Vorlesungsprüfung aus Systemprogrammierung 2002-03-20

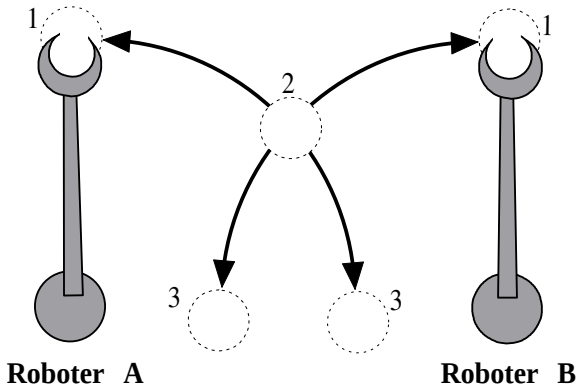
KNr. MNr. Zuname, Vorname

Ges.)(100)	1.)(30)	2.)(25)	3.)(25)	4.)(20)	Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Synchronisation (30)

In einer Fabrik sind zwei Roboter so angeordnet, dass sich ihre Arbeitsbereiche teilweise überlappen. Jeder Roboter kann eine von drei Arbeitspositionen anfahren, um Objekte aufzunehmen oder abzulegen.



Implementieren Sie die Teile eines Programms `Transport`(*von*, *nach*), welches einen Roboter ein Objekt von Position *von* nach Position *nach* legen lässt.

Ihnen stehen folgende Funktionen zur Verfügung:

`Grab()` lässt den Roboter ein Objekt von jener Position aufnehmen, auf welcher der Roboterarm gerade steht.

`Release()` lässt den Roboter ein Objekt an jener Position ablegen, auf welcher der Roboterarm gerade steht.

`GoTo(`***Position***`)` Fährt den Roboterarm an die angegebene Position.

Beachten Sie folgende Randbedingungen:

- Die Arme der Roboter A und B dürfen sich niemals überkreuzen. Weiters dürfen sich beide Arme nie gleichzeitig auf Position 2 oder 3 befinden. Position 2 kann aber z. B. angefahren werden, wenn der andere Greifer auf Position 3 steht.
- Gehen Sie davon aus, dass bei Aufruf der Funktion `Transport`() ein Objekt auf *von* vorhanden ist, die Robotergreifer auf Position 1 stehen und kein Objekt gegriffen wurde.
- Es können beliebig viele Objekte an einem Ort abgelegt werden, d.h es muss *nicht* überprüft werden, ob eine Position frei ist.
- Stellen Sie den Greifer nach erfolgter Aktion in die Position 1 zurück.
- Gehen Sie davon aus, dass je eine Instanz von Transport() auf jedem der Roboter gleichzeitig laufen kann.
- Stellen Sie sicher, dass während des Be- oder Entladens der andere Roboter die nicht blockierten Positionen anfahren kann, sofern dadurch kein Deadlock entstehen kann.
- Verwenden Sie ausschließlich Semaphore zur Synchronisation. Es sind möglichst wenig Synchronisationskonstrukte zu verwenden, die Verwendung von globalen Variablen ist verboten.

Verständnisfrage

Zu welchen Problemen kann es kommen, wenn Roboter A nicht jedesmal auf Position 1 zurückgestellt wird, sondern auf der letzten Entladeposition stehenbleiben würde? Kreuzen Sie die richtige(n) Antwort(en) an und begründen Sie Ihre Antwort!

◯ Deadlock ◯ Starvation ◯ Lifelock

Implementierung

Legen Sie die notwendigen Synchronisationskonstrukte mit der Funktion InitSem(*Semaphorname*,*Initialwert*) an und initialisieren Sie sie entsprechend der Situation auf dem Bild.

