1 Prozesse und Scheduling (13 Punkte)

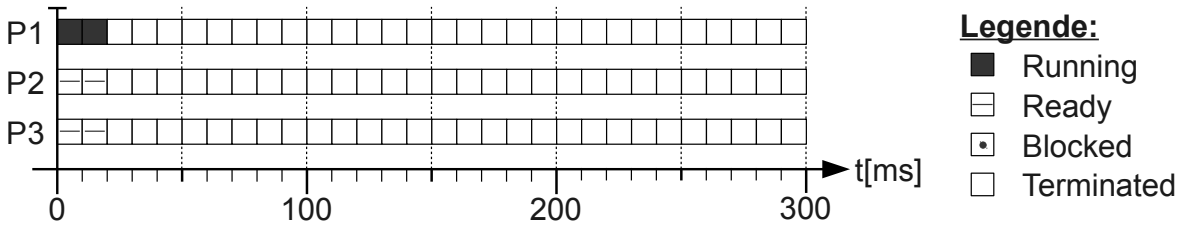
a) Round Robin (6 Punkte) Ein Betriebssystem verwaltet die drei Prozesse P1, P2 und P3. Die Prozesse treffen in dieser Reihenfolge im System ein und sind alle zum Zeitpunkt t=0 rechenbereit. Die Bedienzeiten der Prozesse und die Zeitpunkte und Dauer von E/A-Operationen sind in der folgenden Tabelle angegeben:

Prozess	P1	P2	P3
Bedienzeit	70 ms	110 ms	90 ms
E/A-Zeitpunkt	20 ms	30 ms	60 ms
E/A-Dauer	80 ms	30 ms	50 ms

Bei der „Bedienzeit“ handelt es sich um die reine Rechenzeit. Für die Gesamtzeit kommt noch die Dauer der E/A-Operationen hinzu. Der E/A-Zeitpunkt ist relativ zur Bedienzeit angegeben. Prozess P1 beginnt also nach 20 ms Rechenzeit mit seinen E/A-Operationen, Prozess P2 wird nach seinen ersten 30 ms blockiert, etc.

Zeichnen Sie in das folgende Gantt-Diagramm ein, wie die drei Prozesse P1, P2 und P3 bei Scheduling nach der *„Round Robin“*-Strategie mit einer Zeitscheibendauer von *40 ms* abgearbeitet werden. Jeder Prozess führt *genau einen* E/A-Vorgang durch. Zeitpunkt und Dauer sind in der Tabelle angegeben. Die Prozessumschaltzeit kann vernachlässigt werden. Markieren Sie in dem folgenden Diagramm die Prozesszustände entsprechend der Legende.

Hinweis: Die ersten 20 ms sind bereits fertig ausgefüllt.



b) Prozesserzeugung (7 Punkte) Was geben die folgenden C-Programme aus? Wieso ist die Ausgabe unterschiedlich? Fehlerabfragen und die *#include*-Zeilen zur Einbindung der Systemheader wurden der Einfachheit halber weggelassen. Gehen Sie davon aus, dass alle Systemaufrufe fehlerfrei abgearbeitet werden und keine Race Condition auftritt.

```
1  int pferd;
2
3  void* erzeugePferd(void* param) {
4          pferd++;
5          printf("%d\n",pferd);
6
7          return NULL;
8  }
9
10 int main(void) {
11         pferd = 42;
12
13         if (fork() == 0) {
14                 erzeugePferd(NULL);
15         } else {
16                 erzeugePferd(NULL);
17         }
18         return 0;
19 }
```

Ausgabe:

```
1  int pferd;
2
3  void* erzeugePferd(void* param) {
4          pferd++;
5          printf("%d\n",pferd);
6
7          pthread_exit(NULL);
8  }
9
10 int main(void) {
11         pferd = 42;
12         pthread_t id;
13
14         pthread_create(&id,
15                 NULL,
16                 erzeugePferd,
17                 NULL);
18         erzeugePferd(NULL);
19         return 0;
20 }
```

Ausgabe:

2 Synchronisation und Verklemmungen (14,5 Punkte)

a) Gegeben ist ein Programm, das mittels zwei Semaphoren einen Schreiber gegen mehrere Leser absichern soll. Zu Beginn ist der gemeinsame Speicher leer. Initialisieren Sie die Variable readcount und die Semaphoren `mutex` und `wrt` geeignet. Verwenden Sie die Semaphor-Operationen `signal` und `wait`, um die Prozesse gegeneinander abzusichern. Es soll möglich sein, dass mehrere Leser gleichzeitig arbeiten, der Schreiber währenddessen aber blockiert ist. Ebenso dürfen Leseprozesse nur arbeiten, wenn kein Schreiber aktiv ist. **(10 Punkte)**

```
/* Gemeinsamer Speicher */
Semaphore mutex = [      ];
Semaphore wrt = [      ];
int readcount = [      ];
```

```
void schreiber() {
    [                    ]
    schreibe_Daten();
    [                    ]
}
```

```
void leser() {
    [                    ]
    readcount = [                    ]
    if ( readcount == [          ] ) {
        [                    ]
    }
    [                    ]
    lese_Daten();
    [                    ]
    readcount = [                    ]
    if ( readcount == [          ] ) {
        [                    ]
    }
    [                    ]
}
```

b) Verklemmungen (4,5 Punkte) Wenn eine geschlossene Kette wechselseitig wartender Prozesse existiert (*circular wait*, also ein Zyklus im Betriebsmittelbelegungsgraphen), liegt eine Verklemmung vor. Nennen Sie stichpunktartig die drei *Vorbedingungen*, die erfüllt sein müssen, damit es überhaupt zu einer Verklemmung kommen kann, und erklären Sie diese jeweils kurz mit eigenen Worten.

3 Speicherverwaltung und Virtueller Speicher (8 Punkte)

a) Adressabbildung (4 Punkte) In einem System mit Seitenadressierung (*paging*) befindet sich die Seitenkacheltabelle im unten angegebenen Zustand. Die Adresslänge beträgt 16 Bit, wovon 12 Bit für den Offset innerhalb der Seite verwendet werden (Seitengröße: 4096 Bytes). Bilden Sie die logischen Adressen $6AB1_{16}$ und $F1B7_{16}$ auf ihre physikalischen Adressen ab. (Hinweis: Eine Hexadezimalziffer stellt immer genau vier Bit der Adresse dar.)

Seitennummer	Startadresse
0	$F000_{16}$
1	3000_{16}
2	8000_{16}
3	1000_{16}
4	$C000_{16}$
5	2000_{16}
6	4000_{16}
7	$B000_{16}$
...	...
15	5000_{16}

logische Adresse: $6AB1_{16}$
$\rightarrow$ physikalische Adresse: ☐

logische Adresse: $F1B7_{16}$
$\rightarrow$ physikalische Adresse: ☐

b) Buddy-Verfahren (4 Punkte) Die jeweils zweite Zeile der folgenden Szenarien zeigt die momentane Speicherbelegung eines Arbeitsspeichers der Größe 32 MiB. Ergänzen Sie die folgenden Tabellen um Markierungen für die vorgegebenen Anfragen. Nutzen Sie das *Buddyverfahren* zur dynamischen Speicherverwaltung.
Hinweis: Falls eine Belegung/Freigabe *nicht* erfüllt werden kann, kennzeichnen Sie die betreffende Zeile geeignet.

Szenario 1: Prozess C belegt 3 MiB

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
A	A	A	A							B	B				

Szenario 2: Prozess D belegt 12 MiB

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
A	A														

Szenario 3: Prozess E belegt 14 MiB

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
		B	B									A	A		

Szenario 4: Prozess F belegt 7 MiB

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
A	A	A	A	B	B										

4 Ein-/Ausgabe und Dateisysteme (9,5 Punkte)

a) Block-Buffer-Cache (3 Punkte) Nennen und erläutern Sie drei Ereignisse, die das Rückschreiben des Block-Buffer-Caches auslösen.

b) Kontinuierliche Speicherung (2 Punkte) Nennen Sie je einen Vor- und einen Nachteil der kontinuierlichen Speicherung von Dateien.

c) I/O-Scheduling (4,5 Punkte) Gegeben sei ein Plattenspeicher mit 8 Spuren. Der dazugehörige I/O-Scheduler bekommt immer wieder Leseaufträge für eine bestimmte Spur. Die Leseaufträge in L_1 sind dem I/O-Scheduler bereits bekannt. Nach drei bearbeiteten Aufträgen erhält er die Aufträge in L_2. Nach weiteren drei (d.h. nach insgesamt sechs) bearbeiteten Aufträgen erhält er die Aufträge in L_3. Zu Beginn befindet sich der Schreib-/Lesekopf über Spur 0.
$L_1 = \{1, 4, 7, 2\}, L_2 = \{3, 6, 0\}, L_3 = \{5, 2\}$
Der I/O-Scheduler arbeitet nach der **Fahrstuhlstrategie** (Elevator). Tragen Sie die Reihenfolge der gelesenen Spuren ein.
