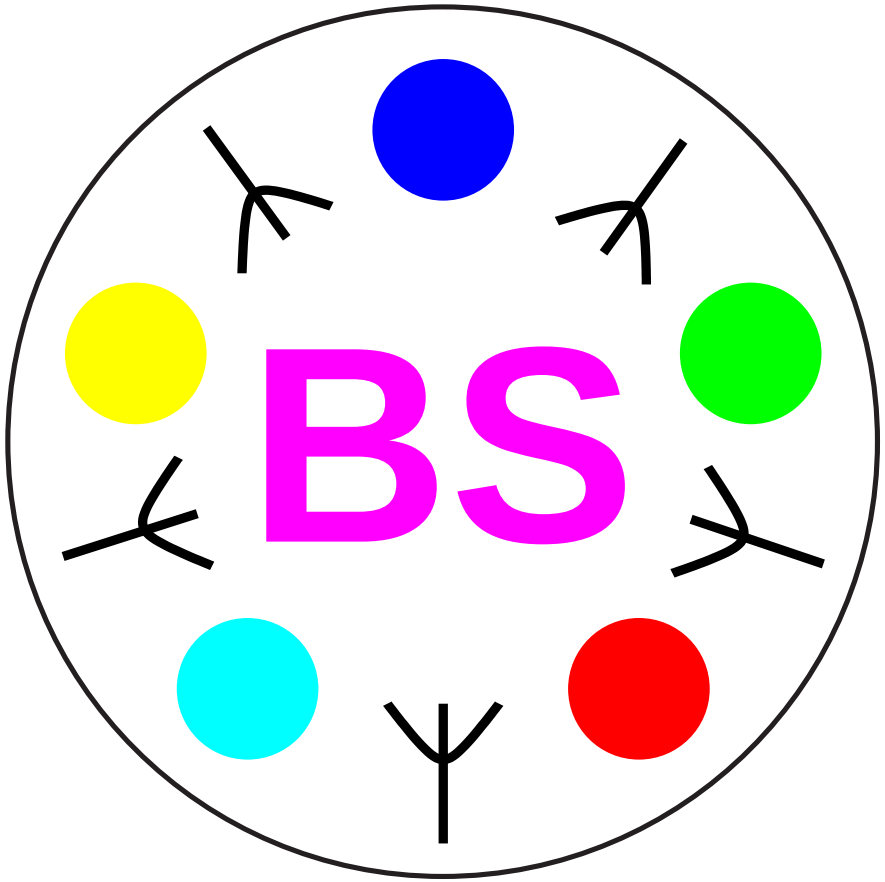
Rechnerarchitektur / Betriebssysteme



Übungen zur Vorlesung

Prof. Dr. Henning Pagnia

Aufgabe 1: Hardware und Interrupts

Bitte kreuzen Sie alle korrekten Antworten an; es können immer auch mehrere Antworten richtig sein.

(1)
- ☐ Unterbrechungen sind erzwungene Unterprogrammaufrufe.
- ☐ Unterbrechungen können nur durch externe Geräte (Tastatur, Festplatte, usw.) angefordert werden.
- ☐ Die Annahme eines Interrupts beim Eintreffen einer Netzwerknachricht und der Sprung in die Behandlungsprozedur werden durch die Hardware veranlasst.
- ☐ Nichts davon.

(2) Cache-Speicher

- ☐ ist eine alternative Bezeichnung für das Main-Memory.
- ☐ sind i. Allg. umso kleiner, je kleiner ihrer Level-Nummer ist.
- ☐ sind Assoziativspeicher.
- ☐ Nichts davon.

(3 Der vom Prozessor verwaltete Stack

- ☐ befindet sich im Stack-Pointer-Register des Prozessors.
- ☐ enthält die Rücksprungadressen aller noch nicht beendeter Unterprogrammaufrufe.
- ☐ und der Code-Bereich wachsen während einer Programmausführung gegenläufig
- ☐ Nichts davon.

(4)
- ☐ Ein Multicore-Prozessor enthält mehrere vollständige CPUs.
- ☐ Bei einem Quad-Core-Prozessor teilen sich die CPUs den First-Level-Cache.
- ☐ Hyperthreading-CPUs sind performanter als Multicore-Prozessoren.
- ☐ Nichts davon.

