Lehrstuhl für Informatik 6
der RWTH Aachen
Prof. Dr.-Ing. H. Ney

Aachen, den 11. April 2011

S. Peitz, M. Freitag, J. Bitsch

Betriebssysteme und Systemsoftware

0. Übung

Aufgabe 0.1 Linux- und Shell-Playground (0 Punkte)

Wir fangen mit ein paar Übungen zum Warmwerden an, um uns ein bißchen mit Linux und der Shell vertraut zu machen.

a) Installiere Linux
Es gibt zwar CygWin für Windows, wie in der Vorlesung erwähnt wurde, aber warum nicht mal ein richtiges Linux ausprobieren? Also einfach eine Linux-Distribution herrunterladen und installieren. Wer sich einen Überblick über verschiedene Distributionen verschaffen will, sollte hier nach schauen: `http://distrowatch.com/dwres.php?resource=major`

Wer lieber bei Windows bleiben und/oder nicht gleich einen Dualboot oder Ähnliches anlegen will, kann eine vollständige Linux-Distribution auch einfach virtuell z.B. innerhalb von VirtualBox installieren: `http://www.virtualbox.org/`

b) RTFM - Read The 'Friendly' Manual!!!
Wer Probleme mit den Shell-Aufgaben hat, kann natürlich Google oder Ähnliches zur Hilfe heranziehen - es geht aber auch professioneller. Öffnet dazu eine Shell und gebt die folgenden Befehle ein:

man man
man grep
...

Hinweis: Lies die Manuals!!!

Aufgabe 0.2 Einfache Shell Commands (0 Punkte)

Was machen folgende Befehle bzw. was ist deren Ausgabe?

a) echo Hallo Welt | sed 's/l//g' | wc

b) name="BUS 2011"; echo $name | awk ' { print $2*5 } '

c) echo Das Haus > datei1.txt; echo am See > datei2.txt; paste -d '_' datei1.txt datei2.txt

d) man man | grep manual | head -n2 | sed 's/$/.../g' | wc

e) Was bewirkt Parameter -r bei remove?

f) Was bewirkt Parameter -d bei paste?

g) Inhalt des aktuellen Ordners: datei1 datei2 datei3 ordner1
find . -name 'd*' | xargs rm
Was passiert hier?
Achtung probiert diese Aufgabe nicht aus, sondern lest dazu nur die Manuals!!

Aufgabe 0.3 Textverarbeitung in UNIX (0 Punkte)

UNIX bietet eine Vielzahl kleiner Utility-Programme an, die zur Manipulation von Textfiles geeignet sind (z.B. `awk`, `grep`, `cut`, `paste`, `sort`, `uniq`, `diff`, `tr`, `wc`, `cat`, `...`). Viele Aufgaben, zu denen man bei anderen Betriebssystemen komplette Programme schreiben müsste, lassen sich unter UNIX durch geschickte Verknüpfung dieser Tools über Pipes direkt auf der Kommandozeile einer Shell lösen.

Löst die Aufgaben unter Verwendung der genannten UNIX–Tools. Gebt neben der Lösung die verwendete Befehlsfolge an.

a) Im L^2P findet ihr einen Ausschnitt aus 'The War of the Worlds ' (`wotw.txt`):
   - Ladet das File herunter
   - Entfernt die Zeichen `" ? . ! : ; , + & ‘`
   - Ersetzt die Zeichen Apostroph `’` und Bindestrich `-` durch Leerzeichen
   - Entfernt alle Leerzeilen

b) Wieviele Zeilen und Wörter hat der Text nun? Wieviele Zeilen enthalten die Zeichenfolge “the” nicht (dabei nicht nach Groß- und Kleinschreibung unterscheiden)?

Aufgabe 0.4 Verteilte Versionskontrolle mit GIT (0 Punkte)

Für das verteilte Bearbeiten von Texten und Quellcode werden häufig sogenannte Verteilte Versionskontrollsysteme verwendet. Damit lassen sich Zwischenstände sichern und die Zusammenarbeit zwischen verschiedenen Autoren besser abstimmen.

Ein Beispiel für ein solches Werkzeug, das z.B. für die Entwicklung des Linuxkerns verwendet wird ist GIT. Eine gute Einführung findet ihr unter: `http://gitref.org/`

Löst folgende Aufgaben unter Verwendung von GIT. Gebt die verwendete Befehlsfolge an.

a) Legt ein Repository für eure Übung an. (init)

b) Nachdem ihr verschiedene Dateien bearbeitet haben, speichert diese und registriert sie mit dem Versionsverwaltungssystem. (add/commit)

c) Schaut sich die Unterschiede zu vorherigen Versionen an (diff/log). Wie finden ihr heraus wer Zeile 4 zuletzt editiert hat? (annotate)

d) Ihr arbeitet im Team. Wie finden die Änderungen der Anderen ihren Weg zu euch? (push/pull/merge)

Lehrstuhl für Informatik 6
der RWTH Aachen
Prof. Dr.-Ing. H. Ney

Aachen, den 18. April 2011

S. Peitz, M. Freitag, J. Bitsch

Betriebssysteme und Systemsoftware

1. Übung

Abgabe der Lösungen: So., den 24. April 2011 im L2P.

Aufgabe 1.1 Bash (2+2 = 4 Punkte)

a) Erstellt ein Shell Skript, dass als Eingabe 2 Zahlen bekommt z.B. ./myScript.sh 2 8 und als Ausgabe eine Zahlen Reihenfolge ausgibt, die alle Zahlen dazwischen beinhaltet (hier z.B. 3,4,5,6,7). Beachtet auch den Fall der Eingabe ./myScript 8 2 mit der selben Ausgabe.

b) Nehmt den Text wotw.txt (Übung 0) und löst folgende Aufgaben nur mit Hilfe der Shell:
Gebt sowohl die Ausgabe als auch euren Lösungsweg an.

- Wie viele Zeilen im Text wotw.txt fangen mit dem Wort The an?
- Wie viele Zeilen fangen mit verschiedenen Wörtern an?
- Verändert den Text nun so, dass alle Buchstaben klein geschrieben sind. Wie viel verschiedene Satzanfänge gibt es nun?
- Ersetzt nun alle Vorkommen des Buchstaben t durch s, wie viele verschiedene Zeilenanfänge gibt es nun?

  Hinweis: Benutzt Google/Manpages (tr,sort,uniq,grep,awk ...)

Aufgabe 1.2 Einfuehrung in C - Typenkonvertierung (1 Punkt)

In der Einführung in C habt ihr die verschiedenen Zahlentypen mit unterschiedlichen Größen kennengelernt. Des Weiteren wird zwischen implizite und explizite Typkonvertierung unterschieden. Implizite Typkonversionen stellen eine erhebliche Fehlerquelle dar. Man sollte daher die Konvertierung explizit durch sog. *casting* anweisen:

```
int i=3;
float f=2.5;
f = (float)i;
```

Übersetzt das Programm mit dem Namen `u1_2`. Der zugehörige Quelltext `u1_2.c` ist im L^{2}P verfügbar. Überprüft die Umrechnung von Pferdestärke (PS) in kilo Watt (kW) nach der genäherten Formel

$$kW = \frac{3}{4} \cdot PS$$

und beseitigt die Fehler im Programm.

Aufgabe 1.3 Einfuehrung in C - Verbundtypen, typdef und Pointer (1+1+1+2 = 5 Punkte)

Ein sehr wichtiges Konzept in C sind sowohl die Verbundtypen als auch Pointer. Zusätzlich macht *typedef* den Programmier-Alltag angenehmer. Übersetzt auch das Programm mit dem Namen `u1_3`. Der zugehörige Quelltext `u1_3.c` ist im L^{2}P verfügbar. Gegeben ist ein Verbundtyp *car* mit der Variable *color*. Im Programm wird die Farbe von *myCar* auf *b* (blue) gesetzt. Verändert das Programm folgendermaßen:

a) Nutzt ein *typedef* für den Verbundtypen *car*.

b) Erweitert den Verbundtypen *car* mit der Variable *ps*. Welcher Typ ist hier der geeignetste?

c) Definiert einen Pointer, der auf *myCar* zeigt.

d) Verändert über den Pointer die Farbe zu rot. Außerdem hat das Auto eine Leistung von 115 PS. Gebt die Leistung mit *printf* aus.

Aufgabe 1.4 Grundlagen von Betriebssystemen (1+1+1+3 = 6 Punkte)

a) Das Betriebssystem wird benötigt, um Programme auszuführen – das Programm ist auf einem Datenträger gespeichert und muss zunächst in den Hauptspeicher geladen werden, um durch die CPU bearbeitet zu werden. Das Problem ist nun: das Betriebssystem selbst ist auch ein Programm, welches in den Hauptspeicher geladen werden muss (bzw. eine Sammlung von Programmen)! Hier haben wir das 'Henne-Ei'-Problem: wie kann das Betriebssystem geladen werden, bevor es geladen wurde?

   Erläutert knapp, wie dieses Problem in der Praxis gelöst wird!

b) Was ist ein großer Vorteil des Schichtenansatzes für das Design eines Betriebssystems? Was ist ein Nachteil dieses Schichtenansatzes?

c) Was ist der Hauptunterschied zwischen einer Unterbrechung durch einen Trap-Befehl und einem Interrupt?

d) Welche der folgenden Befehle sollten nur im Kernmodus ausgeführt werden? Begründet eure Wahl:

   – Ändern von Einträgen in den Gerätezustandstabellen
   – Lesen der Uhr
   – Ändern von Speicherzuordnungstabellen
   – Anweisen einer Trap-Instruktion
   – Abschalten von Interrupts
   – Zugriff auf I/O-Geräte

Lehrstuhl für Informatik 6
der RWTH Aachen
Prof. Dr.-Ing. H. Ney

Aachen, den 25. April 2011

S. Peitz, M. Freitag, J. Bitsch

Betriebssysteme und Systemsoftware

2. Übung

Abgabe der Lösungen: So., den 1. Mai 2011 im L2P.

Aufgabe 2.1 Einfache Scheduling-Strategien (1+1+1+2 = 5 Punkte)

In der Vorlesung habt ihr einige Scheduling-Strategien kennen gelernt (FCFS/LCFS/SJF/SRTF). In den folgenden Aufgaben geht es darum diese auf Daten anzuwenden.

In der Warteschleife (Queue) seien N Jobs mit Bearbeitungszeiten $t_1 ... t_N$. Index n gibt die Reihenfolge des Ankommens in der Queue an: d.h. Job P_n (mit Bearbeitungszeit t_n) kommt vor Job P_{n+1} (mit Bearbeitungszeit t_{n+1}) an. Folgende Jobs warten in der Queue darauf, abgearbeitet zu werden:

Job	Bearbeitungszeit (t_n)
P_1	6
P_2	2
P_3	7
P_4	3
P_5	4
P_6	6

a) Wie werden die Jobs nach FCFS abgearbeitet und wie groß ist die durchschnittliche Wartezeit?

b) Wie werden die Jobs nach LCFS abgearbeitet und wie groß ist die durchschnittliche Wartezeit?

c) Wie werden die Jobs nach SJF abgearbeitet und wie groß ist die durchschnittliche Wartezeit?

d) Nun haben wir zusätzlich zur Bearbeitungszeit noch eine Ankunftszeit. Das heißt die Jobs können erst ab dem Zeitpunkt abgearbeitet werden, ab dem sie auch angekommen sind.

Job	Ankunftszeit	Bearbeitungszeit (t_n)
P_1	00	6
P_2	01	2
P_3	02	7
P_4	03	3
P_5	04	4
P_6	11	2

Wie werden die Jobs jetzt nach SRTF (Shortest Remaining Time First) abgearbeitet und wie groß ist die durchschnittliche Wartezeit?

Hinweis: Zur Abgabe orientiert euch an den Bildern der Vorlesung (Folie 45).

Aufgabe 2.2 Round Robin (4 Punkte)

Folgende acht Prozesse seien gegeben und zum Zeitpunkt 0 in der Ready-Queue in der Reihenfolge ihrer Prozessnummern eingereiht:

Job	Bearbeitungszeit (t_n)
P_1	6
P_2	13
P_3	7
P_4	3
P_5	4
P_6	9
P_7	10
P_8	11

Der Scheduler verwendet ein Zeitquantum Q, nach dessen Ablauf zum jeweils nächsten Prozess umgeschaltet wird. Die Zeiten für diese Kontextwechsel zwischen den Prozessen werden vernachlässigt.

