Technische Universität Dortmund

Klausur „Betriebssysteme“

	erreichbare Punkte	erhaltene Punkte						
Aufgabe 1	12	**a**	**b**	**c**	**d**			
Aufgabe 2	13	**a**	**b**					
Aufgabe 3	14	**a**	**b**	**c**				
Aufgabe 4	10	**a**	**b**	**c**				
Aufgabe 5	10	**a**	**b**					
Aufgabe 6	31	**a**	**b**	**c**	**d**	**e**	**f**	
Summe	90							
Spickzettel vorhanden?								

(Matrikel-Nr.)

(Nachname) (Vorname)

Durch meine Unterschrift bestätige ich
- den Empfang der vollständigen Klausur (21 Seiten inklusive Deckblatt),
- den Empfang der Manualseiten (4 Blätter mit 7 Manualseiten: close, malloc, open, read, sigaction, time und write),
- die Kenntnisnahme der Hinweise auf Seite 2.

Dortmund, 29.09.2025

(Unterschrift)

Hinweise

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

- Es können **90 Punkte** erreicht werden; 50% der Gesamtpunktzahl sind zum Bestehen der Klausur hinreichend. Zur Bearbeitung stehen insgesamt **90 Minuten** zur Verfügung.
- Als Hilfsmittel ist lediglich ein **von Ihnen eigenhändig geschriebenes** (Handschrift, kein Ausdruck oder Kopie) **A4-Blatt** (beidseitig), 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 in der Klausur 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 der Klausur verwenden. Verweisen Sie an der für die Lösung vorgesehenen Stelle auf die genutzte Seite.
- Wir bewerten ausdrücklich gemäß der aus Vorlesung und Übungen sowie der Begleitliteratur bekannten Definitionen (Algorithmen und Konzepten) und Begrifflichkeiten.

Aufgabe 1: Ankreuzfragen (12 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) **Manpage:** Was steht üblicherweise in den Linux-Manpages unter Sektion #3?

3 Punkte

- ☐ Dateiformate und Konventionen
- ☐ Systemaufrufe (Funktionen die der Betriebssystemkern bereitstellt)
- ☐ Ausführbare Programme und Shell-Kommandos
- ☐ Bibliotheksfunktionen (Funktionen in Programm-Bibliotheken)

b) **Prozessverhalten:** Welche Aussage ist **nicht** korrekt?

3 Punkte

- ☐ Prozesse können vom Zustand READY in den Zustand BLOCKED wechseln.
- ☐ Prozesse können vom Zustand RUNNING in den Zustand BLOCKED wechseln.
- ☐ Prozesse können vom Zustand RUNNING in den Zustand READY wechseln.
- ☐ Prozesse können vom Zustand READY in den Zustand RUNNING wechseln.

c) **UNIX-Pipes:** Welche Aussage ist korrekt?

3 Punkte

- ☐ Eine Pipe ermöglicht eine bidirektionale Kommunikation zwischen zwei Kommunikationspartnern.
- ☐ Der Versuch in eine volle Pipe zu schreiben führt zu Datenverlust.
- ☐ Pipes können auch als Spezialdateien im Dateisystem angelegt werden.
- ☐ Unbenannte Pipes sind ausschließlich in der Shell verfügbar.

d) **Virtueller Speicher:** Welche Aussage ist korrekt?

3 Punkte

- ☐ Die Anzahl an nötigen Seiteneinlagerungen kann mit steigender Seitenrahmenanzahl trotz gleicher Ersetzungsstrategie und Referenzfolge zunehmen.
- ☐ Die Seitenumlagerung ermöglicht konstante effektive Zugriffszeiten auch bei hoher Seitenfehlerrate.
- ☐ Der Begriff Seitenflattern bezeichnet eine spezielle Strategie zur Seitenumlagerung.
- ☐ Virtueller Speicher in Betriebssystemen wird oft mit Cloud-Speichersystemen realisiert.

Aufgabe 2: Prozesse und Scheduling (13 Punkte)

a) UNIX-Systemaufrufe

5 Punkte

Geben Sie die Ausgabe des folgenden C-Programms an.
Hinweis: Das Einbinden der Header-Dateien sowie ein Teil der Fehlerbehandlung wurden ausgelassen. Gehen Sie von einem fehlerfreien Ablauf aus. Das Symbol ␣ steht für ein Leerzeichen.

```
int x = 16;
int divisor = 8;

void do_division(char* msg, int divisor) {
    if(divisor == 0) {
        printf("%s:NaN,", msg);
    } else {
        x = x / divisor;
        printf("%s:%d,", msg, x);
    }
}

int main() {
    pid_t pid = fork();
    printf("s,");
    fflush(NULL);
    if(pid < 0) {
        divisor = 2;
        do_division("t", 4);
    } else if(pid > 0) {
        wait(NULL);
        do_division("a", divisor);
    } else if(pid == 0) {
        int x = 64;
        do_division("r", 32);
    }
    do_division("e", x);
    return 0;
}
```

Ausgabe:

