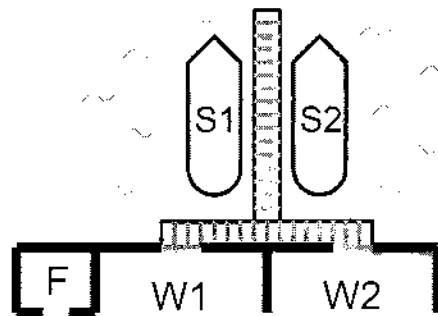


Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation mit Semaphoren (30)

Sie sollen einen Softwareentwickler beim Schreiben einer Simulationssoftware für eine Schiffsanlegestelle (siehe Abbildung) unterstützen. Diese Software simuliert die Aktionen von Schiffen und Passagieren durch Prozesse, die mit Semaphoren synchronisiert werden.



An der Schiffsanlegestelle gibt es Anlegeplätze für zwei Schiffe, S1 und S2, zwei Wartezonen, W1 und W2, die jeweils einem Schiff zugeordnet sind, und einen Fahrkartenschalter F.

Personen, die mit einem der Schiffe, S1 oder S2, verreisen, lösen am Schalter F eine Fahrkarte für ihr Schiff und gehen dann in die entsprechende Wartezone. Dort warten sie, bis sie an die Reihe kommen und gehen dann über den Steg auf ihr Schiff, um ihre Schiffsreise anzutreten.

Schiffe kommen jeweils leer am Steg an, warten dann, bis die erlaubte Anzahl von Passagieren an Bord ist, und verlassen dann den Steg wieder.

Ergänzen Sie in den Codestücken die fehlenden Semaphoreoperationen, um die Prozesse unter folgenden Rahmenbedingungen zu synchronisieren.

- Ein Schiff kann N Passagiere aufnehmen. Schiffe fahren leer zur Anlegestelle, nehmen dort N Passagiere auf und fahren dann wieder ab.
- Für die Durchgänge von den Warteräumen zum Steg gilt, dass jeder Durchgang von maximal zwei Personen gleichzeitig passiert werden kann.
- Die Einstiegstelle der Schiffe ist sehr schmal. Passagiere können die Schiffe daher nur einzeln besteigen.

- Aus Sicherheitsgründen dürfen sich zu jedem Zeitpunkt maximal K Personen auf dem Steg befinden.
- Beim Anlegen bzw. Ablegen eines Schiffes dürfen sich keine Passagiere am Steg befinden.
- Schiffe und Passagiere sollen unter Einhaltung der angegebenen Regeln möglichst unabhängig und möglichst ohne Einschränkungen der Parallelität agieren können.

(a) Initialisierungen (vor Start der Prozesse)

```

init(SP, 0);      init(S1, 2);      init(P1, 0);      init(S2, 0);
init(S1, 0);      init(ES1, 1);      init(P2, 0);
init(S2, 0);      // ...

```

(b) Prozesse für Schiffe

Bedeutung der verwendeten Funktionen:

anlegen(S) Schiff S legt am Steg an

abfahren(S) Schiff S legt vom Steg ab

```
/** Schiff S1 **/
```

```

// ...
P(SP); // Schiff für ...
do K times: P(st);
anlegen(S1); // ...
do N times: V(S1); // ...
for (i=0; i<P; i++) P(Pi); // ...
do K times: P(st);
abfahren(S1);
do K times: V(st);
V(SP);
// ...

```

```
/** Schiff S2 **/
```

```

// ...
P(S2);
anlegen(S2);
// ...
abfahren(S2);
// ...

```

(c) Prozesse für Passagiere

Bedeutung der verwendeten Funktionen:

fahrkarte_kaufen() der Passagier kauft eine Fahrkarte
warteraum_betreten(W) der Passagier betritt den Warteraum W
steg_betreten() der Passagier betritt und geht auf dem Steg
schiff_betreten(S) der Passagier betritt das Schiff S

/** Passagier fuer S1 **/

/** Passagier fuer S2 **/

fahrkarte_kaufen()

fahrkarte_kaufen()

warteraum_betreten(W1)

warteraum_betreten(W2)

steg_betreten()

steg_betreten()

schiff_betreten(S1)

schiff_betreten(S2)

2 Page Replacement (25)

Gegeben ist ein Arbeitsspeicher mit vier Frames, dessen Seiten mit unterschiedlichen Ersetzungsstrategien ersetzt werden sollen: mit OPT, LRU, bzw. mit dem Clock Algorithmus. Geben Sie in den Tabellen für jeden Algorithmus die Speicherinhalte für jeden Frame nach jedem Zugriff der angegebenen Seitenzugriffsfolge an. Die Seitenzugriffsfolge ist für alle Algorithmen gleich. Sie ist jeweils in der Kopfzeile der Tabelle gegeben. Geben Sie in den Spalten die Speicherbelegung nach dem entsprechenden Seitenzugriff an und kennzeichnen Sie in der letzten, mit *PF* markierten Zeile das Auftreten von Page Faults. Der Arbeitsspeicher ist am Beginn leer.

