# COMP3500: Synchronization 3: Semaphore

**🟊: >85%, 🟊🟊: 70-85%, 🟊🟊🟊: 55-70%, 🟊🟊🟊🟊: 40-55%, 🟊🟊🟊🟊🟊: < 40%**

**Exercise 1:** Read the following source code. Can you figure out how the bounded-waiting mutual exclusion is implemented? (2 minutes).

do {  
 waiting[i] = true;  
 key = true;  
 while (waiting[i] && key)

key = test\_and\_set(&lock);

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);

**🟊🟊🟊🟊 Exercise 1.1 (Plickers):** Process Pi can enter its critical section only if

1. Either waiting[i] == false or key == false
2. waiting[i] == true
3. key == false
4. waiting[i] == true and key == true

**🟊🟊🟊🟊🟊 Exercise 1.2 (Plickers):** The variable waiting[i] **(i.e., the first one in the while statement)**becomes false only if (**Assumption:** key == true).

1. Another process enters its critical section
2. Process i enters its critical section
3. Another process leaves its critical section
4. Process i leaves its critical section

**Exercise 2:** Please complete the following code. (1 minute)

release() {

available = \_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

do {

acquire \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

critical section

release \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

remainder section

} while (true);

**Exercise 3:** what Please explain the wait() and signal() operations below. How can a waited process be signaled to proceed? (2 minutes)

wait(S) {

while (S <= 0)

; // busy wait

S--;

}

signal(S) {

S++;

}

**Exercise 4:** In the following code of P1 and P2, it is required that S1 must happen before S2. How to initialize semaphore sync? **Hint:** consider the initial value of sync. (1 minute plus 1-minute group discussion)

P1:

S1;

signal(sync);

P2:

wait(sync);

S2;

**Exercise 5:** Please complete the following code, where the data structureof semaphore is given below. (2 minute)

typedef struct {

int value;  
 struct process \*L;  
} semaphore;

wait(semaphore \*S) {

S->value = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

if (S->value \_\_\_\_\_\_\_\_ 0) {  
 add this process to \_\_\_\_\_\_\_\_\_\_\_\_\_;

block();

}

}

signal(semaphore \*S) {

S->value = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

if (S->value \_\_\_\_\_\_\_\_ 0) {  
 remove a process P from \_\_\_\_\_\_\_\_\_\_\_\_;

wakeup(P);

}

}