Midterm

donn, ZHAW Dauer: 45 Minuten

Name:
Klasse:

Aufgabe	Max. Punkte	Punkte
1	1.0	
2	1.0	
3	2.0	
4	1.5	
5	2.0	
6	2.0	
7	3.0	
Total	12.5	
	Note	

Hinweise

- Korrektur: Resultate werden nur als richtig oder falsch bewertet - bei Multiple Choice Aufgaben kann pro Teilaufgabe mehr als eine Antwort zutreffen, eine Teilaufgabe gilt als richtig, wenn sämtliche Antworten der Teilaufgabe korrekt sind.

Hilfsmittel

- erlaubt sind Taschenrechner, 2 Blätter selbstverfasste Notizen - nicht erlaubt sind PCs (Laptops), Handhelds/xPads und Handys.

Bedingungen

- kein Bleistift, keine roten Stifte und kein TipEx
- nur die Resultate auf den Aufgabenblättern eintragen
- Zusatzblätter werden nicht eingesammelt oder korrigiert
- Resultate als ganze Zahlen resp. mit 2 Stellen nach dem Komma

Aufgabe 1 Cache Zugriffe 1.0P

Gegeben ist folgender Programmausschnitt:

```
...
#define N (10*1000*1000)
...
int arVL[N];
...
for (int i = 0; i < N; i++) {
  sum += arVL[i];
}
...
```

Wie gross ist auf einem 32-Bit Prozessor die Hitrate h_c während dem Zugriff auf arV[i], wenn das Data Cache eine Blockgrösse (Cache Line Size) von 64 Bytes hat? Hinweis: es genügt die Überlegung für ein Rechnersystem mit einem Cache Level zu machen.

h_c =

Aufgabe 2 Speicherzugriffszeit 1.0P

Ein 2 GHz Prozessor (2 GHz entspricht einem Clockzyklus von 0.5 ns) hat eine dreistufige Cache Hierarchie, wobei folgende Daten gegeben sind:

Cache Level 1 Zugriffszeit 4 Clockzyklen
Cache Level 2 Zugriffszeit 10 Clockzyklen
Cache Level 3 Zugriffszeit 40 Clockzyklen
Hauptspeicher Zugriffszeit 60 ns

Wie gross ist die mittlere Speicherzugriffszeit T_a, wenn die Hitrate auf jedem Level 90% beträgt?

T_a =

Aufgabe 3 Make

a) 1.0P, b) 1.0P

Gegeben ist folgendes Makfile:

```
CFL = -g
CMP = gcc $(CFL)
app:      main1.o mythread.o scheduler.o queues.o mylist.o
          $(CMP) main1.o mythread.o scheduler.o queues.o mylist.o -o $@.e
.c.o:
          $(CMP) -c $<
```

und ein Listing (ls -al) des Verzeichnisses in dem make ausgeführt wird:

```
total 144
drwxr-xr-x 2 usr usr 4096 Feb 24 09:21 ./
drwxr-xr-x 3 usr usr 4096 Feb 24 09:32 ../
-rw-r--r-- 1 usr usr 1894 Feb 24 09:16 main1.c
-rw-r--r-- 1 usr usr 4220 Feb 24 09:21 main1.o
-rw-r--r-- 1 usr usr 642 Feb 24 09:19 makefile
-rw-r--r-- 1 usr usr 4152 Feb 24 09:31 mylist.c
-rw-r--r-- 1 usr usr 2317 Feb 24 09:16 mylist.h
-rw-r--r-- 1 usr usr 6428 Feb 24 09:21 mylist.o
-rw-r--r-- 1 usr usr 3975 Feb 24 09:16 mythread.c
-rw-r--r-- 1 usr usr 5224 Feb 24 09:16 mythread.h
-rw-r--r-- 1 usr usr 7448 Feb 24 09:21 mythread.o
-rw-r--r-- 1 usr usr 5264 Feb 24 09:25 queues.c
-rw-r--r-- 1 usr usr 891 Feb 24 09:16 queues.h
-rw-r--r-- 1 usr usr 7648 Feb 24 09:21 queues.o
-rw-r--r-- 1 usr usr 11135 Feb 24 09:16 scheduler.c
-rw-r--r-- 1 usr usr 2434 Feb 24 09:16 scheduler.h
-rw-r--r-- 1 usr usr 10432 Feb 24 09:21 scheduler.o
```

