VO 182.022 28. Jänner 2004

Prüfung Betriebssysteme

KNr. MNr. Zuname, Vorname

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

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Speicherverwaltung (30)

a) Lokalität von Prozessen – Working Sets (8)

Das Working Set Modell beschreibt das Speicherzugriffsverhalten von Prozessen. In diesem Modell ist das Working Set $W(t, \Delta)$ definiert als die Menge der Speicherseiten, die in den letzten Δ Zeiteinheiten vor dem Zeitpunkt t referenziert wurden.

Die folgende Tabelle beschreibt eine Folge von Speicherseitenzugriffen eines Prozesses. In Spalte 2 jeder Zeile steht die Nummer der Seite, auf die in der Zeitscheibe $(t-1, t]$ zugegriffen wird. Geben Sie in den leeren Feldern die Working Sets $W(t, \Delta)$ für Δ gleich 3, bzw. 4 an.

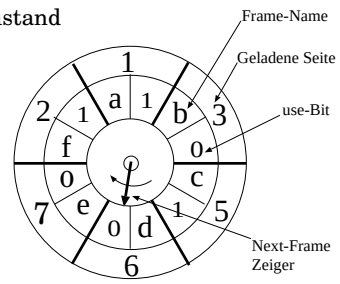
t	*Seitennr.*	$W(t,3)$	$W(t,4)$
1	0x23		
2	0x18		
3	0x19		
4	0x18		
5	0x23		
6	0x19		
7	0x18		
8	0x3F		
9	0x23		
10	0x18		

b) Replacement Policy (11)

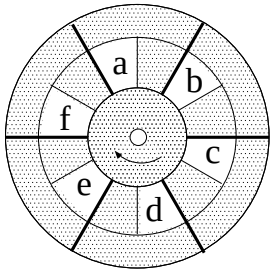
Simulieren Sie das Verhalten einer *clock policy* zum Auslagern von Speicher-Seiten, wenn neue Seiten geladen werden müssen.

Für jede Seite im Speicher existiert ein *use*-Bit. Die einzelnen Plätze im Hauptspeicher sind in diesem Beispiel mit Buchstaben von `a` ... `f` benannt. Der *next frame*-Zeiger bewegt sich in den verwendeten Grafiken im Uhrzeigersinn weiter.

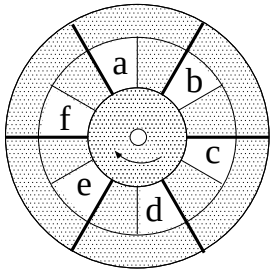
Anfangszustand



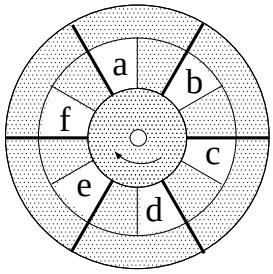
1.Zugriff:
`access[1]`



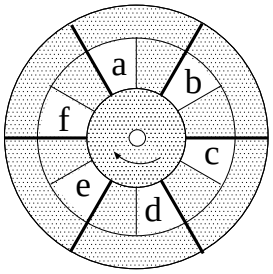
2.Zugriff:
`access[8]`



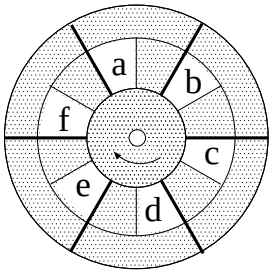
3.Zugriff:
`access[4]`



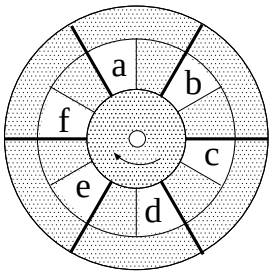
4.Zugriff:
`access[2]`



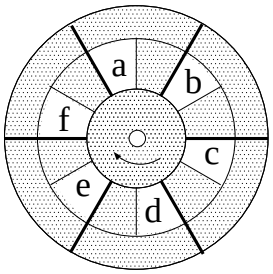
5.Zugriff:
`access[7]`



6.Zugriff:
`access[3]`



7.Zugriff:
`access[1]`



Zeichnen Sie nun in den obigen Grafiken

- die Position des *next frame*-Zeigers
- die Inhalte aller Frames (geladene Seiten)

für die Ausführung von Befehlen mit Speicherzugriffen ein. Ein Befehl `access[x]` steht für einen Speicherzugriff auf die Seite `x`. Geben Sie den gefragten Speicherzustand **nach** der Ausführung des jeweiligen Befehls an, wobei Sie für die Befehle von einer sequentiellen Reihenfolge ausgehen können.

