BuS-Übungsgruppe 7 Übungsblatt 4 Felix Kiunke, 357322
Philipp Hochmann, 356148
Daniel Schleiz, 356092

6 Punkte

5.1

a) 1

First Army	**Second Army**	**Variable (flag = false)**
while (true)		
while (flag !=false)		
temp = true		
	while (true)	
	while (flag !=false)	
	temp = true	
	flag = temp	flag = true
	critical_section()	
flag = temp		Flag = true
critical_section()		

b)

Five-Headed Dragon	**Three-Headed Dragon**	**Variable (counter = 0)**
	while (true)	
	temp = counter +1	
	counter = temp	counter = 1
	if (counter == 3)	
	while (true)	
	temp = counter +1	
	counter = temp	counter = 2
	if (counter == 3)	
	while (true)	
	temp = counter +1	
	counter = temp	counter = 3
	if (counter == 3)	
	critical_section()	
while (true)		

2

temp = counter +1		
counter = temp		counter = 4
if (counter == 5)		
while (true)		
temp = counter +1		
counter = temp		counter = 5
if (counter == 5)		
critical_section()		

c) 1

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

d)

2

Thread 0	**Thread 1**	**SemaphoreSlim ss [counter: 0]**
	while (true)	
	if (ss.Wait(500))	
	else	
	ss.Release()	[counter: 1]
while (true)		
ss.Wait()		[counter: 0]
critical_section()		
	while (true)	
	if (ss.Wait(500))	
	else	
	ss.Release()	[counter: 1]
	while (true)	
	if (ss.Wait(500))	[counter: 0]
	critical_section()	

3 Punkte

5.2

Diese Lösung funktioniert nicht immer wie gewünscht, da ein "Reporterprozess" zwischen den beiden 'wait(politicianReady);' Anweisungen unterbrochen werden kann. So kann es zum Beispiel zu dem Fall kommen, dass zwei Politiker zur Verfügung stehen (also politicianReady == 2 ist), anschließend treffen nahezu gleichzeitig zwei Reporter ein; der erste Reporter führt 'wait(politicianReady);' aus und wird dann durch den zweiten Reporter unterbrochen, welcher ebenfalls sein erstes 'wait(politicianReady);' ausführt. Dann ist politicianReady == 0 und nach Aufgabenstellung soll jede Wartezeit vermieden werden, jedoch sind zwei Politiker vorhanden und ebenfalls zwei Reporter, demnach sollten die zwei Politiker und ein Reporter den Aufzug nehmen, dazu kommt es jedoch nicht, da beide Reporter noch einmal 'wait(politicianReady);' ausführen müssen.

Eine Lösungsmöglichkeit wäre das Einführen einer FIFO-Queue für die Reporter; beim Aufruf von reporterArrives() wird der aufrufende Prozess schlafen gelegt und in die Queue eingereiht. Beim Einfügen eines Reporters wird geprüft, ob 'politicianReady >= 2'. Ist dies der Fall, so wird der vorderste Reporter in der Queue geweckt und führt dann 'EnterElevator();' und den Code danach aus.
Führt ein Politiker 'signal(politicianReady);' aus, so wird nach Inkrementierung von politicianReady noch geprüft, ob 'politicianReady >= 2' und ob die Reporter-Queue nicht-leer ist. Sind diese beiden Bedingungen erfüllt, so wird politicianReady um 2 dekrementiert und es wird der vorderste Reporter in der Queue geweckt, welcher dann 'EnterElevator();' und den Code danach ausführt.

Korrekt, einfacher wäre aber noch die beiden Aufrufe von wait in reporterArrives() zum Beispiel durch einen Mutex abzusichern, sodass diese in einem Block ausgeführt werden.

11 Punkte

5.3

3

a)

```
// Zählsemaphor, Queue ist voll wenn die Variable 0 ist, Maximum n
int QueueSpace = n;
// Zählsemaphor, Anzahl der Elemente in der Queue, Maximum n
int QueueCount = 0;

int QueueAccess = 1; // Mutex, nur ein Queuezugriff zur gleichen Zeit
int enqMutex = 1;    // Mutex, nur ein einfügender Prozess zur gleichen Zeit
int deqMutex = 1;    // Mutex, nur ein entnehmender Prozess zur gleichen Zeit

list queue;

void enqueue(element e) {
   wait(enqMutex);
   // Warte, bis in der Queue Platz zum Einfügen ist
   wait(QueueSpace);

   wait(QueueAccess);
   // Füge element hinzu, erhöhe QueueCount, da nun ein Element mehr drin
   queue.add(e);
   signal(QueueCount);
   signal(QueueAccess);

   signal(enqMutex);
}

element dequeue() {
   wait(deqMutex);
   // Warte, bis in der Queue ein Element zum Entnehmen ist
   wait(QueueCount);

   wait(QueueAccess);
   // Entnehme Element, inkrementiere QueueSpace, da neuer Platz vorhanden
   element e = queue.pop();
   signal(QueueSpace);
   signal(QueueAccess);

   signal(deqMutex);
   return e;
}
```

