count = 0;
shared item_t buffer[N]; semaphore mutex = 1, empty = N, full = 0;

void producer()                    void consumer()
{                                  {
   produce(&item);                    down(&full);
   down(&empty);                      down(&mutex);
   down(&mutex);                      item = buffer[out];
   buffer[in] = item;                 out = (out + 1) % N;
   in = (in + 1) % N;                 up(&mutex);
   up(&mutex);                        up(&empty);
   up(&full);                         consume(item);
}
```

- The order of the two up() operations of the producer can be changed without causing significant side effects., Changing the order of the two down() operations of the consumer will cause a deadlock when the buffer is empty., Moving the produce(&item) call behind the two down() operations reduces concurrency but does not lead to a synchronization error.

- about semaphores
    - A down operation on a semaphore with the value 0 will block the thread performing the down operation., An up operation on a semaphore always dequeues a thread if there are any threads queued on the semaphore., Semaphores can be used to solve both mutual exclusion and coordination problems.
- about the semaphore pattern
    - The multiplex pattern is a generalization of the mutual exclusion pattern., In the double barrier pattern, each thread passing through the turnstile opens the turnstile for the next threat.
- A proper solution of the critical section problem must satisfy the following requirements.
    - Mutual exclusion, Progress, Bounded waiting
- about POSIX mutexes and condition variables
    - Condition variables should always be waited for in a while loop since there is no guarantee that the condition is true when the call of pthread_cond_wait() returns., The mutex passed to pthread_cond_wait() is released while waiting for a signal and it is aquired again before the call returns., Mutexes are essentially the same as binary semaphores., The following program will deadlock.

```
#include

int main(void)
{
   pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
   pthread_cond_t  c = PTHREAD_COND_INITIALIZER;

   (void) pthread_mutex_lock(&m);
   (void) pthread_cond_signal(&c);
   (void) pthread_cond_wait(&c, &m);
```

```
    (void) pthread_mutex_unlock(&m);

    return 0;
}
```

The following program will deadlock.

```
#include

int main(void)
{
    pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
    pthread_cond_t  c = PTHREAD_COND_INITIALIZER;

    (void) pthread_mutex_lock(&m);
    (void) pthread_cond_wait(&c, &m);
    (void) pthread_cond_signal(&c);
    (void) pthread_mutex_unlock(&m);

    return 0;
}
```

- synchronization in Java
  - Java uses the special keyword synchronized to mark methods as critical sections., The mutual exclusion is associated to an object instance in Java., Java provides wait() and notify() methods that resemble condition variables in the POSIX thread API.
- synchronization in Go
  - The Go language supports go routines, which are lightweight concurrent threads managed by the Go runtime., The Go runtime maps a potentially large number of concurrent go routines to a much smaller pool of operating system level threads., Go provides communication channels to solve coordination problems.
- proper definitions of critical sections
  - A critical section is a part of the program code where only one thread may execute at the same point in time.