c) Verständnisfragen (11)

Kreuzen Sie bei den folgenden Aussagen an, ob diese richtig oder falsch sind.

- ◯ *richtig* ◯ *falsch* — Durch Vergrößerung des *Working Set*s von Prozessen kann man das Risiko von "thrashing" vermindern.

- ◯ *richtig* ◯ *falsch* — Die Clock Replacement Strategie liefert im Durchschnitt weniger Page Faults als die Least Recently Used Replacement Strategy.

- ◯ *richtig* ◯ *falsch* — Große Seitengrößen bei reinem Paging führen zu kleinen Seitentabellen (page tables).

- ◯ *richtig* ◯ *falsch* — Um die externe Fragmentierung zu reduzieren, muss man die *Page Size* vergrößern.

- ◯ *richtig* ◯ *falsch* — Unter "prepaging" versteht man einen Mechanismus, bei dem *nur* Pages, die in Zukunft referenziert werden, (und sonst keine) und damit benötigte "Pages" (Seiten) vor ihrer Verwendung in den Hauptspeicher gebracht werden.

- ◯ *richtig* ◯ *falsch* — Beim FIFO-Verfahren kann eine Vergrößerung der Anzahl der Arbeitsspeicherseiten zu einer Zunahme der Seitenfehler führen.

- ◯ *richtig* ◯ *falsch* — Eine virtuelle Adresse verweist immer auf eine Seite auf dem Sekundärspeicher.

- ◯ *richtig* ◯ *falsch* — Die Free Frame List verwaltet bei der Segmentierung die freien Frames im Speicher.

- ◯ *richtig* ◯ *falsch* — Bei der Clock Policy handelt es sich um eine Demand Paging Strategie.

- ◯ *richtig* ◯ *falsch* — Die LRU-Strategie wird in der Praxis wenig verwendet, da sie gegenüber den anderen Ansätzen eine hohe Seitenfehlerrate produziert.

- ◯ *richtig* ◯ *falsch* — Für einen gleich großen Hauptspeicher nutzt eine kleine Seitengröße das Lokalitätsprinzip besser aus als eine große Seitengröße.

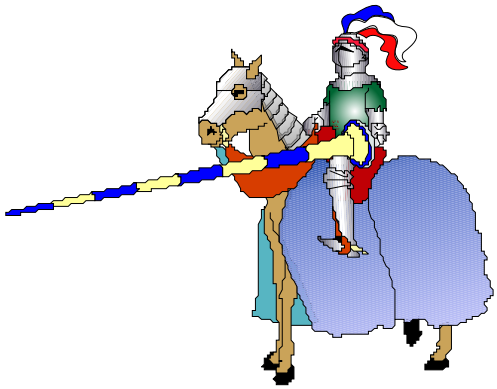
- ◯ *richtig* ◯ *falsch* — Ein virtueller Speicher kann sowohl mit Paging als auch mit Segmentierung implementiert werden.

- ◯ *richtig* ◯ *falsch* — Beim Buddy System kann es zu externer Fragmentierung kommen.

- ◯ *richtig* ◯ *falsch* — Beim Buddy System kann es zu interner Fragmentierung kommen.

