VO 182.711 17. Januar 2014

Prüfung Betriebssysteme

KNr. MNr. Zuname, Vorname

Ges.)(100) 1.)(35) 2.)(20) 3.)(45) Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation mit Semphoren (35)

In einem entlegenen Skigebiet gibt es zwei Berggipfel, von denen Skifahrer abfahren können. Zu jedem Gipfel gibt es eine Talstation, von wo aus die Skifahrer mit Pistentaxis zum eben diesem Gipfel gebracht werden. Von jedem der beiden Gipfel gibt es zwei unterschiedliche Pisten, die zu den beiden Talstationen führen, d.h., Skifahrer müssen sich am Gipfel entscheiden, zu welcher Talstation sie fahren. Um die Pistentaxis gleichmäßig auszulasten, gibt es auf den Gipfeln Anzeigetafeln, von denen Skifahrer eine Empfehlung ablesen können, zu welcher der beiden Talstationen sie abfahren sollen.

Der Betrieb im Skigebiet wird in einem Softwarepaket durch 3 Arten von Prozessen simuliert.

- *Skifahrer* simuliert einen Skifahrer, der beliebig oft mit einem Pistentaxi zu einem Gipfel fährt, dann einen Blick auf die Anzeigetafel (realisiert durch ein Shared Memory) wirft und anschließend zur entsprechenden Talstation abfährt. Es kann beliebig viele Prozesse dieses Typs geben.
- *Taxi(nr)* simuliert das Pistentaxi mit der Nummer nr ($nr \in \{1, 2\}$), das Skifahrer von der Talstation nr zum Gipfel nr bringt.
- *Update* wird immer wieder gestartet, um die Information der Anzeigetafeln zu aktualisieren (schreibt auf das Shared Memory).

Ergänzen Sie die angegebenen Prozesstemplates, um die Prozesse durch *Semaphore* zu synchronisieren. Beachten Sie folgende Punkte:

- Alle Anzeigetafeln werden durch ein einziges Shared Memory simuliert. Von diesem Shared Memory sollen die Skifahrer gleichzeitig lesen können.
- Aktualisierungen des Shared Memory durch den *Update* Prozess sollen möglichst wenig verzögert werden.
- In ein Taxi passen K Leute. Das Taxi fährt von der Talstation ab, wenn es voll ist und lässt am Gipfel alle Leute aussteigen, bevor es wieder zu Tal fährt. Der Code für diesen Prozess ist gegeben. Zur Synchronisation mit dem Taxi NR werden Semaphore mit Namen $<Sem\_Name>\_NR$ verwendet.

Code für Prozesse und Initialisierungen

```
void Skifahrer(void)
{
  int nr = 1;
  for(;;) {
    /* Warten auf Taxi */
```



```
    einsteigen_Taxi(nr);
```



```
    aussteigen_Taxi(nr);
```



```
    nr = ShM_Anzeige;
```



```
    fahre_Piste(nr);
  }
}
```

```
void Update(void)
{
```



```
    /* Anzeige ShM aktualisieren */
    ShM_Anzeige = eval_sensors();
```



```
}
```

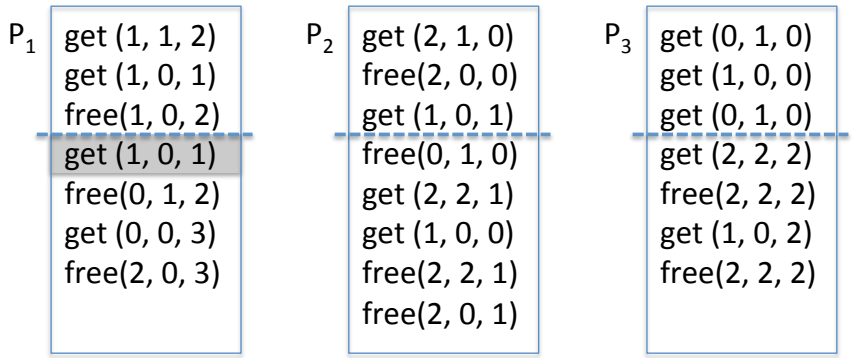
```
void Taxi(int NR)
{
  int cnt;
  for(;;) {
    /* Talstation NR */
    /* freie Plaetze signalisieren */
    for(cnt=0; cnt<K; cnt++) {
        V(in_NR);
    }
    /* Warten auf Einsteigen */
    for(cnt=0; cnt<K; cnt++) {
        P(entered_NR);
    }

    bergfahrt_NR();

    /* Bergstation NR */
    /* Freigabe zum Aussteigen */
    for(cnt=0; cnt<K; cnt++) {
        V(out_NR);
    }
    /* Warten Aussteigen */
    for(cnt=0; cnt<K; cnt++) {
        P(exited_NR);
    }

    talfahrt_NR();
  }
}
```

```
/**  Semaphorinitialisierungen  **/
```

