Technische Universität Dortmund

# Klausur „Betriebssysteme“

| | erreichbare Punkte | erhaltene Punkte | | | | |
|---|---|---|---|---|---|---|
| **Aufgabe 1** | 8 | **a** | **b** | **c** | **d** | |
| | | | | | | |
| **Aufgabe 2** | 8 | **a** | **b** | | | |
| | | | | | | |
| **Aufgabe 3** | 8 | **a** | **b** | | | |
| | | | | | | |
| **Aufgabe 4** | 10 | **a** | **b** | **c** | | |
| | | | | | | |
| **Aufgabe 5** | 6.5 | **a** | **b** | | | |
| | | | | | | |
| **Aufgabe 6** | 19.5 | **a** | **b** | | | |
| | | | | | | |
| **Summe** | 60 | | | | | |
| **Spickzettel vorhanden?** | | | | | | |

(Matrikel-Nr.)

(Name) (Vorname)

Durch meine Unterschrift bestätige ich

- den Empfang der vollständigen Klausur (17 Seiten inklusive Deckblatt),
- den Empfang der Manualseiten (3 Blätter mit 7 Manualseiten: pthread_create/pthread_exit, pthread_join, scanf, sem_destroy, sem_init, sem_post und sem_wait),
- die Kenntnisnahme der Hinweise auf Seite 2.

Dortmund, 15.09.2023

(Unterschrift)

# Hinweise

Bitte lesen Sie die folgenden Informationen **aufmerksam** und unterschreiben Sie die Erklärung auf der ersten Seite.

- Es können **60 Punkte** erreicht werden; 50% der Gesamtpunktzahl sind zum Bestehen der Klausur erforderlich. Zur Bearbeitung stehen insgesamt **60 Minuten** zur Verfügung.
- Als Hilfsmittel ist lediglich ein **von Ihnen eigenhändig geschriebenes** (Handschrift, kein Ausdruck oder Kopie) **A4-Blatt** (beidseitig) erlaubt, welches eingesammelt wird. Ansonsten dürfen **keine** weiteren Hilfsmittel verwendet werden.
- Die Antworten sind in **deutscher** oder **englischer** Sprache zu verfassen.
- Notieren Sie Ihre Lösungen direkt auf den ausgeteilten Aufgabenblättern unter Verwendung eines dokumentenechten, schwarzen oder blauen Stifts. Entfernen Sie **nicht** die Heftung der Blätter. Falls der vorgesehene Platz nicht ausreichen sollte, können Sie auch die Rückseiten der Blätter und die Reserveseiten am Ende verwenden. Notieren Sie in diesem Fall aber an der für die Lösung vorgesehenen Stelle einen Verweis auf die Seite.
- Da es unterschiedliche Sprachgebräuche sowie Algorithmen- und Konzept-Ausprägungen gibt, werden hier ausdrücklich die aus Vorlesung und Übungen bekannten Ausdrucksweisen und Ausprägungen zu Grunde gelegt.

**Aufgabe 1: Ankreuzfragen (8 Punkte)**

Bei den Einfachauswahlfragen in dieser Aufgabe ist jeweils nur **eine** richtige Antwort eindeutig anzukreuzen. Auf die richtige Antwort gibt es die angegebene Punktzahl.

Wollen Sie eine Antwort korrigieren, streichen Sie bitte die falsche Antwort mit drei waagrechten Strichen durch (☒) und kreuzen die richtige an.

Lesen Sie die Frage genau, bevor Sie antworten.

a) **Virtueller Speicher:** Welche Aussage zur Seitenumlagerung (demand paging) in virtuellen Adressräumen ist richtig? 2 Punkte

- ☐ Unter Seitenflattern (thrashing) versteht man das wiederholte Ein- und Auslagern von Speicherseiten, wenn der physische Hauptspeicher nicht ausreicht.
- ☐ Bei der Least-Recently-Used-Strategie (LRU) wird die Seite ausgelagert, auf die in der Vergangenheit am seltensten zugegriffen wurde.
- ☐ Für einen schnelleren Zugriff können Speicherseiten im Seitendeskriptor mit einem Bezeichner versehen werden.
- ☐ Die Seitenersetzungsstrategie Second Chance (clock) ist nur in der Theorie interessant, weil ihre Implementierung in der Praxis viel zu aufwendig ist.