Transport()

Das Programm Transport(*von*, *nach*) verwendet die Unterprogramme Transport1To(*nach*), Transport2To(*nach*) und Transport3To(*nach*). Implementieren Sie diese, so dass Transport() wie angegeben funktioniert:

```
Transport(von,nach)

{

  switch(von) {

    case 1:

      Transport1To(nach);

      break;

    case 2:

      Transport2To(nach);

      break;

    case 3:

      Transport3To(nach);

      break;

  }

}
```

```
Transport1To(nach)

{

}
```

```
Transport2To(nach)
{
}
```

```
Transport3To(nach)
{
}
```

2 Memory Management - Replacement Policies (25)

Gegeben ist eine virtuelle Speicheradressierung (Paging mit Frame Allocation Size von 3) und eine Folge von Page-Zugriffen mit den folgenden Page-Nummern: 3,2,5,4,3,5,1,5,3,4,5. Es sind die Inhalte der Page-Frames nach jedem Page-Zugriff zu bestimmen. Zusätzlich sind eventuelle Page-Faults zu markieren.

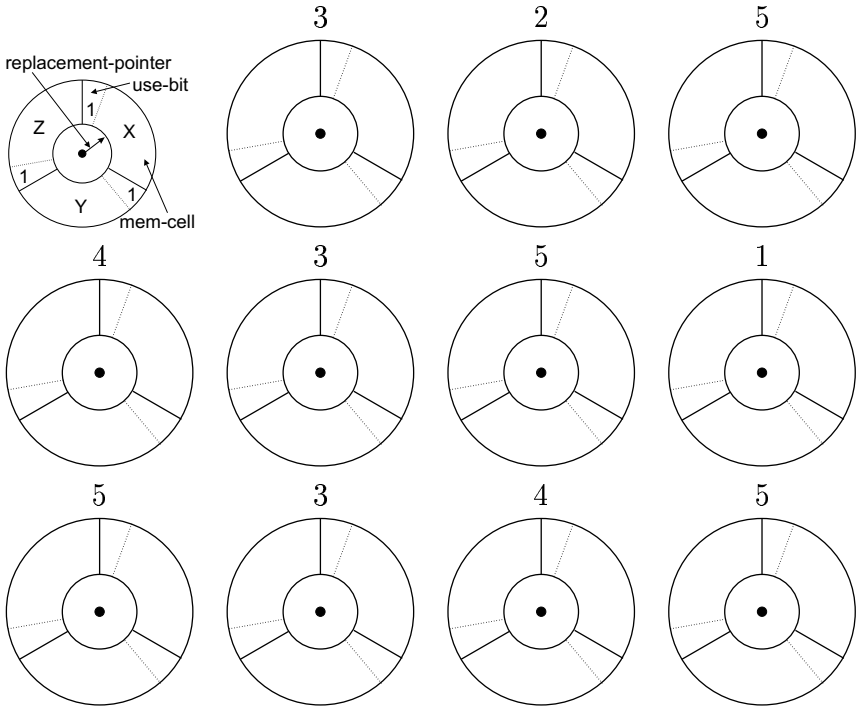
Tragen Sie in folgender Tabelle die Inhalte der Page-Frames ein, entsprechend der folgenden Replacement Policies:

- *Optimal* (OPT)
- *Least Recently Used* (LRU)
- *First-in,First-out* (FIFO)
- *Simple Clock Policy* (CLOCK)

