Vorlesungsprüfung aus ◯ Betriebssysteme ◯ Systemprogrammierung 2001-01-17

KNr. MNr. Zuname, Vorname

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

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

1 Deadlock (25)

Gegeben sind zwei Prozesse P_1, P_2 und die Ressourcen R_1, R_2 und R_3. Benötigt ein Prozess eine vom anderen Prozess belegte Ressource, so wird er auf jeden Fall bis zum Freiwerden der Ressource verzögert. Zu Beginn sind alle Ressourcen verfügbar.

Das Diagramm in Abbildung 1 zeigt die Ressource-Anforderungen der beiden Prozesse. Die Anforderungen von Prozess P_1 sind entlang der x-Achse, die Anforderungen von Prozess P_2 entlang der y-Achse aufgetragen.

Der Fortschritt von P_1 und P_2 bei der (quasi)parallelen Abarbeitung kann als Kantenzug zwischen den Punkten *start* und *end* in der Grafik eingetragen werden (siehe Buch zur Vorlesung: W. Stallings, Operating Systems).

1. Umranden und schraffieren Sie in der Grafik jene Bereiche, durch die ein solcher Kantenzug aufgrund von Ressourcenkonflikten nicht gehen kann.

2. Kennzeichnen Sie auf unterschiedliche Weise die Bereiche, die von einem Kantenzug nicht passiert werden dürfen, wenn eine Abarbeitung von P_1 und P_2 deadlockfrei erfolgen soll. Beschriften Sie diese Bereiche deutlich mit einem "D".

3. Zeichnen Sie einen Kantenzug für eine gültige, deadlockfreie Abarbeitung von P_1 und P_2 in der Grafik ein. Dieser Kantenzug soll durch möglichst viele Punkte (S_1-S_6) führen.

Entscheiden Sie für jeden der 6 Punkte (S_1-S_6) im Diagramm, ob der Punkt erreicht werden kann, oder nicht und kreuzen Sie dementsprechend in der untenstehenden Tabelle an.

Punkt	erreichbar	nicht erreichbar
S_1	◯	◯
S_2	◯	◯
S_3	◯	◯
S_4	◯	◯
S_5	◯	◯
S_6	◯	◯