Aufgabe 2: 2-aus-3-Problem (Semaphorkonzept)

Geg. seien 3 exkl. benutzbare Betriebsmittel Platte, Drucker, Bandgerät sowie 3 kritische Abschnitte `csA`, `csB` und `csC`:

- in `csA` werden Platte und Drucker benötigt
- in `csB` werden Platte und Bandgerät benötigt
- in `csC` werden Drucker und Bandgerät benötigt

a) Beschreiben Sie, welche Ausschlussbeziehungen zwischen den drei kritischen Abschnitten bestehen und stellen Sie diese grafisch dar.

b) Diskutieren Sie die folgende Lösung des Problems. Wenn Sie einen Fehler entdecken, dann verbessern Sie die Lösung.

```
public class TwoOfThreeExercise {
  private Semaphore disk    = new Semaphore (1);
  private Semaphore printer = new Semaphore (1);
  private Semaphore tape    = new Semaphore (1);
  public void getDiskPrinter(){
    disk.p();
    printer.p();
    // csA;
    disk.v();
    printer.v();
  } // getDiskPrinter
  public void getTapeDisk(){
    tape.p();
    disk.p();
    // csB;
    tape.v();
    disk.v();
  } // getTapeDisk
  public void getPrinterTape(){
    printer.p();
    tape.p();
    // csC;
    printer.v();
    tape.v();
  } // getPrinterTape
} // TwoOfThreeExercise
```

Aufgabe 3: Semaphore und Ein-/Ausgabeoperationen innerhalb der Prozessverwaltung

Gegeben sei ein System mit *6 Prozessen* mit den *Prozessidentifikationen* `0` bis `5`, zwei Semaphoren `mutex` und `resource` sowie einer *Festplatte*.
In der Prozessverwaltung ist das Semaphorkonzept sowie ein *Zeitscheibenverfahren* mit *Round-Robin*-Strategie implementiert.
Außerdem wird eine *Ausgabeoperation* `writeDisk` unterstützt, durch die ein Plattenblock auf die Festplatte geschrieben wird:

> Ein Aufruf von `writeDisk` wird *synchron* ausgeführt, d.h. der aufrufende Prozess muss in der Prozesswarteschlange `disk queue` warten, bis sein Plattenauftrag ausgeführt wurde.

Die Beendigung eines `writeDisk`-Auftrags wird von der Festplatte über eine *Unterbrechung* gemeldet, die von der *Interruptroutine* `diskItrHdler` (als erzwungener Prozeduraufruf) wie folgt behandelt wird:

> Der zugehörige Prozess wird wieder in den `ready`-Zustand versetzt und der nächste Plattenauftrag – sofern vorhanden – an die Festplatte geschickt.

Den momentan aktuellen Systemzustand erkennen Sie in der ersten (grau hinterlegten) Zeile der nachfolgenden Tabelle. Bei den Warteschlangen ist das älteste Element jeweils links angegeben. Die angegebenen Schnittstellenoperationen der Prozessverwaltung werden nacheinander aufgerufen (vom jeweils laufenden Prozess) und ausgeführt, wodurch sich der Systemzustand ändert.

Tragen Sie in die Tabelle den jeweils *nach dem Aufruf vorliegenden* Systemzustand ein (`rp` steht für `runningProcess`).

ausgeführte Schnittstellen-operation	Zustand der Prozessverwaltung						
	rp	ready queue	mutex		resource		disk queue
			ctr	queue	ctr	queue	
	0	1,2,3	0	-	-1	4	5
mutex.p()							
timerItrHdler()							
diskItrHdler()							
resource.v()							
writeDisk()							
timerItrHdler()							
resource.v()							
writeDisk()							
diskItrHdler()							
mutex.p()							

Aufgabe 4: Speicherverwaltung

Die Speicherverwaltung teilt Speicherbereiche variabler Länge zu. In der Freispeicherliste befinden sich vier Bereiche. Nacheinander treffen fünf Anforderungen ein. Tragen Sie in die Tabelle ein, wie sich die Freispeicherliste jeweils bei der Bedienung der Anforderungen unter den angegebenen Zuteilungsstrategien verändert.