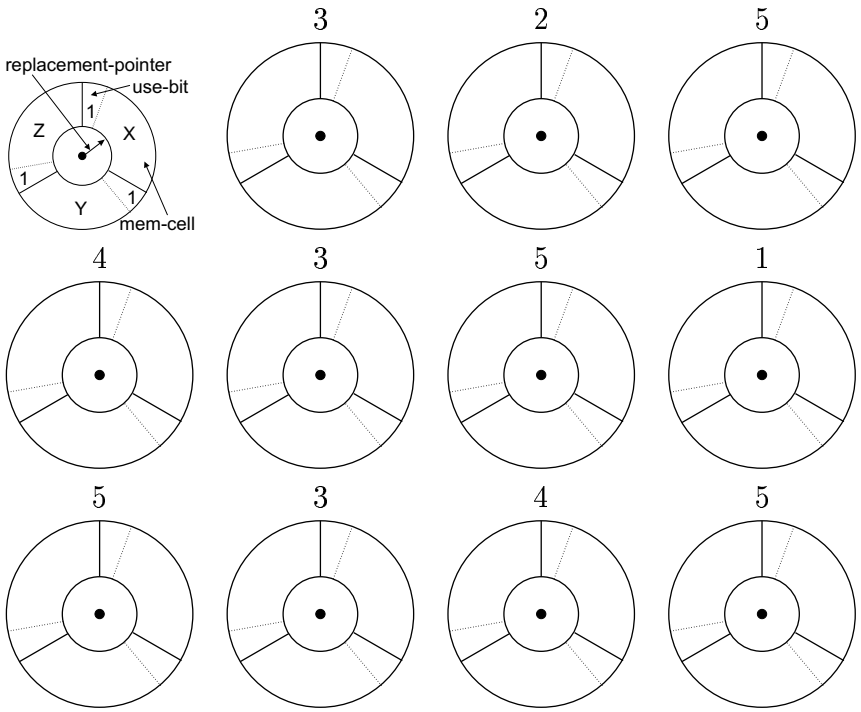
Weiters sind nachfolgende Page Faults mittels ⊗ zu markieren (für die CLOCK-Policy sind sämtliche Schritte in den Kreisdiagrammen auf der nächsten Seite einzuzeichnen - inklusive Replacement-Pointer und Usage-Bit). Sollte es gültige Alternativlösungen geben, so ist eine Variante davon auszuwählen.

	3	2	5	4	3	5	1	5	3	4	5
OPT											
	-										
	-	-									
miss	-	-	-	◯	◯	◯	◯	◯	◯	◯	◯
LRU											
	-										
	-	-									
miss	-	-	-	◯	◯	◯	◯	◯	◯	◯	◯
FIFO											
	-										
	-	-									
miss	-	-	-	◯	◯	◯	◯	◯	◯	◯	◯
CLOCK											
	-										
	-	-									
miss	-	-	-	◯	◯	◯	◯	◯	◯	◯	◯

Templates für die CLOCK-Policy (die Felder sind in der Reihenfolge *rechts - unten - links* aufzufüllen). Die Pages sind mit den Werten $\{X, Y, Z\}$ vorbelegt, welche zu den Werten $\{1, 2, 3, 4, 5\}$ paarweise disjunkt sind.



Reserve-Vorlage (die ungültige Version ist **durchzustreichen**):



Vergleich der Page-Faults

Übertragen Sie aus dem vorhergehendem Beispiel die Anzahl der Page-Faults.

Policy	Anzahl der Page-Faults
OPT	
LRU	
FIFO	
CLOCK	

Diese vier Replacement-Policies lassen sich für den allgemeinen Fall bezüglich der Effizienz mit den Bewertungen *besser/gleich/schlechter* vergleichen.

- Tragen Sie nun entsprechend den Page-Faults aus obiger Tabelle die Kriterien "<" (*besser*), "=" (*gleich*) oder ">" (*schlechter*) in die linken Teilspalten in der folgenden Tabelle ein. Die Bewertungen sind dabei jeweils vom Zeilennamen ausgehend zum Spaltennamen zu lesen (Beispiel: sollte OPT schlechter als LRU sein, tragen Sie dafür ">" im entsprechenden Feld ein).

- Markieren Sie mittels ⊗ in folgender Tabelle jene Paare von Policies, für welche die aus den **konkreten Zugriffen des obigen Beispieles** gewonnenen Aussage *besser/gleich/schlechter* **auch** für den **allgemeinen Fall** mit beliebigen Page-Zugriffen gilt.

	LRU		FIFO		CLOCK	
OPT		◯		◯		◯
LRU	-			◯		◯
FIFO	-		-			◯

3 Filemanagement (25)

Files bestehen aus mehreren Gruppen von sequentiell aufeinanderfolgenden Blöcken. In der Tabelle 1 finden Sie die Namen von fünf Files und die Längen von bis zu vier Gruppen von sequentiellen Blöcken pro File.