b) **Prozesse:** Welche Aussage zu Prozesszuständen ist auf einem Monoprozessorsystem richtig? 2 Punkte

- ☐ Bei Eintreffen eines Interrupts wird der aktuell laufende Prozess für die Dauer der Interrupt-Abarbeitung in den Zustand *blockiert* (blocked) überführt.
- ☐ Im Rahmen der mittelfristigen Einplanung kann ein Prozess von Zustand *laufend* (running) in den Zustand *schwebend laufend* (suspended running) wechseln.
- ☐ Ein Seitenfehler (page fault) kann dazu führen, dass der aktuell laufende Prozess in den Zustand *beendet* (exit) überführt wird.
- ☐ Bei kooperativem Scheduling ist ein direkter Übergang vom Zustand *laufend* (running) in den Zustand *bereit* (ready) unmöglich.

c) **Speicher:** Was ist ein Stack-Frame? 2 Punkte

- ☐ Der Speicherbereich, in dem der Programmcode einer Funktion abgelegt ist.
- ☐ Ein Bereich des Speichers, in dem u.a. lokale (automatic) Variablen einer Funktion abgelegt sind.
- ☐ Der Seitenrahmen (page frame) im Hauptspeicher der den Stack-Speicher enthält.
- ☐ Ein spezieller Registersatz des Prozessors zur Bearbeitung von Funktionen.

d) **Synchronisation:** Welche Aussage zum Thema Synchronisation ist richtig?

2 Punkte

- ☐ Die V-Operation (signal) kann auf einem Semaphor nur von dem Prozess aufgerufen werden, der zuvor auch die P-Operation (wait) aufgerufen hat.
- ☐ Durch den Einsatz von Semaphoren kann ein gegenseitiger Ausschluss (mutual exclusion) erzielt werden.
- ☐ Ein Semaphor kann ausschließlich für mehrseitige Synchronisation (z.B. Leser/Schreiber-Problem) verwendet werden.
- ☐ Einseitige Synchronisation (z.B. Produzent/Konsument) erfordert immer Betriebssystem-Unterstützung.

**Aufgabe 2: Threads und Scheduling (8 Punkte)**

a) POSIX-Threads

3 Punkte

Geben Sie die Ausgabe des folgenden C-Programms an.

**Hinweis:** Der Code wurde von uns zum besseren Verständnis vereinfacht. Gehen Sie von einem fehlerfreien Ablauf aus. Das Symbol ␣ steht für ein Leerzeichen.

```
int x = 5;
pthread_t a,b;
void* next() {
    printf("%d,␣", x++);
    return 0;
}
void* back() {
    pthread_join(a, NULL);
    x = 0;
    printf("b%d,␣", x++);
    return 0;
}
int main() {
    next();
    if(pthread_create(&a, NULL, &next, NULL)) {printf("Oh,␣");}
    if(pthread_create(&b, NULL, &back, NULL)) {printf("Schade,␣");}
    pthread_join(b, NULL);
    printf("j,␣");
    int x = 10;
    next();
    return 0;
```

**Ausgabe:**

---

b) Scheduling-Verfahren

5 Punkte

Gegeben sind **drei zyklisch** arbeitende Prozesse P1, P2 und P3. Die Prozesse treffen mit der in der folgenden Tabelle angegebenen *Ankunftszeit* ein und führen erst einen (vollständigen) *CPU-Stoß* und dann einen (vollständigen) *E/A-Stoß* aus.

| Prozess | Ankunftszeit | CPU-Zeit | E/A-Zeit |
|---|---|---|---|
| P1 | 0 | 4 | 4 |
| P2 | 1 | 1 | 3 |
| P3 | 2 | 3 | 3 |

Das Scheduling arbeitet nach der **Round-Robin-Strategie** mit einer Zeitscheibendauer von **2** Zeiteinheiten. Die ersten drei Zeiteinheiten sind von uns bereits vorgegeben. Zeichnen Sie entsprechend der Legende in das folgende Gantt-Diagramm ein, wie P1, P2 und P3 im folgenden abgearbeitet (Prozesszustände) werden.