In diesem Verzeichnis wird nun der Befehl make ausgeführt:

a) Geben Sie die Namen derjenigen C-Files an, die durch den Aufruf von make übersetzt werden:

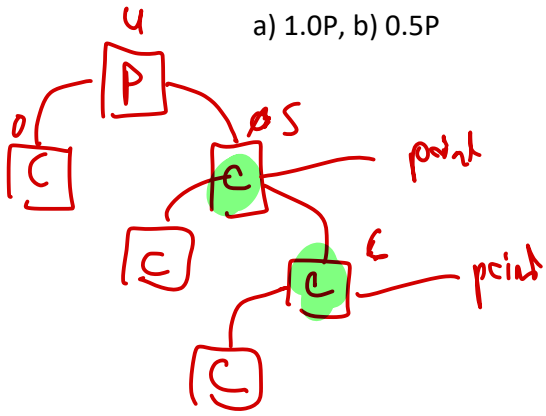
b) Wie heisst das ausführbare File, das durch den Aufruf von make erzeugt wird:

Aufgabe 4 Prozesserzeugung

a) 1.0P, b) 0.5P

Gegeben ist folgender Codeausschnitt:

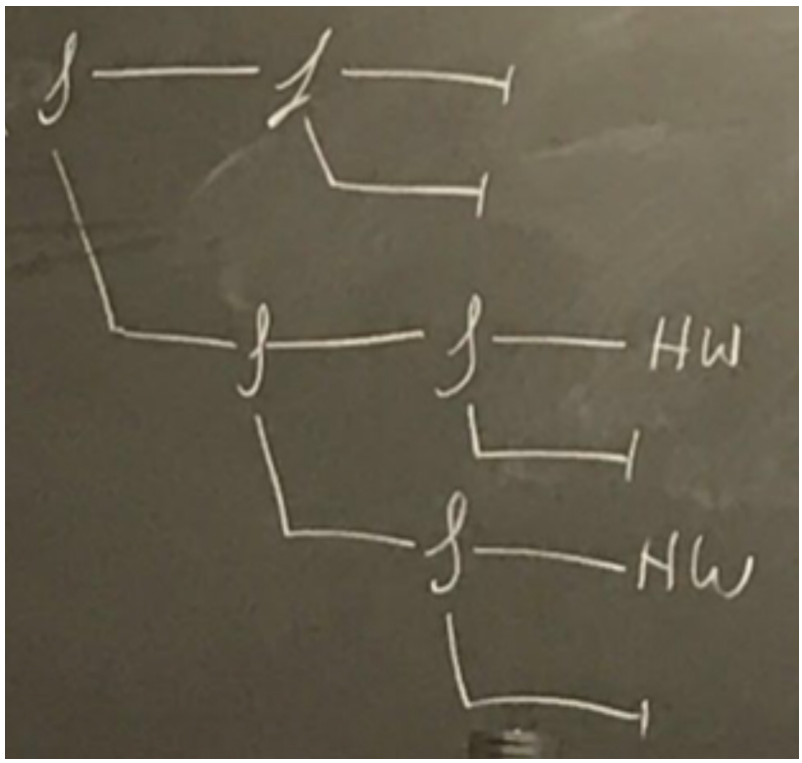
```
if (fork() > 0)
        fork();
else {
        fork();
        if (fork() > 0)
        printf("Hello World\n");
}
```