In Abbildung 1 ist ein Speichermedium schematisch als Matrix von Blöcken dargestellt. Die einzelnen Blöcke sind nummeriert von 0, links oben, bis 99, rechts unten.

Tragen Sie in Abbildung 1 die Files aus der Tabelle 1 ein. Die Files sollen nach der Strategie *Indexed allocation mit variabler Länge* im Speicher abgelegt werden. Markieren Sie dazu alle benötigten Blöcke (für File$X$ tragen Sie bitte X in das Rechteck ein (X={A...E}), und zeichnen Sie für jeden Beginn einer Blocksequenz einen Pfeil vom dazugehörigen Index-Block zum Anfangsblock der Blocksequenz. Tragen Sie die Addressen jedes Index-Blocks in die Tabelle 1 ein.

Die Files werden nacheinander abgearbeitet, wobei bei jedem Schritt EINE Gruppe von Blöcken der in Tabelle 1 angegebenen Länge an das entsprechende File angehängt wird. Bei Länge 0 wird nichts angehängt.

Es soll eine möglichst große Sequenz von freien Blöcken am Ende des Speichers übrigbleiben.

Beschreiben Sie den Algorithmus den Sie zum Lösen dieses Beispieles angewandt haben in Pseudo-Code.

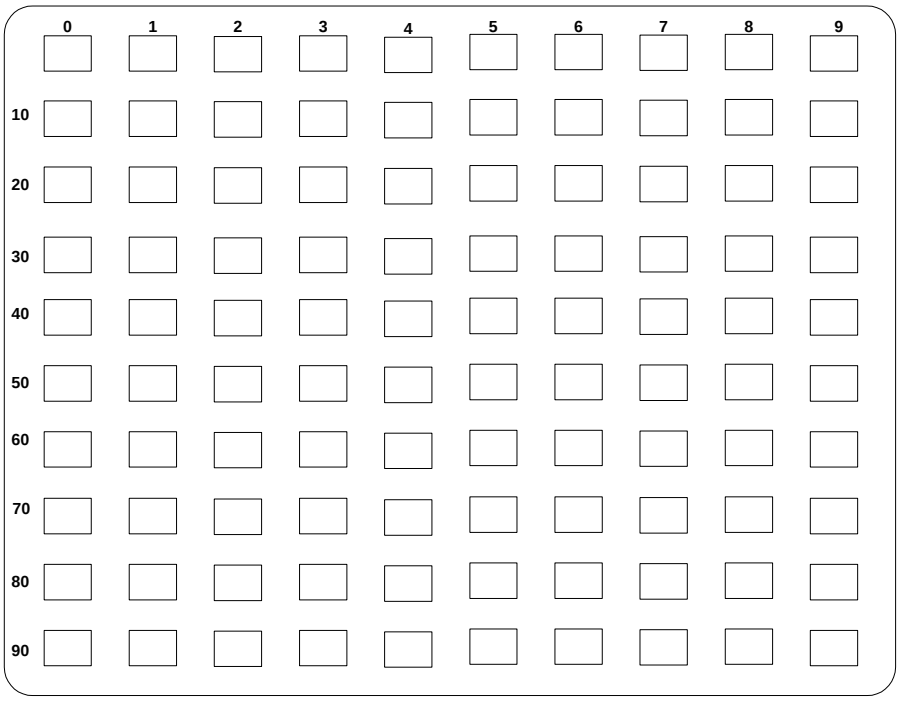


Abbildung 1: Speicher

File Name	Index Block	Durchlauf 1 Length 1	Durchlauf 2 Length 2	Durchlauf 3 Length 3	Durchlauf 4 Length 4
FileA		10	2	10	0
FileB		2	5	2	5
FileC		4	0	0	0
FileD		8	4	0	0
FileE		7	3	0	0