	Freispeicherliste mit Länge der Bereiche (1. Element links)							
	First-Fit				Best-Fit			
aktuelle Listen	90	35	60	45	90	35	60	45
Anforderung								
25								
40								
30								
50								
45								

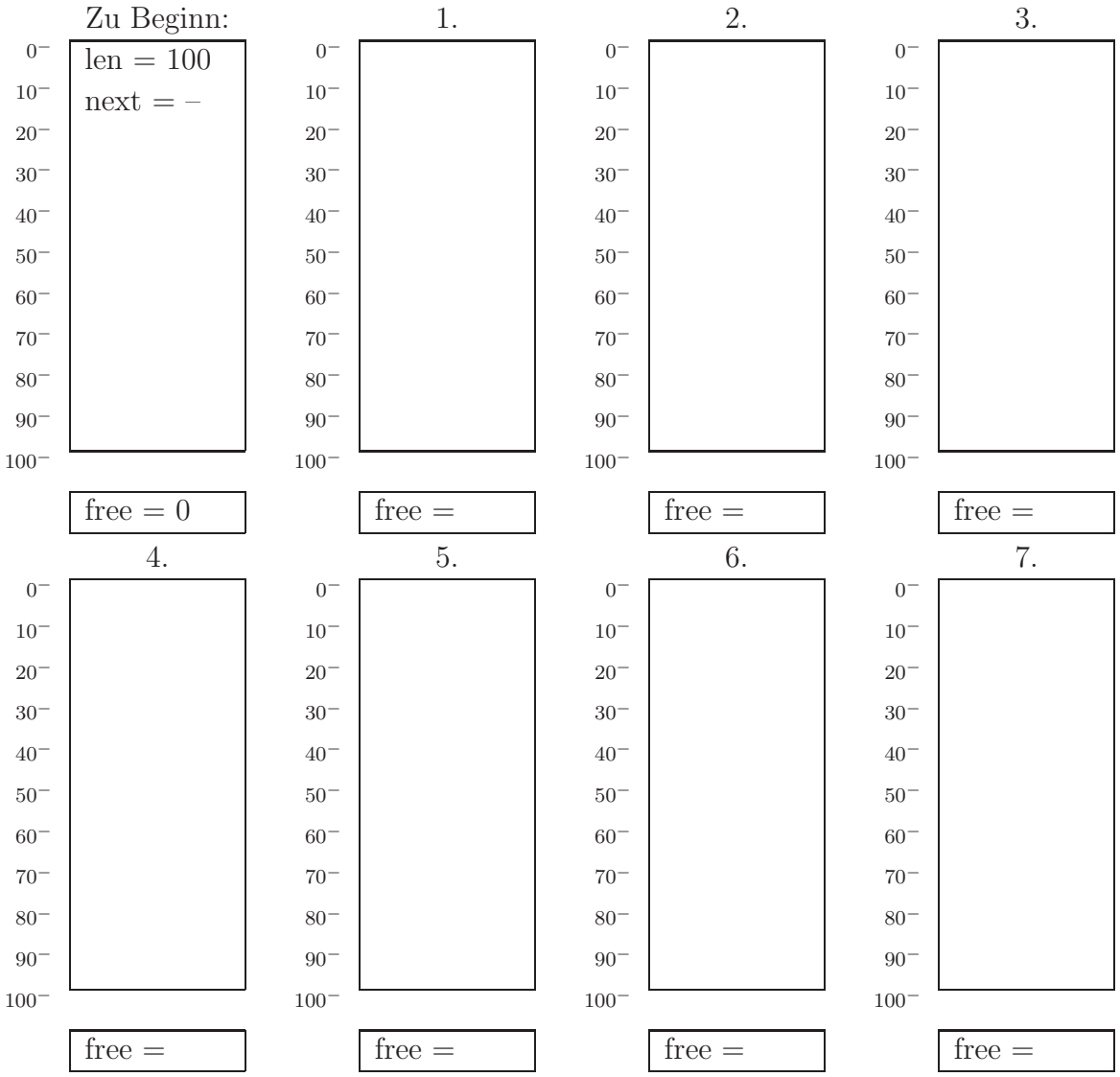
Aufgabe 5 : Speicherzuteilung und -freigabe

Die Hauptspeicherverwaltung teilt Speicherbereiche variabler Länge zu und führt über die freien Bereiche in der Freispeicherliste Buch. Der Listenkopf `free` benennt die Speicheradresse des Listenanfangs. In jedem freien Block steht dessen Länge sowie die Adresse des nächsten freien Blocks. Zu Beginn sei in der Freispeicherliste der *gesamte* Speicher als *zusammenhängender Bereich der Länge 100*.