Schreibt ein Bash-Script oder C-Programm, das eine Tabelle erzeugt, in der für alle sinnvollen ganzzahligen Zeitquanten Q (d.h. in diesem Beispiel bis zum Wert 13) angegeben wird, wann die einzelnen Prozesse (P_i) beendet wurden und wie groß die durchschnittliche Wartezeit war.

Testet euer Programm anhand der obigen Prozesse. Die Ausgabe sollte wie im folgenden Beispiel aussehen:

```
#    Q   P1    P2    P3    P4    P5    P6    P7    P8  Avg.Time
#-------------------------------------------------------------
     01  38    63    45    20    28    54    58    61    38.00
     .                                                      .
     .                                                      .
     .                                                      .
     13  06    19    26    29    33    42    52    63    25.87
```

Aufgabe 2.3 C - SJF (2 + 1 + 2 = 5 Punkte)

In dieser Aufgabe soll die Scheduling-Strategie SJF implementiert werden. Der zugehörige Quelltext `u2_3.c` ist im L^{2}P verfügbar.
Bereits implementiert ist die Datenstruktur mit der ihr arbeiten sollt. Erweitert das Programm folgendermaßen:

a) Implementiert eine Funktion *insertJob*, die einen neuen Job in die Queue steckt. Dabei sollen die Jobs direkt nach Bearbeitungszeit (*runtime*) sortiert werden. Wichtig ist auch, dass Jobs mit gleicher Bearbeitungszeit nach Job-Nummer (*nr*) sortiert werden.

b) Fügt 100 Jobs mit zufälliger Wartezeit (zwischen 1 und 400) in Queue.
Hinweis: Nutzt eine Schleife und die Funktion *rand()*!

c) Implementiert eine Funktion *processQueue*, die alle Jobs in der Queue abarbeitet. Dabei sollen die Jobs aus der Queue gelöscht werden. Achtet darauf, dass zuvor reserviert Speicherplatz wieder frei gegeben wird! Gebt auch aus, welcher Job gerade bearbeitet wird und die durchschnittliche Wartezeit.

Aufgabe 2.4 Praktisches Prozessmanagment (2 Punkte)

Im L2P findet ihr ein kleines Beispielprogramm `u2_4.c`. Kompiliert es mit dem Befehl:

```
gcc -w u2_4.c -o u2_4
```

Wenn ihr das Programm ausführt, werdet ihr merken, dass es in einer Enslosschleife hängt.

a) Wie wird das Programm wieder abgebrochen? Tipp: `man kill`

b) Wie lassen sich die aktuell laufenden Prozesse auf dem Rechner anzeigen? Tipp:`man ps` und `man top`

c) Warum hängt das Programm in einer Endlosschleife? Tipp: Lasst das `-w` beim kompilieren weg.

Lösung 3.1 Multilevel-Feedback-Queue-Scheduling (5 Punkte)

t	Kl. 0 RR(2)	Kl. 1 RR(4)	Kl. 2 RR(8)	Kl. 3 FIFO	Incoming	Running
0	-	-	-	-	A(4),B(3)	-
1	A(4),B(3)	-	-	-	C(7)	A(4)
2	A(3),B(3)	C(7)	-	-	D(6)	A(3)
3	B(3)	C(7),A(2)	D(6)	-	E(2),F(1),G(5)	B(3)
4	B(2),E(2),F(1)	C(7),A(2)	D(6),G(5)	-	H(16)	B(2)
5	E(2),F(1)	C(7),A(2),B(1)	D(6),G(5)	H(16)	-	E(2)
6	E(1),F(1)	C(7),A(2),B(1)	D(6),G(5)	H(16)	I(4)	E(1)
7	F(1)	C(7),A(2),B(1),I(4)	D(6),G(5)	H(16)	-	F(1)
8	-	C(7),A(2),B(1),I(4)	D(6),G(5)	H(16)	-	C(7)
9	-	C(6),A(2),B(1),I(4)	D(6),G(5)	H(16)	-	C(6)
10	-	C(5),A(2),B(1),I(4)	D(6),G(5)	H(16)	-	C(5)
11	-	C(4),A(2),B(1),I(4)	D(6),G(5)	H(16)	-	C(4)
12	-	A(2),B(1),I(4)	D(6),G(5),C(3)	H(16)	-	A(2)
13	-	A(1),B(1),I(4)	D(6),G(5),C(3)	H(16)	-	A(1)
14	-	B(1),I(4)	D(6),G(5),C(3)	H(16)	-	B(1)
15	-	I(4)	D(6),G(5),C(3)	H(16)	-	I(4)
16	-	I(3)	D(6),G(5),C(3)	H(16)	-	I(3)
17	-	I(2)	D(6),G(5),C(3)	H(16)	-	I(2)
18	-	I(1)	D(6),G(5),C(3)	H(16)	-	I(1)
19	-	-	D(6),G(5),C(3)	H(16)	-	D(6)
20	-	-	D(5),G(5),C(3)	H(16)	-	D(5)
21	-	-	D(4),G(5),C(3)	H(16)	-	D(4)
22	-	-	D(3),G(5),C(3)	H(16)	-	D(3)
23	-	-	D(2),G(5),C(3)	H(16)	-	D(2)
24	-	-	D(1),G(5),C(3)	H(16)	-	D(1)
25	-	-	G(5),C(3)	H(16)	-	G(5)
26	-	-	G(4),C(3)	H(16)	-	G(4)
27	-	-	G(3),C(3)	H(16)	-	G(3)
28	-	-	G(2),C(3)	H(16)	-	G(2)
29	-	-	G(1),C(3)	H(16)	-	G(1)
30	-	-	C(3)	H(16)	-	C(3)
31	-	-	C(2)	H(16)	-	C(2)
32	-	-	C(1)	H(16)	-	C(1)
33	-	-	-	H(16)	-	H(16)
.						
.						
.						
48	-	-	-	H(1)	-	H(1)
49	-	-	-	-	-	-

Lösung 3.2 Erzeuger-Verbraucher-Problem (1 + 2 = 3 Punkte)

a)
- Messagesystem (Interprozesskommunikation)
- Festplattencache
- Tastaturpuffer

Alle drei Beispiele folgen dem gleichen Prinzip. Es gibt ein Buffer, in den der Erzeuger (Prozess 1, Festplatte, Tastatureingabe) schreibt und aus dem der Verbraucher liest (Prozess 2, Rest des Systems, Betriebsystem).

b) Konsistenzprobleme treten genau dann auf, wenn zwei verschiedene Prozesse eine Variable modifizieren. In diesem Fall haben der Erzeuger und der Verbraucher jedoch jeweils eine eigene Variable zur Steuerung des 'Lagerbestandes' (in und out), so dass es keine Konflikte gibt.

Sobald mehrere Erzeuger oder Verbraucher auftreten koennen, gibt es wieder mehrere Prozesse, die eine der gemeinsamen Variablen in oder out veraendern. Dementsprechend kann es Konsistenzprobleme geben.

Beispiel: Es ist nur noch ein Element im Lager und 2 Verbraucher testen nahezu gleichzeitig, ob das Lager leer ist. Beide koennen die Schranke ueberwinden, da noch ein Element da ist. Danach kommt es zur Kollision, weil es eben nur fuer einen Verbraucher etwas zu tun gibt.

Lösung 3.3 Wechselseitiger Ausschluss (3 + 1 = 4 Punkte)

a) (a) Das Progress Requirement wird verletzt: P_0 war in seinem kritischen Bereich, setzt `turn` auf 1, verlässt den kritischen Bereich und stirbt. P_1 kann jetzt in den kritischen Bereich, setzt `turn` auf 0, verlässt den kritischen Bereich und kann nun nie mehr zurück, da P_0 nicht mehr lebt, um `turn` auf 1 zu setzen.

t	P_0	P_1	turn
1	crit.		0
2	turn=1		1
3	remai..		1
4	stirbt	crit..	1
5	-	turn=0	0
6	-	remai.	0
7	-	while	0

(b) Mutual Exclusion verletzt: Wenn die CPU die Prozesse jeweils einmal nach der Erfüllung der Schleifen-Bedingung wechselt, setzen beide Prozesse ihre Flags auf TRUE und betreten u.U. gleichzeitig den kritischen Bereich.

t	P0	P1	flag[0]	flag[1]
1	flag[0]=FALSE;		FALSE	
2		flag[1]=FALSE;	FALSE	FALSE
3	while(flag[1])		FALSE	FALSE
4		while(flag[0])	FALSE	FALSE
5	flag[0]=TRUE;		TRUE	FALSE
6		flag[1]=TRUE;	TRUE	TRUE
7	critical..		TRUE	TRUE
8		critical..	TRUE	TRUE

(c) Progress Requirement verletzt: P_0 setzt sein Flag auf TRUE. Anschliessend findet Prozesswechsel statt und P_1 setzt sein Flag auf TRUE. Er betritt die Schleife und wartet darauf, dass P_0 sein Flag auf FALSE setzt. P_0 betritt ebenfalls seine Schleife und wartet hier darauf, dass P_1 sein Flag auf FALSE setzt $\Rightarrow$ der Deadlock ist perfekt.

t	P0	P1	flag[0]	flag[1]
1	flag[0]=FALSE		FALSE	
2		flag[1]=FALSE	FALSE	FALSE
3	flag[0]=TRUE		TRUE	FALSE
4		flag[1]=TRUE	TRUE	TRUE
5	while(flag[1])		TRUE	TRUE
6		while(flag[0])	TRUE	TRUE

b) Das Verfahren arbeitet nicht korrekt. Hier die Begründungen, von denen eine kommen kann, im einzelnen:

a) Mutual Exclusion wird nicht eingehalten:
Angabe eines möglichen Schedules, der eine Möglichkeit fÃ¼r beide Prozesse zeigt, gleichzeitig in den kritischen Bereich zu kommen.

```
//  initial: turn=i, flag[i]=FALSE, flag[j]=FALSE;
//   P_i                  P_j

                        // P_j signalisiert Wunsch
                        flag[j]=TRUE;
                        // turn == i --> Eintritt
                        while (turn!=j){
                        // flag[i] == FALSE --> kein noop
                        while (flag[i]);
```

```
// P_i signalisiert Wunsch
flag[i]=TRUE;
// turn == i --> critical section
while (turn!=i){
criticalSection(Pi);
                    // test war ok
                    turn=j;
                    // nach "au"serer while-schleife
                    criticalSection(Pj)
                    ...
...
```

Dann sind beide Prozesse in ihrem kritischen Bereich.

b) Bounded Waiting wird nicht eingehalten:

Die Bedinung verletzendes Szenario: P_i im unkritischen Bereich (UKB), P_j kommt an und betritt kritischen Bereich (KB), wo er kurz bleibt. Währenddessen kommt P_i an, findet `turn` auf `j` und `flag[j] == TRUE`. Unmittelbar darauf bleibt er stehen. P_j kommt aus KB, stellt `flag[j]` auf FALSE, durchläuft UKB, kommt wieder an, stellt `flag[j]` auf TRUE und betritt wieder KB. Jetzt findet P_i wiederum `flag[j] == TRUE`, usw. Damit kann P_i beliebig oft in der inneren Schleife überholt werden.

c) Progress Requirement wird eingehalten (gut, das nur zur Info, zählt natürlich nicht als Begründung ;-)):

Sobald ein Prozess sein Flag auf TRUE gesetzt hat, ist es für den anderen nicht mehr möglich, die `turn` Variable zu ändern. Gilt oBdA initial `turn == i`, und P_i betritt als erster die Schleife, so gelangt P_i *immer* in seinen KB, da P_j `turn` nicht mehr ändern kann. Gilt initial `turn == j`, und P_i betritt als erster die Schleife, so hat er sein Flag auf TRUE gesetzt. Entweder er kann die `turn` Variable auf `i` setzen, und gelangt danach in den KB, oder er wird von P_j überholt (siehe Teil b)), und P_j gelangt in den KB. Eine Blockade ist also nicht möglich.

Lehrstuhl für Informatik 6
der RWTH Aachen
Prof. Dr.-Ing. H. Ney

Aachen, den 9. Mai 2011

S. Peitz, M. Freitag, J. Bitsch

Betriebssysteme und Systemsoftware

4. Übung

Abgabe der Lösungen: So., den 15. Mai 2011 im L2P.