Annahme: alle fork()'s werden erfolgereich ausgeführt

a) Wie viele Prozesse werden mit diesem Codesausschnitt **zusätzlich** erzeugt?
Anzahl 5

b) Wie oft wird Hello World ausgegeben?
Anzahl: 2

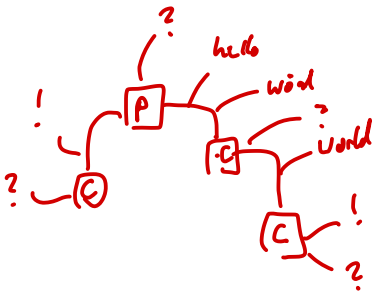


Aufgabe 2 Prozesserzeugung

Gegeben ist unten stehendes Programm:

```
int main(void) {
    printf("hello "); fflush(stdout);
    fork();
    printf("world "); fflush(stdout);
    if (fork() == 0)
        printf("! ");
    fflush(stdout);
    printf("? ");
}
```

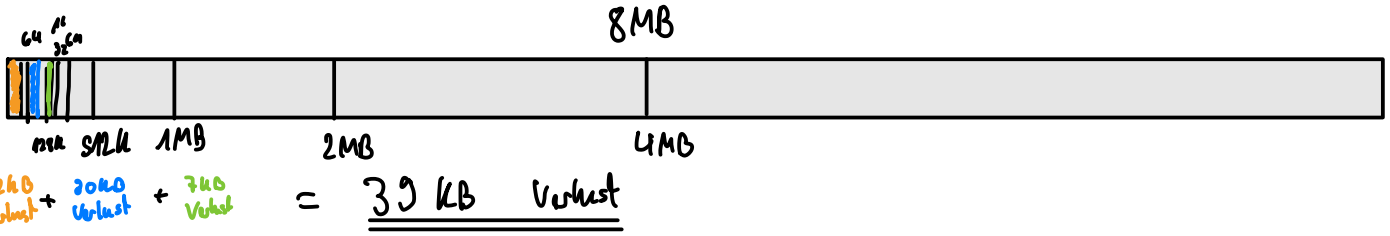
Was gibt das Programm aus, wenn alle `fork()`-Calls erfolgreich sind? ausreichend.



hello world world!! ????!

Aufgabe 4 Buddy System 1.0 P

Ein Betriebssystem-Kernel verwaltet seine Datenbuffer mit einem Buddy System, wobei insgesamt 8MByte Speicher zur Verfügung stehen. Zur Zeit sind folgende Buffer mit 62KByte, 34KByte und 9KByte alloziert worden. Wie viel Speicher geht dabei durch interne Fragmentierung insgesamt verloren, Angabe in KByte:



2kB Verlust + 30kB Verlust + 7kB Verlust = 39 kB Verlust

Aufgabe 5 Page Tabellen

Ein Prozessor besitzt eine Wortbreite von 32Bit. Pointer (Adressen) werden in 32Bit Worten gespeichert, aber nur die 24 tieferwertigen Bits werden für die Adressbildung verwendet (Bits 25-31sind auf 0 gesetzt).
Die logische Adresse ist wie folgt strukturiert:

6-Bit Page Directory	8-Bit Page Nummer	10-Bit Offset

a) Wie gross ist eine Page, Angabe in KBytes?

10 Bit → 1024 Adressen
~~32 Bit = 4 Byte pro Adresse~~ → 1 kB Page size

b) Wie viele Bytes enthält das Page Directory, wenn pro Eintrag ein Pointer (Adresse) auf eine Page Tabelle eingetragen wird, Angabe in KBytes?

6 Bit → 64 Adressen
4 Byte pro Adresse → 64 · 4 = 256 Bytes

c) Wie viele Page Tabellen kann ein Prozess maximal haben?

6 Bit → 64 Page Tabellen

d) Wie viele Frames kann das System maximal haben?

Aufgabe 6 Page Replacement 2.0 P

Ein Prozess referenziert der Reihe nach folgende Pages:

8 7 5 8 7 3 5 1 3 4 2 1 8 3 1 2

Gehen Sie davon aus, dass zu Beginn keine Pages im Speicher stehen und dass auch das erstmalige Laden einer Page als Page Fault gezählt wird (demand paging). Pro Prozess stehen 4 Frames zur Verfügung.