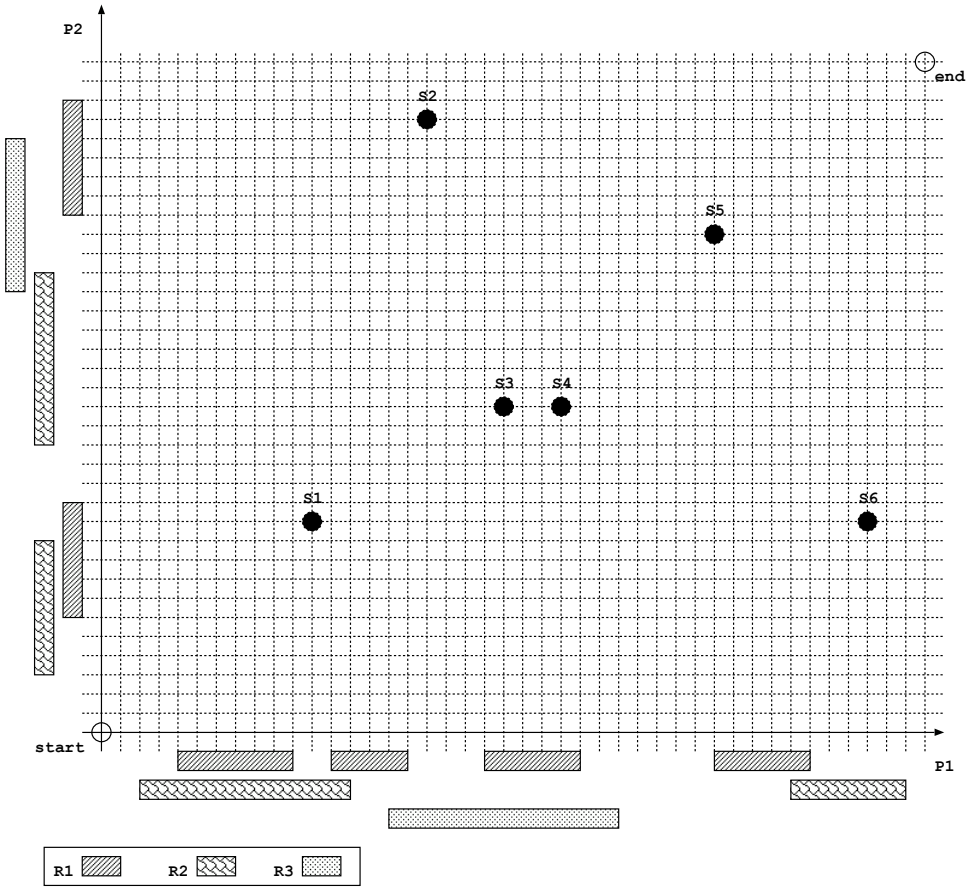

Abbildung 1: Abarbeitungs Diagramm

- Beschreiben Sie den Unterschied zwischen Process Initiation Denial und Resource Allocation Denial:

- Was ist Starvation:

2 Synchronisation (30)

Zum 5. Dezember wollen Sysprog Studenten ein "Krampuskränzchen" feiern. Dazu beabsichtigen sie, Maroni zu braten sowie Glühwein zu kochen. Folgende Randbedingungen müssen berücksichtigt werden.

- Es gibt 5 Kochtöpfe (à 70 Portionen) ...
- sowie 3 Backbleche (à 120 Portionen).
- Der Ofen hat 3 Herdplatten und 1 Backrohr.
- Im Backrohr hat genau ein Backblech Platz.
- Wenn das Backrohr in Betrieb ist, darf nur 1 Herdplatte verwendet werden. Wird das Backrohr nicht verwendet, können alle 3 Herdplatten verwendet werden.
- Zu Beginn des "Krampuskränzchens" sind alle Backbleche und Töpfe leer.
- Erst wenn ein Topf bzw. Blech vollständig geleert ist, wird vom Nächsten entnommen.
- Die Gäste geben ein Backblech bzw. einen Topf nur vollständig entleert wieder heraus.
- Die Lösung soll ein hohes Maß an Parallelität besitzen und dabei keine globalen Variablen verwenden.

Es stehen Ihnen die folgenden Routinen für die Implementierung der drei Programme `hungriger_Student`, `Maroni_Brater` und `Gluehwein_Koch` zur Verfügung:

- `SInit(`*SQ*`,`*val*`)` Initialisiert den Sequencer *SQ* mit dem Wert *val*.
- `STicket(`*SQ*`,`*val*`)` Erhöht den Sequencer *SQ* um den Wert *val* und retourniert den **neuen** Wert.
- `EInit(`*EC*`,`*val*`)` Initialisiert den Eventcounter *EC* mit dem Wert *val*.
- `EWait(`*EC*`,`*val*`)` Ein Aufruf der Routine `EWait` kehrt zurück, sobald der Wert des Eventcounters $EC \geq val$ ist.
- `EAdvance(`*EC*`,`*val*`)` Erhöht den Wert des Eventcounters *EC* um *val*.
- `HoleGluehwein()` Ein durstiger Student kann durch Aufruf dieser Prozedur an Glühwein herankommen. Die Prozedur blockiert solange bis Glühwein vorhanden ist.
- `HoleMaroni()` Mit dieser Prozedur versorgt sich ein hungriger Student mit Maroni. Auch diese Prozedur blockiert bis Maroni verfügbar sind.
- `BlechHolen()` Ein Maronibrater nimmt durch diese Prozedur ein leeres Blech und stellt es bereit für den nächsten Arbeitsschritt.
- `MaroniAufsBlech()` Richtet **120 Portionen** Maroni auf einem bereitgestellten Backblech an.

- `BlechInDenOfen()` Stellt das Backblech in den Ofen, nimmt es in Betrieb und wartet bis die Maroni fertig sind. Beim Terminieren ist das Backrohr bereits abgedreht und frei.
- `TopfHolen()` Nimmt einen Topf und stellt diesen zum Glühweinmixen bereit.
- `GluehweinMixen()` Gibt Gewürze sowie Wein & Wasser für **70 Portionen** Glühwein in einen bereitgestellen Topf.
- `TopfAufDenOfen()` Setzt einen gefüllten Topf auf eine freie Herdplatte, nimmt diese in Betrieb und terminiert wenn der Glühwein fertig ist. Beim Terminieren ist die Herdplatte bereits abgedreht und frei.