Aufgabe 4.1 Scheduling (4 + 1 + 1 = 6 Punkte)

a) Schreibt ein Schedulingverfahren in C, dass mit Quantum 2 denjenigen Prozess (Job) X als nächsten auswählt, der das Minimum der folgenden Funktion erzeugt:

$$\texttt{priority}_t(X) = \Big(0.00001 + \lambda_t \cdot \texttt{waitingtime}(X) + (1 - \lambda_t) \cdot \texttt{remainingtime}(X)\Big)^{-1}$$

Setzt den Parameter

$$\lambda_t = \frac{\sum\limits_X \texttt{remainingtime}(X)}{\sum\limits_X \texttt{runtime}(X)}.$$

Beachtet, dass λ zu unterschiedlichen Zeitpunkten unterschiedliche Werte hat. Als Basis ist der Quelltext `u4_1.c` im L^{2}P verfügbar.

b) Fügt 6 Prozesse hinzu mit Laufzeit 12, 16, 10, 3, 3, 6 und Ankunftszeit 0, 1, 5, 10, 3, 25. Gebt am Ende des Scheduling die Wartezeit für jeden Job und die durchschnittliche Wartezeit aus.

c) Gebt ein Urteil über das Schedulingverfahren: Wie vergleicht sich das Verfahren mit SJF und Multilevel Queue Scheduling?

Aufgabe 4.2 Bakery-Algorithmus (1 + 2 + 2 + 2 = 7 Punkte)

Der Bakery-Algorithmus ist als Lösung des wechselseitigen Ausschlussproblems für n Prozesse bekannt. Der Algorithmus für einen Prozess P_i ist folgendermaßen:

```
01 repeat
02     choosing[i] := true;
03     number[i] := max(number[0,...,n-1])+1;
04     choosing[i] := false;
05     for j:=0 to n-1 do {
06         while (choosing[j]) do noop;
07         while (number[j]!=0 and
08                 ((number[j]<number[i]) or
09                 (number[j]=number[i] and j<i))) do noop;
10     }
11     critical_section(Pi);
12     number[i] := 0;
13     remainder_section(Pi);
14 until false;
```

a) Wozu sind die Felder `choosing` und `number` vorhanden?

b) Zeigt: Wenn P_i im kritischen Bereich ist und für P_k ($k \neq i$) der Wert `number[`k`]` $\neq 0$ ist, dann gilt `number[`i`]` $<$ `number[`k`]` oder `number[`i`]` $=$ `number[`k`]` und $i < k$.

c) Ist die Bedingungen *Bounded Waiting* beim Bakery-Algorithmus erfüllt? Begründet eure Antwort und gebt eine obere Schranke für das *Bounded Waiting* an!

d) Zeigt am Beispiel von 2 Prozessen, dass der wechselseitige Ausschluss nicht mehr gewährleistet ist, wenn man auf das `choosing`-Feld verzichtet. Gebt hierzu auch die Zeilennummern der Prozesse zu den interessanten Zeitpunkten an.

Aufgabe 4.3 Wechselseitiger Ausschluss (3 + 3 + 1 = 7 Punkte)

Eine Landstraße verbindet die Orte A und B. Auf halbem Weg befindet sich eine einspurige Brücke. In dieser Aufgabe sollt ihr eine Verkehrsregelung für die Brücke schaffen.

Dafür zieht folgende Lösungen in Betracht:

- First come, first served: Ihr reguliert den Verkehr überhaupt nicht sondern geht davon aus, dass derjenige zuerst fährt, der zuerst ankommt.
- Ihr sperrt die Brücke wegen Baufälligkeit.
- Ihr baut an jeder Seite eine Ampel auf, die im Minutentakt umschaltet.
- Ihr legt eine Vorrangrichtung fest und stellt die entsprechenden Verkehrsschilder auf.

Der Einfachheit halber wird angenommen, dass die Gelbphase der Ampel so lange dauert, dass ein Auto die gerade befahrene Brücke noch überqueren kann, bevor der Gegenverkehr kommt. Es kann jedoch vorkommen, dass ein Auto, bevor es auf die Brücke fährt stirbt (stehen bleibt). Diese toten Autos können von anderen Autos überholt werden. Jedoch sehen die Autos auf der anderen Seite der Brücke nicht, ob ein Auto gestorben ist oder nur wartet.

a) Welche der vier Lösungen realisieren den wechselseitigen Ausschluss? Beachtet dabei, dass ein Konflikt nur dann auftritt, wenn zwei Autos aus entgegengesetzten Richtungen die Brücke gleichzeitig befahren. Begründet eure Aussagen.

b) Ihr habt euch für die Ampeln entschieden. Euch gefällt jedoch nicht, dass sie auch bei geringem Verkehr stur nach dem festen Zeittakt geschaltet sind und die Autos oftmals vor der leeren Brücke warten müssen. Ihr rüstet die Ampeln daher mit Sensoren für ankommende Fahrzeuge aus. Gebt einen Algorithmus (Pseudocode) für die Ampelschaltung mit Sensoren an, der einen wechselseitigen Ausschluss garantiert. Er sollte sowohl für wenig als auch für viel Verkehr gut funktionieren (guter Durchsatz bei viel Verkehr, geringe Wartezeit bei wenig Verkehr). Die oben gemachte Annahme zu der langen Gelbphase ist weiterhin gültig.

c) Leider stellt sich heraus, dass die Sensoren störanfällig und damit unbrauchbar sind, so dass ihr doch mit einem festen Zeittakt arbeiten müsst. Nun wollt ihr die Schaltung wenigstens so einstellen, dass die mittlere Wartezeit für alle Kraftfahrer minimal wird. Im Laufe der Voruntersuchung habt ihr festgestellt, dass Vormittags doppelt so viele Autos von A nach B fahren als umgekehrt. Am Nachmittag kehrt sich der Trend um. In welchem Verhältnis sollten die Zeittakte der Grünphasen für die Richtungen $A \rightarrow B$ und $B \rightarrow A$ Vormittags und Nachmittags stehen?

Lehrstuhl für Informatik 6
der RWTH Aachen
Prof. Dr.-Ing. H. Ney

Aachen, den 16. Mai 2011

S. Peitz, M. Freitag, J. Bitsch

Betriebssysteme und Systemsoftware

5. Übung

Abgabe der Lösungen: So., den 22. Mai 2011 im L2P.

Aufgabe 5.1 Semaphoren (1.5 + 2.5 + 2 = 6 Punkte)

In einem Hotel gibt es häufig das Problem, dass es nur eine begrenzte Anzahl an Liegestühle um den Pool gibt. Als engagierter Werkstudent wollt ihr das Konzept der Semaphoren auf dieses Problem übertragen, um niemanden mehr einen Liegestuhl anzubieten, wenn alle belegt sind.

Ihr wisst, dass eine Semaphore in etwa einer gekapselten Integer-Variablen entspricht, auf die mittels der drei Operationen `init`, `wait` und `signal` zugegriffen werden kann.

a) Stellt dar, was mit den drei Operationen auf der Integer-Variablen gemacht wird und inwiefern Semaphoren für das Pool-Problem geeignet sind.

b) Schreibt Algorithmen (in Pseudocode), mit denen der Pool mit 42 Liegen verwaltet wird. Der Pool wird morgens geöffnet und abends, nachdem der letzte Gast von der Liege aufgestanden ist, wieder geschlossen.

c) Momentan laufen die Urlauber, für die keine Liege frei ist, so lange um den Pool, bis eine Liege frei wird. Um das zu verhindern, soll ein Wartebereich vor dem Pool eingerichtet werden. Erweitert die Algorithmen dementsprechend.

Aufgabe 5.2 5 Philosophen Problem (1 + 1.5 + 1 + 1.5 = 5 Punkte)

In der Vorlesung habt ihr das *Fünf-Philosophen-Problem (Dining Philosophers Problem)* kennengelernt. Hier gibt es im wesentlichen zwei Probleme: Vermeidung von *Verklemmungen (deadlocks)* und Vermeidung des *Verhungerns (starvation)* von Philosophen. Es gilt:

- Tisch mit 5 Philosophen, Tellern und Stäbchen
- Abwechselnde Ess- und Denkphase
- Essen ist nur möglich, wenn die beiden Stäbchen links und rechts eines Philosophen frei sind
- alle Philosophen sind Linkshänder

a) Warum kann ein deadlock auftreten?

b) Wie können deadlocks verhindert werden?

c) Warum kann ein Philosoph verhungern?

d) Einer der Philosophen wird von euch mit Gewalt zum Rechtshänder umgepolt. Was passiert jetzt? Ist diese Idee der Asymmetrie eine alltagstaugliche Lösung?

Hinweis: Bitte beantwortet die Fragen knapp und präzise. Für Aufgabenteil a) und c) genügt ein Beispiel.

Aufgabe 5.3 I/O Geräte (1 + 1 + 1 = 3 Punkte)

I/O Geräte sind ein wichtiger Bestandteil des Betriebssystems. In der Vorlesung haben wir drei verschiedene Ansätze für I/O Software kennen gelernt: Busy Waiting, Interrupt gesteuert und DMA. Beschreibt diese drei Methoden. Was sind deren Unterschiede, was sind deren Vor- und Nachteile?

Lehrstuhl für Informatik 6
der RWTH Aachen
Prof. Dr.-Ing. H. Ney

Aachen, den 23. Mai 2011

S. Peitz, M. Freitag, J. Bitsch

Betriebssysteme und Systemsoftware

6. Übung

Abgabe der Lösungen: So., den 29. Mai 2011 im L2P.

Aufgabe 6.1 Semaphoren in C (0 + 2 + 4 = 6 Punkte)

Im L2P findet ihr das Programm `u6_1.c`. Es handelt sich um eine Implementierung des Erzeuger-Verbraucher-Problems mit zwei Verbrauchern. Die Synchronisation der Verbraucherprozesse erfolgt über UNIX-Semaphoren.

a) Informiert euch über die Handhabung von Semaphoren unter UNIX. Lest dazu die man-Pages von `semget()`, `semctl()` und `semop()` und studiert den gegebenen Quelltext. Testet das Programm.

b) Erweitert das Programm um einen weiteren Erzeugerprozess. Stellt bei eurer Lösung durch eine zweite Semaphore sicher, dass alles richtig funktioniert.

c) Implementiert eine Simulation des *5-Philosophen-Problems* mit der diskutierten Rechtshänderlösung (siehe Übung 5) unter Verwendung von UNIX-Semaphoren. Die Zeiten, die ein Philosoph isst oder denkt, sollen Zufallszahlen sein.

Ein Ergebnisprotokoll (Ausgaben des Programms) soll zeigen, dass das Programm einwandfrei funktioniert.

Aufgabe 6.2 Deadlocks (0,5 + 2 + 0,5 + 1 + 1 = 5 Punkte)

Es ist Sommer! Die Sonne scheint, die Temperaturen sind super: Das perfekte Grillwetter. Thomas und Martina planen eine Grillparty. Doch schon gibt es die ersten Probleme. Benjamin, der beste Freund von Thomas, und seine Freundin Maria sind Vegetarier. Fleisch gleichzeitig auf dem Grill mit ihrem Käse und gefüllten Champignons geht gar nicht. Auch mit dem großen Messer, und den Spießen gibt es Probleme. Alle brauchen alles gleichzeitig.

Zum Glück habt ihr in der Vorlesung die Prozessfortschrittsdiagramme kennengelernt und könnt Thomas und Martina helfen.

a) Wieviele Betriebsmittel und Prozesse müssen an einer Deadlocksituation beteiligt sein? Welche Voraussetzungen sind außerdem nötig?

b) Gegeben sind die Prozesse A und B, die jeweils 15 Zeittakte arbeiten, und vier exklusiv zu nutzende Betriebsmittel BM1 bis BM4. Tragt die Situation (sicher, unsicher, unmöglich) in ein Prozessfortschrittsdiagramm ein (Prozess A: y-Achse, Prozess B: x-Achse).

	Vegetarier (Prozess A)	Fleischesser (Prozess B)
großes Messer	2-4	1-6
Gewürze	2-3	5-6
Grill	10-11	9-14
Spieße	8-12	13-14

Beispiel: Der Tabelleneintrag 2-4 heißt, dass die Vegetarier das große Messer von der 2. bis zur 4. Zeiteinheit benötigen.