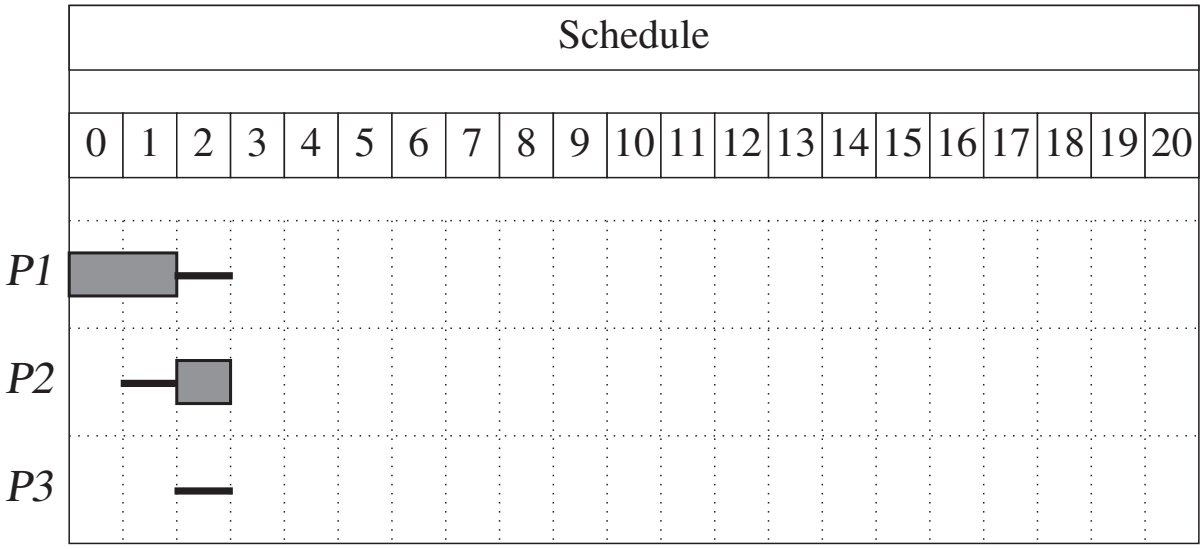
**Hinweis:** Die Prozessumschaltzeit kann vernachlässigt werden. Es gibt nur eine CPU, die E/A-Vorgänge der Prozesse können parallel ausgeführt werden. Es genügt, wenn Sie die Flächen für *Running* geeignet schraffieren.

Legende

*Running*

*Blocked*

*Ready*




(Ersatzdiagramme auf dem Reserveblatt: Streichen Sie ungültige Lösungen deutlich durch!)

**Aufgabe 3: Synchronisation und Verklemmungen (8 Punkte)**

a) Leser-Schreiber-Problem

6 Punkte

Wie beim gegenseitigen Ausschluss, wird auch beim Leser-Schreiber-Problem ein kritischer Abschnitt geschützt. Allerdings werden die Prozesse in zwei Arten von konkurrierenden Prozessen aufgeteilt, den *Schreibern* und den *Lesern*. In der unten vorgestellten Implementierung haben sich jedoch drei Fehler eingeschlichen, die dafür sorgen, dass die Synchronisierung zwischen den konkurrierenden Prozessen nicht reibungslos abläuft. Markieren Sie die drei Fehler im Quellcode mit **1**, **2**, und **3**, und beschreiben Sie kurz das jeweilige Problem.

```
/* gemeinsamer Speicher */

int readcount = 0;

Semaphore mutex = 1;

Semaphore wrt = 0;
```

```
/* Schreiber */

p(&wrt);

... schreibe

v(&wrt);
```

```
/* Leser */

readcount++;
p(&mutex);
if (readcount == 1)
p(&wrt);
v(&mutex);

... lese

p(&mutex);
readcount--;
if (readcount == 0)
v(&wrt);
```

Fehler 1 (1 Punkt)

Fehler 2 (1 Punkt)

Fehler 3 (1 Punkt)

b) Sprachgestützte Synchronisation

2 Punkte

Wie wird eine Sprachunterstützung zum gegenseitigen Ausschluss genannt? Diese wird beispielsweise in der Programmiersprache `Java` eingesetzt, um gegenseitigen Ausschluss beim Zugriff auf Klassen sicherzustellen.