2.1 Initialisierung (5)

2.2 Hungriger Student (3)

Implementieren Sie das Programm `hungriger_Student`, das auf allen Gästen parallel abläuft.

```
while(hungrig || durstig){
  if(durstig){
```



```
    durstig = Trinken() /* trinkt den Gluehwein & aktualisiert durstig */
  }
  if(hungrig){
```



```
    hungrig = Essen() /* Iszt die Maroni & aktualisiert hungrig */
  }
}
Heimtorkeln()
```

2.3 Maronibrater (11)

Implementieren Sie das Programm `Maroni_Brater` das auf allen Maronibratern parallel exekutiert wird.

```
while(fest == IM_GANGE){
```



```
  GebtDenHungerden()
}
Aufraeumen()
```

2.4 Glühweinkoch (11)

Implementieren Sie das Programm `Gluehwein_Koch` das auf allen Glühweinköchen parallel abgearbeitet wird.

```
while(fest == IM_GANGE){
```



```
  GebtDenDurstigen()
}
Aufraeumen()
```

3 Speicherverwaltung (25)

a) Translation Lookaside Buffer (12)

Das gegebene Speicherverwaltungssystem basiert auf der Paging-Technik und verwendet einen *Translation Lookaside Buffer (TLB)* mit drei Einträgen. Der TLB wird mit der **FIFO** Strategie verwaltet, d.h., wenn für eine neu einzutragende Seite kein Eintrag mehr frei ist, wird jener Eintrag ersetzt, der sich am längsten im TLB befindet. Zu diesem Zweck enthält der TLB für jeden Eintrag ein Feld (mit der Bezeichnung FIFO), das angibt, wie lange sich der entsprechende Eintrag in dem TLB befindet. Bitte beachten Sie, dass der niedrigste Wert dem am längsten im TLB befindlichen Eintrag zugeordnet ist. Der Zähler eines *neuen* Eintrags ist das um eins inkrementierte Maximum der bisherigen FIFO-Counters.

Der TLB und die benötigte Page Table werden mittels **Associative Mapping** angesprochen.

Eine virtuelle Speicheradresse hat folgendes Format:

Page#(8 Bit)	Offset (16 Bit)

Eine physikalische Speicheradresse hat folgendes Format:

Frame#(12 Bit)	Offset (16 Bit)

Die Speicheradressen, Frame- und Pagenummern sind im Hexadezimalsystem angegeben.

Auf der nächsten Seite ist ein Translation Lookaside Buffer und eine Page Table vorgegeben. Simulieren Sie ausgehend von diesen Daten den hintereinanderfolgenden Zugriff auf die virtuellen Speicheradressen auf den folgenden Seiten. Bitte beachten Sie dabei:

- Befindet sich eine Seite nicht im Hauptspeicher, so können Sie die Nummer des Page-Frames für die Page aus den nicht belegten frei wählen. Tragen Sie diese in der Page Table auf der nächsten Seite ein. `frei wählen`
- Bestimmen Sie, ob es zu einem TLB Hit oder Miss kommt und ob es zu einem Main Memory Page Hit kommt oder nicht (Kreuzen Sie entsprechend **Ja** oder **Nein** an). (Hinweis: ein TLB Hit impliziert einen Page Table Hit)
- Geben Sie weiters jeweils den Inhalt des TLB **nach** dem Zugriff auf die Page an.

Ausgangssituation

Translation Lookaside Buffer (TLB):

Page#	Frame#	FIFO
22	0xFAC	2
3	0x723	1
12	0x333	3

3 TLB hit - da die Page 22 im TLB ist

Page Table:

Page#	Frame#
0	0x2FF
1	0x341
2	0x121
3	0x723
...	...
12	0x333
...	...
22	0xFAC
23	
24	
25	
...	...
37	
38	
39	
...	...
6A	0xA1C
6B	0xAFF

4 in der Page Table wird nicht nachgeschaut da sich die Page schon im TLB befindet

Page# 1

Virtuelle Adresse

offset 2

0x223ABC

Physikalische Adresse

0xFAC3ABC

6

5

TLB Hit ⊗ Ja ○ Nein

Page Hit ○ Ja ⊗ Nein

Frame# 0xFAC den wir im TLB gefunden haben (bei der Page# 22)

Offset aus der virtuellen Adresse

Translation Lookaside Buffer

Page#	Frame#	FIFO
22	0xFAC	3
3	0x723	1
12	0x333	3

um ein 1 erhöht da TLB hit

7

Virtuelle Adresse

0x3921C3

Physikalische Adresse

0x11121C3

1

TLB Hit ○ Ja ⊗ Nein

Page Hit ○ Ja ⊗ Nein

Page# 0x39 befindet sich nicht im TLB und auch in der Page Table ist unter 39 nichts zu finden. Also müssen wir nachladen

Translation Lookaside Buffer

Page#	Frame#	FIFO
22	0xFAC	3
39	0x111	1
12	0x333	3

2

wir ersetzen die Page# 0x3 da diese am lägsten nicht gebraucht wurde. An ihrer Stelle kommt kommt die Page# 0x39 und ihr FIFO Wert wird auf 1 gesetzt.