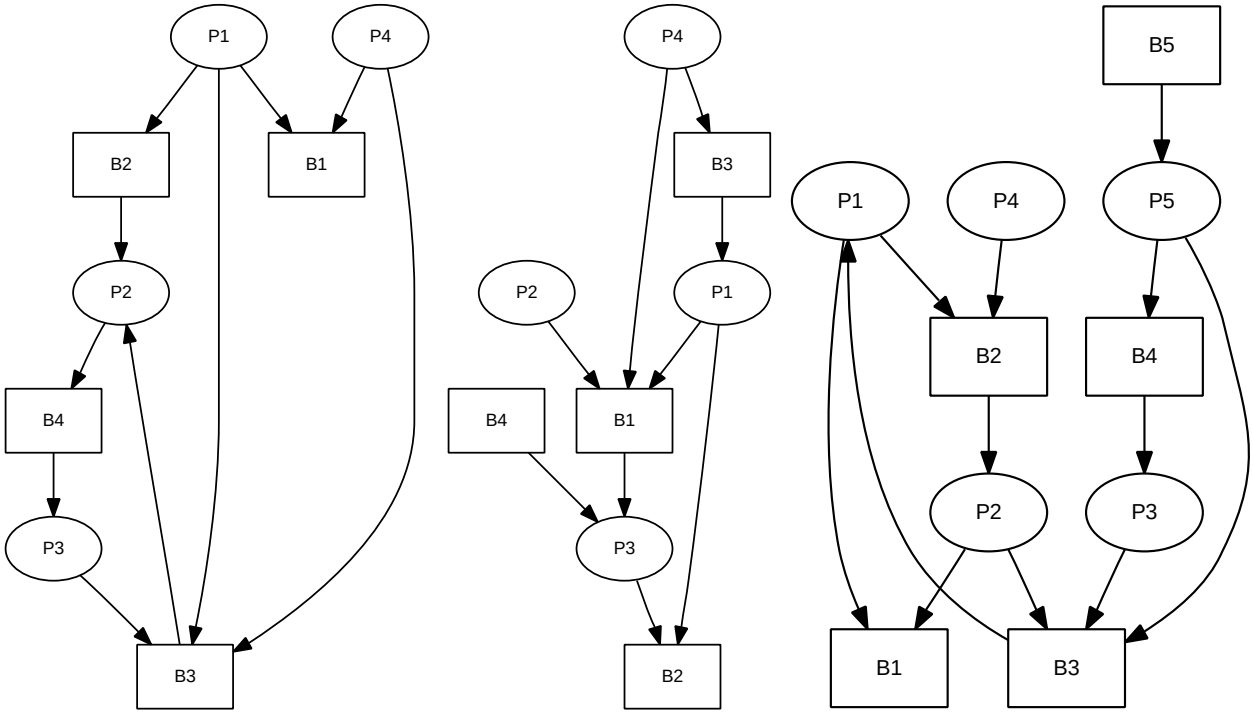
c) Entscheidet, welchen Status die folgenden Zeitpunkte haben:

(2,3); (4,6); (8,3); (12,9); (12,14)

d) Gebt einen Algorithmus an (Prosa, kein Code), mit dem unsichere Bereiche markiert werden können.

e) Schreibt einen sicheren Schedule für die beiden Prozesse und zeichnet ihn in das Diagramm ein.

Aufgabe 6.3 Deadlocks (3 + 1= 4 Punkte)



a) Gegeben seien die drei oben abgebildeten Graphen. $B_1 ... B_m$ sind die verschiedenen Betriebsmittel, $P_1 ... P_n$ sind die verschiedenen Prozesse. Eine Kante $P_i \rightarrow B_j$ zeigt, dass P_i das Betriebsmittel B_j braucht. Eine Kante $B_j \rightarrow P_i$ zeigt, dass das Betriebsmittel B_j bereits dem Prozess P_i zugeteilt wurde. Entscheidet und begründet jeweils, ob in dem Graphen ein Deadlock vorliegt.

b) Nehmen wir nun an, dass jedes Betriebsmittel nun zweimal verfügbar ist. Bestehen weiterhin die Deadlocks? Jedes zugeteiltes Betriebsmittel ist hierbei auch nur einmal zugeteilt.

Lehrstuhl für Informatik 6
der RWTH Aachen
Prof. Dr.-Ing. H. Ney

Aachen, den 30. Mai 2011

S. Peitz, M. Freitag, J. Bitsch

Betriebssysteme und Systemsoftware

7. Übung

Abgabe der Lösungen: So., den 05. Juni 2011 im L2P.

Aufgabe 7.1 Banker's Algorithmus (2 + 2 = 4 Punkte)

Gegeben seien drei Prozesse P_1, P_2 und P_3, die drei exklusive Betriebsmittel BM_1, BM_2 und BM_3 benutzen wollen. Es existieren 12 Exemplare von BM_1, 9 von BM_2 und 8 von BM_3. Zum Zeitpunkt 0 sei der Vektor freier Betriebsmittel (BM_1, BM_2, BM_3) gegeben mit:
Available(0) $= (Available[1](0), Available[2](0), Available[3](0)) = (9, 6, 6)$.

Weiterhin gilt:

- $max[1](0) = (max[1,1], max[1,2], max[1,3]) = (6, 3, 3)$,
  $Allocation[1](0) = (Allocation[1,1](0), Allocation[1,2](0), Allocation[1,3](0)) = (0, 2, 0)$
- $max[2](0) = (4, 3, 5)$, $Allocation[2](0) = (1, 0, 1)$
- $max[3](0) = (5, 1, 1)$, $Allocation[3](0) = (2, 1, 1)$

a) Prüft mit dem in der Vorlesung vorgestellten *Banker-Algorithmus* 'Safety Detection', ob sich das System in einem sicheren Zustand befindet!

b) In der Realität erfolgen nicht alle Anforderungen auf einen Schlag, sondern es werden immer eine oder wenige Anforderungen zu einem Zeitpunkt kommen. Der Banker-Algorithmus kann auch dazu verwendet werden, zu solch einer Anforderung zu entscheiden, ob das System in einem sicheren Zustand bleibt oder nicht.

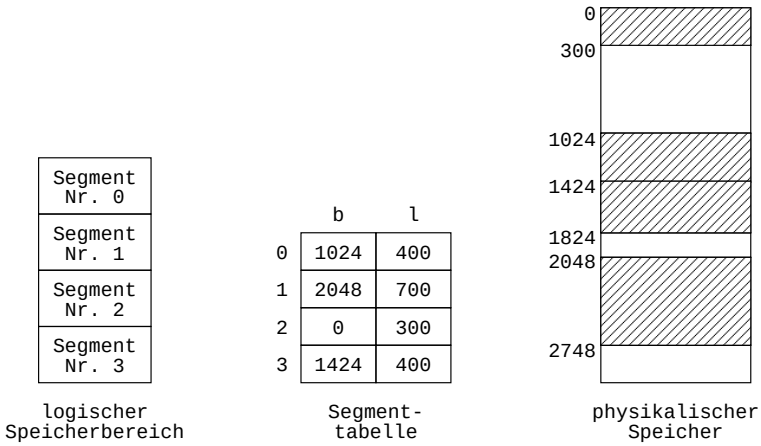
   Zum Zeitpunkt k erfolge nun eine Anforderung. Betrachtet die folgenden drei Alternativen. Welche der Anforderungen führen jeweils vom obigen Zustand ausgehend in einen sicheren Zustand, d.h. welche Anforderungen dürften erfüllt werden?

   (a) $Request[2](k) = (1, 2, 0)$
   (b) $Request[2](k) = (0, 0, 3)$
   (c) $Request[3](k) = (0, 0, 1)$

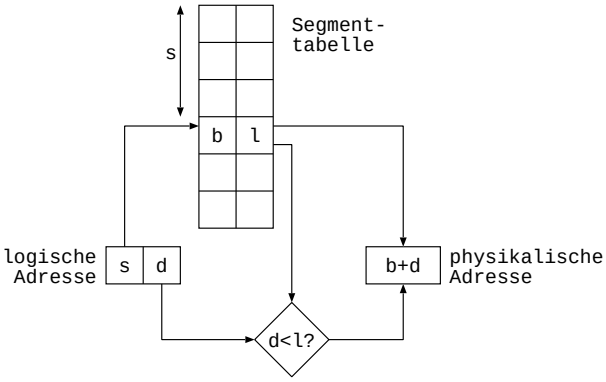
Aufgabe 7.2 Speicherverwaltung (2 + 2 + 2 = 6 Punkte)

Ein großer Vorteil der Speicherverwaltung ist, dass der Programmierer sich nicht um den physikalischen Adressraum kümmern muss, sondern mit logischen Adressen arbeiten kann. Spätestens bei der Ausführung eines Programmes müssen diese logischen Adressen aber in physikalische Adressen umgewandelt werden.

Das Umsetzungsprinzip beim *Segmenting* funktioniert wie folgt: Jedem Programm ist eine Segmenttabelle zugeordnet, in der für jedes logische Segment die physikalische Anfangsadresse b und die Größe des Segments l eingetragen ist, die das Programmsegment im Speicher einnimmt.



Eine logische Adresse besteht nun aus der Segmentnummer `s`, mit der aus der Segmenttabelle die entsprechende Anfangsadresse `b` bestimmt wird, und einem Offset `d`, der angibt, an welcher Stelle in Relation zur Anfangsadresse sich die Adresse, auf die man zugreifen will, befindet. Dabei wird geprüft, ob der Offset kleiner als die Segmentlänge `l` ist, da anderenfalls eine ungültige Adresse angesprochen wird. Die physikalische Adresse ergibt sich also zu `b+d`.



Für die Speicherverwaltung nach dem Segmentierungsverfahren ist für ein Programm folgende Segmenttabelle gegeben (Länge in Speicherworten):

Segment	Basis	Länge
0	1337	210
1	11	11
2	42	20
3	300	23
4	1111	200

a) Wieviele Speicherworte stehen dem Programm im physikalischen Speicher zur Verfügung?

b) Welches ist die kleinste und welches die größte für das Programm verfügbare physikalische Adresse?

c) Berechnet zu den folgenden physikalischen Adressen jeweils die logischen Adressen:

- 310
- 811
- 1338
- 1111

Aufgabe 7.3 Speicherhierarchie (1 + 1 + 1 + 1 + 1 + 1 = 6 Punkte)

In der Vorlesung habt ihr die Speicherhierarchie kennengelernt. So gibt neben dem Hauptspeicher und der Festplatte auch eine Reihe weiterer Speichertypen, wie Register, Level 1/2/3 Cache, RAM, optische Laufwerke und so weiter. In dieser Aufgabe sollt ihr euch mit Hilfe des Internets einen Überblick über die aktuellen Zugriffsgeschwindigkeiten (Random Access Time) und Größen dieser verschiedenen Speicher verschaffen.

a) Caches (Unterschieden nach L1/L2/L2)

b) externes RAM

c) Solid State Disks

d) normale Festplatten

e) DVDs

f) Magnetbänder

Lehrstuhl für Informatik 6
der RWTH Aachen
Prof. Dr.-Ing. H. Ney

Aachen, den 06. Juni 2011

S. Peitz, M. Freitag, J. Bitsch

Betriebssysteme und Systemsoftware

8. Übung

Abgabe der Lösungen: So., den 19. Juni 2011 im L2P.

Aufgabe 8.1 Hauptspeicherverwaltung (3 + 2 = 5 Punkte)

Gegeben sei folgende Belegung eines Speichers:

belegt	512 Byte frei	belegt	1024 Byte frei	belegt	256 Byte frei

Weiterhin seien vier Belegungensmethoden für Speicherplatzanforderungen gegeben:

- *First-Fit* Belege den ersten freien Speicherbereich, der groß genug ist, die Anforderung zu erfüllen.
- *Rotating-First-Fit* Wie First-Fit, jedoch wird von der Position der vorherigen Platzierung ausgehend ein passender Bereich gesucht. Wird das Ende des Speichers erreicht, so wird die Suche am Anfang des Speichers fortgesetzt (maximal bis zur Position der vorherigen Platzierung). Bei der ersten Anforderung beginnt die Suche am Anfang des Speichers.
- *Best-Fit* Belege den kleinsten freien Speicherbereich, in den die Anforderung paßt.
- *Worst-Fit* Belege den größten freien Speicherbereich, in den die Anforderung paßt.

a) Wie sieht der obige Speicher aus, wenn nacheinander vier Anforderungen der Größe 192 Byte, 320 Byte, 256 Byte und 1024 Byte ankommen? Notiert für jede Strategie die freien Speicherbereiche nach jeder Anforderung (z.B. (512, 1024, 256) für die obige Belegung), und gebt an, für welche Methoden die Anforderungen erfüllt werden können!

   Beachtet, dass die Daten jeweils linksbündig in einer Lücke abgelegt und einmal belegte Speicherbereiche nicht wieder freigegeben werden!

b) Ihr dürft nun die Reihenfolge der freien Speicherbereiche und die der Anforderungen vertauschen (aber keine Speicherbereiche zusammenfassen). Gebt für jede Strategie eine Speicherbelegung und die zugehörigen Anforderungen an, die für jeweils nur diese Methode erfolgreich arbeitet und für die übrigen Methoden nicht alle Anforderungen erfüllen kann! Gebt auch die Speicherbelegungen nach Abarbeitung der einzelnen Anforderungen an!