2 Deadlock Avoidance – Banker's Algorithm (20)

In einem Computersystem gibt es drei Prozesse (P_1, P_2, P_3) und drei Arten von Ressourcen (R_1, R_2, R_3). Der *Resource Vector* R=(4, 5, 4) beschreibt die Anzahl der vorhandenen Ressourcen. Prozesse verwenden die Operationen *get(r_1, r_2, r_3)* bzw. *free(r_1, r_2, r_3)*, um r_i Ressourcen der Ressourcenart R_i ($i = 1 \ldots 3$) anzufordern bzw. freizugeben.

Die folgende Abbildung zeigt für jeden der drei Prozesse die Folge von *get* und *free*-Operationen, die bei jeder Prozessabarbeitung durchgeführt werden.



Nehmen Sie an, dass die drei ersten Operationen jedes Prozesses abgearbeitet wurden (d.h., die Abarbeitung jedes Prozesses befindet sich an der strichlierten Linie). Als nächster Schritt soll die vierte Operation von P_1 (grau hinterlegte Operation) durchgeführt werden.

Verwenden Sie den Banker's Algorithmus, um festzustellen, ob die vierte Operation von P_1 durchgeführt werden soll. Geben Sie alle erforderlichen Vektoren und Matrizen an und führen Sie den Banker's Algorithmus schrittweise durch, d.h. geben Sie für jeden Schritt die Werte der Elemente aller relevanten Matrizen und Vektoren an.

3 Fragen zu Betriebssystemen (45)

Skizzieren Sie die Folge der Schritte, die bei der Abarbeitung eines *Synchronen I/O Requests* ablaufen. (5)

Was versteht man unter einem *Kernel Level Thread* und unter einem *User Level Thread*? Beschreiben Sie die beiden Arten der Threadimplementierung und charakterisieren Sie deren Unterschiede. (4)

Was versteht man unter einem *Monitor* zur Prozesssynchronisation? Nennen Sie die wichtigsten Komponenten und Eigenschaften des Monitors. (4)

Nennen Sie die Arten von Optimierungszielen, die ein Scheduler beim Prozess-Scheduling verfolgen kann und geben Sie jeweils Beispiele an. (4)

Was ist *Thrashing*, wodurch kommt es dazu? Wie erkennt das Betriebssystem Thrashing? Wie kann dieses Problem beseitigt werden? (4)

Wozu wird die *Clock Policy* verwendet? Beschreiben Sie deren Funktionsweise. (5)

Was versteht man unter *Blocking* bzw. *Non-Blocking I/O*? Beschreiben Sie die beiden Arten, I/O-Operationen durchzuführen. (3)

Was versteht man unter einer *File Allocation Table*? Wie ist diese organisiert? (2)

Beschreiben Sie das Prinzip einer Sicherheitsattacke durch Stack/Buffer Overflow. Wodurch kann man sich bei der Implementierung eines Betriebssystems vor einen solchen Angriff schützen? (4)

Was beschreibt das Modell von Bell und LaPadula? Geben Sie die vom Modell geforderten Eigenschaften an. (4)

Nennen Sie die vier Layer eines TCP/IP Stacks und beschreiben Sie kurz deren Aufgaben. Geben Sie evtl. Beispiele an. (4)

Was versteht man in einem Distributed File System unter *Location Transparency* bzw. *Location Independence*. Erklären Sie die beiden Begriffe. (2)