Gruppe **20**
Betriebssysteme und Systemsoftware

Patrick Arens, 377536
Georg Manthey 376854
Benedikt Gerlach, 376944

Übung 5

Aufgabe 5.1

a)

Thread 0	Thread 1	Variable (flag = false)
while (true) {		
while (flag != false) {		
	while(true) {	
	while (flag != false)	
flag = true		flag = true
	flag = true	flag = true
critical_section()		
	critical_section()	

b)

Thread 0	Thread 1	Variable (counter = 0)
	while (true) {	
{	counter++	counter = 1
	if (counter == 3) {	
	}	
	}	
	while (true) {	
{	counter++	counter = 2
	if (counter == 3) {	
	}	
	}	
	while (true) {	
{	counter++	counter = 3
	if (counter == 3) {	
	critical_section	
while (true) {		
counter++		counter = 4
if (counter == 5) {		
}		
}		
while (true) {		
counter++		counter = 5
if (counter == 5) {		
critical_section()		

c)

Thread 0	Thread 1	Variable (mutex (unlocked))	Variable (mutex2 (unlocked))
Monitor.Enter(mutex)		mutex (locked by thread 0)	
	Monitor.Enter(mutex)		mutex2 (locked by thread 1)

Thread 0	Thread 1	Variable (semaphore [counter: 0])
while (true) {		
	while (true) {	
	if (semaphore.Wait(500)) {	
	semaphore.Release()	semaphore [counter: 1]
semaphore.Wait();		semaphore [counter: 0]
	}	
	}	
	while (true) {	
	semaphore.Release()	semaphore [counter: 1]
	while (true) {	
	if (semaphore.Wait(500)) {	semaphore [counter: 0]
	critical_section()	
critical_section()		

Aufgabe 5.2

a) Bounded Waiting
Turn wird mit 0 initialisiert, somit geht P0 nach der Prüfung der Schleife in den kritischen Bereich. P1 wird nun so scheduled das die Schleifenbedingung immer geprüft wird, wenn P0 im kritischen Bereich ist. Turn ist nun immer 0, falls P1 die Schleifenbedingung Prüft . P0 kann damit P1 beliebig oft überholen.

b) Mutal Exclusion
flag[0] und flag[1] werden mit false initialisiert, somit ist es möglich das das P0 die Schleifenbedingung prüft und dabei unterbrochen wird, bevor flag[0] = true ausgeführt wird. P1 prüft dann auch die Schleifenbedingung und setzt flag[1] = true und geht in den kritischen Bereich. P1 wird durch P0 unterbrochen welches flag[0] = true und anschließend in den kritischen Zustand geht. Damit sind 2 Prozesse im kritischen Zustand.

c) Progress
flag[0] und flag[1] werden mit false initialisiert. P0 setzt flag[0] = true, anschließend setzt P1 flag[1] = true. Nun befinden sich beide Prozesse in einer Endlosschleife da sie nun jeweils ihre Schleifenbedingung prüfen und noop ausführen.

Aufgabe 5.3

b)

```
1  signal (x) {
2    testinc (x);
3  }
4
5  wait (x) {
6    if (dectest (x) == -1) {
7      testinc (x);
8    }
9  }
```

Aufgabe 5.4

Die benachbarten Zeilen 07 nutzen nur das gleiche Array aber nicht den selben Speicherbereich darauf.
Würden beide dieser Operationen eine Zeile höher stehen kann es zu einer Mutual Exclusion kommen. Ein Beispielhafter Schedule dazu wäre:

CPU 1	CPU 2	Variable flag=(false, false)/ turn
01 repeat	01 repeat	
02 flag[1] = true		flag=(false, true)
03 turn = 0		turn=0
04 while(flag[0] and turn=0) {		
07 flag[1]=false		flag=(false, false)
	02 flag[0] = true	flag=(true, false)
	03 turn = 1	turn=1
	04 while(flag[1] and turn=1) {	
	07 flag[0]=false	flag=(false, false)
06 critical section 1	06 critical section 0	Mutual Exclusion
09 until false	09 until false	

Aufgabe 5.5

a)

```
reader()
    wait(sync1)
    if(sync2)
    {signal(sync2);}
    init(sync2,1)
    dostuff;
```

```
writer()
    wait(sync2)
    init(sync1, 1)
    dostuff;
    signal(sync1)
    signal(sync2)
```

b)

```
reader()
    wait(sync1)
    if(sync2 && !sync3)
    {signal(sync2);}
    init(sync2,1)
    dostuff;
```

```
writer()
    init(sync3, 1)
    wait(sync2)
    init(sync1, 1)
    dostuff;
    signal(sync1)
    signal(sync2)
```