Aufgabe 8.2 Multilevel-Feedback-Queue-Scheduling (5 Punkte)

Gegeben sei nun ein Multilevel-Feedback-Queue-Scheduling mit vier Prioritätsklassen. Jeder Klasse ist eine eigene Warteschlange und ein eigenes Quantum zugeordnet:

Klasse	Quantum	Priorität
0	1	höchste
1	2	
2	5	
3	∞	niedrigste

Wenn das Zeitquantum eines Prozesses abgelaufen ist, wird er an das Ende der Warteschlange mit nächstniedriger Priorität gestellt. Neuen Prozessen ist eine bestimmte Priorität zugeordnet. Sie werden an das Ende der Warteschlange ihrer Prioritätsklasse angehängt. Es wird am Ende jeder Zeiteinheit überprüft, welcher Prozess am Anfang der nicht leeren-Warteschlange mit der höchsten Priorität steht und dieser dann im nächsten Schritt bearbeitet.

Folgende Prozesse sollen betrachtet werden:

Prozess	Ankunftszeit	Start Klasse	Laufzeit
A	0	0	16
B	0	0	1
C	1	0	3
D	2	0	5
E	3	0	2
F	10	0	1
G	13	0	4
H	14	0	6
I	16	0	1

Die lexikographische Ordnung gibt die Reihenfolge des Ankommens in der Queue an: d.h. Job A kommt vor Job B an und wird somit zuerst bearbeitet. Ein Prozess, der zum Zeitpunkt t ankommt, wird erst ab dem $(t+1)$-ten Zeitpunkt berücksichtigt, d.h. er kann frühestens eine Zeiteinheit nach seiner Ankunft die CPU verwenden. Kontextwechsel werden vernachlässigt.

Gebt für die ersten 40 Zeiteinheiten an, welche Prozesse sich in welcher Warteschlange befinden (in der korrekten Reihenfolge) und welchem Prozess Rechenzeit zugeteilt wird. Verwendet dabei das folgende Tabellenformat für eure Angaben:

t	Kl. 0 RR(1)	Kl. 1 RR(2)	Kl. 2 RR(5)	Kl. 3 FIFO	Running	Incoming
0	-	-	-	-	-	A(16),B(1)
1	...	...	...	...	...	...
2	...	...	...	...	...	...

Aufgabe 8.3 Deadlocks (4 Punkte)

Gegeben sind zwei Prozesse P1 und P2, die beide die Betriebsmittel BM1 und BM2 verwenden. Beide Prozesse benötigen für ihre Ausführung 8 Zeiteinheiten (inklusive der Zeiten für die Nutzung von Betriebsmitteln). Prozess P1 benötigt BM1 während seiner Ausführung in den Zeiteinheiten 1 bis 4 und BM2 in den Zeiteinheiten 3 bis 6. Prozess P2 benötigt BM1 in den Zeiteinheiten 4 bis 7 und BM2 in den Zeiteinheiten 2 bis 5. Beide Prozesse benötigen exklusiven Zugriff auf die Betriebsmittel, ein Entzug von Betriebsmitteln ist unmöglich. Nun werde zunächst P1 für zwei Zeiteinheiten ausgeführt, danach P2 für drei Zeiteinheiten. Tritt in dieser Situation unvermeidbar ein Deadlock auf oder nicht? Begründe deine Antwort.

Lösung 8.1 Hauptspeicherverwaltung (3 + 2 = 5 Punkte)

a)
- FF
  - 192: 320 // 1024 // 256
  - 320: 0 // 1024 // 256
  - 256: 0 // 768 // 256
  - 1024: FAIL
- RFF
  - 192: 320 // 1024 // 256
  - 320: 320 // 704 // 256
  - 256: 320 // 704 // 0
  - 1024: FAIL
- BF
  - 192: 512 // 1024 // 64
  - 320: 192 // 1024 // 64
  - 256: 192 // 768 // 64
  - 1024: FAIL
- WF
  - 192: 512 // 832 // 256
  - 320: 512 // 512 // 256
  - 256: 256 // 512 // 256
  - 1024: FAIL

b)
- FF: 1024, 320, 256, 192
- RFF: 320, 1024, 256, 192
- BF: 1024, 320, 256, 192
- WF: 1024, 192, 320, 256

Lösung 8.2 Multilevel-Feedback-Queue-Scheduling (5 Punkte)

t	Kl. 0 RR(1)	Kl. 1 RR(2)	Kl. 2 RR(5)	Kl. 3 FIFO	Incoming	Running
0	-	-	-	-	A(16),B(1)	-
1	A(16),B(1)	-	-	-	C(3)	A(16)
2	B(1),C(3)	A(15)	-	-	D(5)	B(1)
3	C(3),D(5)	A(15)	-	-	E(2)	C(3)
4	D(5),E(2)	A(15),C(2)	-	-	-	D(5)
5	E(2)	A(15),C(2),D(4)	-	-	-	E(2)
6	-	A(15),C(2),D(4),E(1)	-	-	-	A(15)
7	-	A(14),C(2),D(4),E(1)	-	-	-	A(14)
8	-	C(2),D(4),E(1)	A(13)	-	-	C(2)
9	-	C(1),D(4),E(1)	A(13)	-	-	C(1)
10	-	D(4),E(1)	A(13)	-	F(1)	D(4)
11	F(1)	D(3),E(1)	A(13)	-	-	F(1)
12	-	D(3),E(1)	A(13)	-	-	D(3)
13	-	E(1)	A(13),D(2)	-	G(4)	E(1)
14	G(4)		A(13),D(2)	-	H(6)	G(4)
15	H(6)	G(3)	A(13),D(2)	-	-	H(6)
16	-	G(3),H(5)	A(13),D(2)	-	I(1)	G(3)
17	I(1)	G(2),H(5)	A(13),D(2)	-	-	I(1)
18	-	G(2),H(5)	A(13),D(2)	-	-	G(2)
19	-	H(5)	A(13),D(2),G(1)	-	-	H(5)
20	-	H(4)	A(13),D(2),G(1)	-	-	H(4)
21	-	-	A(13),D(2),G(1),H(3)	-	-	A(13)
22	-	-	A(12),D(2),G(1),H(3)	-	-	A(12)
23	-	-	A(11),D(2),G(1),H(3)	-	-	A(11)
24	-	-	A(10),D(2),G(1),H(3)	-	-	A(10)
25	-	-	A(9),D(2),G(1),H(3)	-	-	A(9)
26	-	-	D(2),G(1),H(3)	A(8)	-	D(2)
27	-	-	D(1),G(1),H(3)	A(8)	-	D(1)
28	-	-	G(1),H(3)	A(8)	-	G(1)
29	-	-	H(3)	A(8)	-	H(3)
30	-	-	H(2)	A(8)	-	H(2)
31	-	-	H(1)	A(8)	-	H(1)
32	-	-	-	A(8)	-	A(8)
33	-	-	-	A(7)	-	A(7)
34	-	-	-	A(6)	-	A(6)
35	-	-	-	A(5)	-	A(5)
36	-	-	-	A(4)	-	A(4)
37	-	-	-	A(3)	-	A(3)
38	-	-	-	A(2)	-	A(2)
39	-	-	-	A(1)	-	A(1)
40	-	-	-	-	-	-

Lösung 8.3 Deadlocks (5 Punkte)

Loesung mit Prozessfortschrittsdiagramm:

Man laeuft in einen unsicheren Bereich, ein Deadlock ist unvermeidbar.

Lehrstuhl für Informatik 6
der RWTH Aachen
Prof. Dr.-Ing. H. Ney

Aachen, den 20. Juni 2011

S. Peitz, M. Freitag, J. Bitsch

Betriebssysteme und Systemsoftware

9. Übung

Abgabe der Lösungen: So., den 26. Juni 2011 im L2P.

Aufgabe 9.1 Paging (3 + 2 + 2 = 7 Punkte)

Untersucht die Zugriffszeiten, die bei unterschiedlichen Arten der Traversierung einer Matrix auftreten. Implementiert dazu ein Programm in C, dass eine Matrix der Größe $n \times n$ anlegt und diese einmal zeilenweise und einmal spaltenweise durchläuft.

- Die Matrix soll in einem zusammenhängenden Speicherblock angelegt werden.
- Initialisiert die Matrix mit Nullen.
- Beim Durchlaufen der Matrix soll diese mit `A(i,j) = i | j` (bitweises ODER) gefüllt werden.
- Kompiliert euer Programm ohne Compiler-Optimierung (`gcc -O0`), um die Messwerte nicht zu verfälschen.

Ihr könnt das Grundgerüst `u10_1.c` verwenden, das ihr im L^{2}P findet.

a) Implementiert eine Methode, um die (durchschnittliche) Zeit zu messen, die für den Zugriff auf ein Matrixelement benötigt wird. Verwendet dazu z.B. die Funktionen `times` oder `getrusage` (`man 2 times`, `man 2 getrusage`). Warum ist die Initialisierung der Matrix wichtig?

   Die Zugriffszeit kann mit einfachen Mitteln nicht exakt gemessen werden, Approximationen sind daher erlaubt.

b) Messt die Zugriffszeiten für den zeilenweisen und den spaltenweisen Zugriff für verschiedene Matrixgrößen. Untersucht auch Matrizen, die mehr Speicherplatz benötigen, als physikalischer Speicher vorhanden ist. Erstellt ein Diagramm an, in dem die Zugriffszeiten über der Matrixgröße aufgetragen sind. Verwendet hierzu Mittelwerte von Messungen mehrerer Programmläufe.

c) Was kann man beobachten? Erklärt eure Beobachtungen!

Aufgabe 9.2 Paging - Speicherverwaltung (6 Punkte)

In der Vorlesung habt Ihr das Prinzip des *Pagings* kennen gelernt. Es beinhaltet, dass im Hauptspeicher eine begrenzte Zahl von Speicherseiten (*Pages*) fester Grösse gehalten wird, die kleiner als die Anzahl der tatsächlich existierenden Seiten ist. Seiten, die gerade nicht im Hauptspeicher gehalten werden können, werden auf ein externes Speichermedium ausgelagert.

Will ein Prozess auf eine solche Seite zugreifen, muss sie zuvor vom externen Speicher in den Hauptspeicher zurückgeholt werden (*Seitenfehler, Page Fault*). Sind noch nicht alle Plätze für Speicherseiten (*Frames*) im Hauptspeicher gefüllt, stellt dies kein Problem dar. Ist jedoch kein Platz mehr vorhanden, muss eine Seite im Hauptspeicher ausgewählt werden, die ausgelagert und durch die neue Seite ersetzt wird.

Für die Auswahl der zu ersetzenden Seite gibt es verschiedene Strategien:

FIFO (First In, First Out): Die jeweils älteste Seite wird aus dem Hauptspeicher ausgelagert.

LRU (Least Recently Used): Es wird die Seite aus dem Hauptspeicher entfernt, auf die am längsten kein Zugriff erfolgt ist.

SC (Second Chance): Diese Strategie ist eine Verbesserung des FIFO Algorithmus. Pro Seite wird ein Use-Bit gespeichert, welches gesetzt wird, wenn nach dem Zugriff, der die Seite angefordert hat, ein weiterer Zugriff erfolgt. Ersetzt wird die älteste Seite mit nicht gesetztem Use-Bit. Alle älteren Seiten als die zu ersetzende Seite verlieren dabei ihr Use-Bit und ihre Ladezeiten werden auf die aktuelle Zeit gesetzt. Ebenfalls verlieren alle Seiten ihr Use-Bit, wenn die Use-Bits aller Seiten gesetzt sind ($\rightarrow$ keine zusätzliche Information mehr durch die Use-Bits).

LFU (Least Frequently Used): Bei dieser Strategie wird die Seite mit der geringsten Nutzungshäufigkeit ausgetauscht. Die Nutzungshäufigkeit kann auf verschiedene Art und Weise ermittelt werden. Für diese Aufgabe wird angenommen, dass die Nutzung ab Beginn des Referenzstrings gezählt wird. Ist für mehrere Seiten die gleiche Nutzungshäufigkeit aufgetreten, wird die älteste dieser Seiten zuerst ausgetauscht.

CLIMB (Aufstieg bei Bewährung): Bei dieser Strategie wird eine Seite, die sich bereits im Speicher befindet und auf die ein erneuter Zugriff erfolgt, um eine Position "verjüngt" (also mit ihrem Vorgänger vertauscht). Die dann älteste Seite wird immer verdrängt.