**Aufgabe 4: Speicherverwaltung und virtueller Speicher (10 Punkte)**

a) Programmsegmente und Variablen

4 Punkte

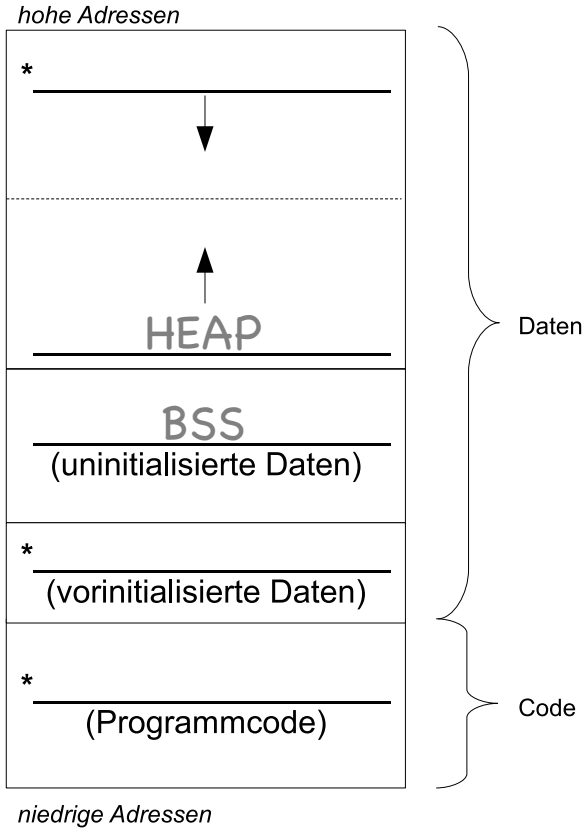
Auf einem x86-System wird das folgende C-Programm übersetzt und gebunden. Vervollständigen Sie das Speicherlayout mit den **Namen** der fehlenden Segmente (mit * markiert) und ordnen Sie die Variablen (**x**, **y**, **z**) und Funktionen (**main**, **add**) den korrekten Segmenten zu.

**Hinweis:** Sie können für die Zuordnung einfach Pfeile aus dem Code zum jeweiligen Segment zeichnen.

```
int x;
int y = 0;

void add(int a, int b) {
  int z = a + b;
  x = z;
}

int main(void) {
  y = 1;
  main(y, 5);
  return x;
}
```




b) Segmentbasierte Addressberechnung

2 Punkte

Geben Sie für die logischen Adressen $1010\ 0001_{16}$ und $0000\ 0123_{16}$ jeweils die physikalische Adresse unter Anwendung der segmentbasierten Adressabbildung an. Die höchstwertigen 8 Bit der logischen Adresse geben die Position innerhalb der Segmenttabelle an. Falls eine Speicheranfrage eine Zugriffsverletzung auslöst, machen Sie dies kenntlich.

Segmenttabelle:

| | Startadresse | Länge |
|---|---|---|
| $00_{16}$ | $0815\ 0000_{16}$ | $00\ 00FF_{16}$ |
| $01_{16}$ | $FA57\ 0000_{16}$ | $01\ DDDD_{16}$ |
| $02_{16}$ | $F00D\ 0000_{16}$ | $50\ 0000_{16}$ |
| $03_{16}$ | $8080\ 0000_{16}$ | $00\ FFFF_{16}$ |
| ... | | |
| $10_{16}$ | $99C0\ 0000_{16}$ | $AF\ FFFF_{16}$ |

logische Adresse: $1010\ 0001_{16}$

→ physikalische Adresse:

logische Adresse: $0000\ 0123_{16}$

→ physikalische Adresse:

c) *Seitenersetzung*

4 Punkte

In einem System mit Seitenadressierung und der Seitenersetzungsstrategie *LRU* (*Least Recently Used*) tätigt ein Prozess Seitenzugriffe entsprechend folgender

**Referenzfolge 1,5,2,3,1,3,5,1,4**.

Das Betriebssystem sieht für diesen Prozess eine feste Anzahl von drei Hauptspeicherkacheln vor. Tragen Sie in die Tabelle unter „Hauptspeicher“ jeweils die Nummer der Seite ein, die zum gegebenen Zeitpunkt in der jeweiligen Kachel eingelagert ist. Die Felder unter „Kontrollzustände“ können Sie zum Notieren von Hilfsinformationen verwenden.

| Referenzfolge | | 1 | 5 | 2 | 3 | 1 | 3 | 5 | 1 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|
| Hauptspeicher | Kachel 1 | | | | | | | | | |
| | Kachel 2 | | | | | | | | | |
| | Kachel 3 | | | | | | | | | |
| Kontrollzustände | Kachel 1 | | | | | | | | | |
| | Kachel 2 | | | | | | | | | |
| | Kachel 3 | | | | | | | | | |

(Ersatzdiagramm auf dem Reserveblatt: Streichen Sie ungültige Lösungen deutlich durch!)