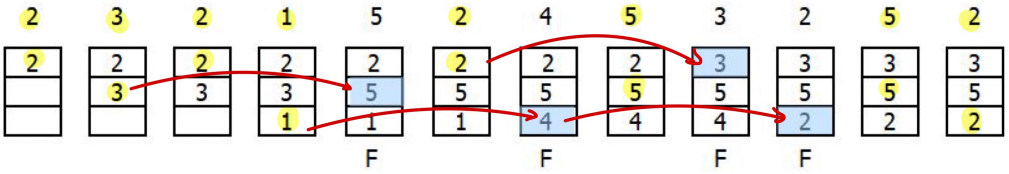
Tragen Sie in unten stehender Tabelle die den Frames zugewiesenen Pages für den **Least Recently Used** Algorithmus. Das Page dessen Zugriff am weitesten Zeitlich zurückliegt wird dann mit ein neues Page ersetzt.

Markieren Sie die Spalten mit einem Stern, wo ein Page Fault auftritt. Nehmen Sie an, dass ausschliesslich Demand Paging verwendet wird. Wenn mehrere Frames für das placement resp. replacement in Frage kommen, muss der Frame mit der kleinsten Nummer gewählt werden.

Referenzen	8	7	5	8	7	3	5	1	3	4	2	1	8	3	1	2
frame 1	8	8	8	8	8	8	8	1	1	1	1	1	1	1	1	1
frame 2		7	7	7	7	7	7	7	7	4	4	4	4	3	3	3
frame 3			5	5	5	5	5	5	5	5	2	2	2	2	2	2
frame 4						3	3	3	3	3	3	3	8	8	8	8
page fault	X	X	X			X		X		X	X		X	X		

neu geladen → compulsory page fault

Least recently used (Linux approach) in Vergangenheit schauen, welche Page verwendet wurde



later: 3 Schritte zurück

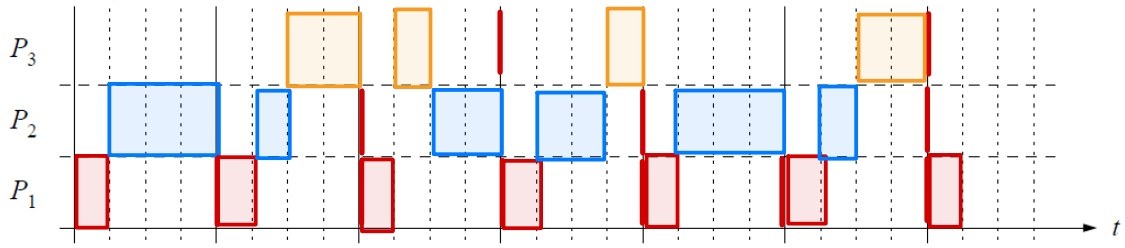
Aufgabe 5 Real Time Scheduling 2.0P

Gegeben sind 3 periodische Echtzeit-Tasks mit folgenden Daten:

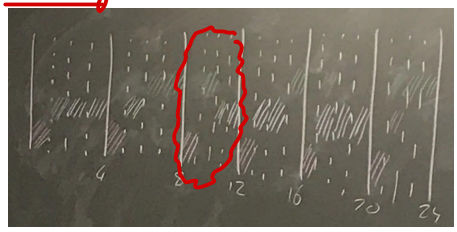
Task	Periode (Prio)	Rechenzeit
1	4ms 1.	1ms
2	8ms 2.	4ms
3	12ms 3.	3ms

Zeichnen Sie das Balkendiagramm für Deadline-Scheduling mit folgenden Bedingungen:
- Re-Scheduling: alle 4 *ms* oder wenn sich ein Task suspendiert
- bei gleichen Deadlines hat der Task mit der kürzeren Periode höhere Priorität

Pre-emptive:



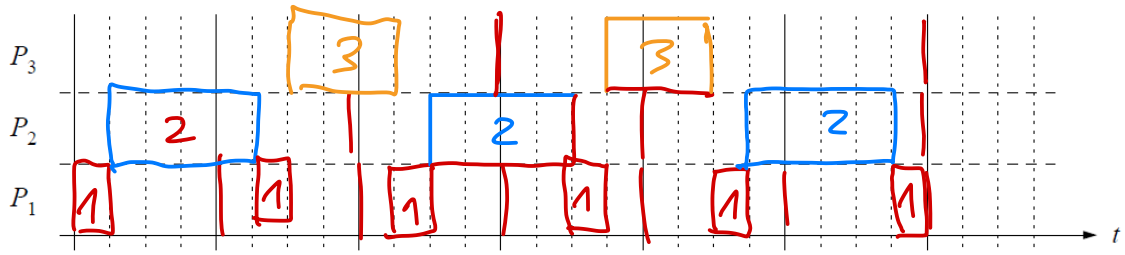
Lösung

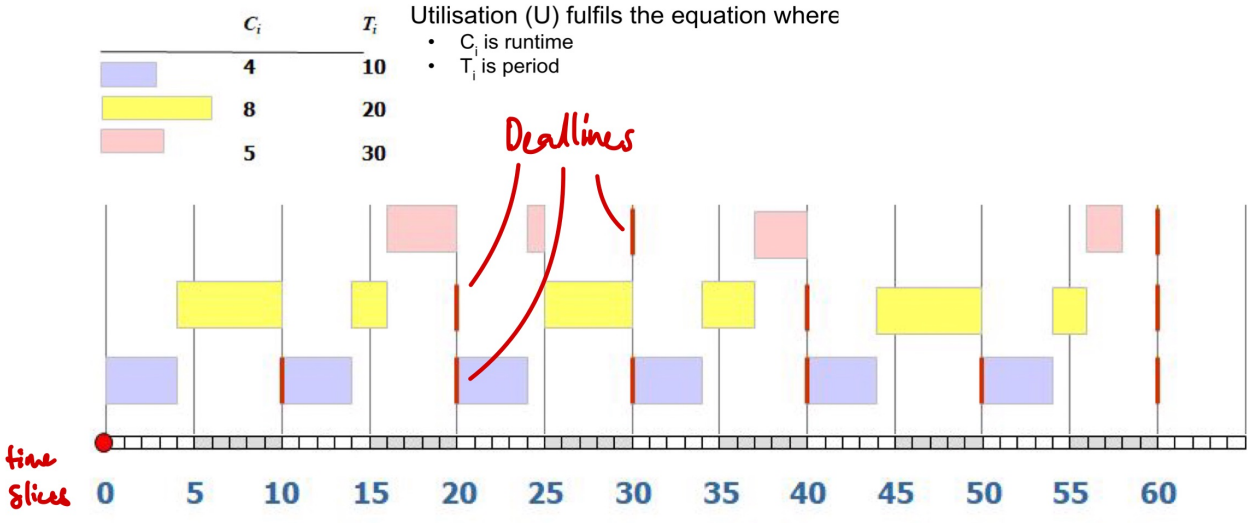


Non - Pre-emptive (Nicht richtig!)

Sollte beim ersten mal Zeichnen etwas schief gelaufen sein:

Priority Queue



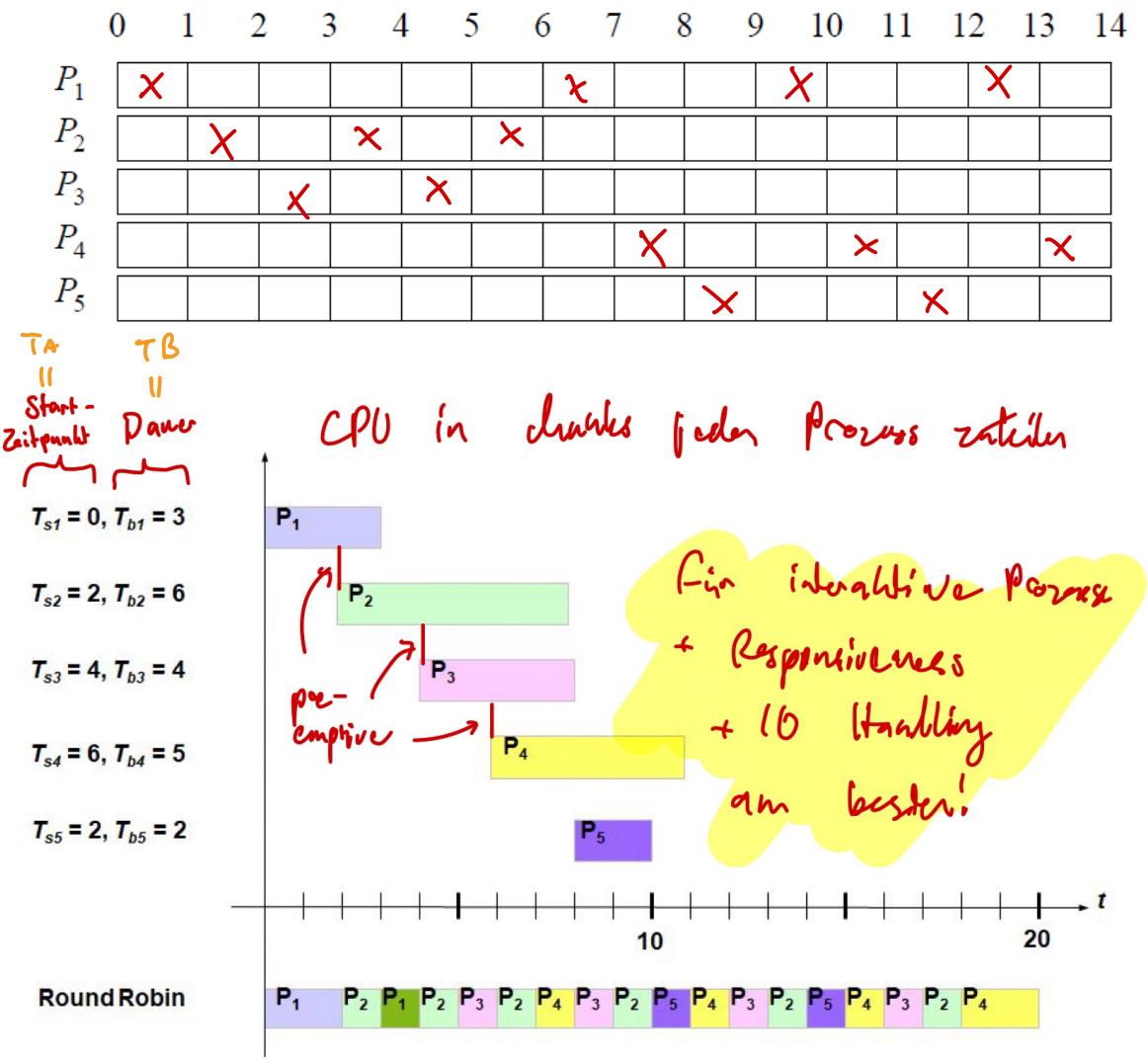


Aufgabe 6 Scheduling 2.0P

Gegeben sind 5 Prozesse mit folgenden Prioritäten (hohe Zahl -> hohe Priorität), Ankunftszeiten T_A und Ausführungszeiten T_B:

Prozess	Priorität	Ankunftszeit TA	Ausführungszeit TB
P1	0	0	4
P2	1	1	3
P3	1	2	2
P4	0	5	3
P5	0	7	2

Annahmen: Multilevel Scheduling (nicht Multilevel Feedback), kein Prozess blockiert, RR mit Timeslot $q = 1$
Skizzieren Sie den Schedule als Balkendiagramm in folgendem Zeitraster:



Aufgabe 7 Kurzfragen

0.5P pro Antwort, total 3P

Pro Teilfrage können eine, mehrere oder keine Antworten zutreffen, kreuzen Sie die richtige(n) Antwort(en) an.