Tabelle 1: File Allocation Table

Pseudo-Code des Algorithmus:

4 Scheduling (20)

Real-Time Scheduling

Klassifizieren Sie die Scheduling-Strategien Rate Monotonic Scheduling (RMS) und Earliest Deadline First (EDF) anhand der folgenden Eigenschaften:

- "preemptive" versus "non-preemptive" Scheduling-Verfahren
- zentrale (centralized) versus verteilte (distributed) Scheduling-Entscheidung (Entscheidung der Abarbeitung von Prozessen)
- dynamisches (= Task Set zur Laufzeit veränderbar) versus statisches (= fixiertes Task Set; Abarbeitung der Tasks wird off-line entschieden) Scheduling
- dynamische versus statische Zuweisung von Prioritäten

Bitte klassifizieren Sie EDF und RMS durch Eintragen von Kreuzen (Kreuze haben die Bedeutung "diese Eigenschaft trifft zu"):

Eigenschaft	**RMS**	**EDF**
preemptive		
nonpremtive		
zentrale Entscheidung		
verteilte Entscheidung		
dynamisches Scheduling		
statisches Scheduling		
dynamische Prioritätenzuweisung		
statische Prioritätenzuweisung		

Kreuzen Sie an, ob die folgenden Aussagen korrekt oder inkorrekt sind:

Scheduling nach dem Round-Roubin Prinzip wird bei Verwendung sehr großer Zeitquanten zu Scheduling nach dem First-In-First-Out Prinzip.

◯ korrekt ◯ inkorrekt

Eine Scheduling-Strategie für eine CPU ist "preemptive", wenn das Betriebssystem einen Prozess *nicht* von der CPU suspendieren kann.

◯ korrekt ◯ inkorrekt

Echtzeitsysteme benutzen generell preemptive CPU Scheduling Verfahren.

◯ korrekt ◯ inkorrekt

Die Antwortenzeiten von Systemen, die "preemptive" Scheduling-Strategien benutzen, sind besser voraussagbar (=Angaben sind zuverlässiger), als die Antwortzeiten von Systemen, die kooperatives (= "non-preemptive") Scheduling benutzen.

◯ korrekt ◯ inkorrekt

Generell kann man sagen, dass rechenintensive Prozesse bei Scheduling nach dem Round-Robin-Prinzip durchschnittlich schneller abgearbeitet werden als I/O-intensive Prozesse.

◯ korrekt ◯ inkorrekt

Scheduling-Strategien, die Prioritäten statisch zuweisen, sind leichter zu implementieren als Scheduling-Strategien, die Prioritäten dynamisch zuweisen.

◯ korrekt ◯ inkorrekt

Scheduling-Strategien, die Prioritäten statisch zuweisen, verlangen mehr Laufzeit-Overhead als Scheduling-Strategien, die Prioritäten dynamisch zuweisen.

◯ korrekt ◯ inkorrekt

Scheduling nach dem Multi-level Feedback Queue Prinzip bevorzugt Prozesse mit langen Ausführungszeiten.

◯ korrekt ◯ inkorrekt

Scheduling nach dem Multi-level Feedback Queue Prinzip bevorzugt I/O-intensive Prozesse, um gute I/O-Geräte-Auslastung zu erreichen.

◯ korrekt ◯ inkorrekt

Deadline Scheduling verlangt einen intensiven Ressourcen-Einsatz; deshalb ist der Overhead im Vergleich zu anderen Scheduling Strategien groß.

◯ korrekt ◯ inkorrekt

Scheduling nach dem Round-Robin-Prinzip unter Verwendung von langen Zeitquanten bevorzugt I/O-intensive Prozesse gegenüber rechenzeitintensiven Prozessen.

◯ korrekt ◯ inkorrekt

Eine Deadline-Verletzung eines "Soft Real-Time" Prozesses hat fatale Folgen; deshalb muss eine solche Deadline von einem Scheduling-Algorithmus auf jeden Fall eingehalten werden.

◯ korrekt ◯ inkorrekt