**Aufgabe 5: UNIX-Zugriffsrechte (6.5 Punkte)**

a) Zugriffsrechte verstehen

4.5 Punkte

In einem UNIX-Dateisystem liegt eine Datei, deren Zugriffsrechte von '`ls -l`' folgendermaßen angezeigt werden:

```
-rw--wxr--  1 europa studi 1448 2023-09-08 11:56 astudio.sh
```

Das Verzeichnis, in dem diese Datei liegt, ist für alle Benutzer:innen lesbar. Die Benutzerin 'kallisto' ist in der Gruppe 'studi', der Benutzer 'guest' dagegen nicht. Welche Benutzer:innen dürfen diese Datei lesen, welche schreiben, und welche ausführen? Schreiben Sie in jedes Tabellenfeld 'J' (ja) oder 'N' (nein), je nachdem, ob der Benutzer das jeweilige Recht besitzt. Füllen Sie jede Zelle aus.

| **Benutzername** | **lesen** | **schreiben** | **ausführen** |
|---|---|---|---|
| europa | | | |
| kallisto | | | |
| guest | | | |

b) Zugriffsrechte abändern

2 Punkte

Ändern Sie die Zugriffsrechte der oben angegebenen Datei so ab, dass alle Benutzer:innen, die nicht die Nutzerin `europa` sind oder Mitglieder der Gruppe `studi` sind, die Datei nur ausführen dürfen. Ferner sollen Mitglieder der Gruppe `studi` sowie die Nutzerin `europa` die Datei nur lesen dürfen. Verwenden Sie dazu die oben bereits abgebildete Notation: `-rw--wxr--`.

---

**Aufgabe 6: Programmierung (19.5 Punkte)**

Implementieren Sie in zwei Schritten das Programm `coffee`. Hierbei sollen zunächst zwei Namen (einer für jeden Thread) eingelesen werden. Ein Thread wird Kaffee produzieren, der andere Thread wird Kaffee trinken.

Hier ein exemplarischer Aufruf von „`./coffee`“mit Ausgabe:

```
Please enter the first name: Hans ⏎
Please enter the second name: Peter ⏎
Hans brewed 1 cups!
Hans brewed 2 cups!
Hans brewed 3 cups!
  Peter drank 3 coffees!
Hans brewed 1 cups!
Hans brewed 2 cups!
Hans brewed 3 cups!
  Peter drank 6 coffees!
Hans brewed 1 cups!
Hans brewed 2 cups!
Hans brewed 3 cups!
  Peter drank 9 coffees!
Hans brewed 1 cups!
  Peter drank 10 coffees!
  Peter: No more coffee for me. :-(
```

**Aufbau der Funktion `main`**

- Korrekte Initialisierung des Semaphors `mutex`.
- Einlesen der beiden Namen von der Standardeingabe in die Variablen `nameA` und `nameB`.
- Erzeugung eines Threads, der die Funktion `a_thread` (konsumiert Kaffee) ausführt.
- Nun wird 10x Kaffee gekocht:
  - Das Brühen dauert jeweils 1 Sekunde (`sleep(1)`). **Achtung:** Schlafen Sie nicht im kritischen Abschnitt!
  - Geben Sie nach jedem Brühvorgang den Namen der kochenden Person sowie die Anzahl der im Moment fertigen Kaffees aus.
  - Zur Koordinierung mit dem Thread wird die geteilte Variable `coffees` genutzt, welche synchronisiert werden muss!
- Warten auf Terminierung des oben erstellten Threads (alle 10 Kaffees sind konsumiert).
- Zerstörung des Semaphors.

**Funktion `a_thread`**

- Die Kaffeetrinker:in soll in einer Endlosschleife Folgendes tun:
  - Jedes Betreten der Küche dauert drei Sekunden (`sleep(3)`. **Achtung:** Schlafen Sie nicht im kritischen Abschnitt!
  - Die Kaffeetrinker:in konsumiert anschließend alle verfügbaren Kaffees. Geben Sie jedes Mal den Namen und die Anzahl der insgesamt schon konsumierten Tassen aus.
  - Der Konsum muss mittels der geteilten Variable `coffees` mitgeteilt werden. D.h. der Wert wird auf 0 gesetzt. Zum Zugriff muss gegenseitiger Ausschluss sichergestellt werden!
  - Ist die Anzahl verfügbarer Kaffees kleiner oder gleich Null, soll das Trinken beendet werden (Ausgabe eines Hinweises).
- Nach der Schleife wird der Thread beendet.