OPT (Optimalstrategie): Diese Strategie ersetzt die Seite zuerst, die in Zukunft am längsten nicht mehr benötigt wird. Die Strategie ist in der Praxis nicht realisierbar, weil dazu alle zukünftigen Zugriffe bekannt sein müssen.

Seitenzugriffe werden (im Modell) als *Referenzstring* angegeben, welcher die zeitliche Reihenfolge der angeforderten Seiten enthält. Ein solcher String ist z.B. "01230544351120675231": Erst erfolgt ein Zugriff auf die Seite 0, dann auf Seite 1, dann auf Seite 2 usw. Seitennummern seien hier immer einstellig!

Gegeben sei ein System mit vier Frames. Gebt für jede der vorgestellten Strategien die Belegung dieser vier Speicherstellen zu jedem Zeitpunkt sowie die Anzahl der auftretenden Seitenfehler an. Legt dabei den oben angegebenen Referenzstring zugrunde! Zu Beginn sei der Hauptspeicher leer.

Aufgabe 9.3 Windows vs. Linux (2 + 2 + 1 + 2 = 7 Punkte)

In der Vorlesung habt ihr die grundlegenden Unterschied und Gemeinsamkeiten von Windows und Linux kennen gelernt. Dies wollen wir jetzt weiter vertiefen und wiederholen.

a) **Designprinzipien:** Linux verwendet einen monolithischen Kernel, während Windows ursprünglich als Mikrokernel geplant wurde, inzwischen aber ein Hybridkernel ist. Was sind die Hauptunterschiede zwischen diesen Kernel Architekturen? Wie ordnen sich andere Betriebssysteme, wie BSD, MacOS, L4, Symbian, Plan 9, in dieses Spektrum ein?

b) **Scheduling:** Beschreibe die Funktionsweise des Linux O(1) Schedulers. Wie werden Prozesse behandelt die sehr IO-lastig sind? Wann wird die IO-Lastigkeit eines Prozesses entschieden?

c) **Deadlock Behandlung:** In der Vorlesung wurden verschiedene Mechanismen zur Erkennung und Vermeidung von Deadlocks vorgestellt. Welche davon werden in der Praxis bei Linux und Windows verwendet?

d) **Dateisysteme:** NTFS und EXT3 kennt ihr ja aus der Vorlesung. Eines der verbreitetsten Dateisysteme ist jedoch FAT, in seinen verschiedenen Variationen, darunter VFAT und FAT32. Was sind die maximalen Größen dieser Dateisysteme? Wie groß dürfen Dateien maximal werden? Wie funktioniert die Benamung von Dateien und Ordnern? Welche Möglichkeiten zur Dateirechte-Verwaltung bestehen?

Lösung 9.1 Paging (3 + 2 + 2 = 7 Punkte)

a)
- Implementierung: siehe u10_1.c
- Wichtig:
  - Matrix in einem zusammenhaengenden Speicherblock
  - Sinnvolle Zeitmessung
- Durch Initialisierung werden alle Speicherseiten einmal angefordert. Sonst wuerden die Messwerte fuer die beiden Zugriffsarten verfaelscht.

b)
- siehe plot.pdf und plot_swap.pdf

c)
- Spaltenweiser Zugriff ist deutlich langsamer, da bei jedem Zugriff eine andere Seite angesprochen werden muss
- Wenn die Matrix nicht mehr in den physikalischen Speicher passt, werden die Effekte durch Seitenfehler noch deutlicher.
- Bonus: Bei Matrixgroessen 2^n beobachtet man Peaks. Bei diesen Matrixgroessen passt eine Zeile immer exakt in ein Vielfaches der Seitengroesse. Dadurch wird der Hardware-Cache (L2-Cache) deutlich schlechter ausgenutzt.

Lösung 9.2 Paging - Speicherverwaltung (6 Punkte)

FIFO

Reference	0	1	2	3	0	5	4	4	3	5	1	1	2	0	6	7	5	2	3	1
Frame 1	0	0	0	0	*	5	5	*	*	*	5	*	5	0	0	0	0	2	2	2
Frame 2		1	1	1	*	1	4	*	*	*	4	*	4	4	6	6	6	6	3	3
Frame 3			2	2	*	2	2	*	*	*	1	*	1	1	1	7	7	7	7	1
Frame 4				3	*	3	3	*	*	*	3	*	2	2	2	2	5	5	5	5

4+11 Seiten Fehler

LRU

Reference	0	1	2	3	0	5	4	4	3	5	1	1	2	0	6	7	5	2	3	1
Frame 1	0	0	0	0	*	0	0	*	*	*	1	*	1	1	1	7	7	7	7	1
Frame 2		1	1	1	*	5	5	*	*	*	5	*	5	5	6	6	6	6	3	3
Frame 3			2	2	*	2	4	*	*	*	4	*	2	2	2	2	5	5	5	5
Frame 4				3	*	3	3	*	*	*	3	*	3	0	0	0	0	2	2	2

4+11 Seiten Fehler

SC

Reference	0	1	2	3	0	5	4	4	3	5	1	1	2	0	6	7	5	2	3	1
Frame	0	1	2	3	3	5	4	4	4	4	1	1	2	0	6	7	5	2	3	1
		0	1	2	2	0	5	5	5	5	3	3	4	1	0	6	7	5	2	3
			0	1	1	3	0	0	0	0	4	4	5	2	1	0	6	7	5	2
				0	0	2	3	3	3	3	5	5	1	4	2	1	0	6	7	5

4+11 Seitenfehler

LFU

Nutzhäufigkeit	1	1	1	1	2	1	1	2	2	2	2	3	2	3	1	1	3	3	3	4
Reference	0	1	2	3	0	5	4	4	3	5	1	1	2	0	6	7	5	2	3	1
Frame 1	0_I	0_I	0_I	0_I	0_{II}	0_{II}	0_{II}	0_{II}	0_{II}	0_{II}	1_{II}	1_{III}	1_{III}	1_{III}	1_{III}	1_{III}	1_{III}	1_{III}	3_{III}	3_{III}
Frame 2		1_I	1_I	1_I	1_I	5_I	5_I	5_I	5_I	5_{II}	5_{II}	5_{II}	5_{II}	0_{III}	0_{III}	0_{III}	0_{III}	0_{III}	0_{III}	1_{IIII}
Frame 3			2_I	2_I	2_I	2_I	4_I	4_{II}	4_{II}	4_{II}	4_{II}	4_{II}	4_{II}	4_{II}	6_I	7_I	5_{III}	5_{III}	5_{III}	5_{III}
Frame 4				3_I	3_I	3_I	3_I	3_I	3_{II}	3_{II}	3_{II}	3_{II}	2_{II}	2_{III}	2_{II}	2_{II}	2_{II}	2_{III}	2_{III}	2_{III}

4+10 Seitenfehler

CLIMB

Reference	0	1	2	3	0	5	4	4	3	5	1	1	2	0	6	7	5	2	3	1
Frame X	0	1	2	3	3	5	~~4~~	4	4	4	1	1	2	0	6	7	5	2	3	1
Frame X		0	1	2	2	3	~~5~~	5	3	5	4	4	1	2	0	6	7	5	2	3
Frame X			0	1	0	2	3	3	5	3	5	5	4	1	2	0	6	7	5	2
Frame X				0	1	0	2	2	2	2	3	3	5	4	1	2	0	6	7	5

4+11 Seitenfehler

OPT

Reference	0	1	2	3	0	5	4	4	3	5	1	1	2	0	6	7	5	2	3	1
Frame 1	0	0	0	0	0	5	5	5	5	5	5	5	5	5	5	5	5	5	5	1
Frame 2		1	1	1	1	1	1	1	1	1	1	1	1	0	6	7	7	7	7	7
Frame 3			2	2	2	2	4	4	4	4	4	4	2	2	2	2	2	2	2	2
Frame 4				3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

4+7 Seitenfehler

Lösung 9.3 Paging (3 + 2 + 2 = 7 Punkte)

a) **Designprinzipien:** Mit unserem guten Freund Wikipedia leicht zu finden.

monolithischer Kernel: Nicht nur IPC,Prozess- und Speicherverwaltung, sondern auch Hardware-Treiber und andere Funktionen direkt im Kernel. Beispiele: Linux, BSD

Mikrokernel: nur IPC,Prozess- und Speicherverwaltung im Kernel. Alles andere ueber abgetrennte Module im eigenen Prozesskontext. Deutlich weniger Befehle und kleiner. Dadurch beweissbar sicherer. Beispiele: L4, Symbian

Hybridkernel: Hybrid zwischen monolithischem und Mikrokernel. Einige Teile von monolithischen Kernel werden in den Kernel integriert, um bessere Performanz zu erreichen. Beispiele: MacOSX, Windows, Plan 9

b) **Scheduling:** Beschreibe die Funktionsweise des Linux O(1) Schedulers.

jeder Prozessor hat eine eignee runqueue in der wartende Prozesse verwaltet werde. Jede Runqueue hat 2 priority arrays: active und expired. innerhalb der arrays sind die Prozesse nach prioritaet geordnet. active array: alle prozesse denen noch CPU-Zeit zusteht. expired array: alle abgelaufenen prozesse. es wird immer der jeweils hoechstprioritisierte Prozess des active array abgearbeitet und dann in den expired array verschoben. sobald alle Prozesse ihre Zeitquanten verbracht haben werden active und expired array ausgetauscht.

Wie werden Prozesse behandelt die sehr IO-lastig sind?

Setzen des nice Werts +/- 5 abhaengig von der IO-Lastigkeit.

Wann wird die IO-Lastigkeit eines Prozesses entschieden?

Neuberechnung der dynamische Prioritaeten erfolgt beim Wechsel eines Prozesses vom active in den expired array

c) **Deadlock Behandlung:** Vogel-Strauss-Taktik: Es findet keine Deadlockbehandlung statt, sondern aufgrund der geringen Wahrscheinlichkeit wird dieses Problem ignoriert.

d) **Dateisysteme:** Was sind die maximalen Größen dieser Dateisysteme? VFAT: 4 GB FAT32: 8 TB (bei 32KB Clustern)

Wie groß dürfen Dateien maximal werden? VFAT: 4 GB FAT32: 4GB

Wie funktioniert die Benamung von Dateien und Ordnern? VFAT: 8.3 + Trick fuer lange Namen. FAT32: 255 Zeichen

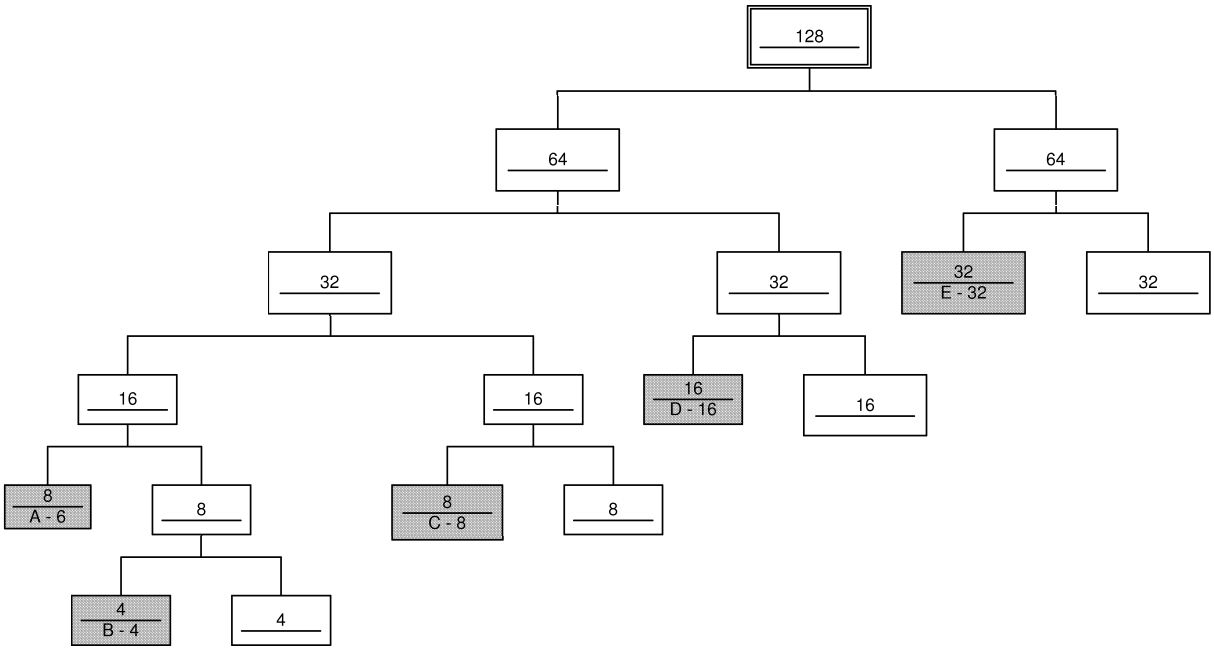
8.3 Alias + zusaetzliche Namensteile Verzeichniseintraege. Gesamtname kann maximal 260 Zeichen haben inklusive Pfad.

