Betriebssysteme und Softwaretechnik

Abgabe: 15.06.2017

Adrian C. Hinrichs Matr.Nr. 367129
Jeremias Merten Matr.Nr. 367626
Georg C. Dorndorf Matr.Nr. 366511

Aufgabe 5

Aufgabe 5.1

a)

Thread 0	Thread 1	Variable (flag = false)
while (flag != false)		
	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)		flag = true
while(true)		
counter++		counter = 5
if (counter == 5)		
critical_section()		

c)

Thread 0	Thread 1	Object mutex;	Object mutex2;
Monitor.Enter(mutex);		mutex; (locked by thread 0)	
	Monitor.Enter(mutex2);		mutex2;(locked by thread 1)

Es tritt ein Deadlock auf, da in den nachfolgenden Programmzeilen die Programme auf die Freigabe des jeweils anderen `mutex` warten.

d)

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

Aufgabe 5.2

a)

Beim gegebenen Ansatz ist es möglich, dass die Progress-Bedingung veletzt wir, wenn der Schduler zum Beispiel P1 nicht zum Zuge kömmen lässt wird `turn` niemals wieder freigegeben und P0 könnte keinen Fortschritt machen.

b)

Beim gegeben Ansatz kann die Mutual Exclusion Bedingung verletzt werde, da die Prozesse unsynchroniesiert voneinander gleichzeitig die Critical-Section bdetreten können.

c)

Beim gegebenen Ansatz wird die Bounded Waiting und die Prgogress Bedingung verletzt, da ein Deadlock auftreten kann, wenn beide Prozesse nacheinander `flag[0]` und `flag[1]` auf True setzen. Dann befinden sich nämliche beide Prozesse in einer noop()-Endlosschleife,

Aufgabe 5.3

a)

b)

Aufgabe 5.4

Aufgabe 5.5

a)

b)