Freigegebene Speicherbereiche werden ggf. *an das Ende* der Freispeicherliste angehängt; Reste, die bei einer Anforderung übrig sind, bleiben in der Freispeicherliste an der Stelle des ausgewählten freien Speicherbereichs. Falls eine Anforderung erfüllbar ist, skizzieren Sie bitte jeweils die Speicherbelegung sowie die Freispeicherliste (`free`) nach deren Bearbeitung. Anderenfalls geben Sie bitte *"nicht erfüllbar"* an.

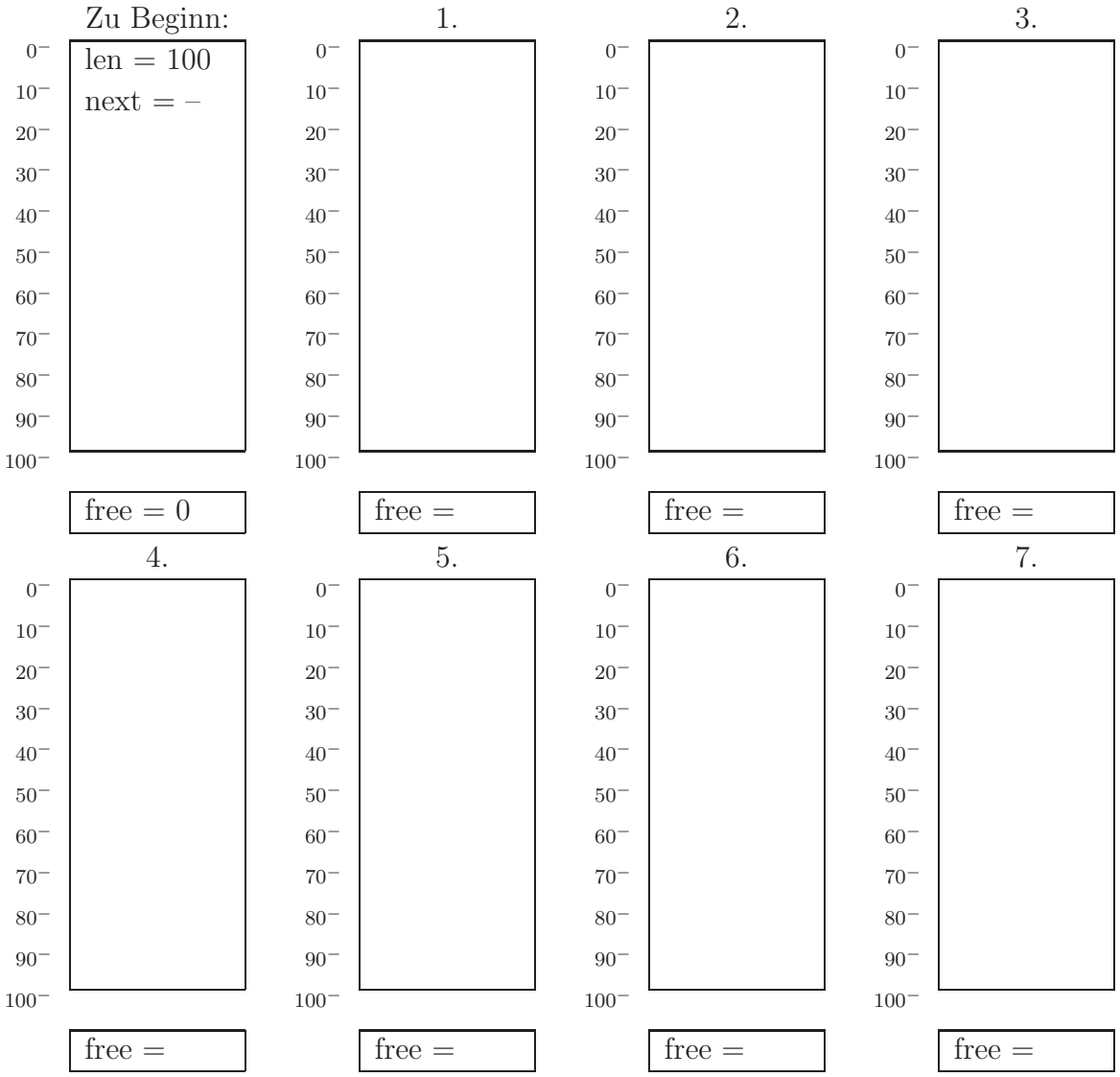
(a) Es wird die ***First-Fit-Strategie*** verwendet und die folgenden Speicheranforderungen und -freigaben treffen ein:

1. A1: anfordern(20)
2. A2: anfordern(20)
3. A3: anfordern(30)
4. freigeben der Anforderung A1
5. A4: anfordern(10)
6. freigeben der Anforderung A3
7. A5: anfordern(50)



(b) Es wird die ***Best-Fit-Strategie*** verwendet und die folgenden Speicheranforderungen und -freigaben treffen ein:

1. A1: anfordern(20)
2. A2: anfordern(20)
3. A3: anfordern(30)
4. freigeben der Anforderung A1
5. A4: anfordern(10)
6. freigeben der Anforderung A3
7. A5: anfordern(50)



Aufgabe 6: Lineare Seitentabelle

Betrachten Sie ein System mit 32-Bit langen virtuellen Adressen sowie einer Seitengröße von 4 KiB und einer Wortlänge von 32 Bit.

(i) Wie viele Seiten umfasst der maximal ausgebaute Adressraum eines Prozesses?

(ii) Wie groß ist dann eine lineare Seitentabelle, wenn ein Seitentabelleneintrag 6 Byte benötigt?

Aufgabe 7: Seitentauschstrategien

Ein Programm besteht aus fünf Seiten im virtuellen Adressraum und hat drei Kacheln im Hauptspeicher zur Ausführung zur Verfügung. Die Referenzreihenfolge der Seiten ist:

1 (`r`), 5 (`w`), 2 (`w`), 4 (`r`), 5 (`w`), 3 (`r`), 4 (`w`), 5 (`r`), 2 (`w`), 4 (`r`), 2 (`w`), 1 (`r`), 4 (`w`), 3 (`r`)

(`r`) steht für einen lesenden, (`w`) für einen schreibenden Zugriff auf die entsprechende Seite, wobei diese Information nur beim Second-Chance-Algorithmus verwendet werden soll. Führen Sie Buch über die Hauptspeicherbelegung. Verwenden Sie dabei folgende Seitenverdrängungsalgorithmen:

a) **Optimale Verdrängung**

b) **First-In-First-Out**-Verdrängung

c) **Least-Recently-Used** (LRU-Algorithmus)

d) **Second-Chance**-Algorithmus

Zählen Sie jeweils die Anzahl der Seitenfehler und vergleichen Sie die Ergebnisse.

Aufgabe 8: Seitenfehler bei Programmausführung

Gegeben ist ein System mit Seitentausch, wobei die Seitengröße 1024 Bytes beträgt. Die zweidimensionale, zeilenweise abgespeicherte Matrix

```
byte[][] m = new byte[256][256];
```

beginnt mit der zweiten Seite des virtuellen Adressraums (d.h. die virtuelle Adresse von `m[0,0]` ist 1024).

Das Programm

Variante (i):

```
for (i=0; i<256; i++)
    for (k=0; k<256; k++)
        m[i][k] = 0;
```