OPT-Strategie:

	A	B	C	D	E	F	C	D	E	C	F	G	H	E	D
0	A	A	A	A	E	E			E	E	E	E	E	E	E
1		B	B	B	B	F		F	F	F	F	G	G	G	G
2			C	C	C	C	C	C	C	C	C	C	H	H	
3				D	D	D	D	D	D	D	D	D	D	D	D
PF		PF	PF	PF	PF	PF						PF	PF		

LRU-Strategie:

	A	B	C	D	E	F	C	D	E	C	F	G	H	E	D
0	A	A	A	A	E	E			E			E	H	E	E
1		B	B	B	B	F		F	F			F	F	F	F
2			C	C	C	C	C	C	C	C	C	C	C	E	E
3				D	D	D	D	D	D	D	D	C	G	G	G
PF	PF	PF	PF	PF	PF	PF						PF	PF	PF	PF

Clock-Algorithmus:

	A	B	C	D	E	F	C	D	E	C	F	G	H	E	D
0	A		A	A	E	F	E	E	E	E	E	E	E	E	E
1		B	B	B	B	F	F	F	F	F	F	F	F	F	F
2			C	C	C	C	C	C	C	C	C	G	G	G	G
3				D	D	D	D	D	D	D	D	D	H	H	H
PF	PF	PF	PF	PF	PF	PF						PF	PF		PF

Vergleichen und diskutieren Sie die Anzahl der bei den Seitenersetzungsstrategien beobachteten Page Faults.

OPT	6
LRU	12
Clock	12

3 Fragen zu Betriebssystemen (45)

Erklären Sie die drei Begriffe *Deadlock*, *Livelock* und *Starvation*. (5)

Handwritten answer:

Deadlock: Zustand, in dem zwei oder mehr Prozesse auf Ressourcen warten, die von anderen Prozessen gehalten werden, die ebenfalls auf Ressourcen warten. Livelock: Zustand, in dem zwei oder mehr Prozesse auf Ressourcen warten, die von anderen Prozessen gehalten werden, die ebenfalls auf Ressourcen warten. Starvation: Zustand, in dem ein Prozess für eine unendliche Zeit auf Ressourcen warten muss, weil andere Prozesse die Ressourcen monopolisieren.

Worin liegt der grundlegende Unterschied zwischen *Prozessen* und *Threads*? Welcher Vorteil ergibt sich aus der Einführung von Threads für den Benutzer und worauf muss der Benutzer achten? (4)

Handwritten answer:

Prozesse sind unabhängige Ausführungsinstanzen, die ihren eigenen Adressraum und Ressourcen haben. Threads sind Ausführungsinstanzen innerhalb eines Prozesses, die den Adressraum des Prozesses teilen. Threads sind leichter zu erstellen und zu zerstören als Prozesse. Threads können die Ausführung eines Prozesses parallelisieren und die Leistung verbessern. Threads müssen synchronisiert werden, um Dateninkonsistenzen zu vermeiden.

Was versteht man unter *Virtual Memory Management*? Welche Vorteile bietet es? (5)

Handwritten answer:

Virtual Memory Management ist die Verwaltung von virtuellem Hauptspeicher. Es ermöglicht die Ausführung von Programmen, die mehr Hauptspeicher benötigen, als physisch vorhanden ist. Vorteile: Erhöhter Hauptspeicher, Schutz vor Speicherüberschreitung, Vereinfachung der Programmierung.

Beschreiben Sie Aufgabe und Funktion eines *Translation Lookaside Buffers*? Worauf hat man bei der Betriebssystemimplementierung bei einem Process Switch zu achten, wenn man einen Translation Lookaside Buffer verwendet? (4)

Ein TLB (Translation Lookaside Buffer) ist eine kleine, schnelle Cache-Struktur, die die aktuellen Übersetzungen von virtuellen zu physischen Adressen speichert. Er wird verwendet, um die Performance von Memory Access zu verbessern, indem er die Suche im Hauptspeicher (RAM) umgeht. Bei einem Process Switch muss der TLB für den neuen Prozess neu befüllt werden, da die alten Übersetzungen nicht mehr gültig sind. Dies kann zu einem TLB-Miss führen, bei dem die Übersetzung neu gesucht werden muss. Um dies zu vermeiden, kann man einen TLB-Schreibschutz (Write-Through) implementieren, bei dem die Übersetzungen direkt in den Hauptspeicher geschrieben werden, oder einen TLB-Invalidate-Mechanismus, der die alten Übersetzungen ungültig macht.

Was versteht man unter *Thrashing*? (2)

Thrashing ist ein Zustand, bei dem ein System so viel Zeit mit dem Verschieben von Daten zwischen dem Hauptspeicher und der Festplatte verbringt, dass keine weiteren Berechnungen durchgeführt werden können. Dies tritt auf, wenn die Arbeitsmenge größer ist als die verfügbare Hauptspeicherkapazität, was zu einem hohen Grad an Fragmentierung und einem hohen Anteil an Seitenfehlern (Page Faults) führt. Thrashing kann durch eine Erhöhung der Hauptspeicherkapazität oder durch eine Reduzierung der Arbeitsmenge vermieden werden.

Wie funktioniert *Round Robin* Scheduling? Welchen wichtigen Parameter gibt es bei diesem Verfahren? Wie wird man diesen Parameter günstiger Weise wählen? (4)

Round Robin Scheduling ist ein Zeit-scheitendes Verfahren, bei dem die CPU-Zeit in kleine Zeitscheiben (Time Slices) unterteilt wird, die den verschiedenen Prozessen im System zugeteilt werden. Der wichtigste Parameter ist die Zeitscheibe (Time Slice), die die maximale Zeit darstellt, die ein Prozess für die Ausführung erhalten darf, bevor er in die Warteschlange zurückgeführt wird. Eine zu kleine Zeitscheibe führt zu einem hohen Kontextwechsel-Overhead, während eine zu große Zeitscheibe zu einer schlechten Reaktionszeit führt. Ein guter Wert für die Zeitscheibe hängt von der Art der Last ab, aber ein typischer Wert liegt zwischen 10 und 100 Millisekunden.

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

Nennen Sie Design Prinzipien für die Konstruktion von sicheren Systemen. Geben Sie für jede Regel ein Beispiel an. (4)