Welche Möglichkeiten zur Dateirechte-Verwaltung bestehen? VFAT,FAT32: Urspruenglich nur Verschiedene Dateiattribute: Schreibgeschuetzt, Versteckt, Systemdatei. Keine UNIX Rechtebits.

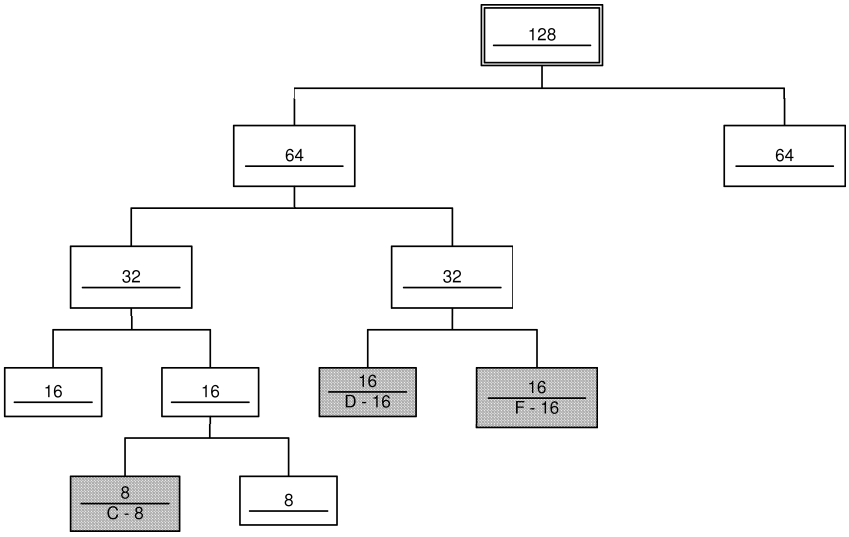
Spaetere Erweiterung exFAT hat auch ACLs.

Lösung 10.1 Speicherverwaltung mit Buddy-Systemen (2 + 2 = 4 Punkte)

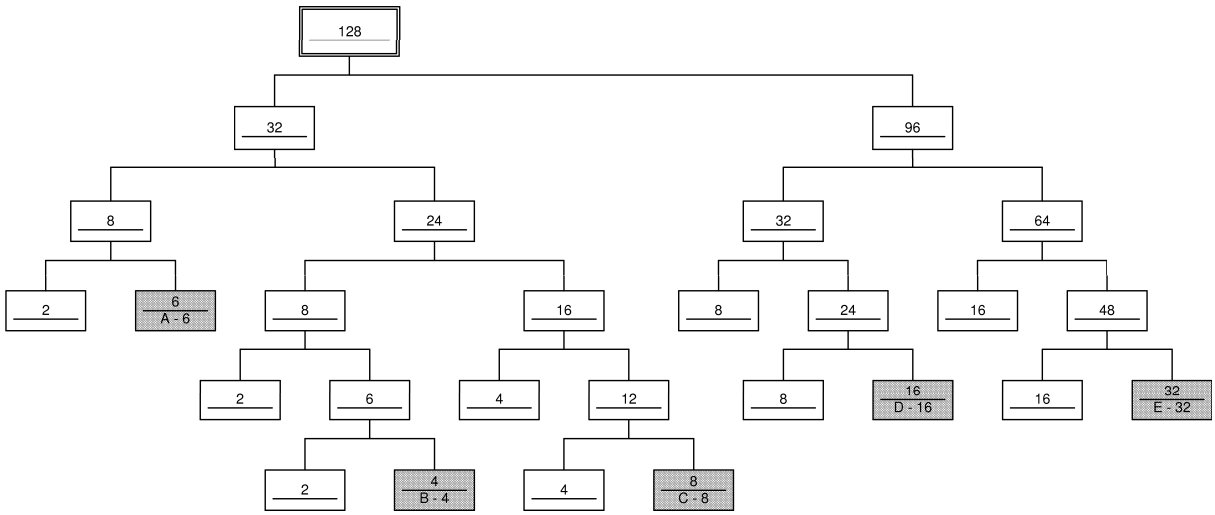
a) Speicherbelegung bis nach der Anforderung von E:



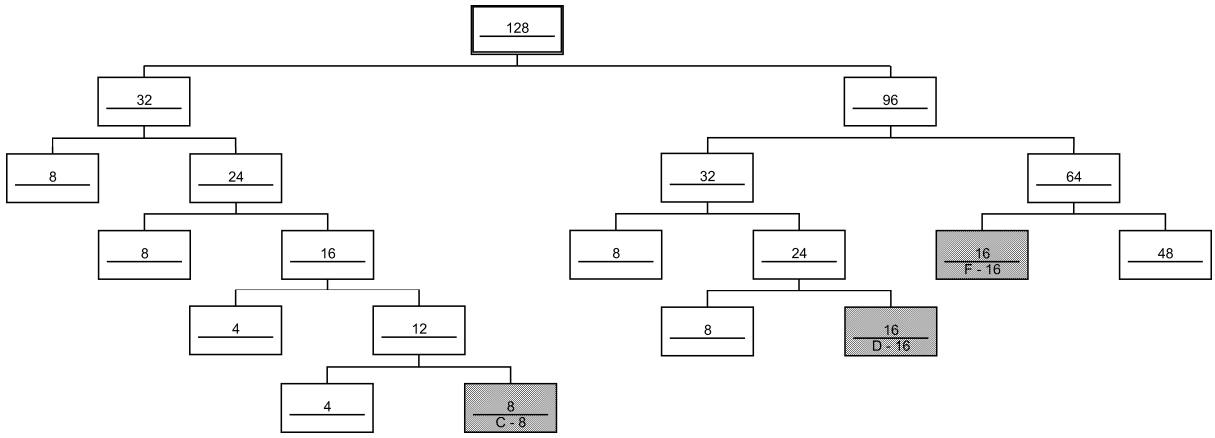
Speicherbelegung am Ende der Tabelle:



b) Speicherbelegung bis nach der Anforderung von E:



Speicherbelegung am Ende der Tabelle:



Lösung 10.2 Seitenersetzungsstrategien (2 Punkte)

a) n

b) $m \geq n : n$ sonst p

Lösung 10.3 Belady-Anomalie (2 Punkte)

Nein, bei einem optimalen Algorithmus tritt die Belady-Anomalie nicht auf, sonst waere er nicht optimal.

Lösung 10.4 (4 Punkte)
Inklusionseigenschaft:

Analog zur Vorlesung: $M_t(m) \subseteq M_t(m+1), \qquad m = 1, \ldots, n-1$

t	1	2	3	4	5	6	7	8	9	10
r_t	1	2	3	4	1	3	2	2	3	1
$M_t(m=4)$	1	2	3	4	4	4	4	4	4	4
	-	1	2	3	3	3	3	3	3	3
	-	-	1	2	2	2	2	2	2	2
	-	-	-	1	1	1	1	1	1	1
$M_t(m=3)$	1	2	3	4	1	1	1	1	1	1
	-	1	2	3	3	3	3	3	3	3
	-	-	1	1	2	2	2	2	2	2
$M_t(m=2)$	1	2	3	4	4	3	3	3	3	1
	-	1	1	1	1	1	2	2	2	2
$M_t(m=1)$	1	2	3	4	1	3	2	2	3	1

man sieht hier leicht, dass die Inklusionseigenschaft erfüllt ist. Als nächstes folgt die Herleitung der Prioritätsliste π_t mit dem Kriterium “wachsende Vorwärtsdistanz” und daraus die Herleitung des Stacks s_t:

t	1	2	3	4	5	6	7	8	9	10
r_t	1	2	3	4	1	3	2	2	3	1
π_t	1	1	1	1	3	2	2	3	1	1
	-	2	3	3	2	3	3	1	3	3
		-	2	2	1	1	1	2	2	2
			-	4	4	4	4	4	4	4
s_t	1	2	3	4	1	3	2	2	3	1
	-	1	1	1	4	1	3	3	2	3
		-	2	3	3	4	1	1	1	2
			-	2	2	2	4	4	4	4
			-	-	-	-	-	-	-	-

Für s_t muss gelten: $M_t(m) = s_t(1), \ldots, s_t(m), \qquad m = 1, \ldots, 4 \ \forall t$

Lehrstuhl für Informatik 6
der RWTH Aachen
Prof. Dr.-Ing. H. Ney

Aachen, den 4. Juli 2011

S. Peitz, M. Freitag, J. Bitsch

Betriebssysteme und Systemsoftware

11. Übung

Abgabe der Lösungen: So., den 10. Juli 2011 im L2P.

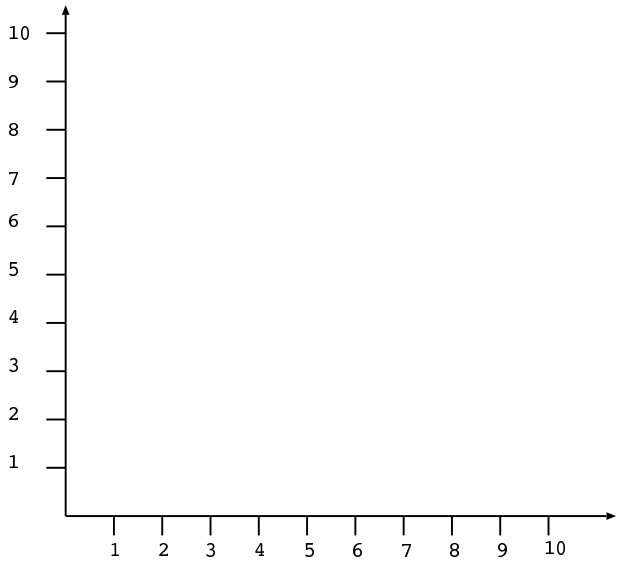
Aufgabe 11.1 Lifetime-Funktion (6 Punkte)

Gesucht wird die Lifetime-Funktion $L(m)$ zu einem Programm, das durch den Referenzstring ω mit

$$\omega = 2\,3\,2\,3\,1\,3\,1\,3\,0\,3\,0\,1\,2\,3\,2\,3\,2\,1\,2\,0\,2\,1\,2\,0$$

gekennzeichnet ist. Die Zeit, die zwischen zwei Seitenanfragen vergeht, soll jeweils eine Zeiteinheit betragen. Die Seitenersetzung erfolge mittels LRU. Gehen Sie davon aus, dass das Working Set zu Beginn leer ist.

Zeichnet den Graphen der Lifetime-Funktion $L(m)$ für $m = 1 \ldots 10$ und gebt zusätzlich die Zahl der Seitenfehler für $m = 1 \ldots 10$ in einer Tabelle an.



Aufgabe 11.2 Working Set (1 + 3 + 2 + 1 = 7 Punkte)

Das Working Set $W(t, h)$ eines Prozesses zur Zeit t ist die Menge der Seiten, die bei den letzten h Zugriffen mindestens einmal referenziert wurden. Wird eine Seite aktuell gebraucht, dann befindet sie sich im Working Set. Wenn sie nicht länger benötigt wird, wird sie nach h Zeiteinheiten aus dem Working Set geworfen.

a) Was sind die negativen Folgen, falls der Wert für h zu hoch bzw. zu niedrig gewählt wird?

b) Es sei der folgende Referenzstring eines Prozesses gegeben:

$$\omega = 12344213566563216$$

Der Wert h soll für diese Aufgabe $h := 6$ betragen! Konstruiert einen Graphen, auf dessen x-Achse die Zeit t (von 0 bis 16) und auf dessen y-Achse die Anzahl der momentan dem Prozess zugeteilten Seiten aufgetragen wird. Geht davon aus, dass das Working Set zu Beginn, also bei $t = 0$, mit keiner Seite gefüllt ist und die Zeit, die zwischen zwei Seitenanfragen vergeht, jeweils eine Zeiteinheit beträgt.

c) Auf dem Graphen sollten sich nun verschiedene lokale Bereiche des Prozesses und die jeweiligen „Phasenwechsel“ (d.h. die Übergänge von einem lokalen Bereich zum nächsten) erkennen lassen. Gebt die Working Sets der lokalen Bereiche an.

d) Was versteht man unter *Thrashing*? Kann es bei Zuteilung von 3 Rahmen zum Thrashing kommen?