bzw. **Variante (ii):**

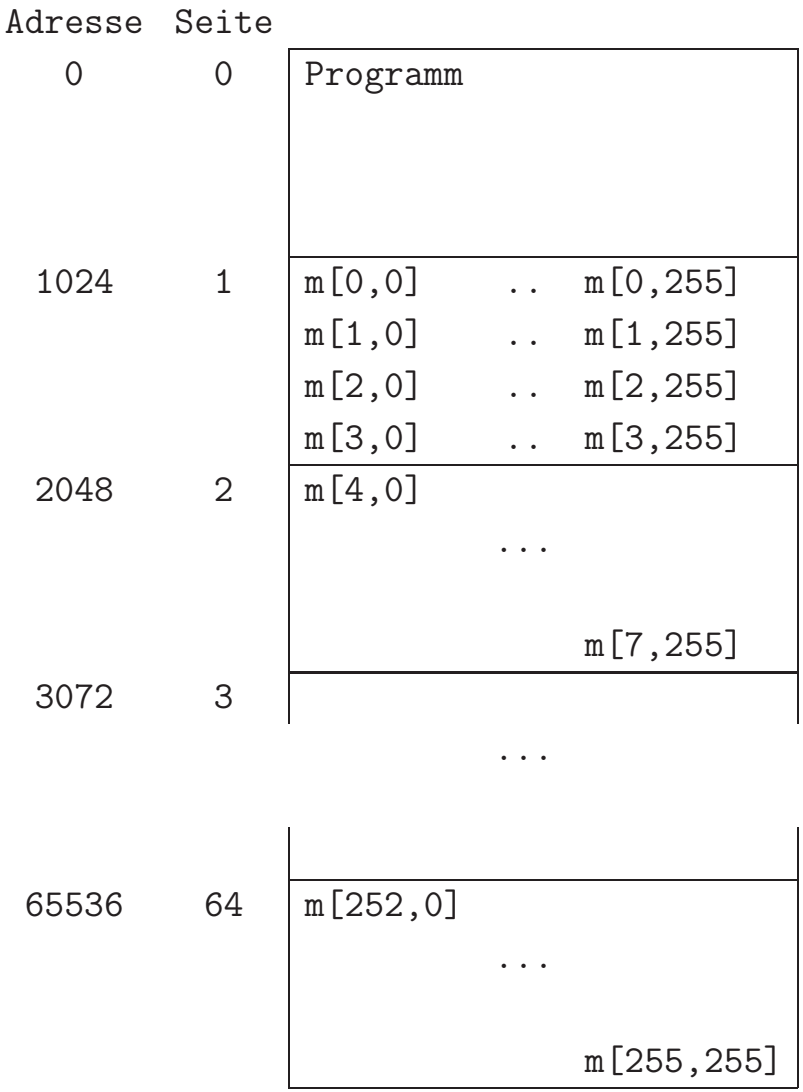
```
for (i=0; i<256; i++)
    for (k=0; k<256; k++)
        m[k][i] = 0;
```

liegt in der ersten Seite des virtuellen Adressraums.

Einem Prozess stehen zur Ausführung des Programms (i) bzw. seiner Variante (ii) jeweils drei Hauptspeicherkacheln zur Verfügung. Das Programm befindet sich bereits in Kachel 1, zwei weitere Kacheln sind noch frei. Die Seitenersetzungsstrategie ist LRU.

Wieviele Seitenfehler werden von den beiden Varianten (i) und (ii) jeweils erzeugt?

Programm und Matrix sind wie folgt im virtuellen Speicher angeordnet:



(**Tipp:** Es wird immer abwechselnd auf die *Programmseite* und eine *Matrixseite* zugegriffen, d.h. direkt vor einem Zugriff auf ein Matrixelement findet ein Programmzugriff statt.)

Aufgabe 9: UNIX-Dateischutz

Gegeben ist ein System mit den drei Benutzern **Meier**, **Müller** und **Schmidt**, wobei Meier und Müller eine *Benutzergruppe* bilden.

Meier gehören die Programmdatei `p1` und die Textdatei `d1`. Schmidt gehören die Programmdateien `p2` und `p3` sowie die Textdatei `d2`. Das Programm `p1` schreibt in die Datei `d1`. Das Programm `p2` liest aus `d1` und schreibt in `d2`. Das Programm `p3` liest aus `d2`.

a) Setzen Sie die Schutzbits so, dass Müller `p1` und `p2` ausführen kann und Schmidt `p2` und `p3` ausführen kann. Vergeben Sie minimale Rechte! Beachten Sie dabei, dass ein Programm nur dann fehlerfrei abläuft, wenn der aufrufende Benutzer die entsprechenden Zugriffsrechte *für alle* benötigten Dateien besitzt.

	(u)			(g)			(o)	
	r	w	x	r	w	x	r	w	x
p1									
p2									
p3									
d1									
d2									

b) Welche Dateizugriffe kann welcher Benutzer über das notwendige Maß hinaus durchführen?