Den Frame# kann man frei wählen (siehe oben). Ich hab 0x111 genommen. Man kann auch 0x123, 0xF11, ... nehmen Hauptsache der Frame hat 12bit

Virtuelle Adresse

0x031274

Physikalische Adresse

0x7231274

TLB Hit ○ Ja ⊗ Nein

Page Hit ⊗ Ja ○ Nein

Translation Lookaside Buffer

Page#	Frame#	FIFO
22	0xFAC	3
3	0x723	1
12	0x333	3

den Rest kann man dan selber lösen

Virtuelle Adresse

0x121100

Physikalische Adresse

TLB Hit ○ Ja ○ Nein

Page Hit ○ Ja ○ Nein

Translation Lookaside Buffer

Page#	Frame#	FIFO

Virtuelle Adresse

0x6A001B

Physikalische Adresse

TLB Hit ◯ Ja ◯ Nein
Page Hit ◯ Ja ◯ Nein

Translation Lookaside Buffer

Page#	Frame#	FIFO

Virtuelle Adresse

0x245677

Physikalische Adresse

TLB Hit ◯ Ja ◯ Nein
Page Hit ◯ Ja ◯ Nein

Translation Lookaside Buffer

Page#	Frame#	FIFO

b) Page-Replacement Algorithms(9)

Ein Prozess referenziert die Pages A,B,C,D und E laut vorgegebenen Zugriffsschema. Simulieren Sie das Verhalten der entsprechenden Page-Replacement Algorithmen in der dafür vorgesehenen Tabelle. Am Anfang ist der TLB leer. Bei mehrdeutigen Lösungsmöglichkeiten geben Sie bitte eine gültige an. Markieren Sie einen Page Fault mit $\surd$ (letzte Zeile).

OPTIMAL

	A	B	A	D	E	B	A	C	A	D	A	C
page fault?												

Anzahl d. page faults:

LRU

	A	B	A	D	E	B	A	C	A	D	A	C
page fault?												

Anzahl d. page faults:

FIFO

	A	B	A	D	E	B	A	C	A	D	A	C
page fault?												

Anzahl d. page faults:

c) Verständnisfragen (4)

- Welche dieser Replacement-Policies ist am einfachsten zu implementieren?
  - ◯ Least recently used
  - ◯ Clock algorithmus
  - ◯ First in - first out
- Um die interne Fragmentierung zu reduzieren, muss man die *Page Size*
  - ◯ verringern
  - ◯ vergrößern
- Wieviele Zugriffe auf die Pagetable benötigt ein System mit Translation Lookaside Buffer im Falle eines TLB Hits, um die virtuelle Adresse aufzulösen?

  Zugriff(e)
- Im Vergleich zu den anderen Replacement-Policies produziert die *optimal policy* bei beschränkter Buffergröße
  - ◯ keine page faults
  - ◯ die wenigsten page faults
  - ◯ die meisten page faults

4 Security (20)

a) (3)

Was beschreibt das Modell von Bell und LaPadula?

b) (4)

Im Modell von Bell und LaPadula gibt es die *Simple Security Property* und die $\star$*-Property*. Beschreiben Sie für jeden der beiden Begriffe was er besagt und was das damit verbundene Konzept bewirken soll.

c) (5)

Beschreiben Sie die Funktionsweise von *Public Key Verschlüsselungsverfahren*. Erklären Sie insbesonders, welche Schüssel man bei diesem Verfahren benötigt, wer welche Schlüssel kennen darf und wie die Schlüssel von den Sendern und Empfängern von Daten verwendet werden.

d) (8)

Geben Sie die Schritte an, die zum Senden und beim Empfangen einer mit einem Public Key Verfahren verschlüsselten und signierten Nachricht notwendig sind. Welche Schlüssel braucht man dabei?