Bewertung: wenn Sie eine Teilfrage richtig beantwortet haben (sämtlichen Kästchen korrekt), erhalten Sie 0.5 Punkte sonst 0 Punkte.

a) Ein Programm führt unten stehende Aktivitäten aus. Für welche dieser Aktivitäten muss vom Betriebssystem ein System-Call zur Verfügung gestellt werden?

- den Wert einer Variablen auf dem Bildschirm ausgeben	?
- den Wert von zwei Variablen vergleichen	
- 1 Sekunde schlafen	X
- den Wert einer Integer Variablen inkrementieren	
- den Wert einer Fliesskomma Variablen um eins erhöhen	

b) Die Systemfunktionen pthread join()

- terminiert alle User-Level Threads	
- kann jederzeit durch sleep(0) ersetzt werden	
- verhindert, dass der Prozess vor Beendigung der Threads terminiert	X
- blockiert die CPU, bis alle Threads terminiert haben	

c) Wieso werden System-Calls werden über Softwareinterrupts aufgerufen?

- weil sie schneller sind als Prozeduraufrufe	
- weil Prozeduraufrufe (CALL) nicht in den System Mode umschalten können	X
- weil Anwenderprogramme die Adressen der System-Call Routinen nicht kennen	(x)

d) Der PCB ist ein Datenbereich

- in dem ein Prozesse seine globalen Daten ablegt	
- vom Betriebssystem Statusinformation zum Prozesse abgelegt wird	X
- ein Prozesse kleine Datenmengen effizient zwischenspeichern kann	
- die Prozessidentifikationsnummer abgespeichert ist	

e) Ein Zombie ist ein

- Thread, der zu keinem Prozess mehr gehört	
- ein Prozess, der keinen Elternprozess hat	
- ein Prozess, der als Eltern den init-Prozess hat	
- auf den der Elternprozess nicht gewartet hat (wait(), waitpid())	✗
- ein Prozess, beim dem alle gestarteten Threads terminiert haben	

f) Ein Daemon ist

- ein Virus auf Linuxsystemen	
- ein Prozsse ohne Kontrollterminal	
- ein Prozess, der eine Shell als Elternprozess hat	✗

b) Welcher der folgenden Aussagen treffen zu?

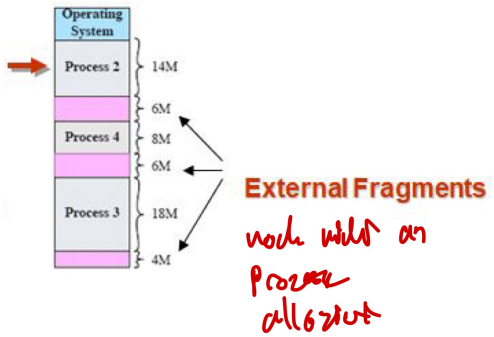
- Pages müssen grosser als Frames dimensioniert werden (gleichgross)
- Es müssen mindestens so viele Pages wie Frames in einen System vorhanden sein → falls nicht haben wir einfach nicht benutzts Memory
- Sowohl interne wie auch externe Fragmentierung treten bei Paging auf (Ja)
- Pages und Frames müssen gleich gross dimensioniert werden (Nein)

c) Welcher der folgenden Aussagen treffen zu?

- Ein MMU übersetzt Logischen Adressen zu Physikalische Adressen
- Swap in bedeutet dass ein Prozess auf die Hard-Disk verlagert wird
- Ein Prozess der auf der Harddisk verlagert worden ist kann sich im Zustand Running befinden Teile vom ~~the~~ running Prozess können aber ausgelagert sein, der Prozess selber aber nicht

d) Welcher der folgenden Aussagen treffen zu?

- Bei Static Partitioning tritt External Fragmentation auf (Internal tritt auf)
- Bei Dynamic Partitioning tritt External Fragmentation auf
- Nur bei Best Fit Allocation wird kein Compaction benötigt → kann man immer brauchen



noch nicht an Prozess allokiert