b) 1

Ein Problem liegt darin, dass eingefahreneWagen nicht synchronisiert ist, zum Beispiel: Angenommen es ist gerade kein Wagen da und es kommen zwei Wagen fast gleichzeitig an. Der Prozess des ersten Wagens springt also nicht in 'while (eingefahreneWagen > 0)' und geht einen Schritt weiter im Program Counter. Nun wird der Prozess durch den des zweiten Wagens unterbrochen, dieser gelangt ebenfalls nicht in die oben genannte while-Schleife, da eingefahreneWagen noch 0 ist, setzt dann eingefahreneWagen auf 1 und benutzt 'fahreAufPlattform()'. Nun wird der Prozess des zweiten Wagens unterbrochen und der erste Wagen ist wieder dran, setzt eingefahreneWagen auf 1 und führt dann auch 'fahreAufPlattform()' aus. Dies ist jedoch nicht erlaubt, da nur ein Wagen auf der Plattform sein kann.

Zweites Problem: verfügbareSitze ist ebenfalls nicht synchronisiert, zum Beispiel: Sei gerade ein Wagen auf der Plattform und eine Person befindet sich darin, also verfügbareSitze = 1. Kommen nun zwei Besucher fast gleichzeitig an, so kann es sein, dass zunächst einmal die erste Person AnkunftBesucher() aufruft, nicht in die while-Schleife springt, da verfügbareSitze = 1, betreteWagen() ausführt, und dann durch den Prozess des zweiten Besuchers unterbrochen wird. Dieser springt ebenfalls nicht in die while-Schleife, betritt den Wagen mit betreteWagen() und dekrementiert verfügbareSitze und ist dann fertig. Nun macht der Prozess der ersten Person weiter und dekrementiert verfügbareSitze und ist dann auch fertig. Nun sind aber theoretisch 3 Personen im Wagen, zudem ist verfügbareSitze negativ, was nicht passieren sollte.

c) 3

```
int plattformPlatz = 1;    // Mutex, 1 wenn Platz auf der Plattform ist
int eingestiegeneBesucher = 0;  // Zählsemaphor, Maximum 2
int verfügbareSitze = 0;   // Zählsemaphor, Maximum 2

void AnkunftWagen() {
   wait(plattformPlatz);
   fahreAufPlattform();
   öffneTüren();
   signal(verfügbareSitze);
   signal(verfügbareSitze);
   wait(eingestiegeneBesucher);
   wait(eingestiegeneBesucher);
   schließeTüren();
   verlassePlattform();
   signal(plattformPlatz);
}
```

```
void AnkunftBesucher() {
   wait(verfügbareSitze);
   betreteWagen();
   signal(eingestiegeneBesucher);
}
```

d)
Siehe bend.c

3

e)
Dieses Problem ließe sich zum Beispiel mit Semaphoren mit einer Warteschlange lösen. Benötigt wäre eine FIFO-Queue, welche eine standardmäßige dequeue() Funktion zur verfügung stellt und außerdem eine enqueue_normal() und eine enqueue_vip() Funktion, wobei die enqueue_normal() einen normalen Besucherprozess standardmäßig an das Ende der Queue einreiht, während enqueue_vip() einen VIP Besucherprozess vor allen normalen Besuchern in die Queue einreiht, aber hinter möglicherweise sich in der Queue befindenden VIP Besuchern.

1

Zudem seien die beiden Zählsemaphoren eingestiegeneBesucher und verfügbareSitze wie aus c) und d) gegeben. Demnach wird ein Prozess, welcher Ankunftbesucher() aufruft, mittels enqueue_normal() in die Queue gereiht, und ein Prozess, welcher AnkunftVIP() aufruft, mittels enqueue_vip() eingereiht. Bei beiden wird nach der Einreihung wait(verfügbareSitze) aufgerufen, was darauf wartet, dass verfügbareSitze > 0, und dann verfügbareSitze um eins dekrementiert und dequeue() aufruft, also den vordersten der Warteschlange freigibt, welcher dann in den Wagen einsteigt. Danach läuft das Programm wie sonst weiter. So wird sichergestellt, dass VIP's Vorrang haben.

Insgesamt 20/20