2 Synchronisation (25)

2.1 Semaphore (20)



Wir schreiben das Jahr 867. Ritter Kunibert ist ein angesehener und gefürchteter Turnierritter. Zum Leidwesen seiner zwei Knechte, ist Ritter Kunibert dem Vergnügen vor dem Turnier nicht abgeneigt. Und so kommt es wie es kommen muss, Ritter Kunibert ist am Morgen des Turniers noch völlig erschöpft von der Turnierparty und benötigt die Hilfe seiner Diener um sich passend einzukleiden. Da nicht mehr viel Zeit bis zu Kuniberts Auftritt bleibt ist es wichtig, dass

- Ritter Kunibert in der richtigen Reihenfolge eingekleidet und für das Turnier vorbereitet wird
- und dies unter Ausnutzung maximaler Parallelität geschieht.

Die folgende Tabelle listet die dafür notwendigen Schritte auf:

ID	Aktion	Funktion	benötigt	Akteur
A	Aufstehen	`stand_up()`	-	Kunibert
B	Stillhalten	`hold_still()`	A	Kunibert
C	Hose anziehen	`pants()`	B	Ruprecht
D	Kettenhemd anziehen	`shirt()`	B	Vladimir
E	Schuh links anziehen und zubinden	`shoe(left)`	C	Ruprecht
F	Schuh rechts anziehen und zubinden	`shoe(right)`	C	Vladimir
G	Beinschoner links befestigen	`guard(shin, left)`	E	Ruprecht
H	Beinschoner rechts befestigen	`guard(shin, right)`	F	Vladimir
I	Armprotektor links befestigen	`guard(arm, left)`	D	Ruprecht
J	Armprotektor rechts befestigen	`guard(arm, right)`	D	Vladimir
K	Brustpanzer fixieren	`protector(front)`	G,H,I,J	Ruprecht
L	Rückenpanzer fixieren	`protector(back)`	K	Ruprecht
M	Helm aufsetzen	`helmet()`	G,H,I,J	Vladimir
N	aufs Pferd setzen	`get_on_horse()`	L, M	Kunibert
O	Schild in die Hand geben	`shield()`	N	Ruprecht
P	Lanze ausrichten	`lance()`	N	Vladimir
Q	Kämpfen	`fight_for_honor()`	O,P	Kunibert

Das Feld `ID` enthält eine Kurzbezeichnung für eine `Funktion`, welche im Feld `Aktion` beschrieben ist. Um eine solche `Funktion` ausführen zu können, müssen davor eine oder mehrere andere Funktionen fertiggestellt worden sein: die IDs dieser Funktionen sind im Feld `benötigt` angegeben. Das Feld `Akteur` gibt die ausführende Person an.

Die Synchronisation ist mittels einer minimalen Anzahl von Semaphoren zu realisieren. Die Verwendung von globalen Variablen und busy waiting ist verboten.

Initialisierung

Benutzen Sie zur Initialisierung der Semaphore die Funktion `init_sem(sem,value)`, wobei `sem` der Name des Semaphores ist und `value` der entsprechende Initialisierungswert.

```
init() {
```



```
}
```

Ritter Kunibert

```
Kunibert() {
```

```
}
```

Knecht Ruprecht

```
Ruprecht() {
```

```
}
```

Knecht Vladimir

```
Vladimir() {
```



```
}
```

2.2 The Producer/Consumer Problem (5)

Welche Grundthematik behandelt das *Producer/Consumer* Problem:

3 Security (20)

a) (2)

Geben Sie mit Begründung an, ob aktive oder passive *Secutity Threats* schwieriger zu erkennen sind.

b) (3)

Was bedeutet das Prinzip *Least Priviledge*?

Nennen Sie ein Beispiel, wo dies verletzt wäre:

c) (6)

Beschreiben Sie das Prinzip des Public-Key Verschlüsselungsverfahrens, sowie die Vor- und Nachteile gegenüber symmetrischen Verschlüsselungsverfahren.

Prinzip:

Vorteile:

Nachteile:

d) (6)

Nennen Sie drei Methoden eines Hackers zum Herausfinden von Passwörtern:

Nennen Sie drei potentielle Schwachstellen des Passwortes "*passwort*":

e) (3)

Was beschreibt das Modell von Bell und LaPadula?