Aufgabe 11.3 Bedienprozesse (1 + 1 + 1 + 2 = 5 Punkte)

In der Vorlesung habt ihr Bedienprozesse kennengelernt. Diese werden im Zusammenhang mit der Warteschlangentheorie benutzt, um Abbarbeitungsprozesse zu modellieren und sind ein wichtiges Werkzeug um beispielsweise Server in Rechenzentren richtig zu dimensionieren. In der Vorlesung wurde der spezielle Fall mit einer Queue $c = 1$ vorgestellt.

a) Was bedeuten die Abkürzungen in der Notation $M/M/1$?

b) Und bei $M/D/1$?

c) Wie berechnet man die erwartete Wartezeit?

d) Was versteht man unter der 50% Regel?

Lehrstuhl für Informatik 6
der RWTH Aachen
Prof. Dr.-Ing. H. Ney

Aachen, den 11. Juli 2011

S. Peitz, M. Freitag, J. Bitsch

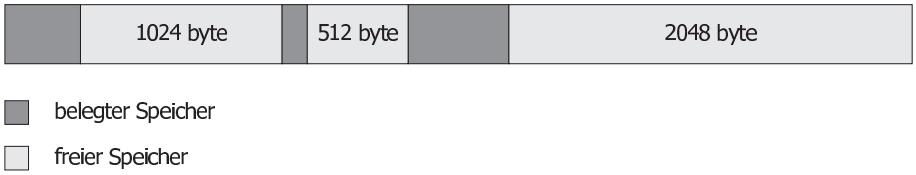
Betriebssysteme und Systemsoftware

12. Übung

Abgabe der Lösungen: im L2P.

Aufgabe 12.1 Speicherbelegung bei Segmentierung (0 Punkte)

Gegeben sei folgende Belegung eines Speichers:



Weiterhin wird Segmentierung zur Belegung des Speichers verwendet und es seien vier Belegungsstrategien für Speicherplatzanforderungen aus der Vorlesung gegeben:

First-Fit	Belege von vorne beginnend den ersten freien Speicherbereich, der groß genug ist, die Anforderung zu erfüllen.
Rotating-First-Fit	Wie First-Fit, jedoch wird vom Ende des zuletzt ausgewählten Speicherbereichs aus ein passender Bereich gesucht. Wird das Ende des Speichers erreicht, so wird die Suche am Anfang des Speichers fortgesetzt. Bei der ersten Anforderung beginnt die Suche am Anfang des Speichers.
Best-Fit	Belege den kleinsten freien Speicherbereich, in den die Anforderung passt.
Worst-Fit	Belege den größten freien Speicherbereich, in den die Anforderung passt.

a) Wie wird obige Speicher belegt, wenn nacheinander vier Anforderungen der Größe 384 Byte, 640 Byte, 512 Byte und 2048 Byte gestellt werden? Notieren Sie für jede der vier Strategien die freien Speicherbereiche nach jeder Anforderung (z.B. in der Form (1024, 512, 2048) für die obige Ausgangsbelegung) und geben Sie an, für welche Strategien die Anforderungen erfüllt werden können!

   Beachten Sie, dass die Daten jeweils linksbündig in einer Lücke abgelegt und einmal belegte Speicherbereiche nicht wieder freigegeben werden!

b) Sie dürfen nun die Reihenfolge der freien Speicherbereiche und die der Anforderungen vertauschen (aber keine Speicherbereiche zusammenfassen). Geben Sie für jede der vier Strategien eine Speicherbelegung und eine Anforderungsreihenfolge an, die für jeweils nur diese Strategie erfolgreich arbeitet und für die übrigen Strategien nicht alle Anforderungen erfüllen kann. Geben Sie auch jeweils für die erfolgreiche Strategie die Speicherbelegung nach Abarbeitung der einzelnen Anforderungen an.

Aufgabe 12.2 Deadlocks (0 Punkte)

Betrachten Sie in dieser Aufgabe das Problem der Erkennung von Deadlocks.

a) Gegeben sei ein System mit drei Prozessen $P_1, \ldots, P_3$ und vier Betriebsmitteln $R_A, \ldots, R_D$, von denen jeweils nur ein Exemplar zur Verfügung steht und die nur von einem Prozess gleichzeitig verwendet werden können. Die Anforderungen von Betriebsmitteln durch die Prozesse ist in der folgenden Tabelle gegeben.

Zeit	P_1	P_2	P_3
1	(A;5)		
2		(B;5)	
3			(D;3)
4			(C;5)
5	(D;1)		(B;3)
6		(A;2)	
7			

Ein Eintrag der Form $(X;a)$ in der Spalte P_i bedeutet, dass das Betriebsmittel R_X für a Zeiteinheiten vom Prozess P_i angefordert wird. Ein Prozess kann seine Arbeit erst dann fortsetzen, wenn ihm das jeweils angeforderte Betriebsmittel zugewiesen wurde.

Sind einem Prozess die angeforderten Betriebsmittel zum Zeitpunkt t zugewiesen worden, werden sie in der Zeit von $t+1$ bis $t+a$ verwendet und stehen zum Zeitpunkt $t+a+1$ wieder zur Verfügung.

Stellen Sie die Situation aus der obigen Tabelle für den Zeitpunkt $t = 7$ anhand eines Resource Allocation Graph dar. Liegt ein Deadlock vor? Begründen Sie Ihre Antwort.

b) Zwei Prozesse A und B benötigen zu ihrer Ausführung je eine gewisse Zeitdauer t_A bzw. t_B und greifen auf die drei Betriebsmittel X, Y und Z zu. t_{IJ} bezeichne die Dauer, die das Betriebsmittel I vom Prozess J irgendwann während der Ausführungszeit benötigt wird. Ein Experte macht nun folgende Aussagen:

i) Ein Deadlock kann nicht auftreten, wenn $t_{XA} \cdot t_{XB} + t_{YA} \cdot t_{YB} + t_{ZA} \cdot t_{ZB} < t_A \cdot t_B$.

ii) Ein Deadlock liegt automatisch vor, wenn $t_{XA} + t_{YA} + t_{ZA} > t_A$ und $t_{XB} + t_{YB} + t_{ZB} > t_B$.

iii) Ein Deadlock liegt automatisch vor, wenn $t_{XA} \cdot t_{XB} + t_{YA} \cdot t_{YB} + t_{ZA} \cdot t_{ZB} > t_A \cdot t_B$.

iv) Ein Deadlock kann nicht auftreten, wenn $t_{XA} + t_{YA} + t_{ZA} < t_A$ und $t_{XB} + t_{YB} + t_{ZB} < t_B$.

Welche dieser Aussagen sind richtig und welche Aussagen stimmen nicht? Begründen Sie zu jeder der einzelnen Aussagen, warum diese gültig sind, oder geben Sie zu jeder Aussage, die nicht gültig ist, ein Gegenbeispiel an (z.B. durch ein Prozessfortschrittsdiagramm).

Aufgabe 12.3 Speicherverwaltung (0 Punkte)

Der Referenz String ω beschreibe die Folge der Seitenzugriffe. Die Tabellen zeigen die Belegung der Seitenrahmen im Hauptspeicher. Es stehen einmal drei und einmal vier Rahmen zur Verfügung. Vervollständigen Sie die Tabellen unter Verwendung der Second Chance Strategie. Markieren Sie die Seitenfehler mit einem $\star$.

Aufgabe 12.4 Stackalgorithmus (0 Punkte)

Zu den Seitenzugriffen eines Programms sei der folgende Referenzstring ω gegeben:

$$\omega = 6\ 5\ 4\ 3\ 2\ 5\ 6\ 3\ 1$$

a) Zeigen Sie anhand des gegebenen Referenzstrings, dass die Seitenersetzungsstrategie LIFO die Inklusionseigenschaft erfüllt.

b) Geben Sie die Prioritätsliste π_t und die resultierende Stack-Belegungsfolge s_t zum Referenzstring ω für die Zeitpunkte $t = 1, \ldots, 9$ an.

Aufgabe 12.5 Working Set (0 Punkte)

Es sei der folgende Referenzstring eines Prozesses gegeben:

$$\omega = 2\ 1\ 5\ 0\ 3\ 4\ 2\ 3\ 2\ 2\ 5\ 6\ 1\ 2\ 3\ 4\ 5\ 6\ 0\ 1\ 6\ 5\ 6\ 1$$

Der Wert h soll für diese Aufgabe $h := 5$ betragen.

a) Konstruieren Sie einen Graphen, auf dessen x-Achse die Zeit t (von 0 bis 24) und auf dessen y-Achse die Anzahl der momentan dem Prozess zugeteilten Seiten aufgetragen wird. Gehen Sie davon aus, dass das Working Set zu Beginn, also bei $t = 0$, mit keiner Seite gefüllt ist und die Zeit, die zwischen zwei Seitenanfragen vergeht, jeweils eine Zeiteinheit beträgt.

b) Geben Sie die Working Sets der lokalen Bereiche zu den Zeitpunkten 10, 15, 19 und 24 an.

Aufgabe 12.6 Lifetime-Funktion (0 Punkte)

Geben Sie die Lifetime-Funktion $L(m)$ zu einem Programm an, das durch den Referenzstring ω mit

$$\omega = 2\,3\,2\,3\,1\,3\,1\,3\,0\,3\,0\,1\,2\,3\,2\,3\,2\,1\,2\,0\,2\,1\,2\,0$$

gekennzeichnet ist. Die Zeit, die zwischen zwei Seitenanfragen vergeht, soll jeweils eine Zeiteinheit betragen. Die Seitenersetzung erfolge mittels LFU-Strategie (Least Frequently Used). Für diese Aufgabe wird angenommen, dass die Nutzung ab Beginn des Referenzstrings gezählt wird. Gehen Sie davon aus, dass das Working Set zu Beginn mit anderen als den angegebenen Seiten gefüllt ist.

Zeichnen Sie den Graphen der Lifetime-Funktion $L(m)$ für $m = 1 \ldots 10$ und geben Sie zusätzlich die Zahl der Seitenfehler für $m = 1 \ldots 10$ in einer Tabelle an.

Aufgabe 12.7 UNIX Inodes (0 Punkte)

In der Vorlesung wurde die indizierte Belegung von UNIX-Dateisystemen (z.B. ext2) vorgestellt. Berechnen Sie die Größe der Daten, die mit den direct / indirect / double indirect und triple indirect Pointern referenziert werden können. Die Blockgröße sei 2 KByte und die Pointeröße sei 2 Bytes. Wieviel Speicherplatz brauchen die Pointer?

Aufgabe 12.8 Einfache Scheduling-Strategien (1+1+1+2 = 5 Punkte)

In der Vorlesung habt ihr einige Scheduling-Strategien kennen gelernt (FCFS/LCFS/SJF/SRTF). In den folgenden Aufgaben geht es darum diese auf Daten anzuwenden.

In der Warteschleife (Queue) seien N Jobs mit Bearbeitungszeiten $t_1 ... t_N$. Index n gibt die Reihenfolge des Ankommens in der Queue an: d.h. Job P_n (mit Bearbeitungszeit t_n) kommt vor Job P_{n+1} (mit Bearbeitungszeit t_{n+1}) an. Folgende Jobs warten in der Queue darauf, abgearbeitet zu werden:

Job	Bearbeitungszeit (t_n)
P_1	5
P_2	3
P_3	6
P_4	2
P_5	4
P_6	6

a) Wie werden die Jobs nach FCFS abgearbeitet und wie groß ist die durchschnittliche Wartezeit?

b) Wie werden die Jobs nach LCFS abgearbeitet und wie groß ist die durchschnittliche Wartezeit?

c) Wie werden die Jobs nach SJF abgearbeitet und wie groß ist die durchschnittliche Wartezeit?

d) Nun haben wir zusätzlich zur Bearbeitungszeit noch eine Ankunftszeit. Das heißt die Jobs können erst ab dem Zeitpunkt abgearbeitet werden, ab dem sie auch angekommen sind.

Job	Ankunftszeit	Bearbeitungszeit (t_n)
P_1	00	5
P_2	01	3
P_3	02	6
P_4	03	4
P_5	04	3
P_6	11	2

Wie werden die Jobs jetzt nach SRTF (Shortest Remaining Time First) abgearbeitet und wie groß ist die durchschnittliche Wartezeit?

Hinweis: Zur Abgabe orientiert euch an den Bildern der Vorlesung (Folie 45).