**Fehlerbehandlung:** Beachten Sie, dass Fehler im gesamten Programm auftreten können und entsprechend behandelt werden müssen. Das Programm darf beim Auftreten eines Fehlers jederzeit abgebrochen werden.

**Relevante Manual-Seiten:** `sem_init`, `sem_destroy`, `sem_wait`, `sem_post`, `pthread_create`, `pthread_join`, `scanf`

**Hinweis:** Einfache syntaktische Fehler (z.B. vergessene Strichpunkte) führen nicht zu Punktabzug, es geht um die semantische Umsetzung.

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

// Sorgt für gegenseitigen Ausschluss
sem_t mutex;

int coffees = 0;

char nameA[43];
char nameB[43];

void* a_thread(void *arg);
```

a) `main`-Funktion (11.5 Punkte)

```c
int main(int argc, const char *argv[]) {
  pthread_t thread;
```

```
  return EXIT_SUCCESS;
}
```

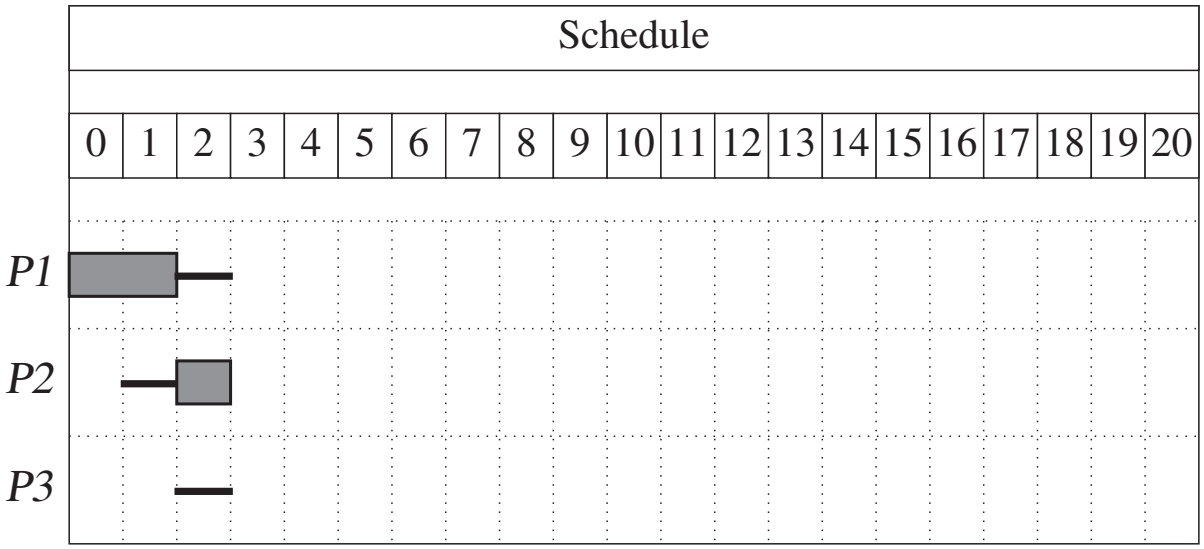
b) Funktion `a_thread` (8 Punkte)

```
void* a_thread(void *arg) {

}
```

# Reserveblatt

| Prozess | Ankunftszeit | CPU-Zeit | E/A-Zeit |
|---|---|---|---|
| P1 | 0 | 4 | 4 |
| P2 | 1 | 1 | 3 |
| P3 | 2 | 3 | 3 |




## Ersatztabelle für Aufgabe 4

| Referenzfolge | | 1 | 5 | 2 | 3 | 1 | 3 | 5 | 1 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|
| Hauptspeicher | Kachel 1 | | | | | | | | | |
| | Kachel 2 | | | | | | | | | |
| | Kachel 3 | | | | | | | | | |
| Kontrollzustände | Kachel 1 | | | | | | | | | |
| | Kachel 2 | | | | | | | | | |
| | Kachel 3 | | | | | | | | | |