4 Deadlock (25)

In einer Firma die sich mit Software und Web Design beschäftigt gibt es folgende Geräte: 4 Computer (C), 1 Scanner (S), 2 Drucker (D) und 3 Telefone (T).

In der Firma gibt es 4 Angestellte, einer ist zuständig für Web Design (WD), der zweite für Datenbanksoftware (DS) der dritte betreut die Kunden (KB), und der vierte ist System Administrator (*root*).

WD benötigt für seine Tätigkeit 3 Computer, 1 Scanner, 1 Drucker, und 2 Telefone. DS benötigt für seine Tätigkeit 1 Computer, 1 Drucker, und 1 Telefon. KB benötigt für seine Tätigkeit 1 Computer, 1 Scanner, 1 Drucker, und 2 Telefone. Root benötigt für seine Tätigkeit 1 Computer und 1 Telefon.

Zur Zeit belegt WD 2 Computer und 1 Telefon. DS belegt 1 Computer und 1 Telefon. KB belegt 1 Computer und 1 Drucker. Root belegt 1 Telefon.

a) (5)

Tragen sie den Resource-Vektor, die Claim Matrix und die Allocation-Matrix ein. Berechnen Sie auch den Availability-Vektor.

$$Resource = \begin{pmatrix} \texttt{C} & \\ \texttt{S} & \\ \texttt{D} & \\ \texttt{T} & \end{pmatrix} \qquad Claim = \begin{array}{c} \begin{matrix} \text{WD} & \text{DS} & \text{KB} & \text{root} \end{matrix} \\ \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix} \end{array}$$

$$Available = \begin{pmatrix} \texttt{C} & \\ \texttt{S} & \\ \texttt{D} & \\ \texttt{T} & \end{pmatrix} \qquad Allocation = \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix}$$

b) (12)

Suchen Sie jetzt eine Abarbeitungsreihenfolge für die Tätigkeiten der Angestellten, bei der alle Tätigkeiten ausgeführt werden können. Verwenden Sie dazu den *banker's algorithm* und geben Sie für *jeden* Schritt die Claim- und Allocationmatrix, sowie den Availability Vector und die als nächstes durchzuführende Tätigkeit (WD, DS, KB oder root) an. Wenn

keine Tätigkeit mehr auszuführen ist, dann schreiben sie ‘fertig’, falls ein Deadlock auftritt, schreiben Sie ‘Deadlock’ in den nächsten Schritt.

$$Claim = \begin{pmatrix} \mathtt{C} & & & & \\ \mathtt{S} & & & & \\ \mathtt{D} & & & & \\ \mathtt{T} & & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \\ \\ \end{pmatrix}$$

$$Claim = \begin{pmatrix} \mathtt{C} & & & & \\ \mathtt{S} & & & & \\ \mathtt{D} & & & & \\ \mathtt{T} & & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \\ \\ \end{pmatrix}$$

$$Claim = \begin{pmatrix} \mathtt{C} & & & & \\ \mathtt{S} & & & & \\ \mathtt{D} & & & & \\ \mathtt{T} & & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \\ \\ \end{pmatrix}$$

$$Claim = \begin{pmatrix} \mathtt{C} & & & & \\ \mathtt{S} & & & & \\ \mathtt{D} & & & & \\ \mathtt{T} & & & & \end{pmatrix} \quad Alloc = \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \\ \\ \\ \end{pmatrix}$$

$$Claim = \begin{pmatrix} \texttt{C} \\ \texttt{S} \\ \texttt{D} \\ \texttt{T} \end{pmatrix} \quad Alloc = \begin{pmatrix} \\ \\ \\ \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \end{pmatrix}$$

$$Claim = \begin{pmatrix} \texttt{C} \\ \texttt{S} \\ \texttt{D} \\ \texttt{T} \end{pmatrix} \quad Alloc = \begin{pmatrix} \\ \\ \\ \end{pmatrix} \quad Avail = \begin{pmatrix} \\ \end{pmatrix}$$

c) (3)

Was ist Deadlock?

d) (5)

Welche 4 Bedienungen müssen erfüllt werden dass es zu einen Deadlock kommt?