---

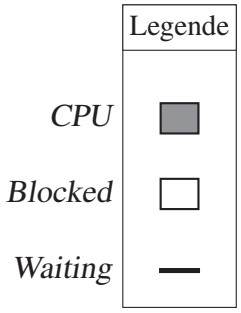
b) Scheduling-Verfahren

8 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	1	3
P2	1	1	2
P3	2	2	1

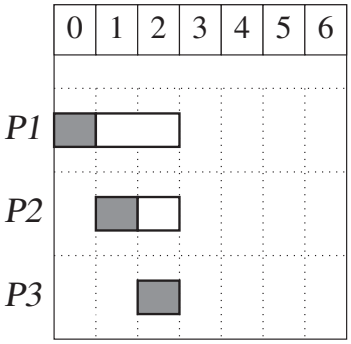
Das Scheduling arbeitet nach der in der jeweiligen Teilaufgabe angegebenen Strategie. Die ersten drei Zeiteinheiten sind von uns bereits vorgegeben. Zeichnen Sie entsprechend der Legende in das folgende Gantt-Diagramm ein, wie P1, P2 und P3 im Folgenden abgearbeitet (Prozesszustände) werden.



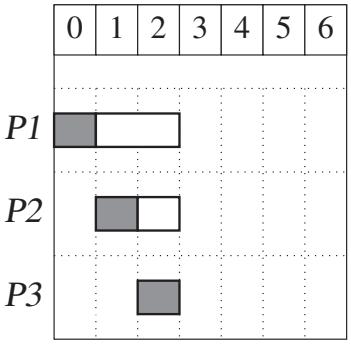
Hinweis: Die Prozessumschaltzeit kann vernachlässigt werden. Es gibt nur eine CPU, die E/A-Vorgänge der Prozesse können parallel ausgeführt werden. Es genügt, wenn Sie die Flächen für Running geeignet schraffieren.

I) Round Robin (2 Punkte)

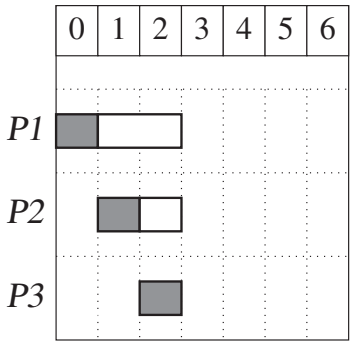
Die Länge der Zeitschreiben beträgt 2 Zeiteinheiten.



II) First Come First Serve (2 Punkte)

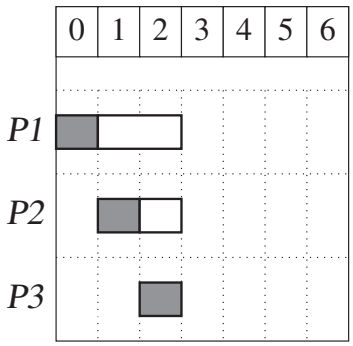


III) Shortest-Process-Next-Strategie (2 Punkte)



IV) Virtual Round Robin (2 Punkte)

Die Länge der Zeitschreiben beträgt 2 Zeiteinheiten.



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

Aufgabe 3: Synchronisation und Verklemmungen (14 Punkte)

a) Betriebsmittel

4 Punkte

Erklären Sie die Begriffe *wiederverwendbares Betriebsmittel* und *konsumierbares Betriebsmittel*. **Nennen** Sie außerdem ein Beispiel und eine typische Zugriffssynchronisation **je** Betriebsmittelart.

Antwort:

b) Verklemmungserkennung

4 Punkte

Das folgende Programm erzeugt zwei Threads, einen Produzenten (producer) und einen Konsumenten (consumer). Der Produzent erzeugt eine zufällige Ganzzahl (int) aus dem Intervall [1, 101]. Der Konsument nutzt den erzeugten Wert und berechnet aus diesem den Kehrwert (reciprocal). Das Programm wird regulär beendet sobald das erste Mal der Wert 101 durch den Produzenten erzeugt wird.

```
01: #include <error.h>
02: #include <pthread.h>
03: #include <semaphore.h>
04: #include <stdio.h>
05: #include <stdlib.h>

06: #define ERROR(sys_call) if(sys_call) { \
07:         perror("A␣system␣call␣failed"); \
08:         exit(EXIT_FAILURE); \
09:     }

10: int data = 100;
11: sem_t data_requested;
12: sem_t data_available;
13: pthread_t producer_handle;
14: pthread_t consumer_handle;

15: void* producer() {
16:     while(data != 101) {
17:         ERROR(sem_wait(&data_requested))
18:         data = rand() % 102;
19:         if(data == 0) {
20:             // Invalid value, try again!
21:             continue;
22:         }
23:         ERROR(sem_post(&data_available))
24:     }

25:     return 0;
26: }

27: void* consumer() {
28:     while(data != 101) {
29:         ERROR(sem_wait(&data_available))
30:         printf("Data:␣%i\n", data);
31:         double reciprocal = 1.0 / data;
32:         ERROR(sem_post(&data_requested))
33:         printf("Reciprocal:␣%f\n", reciprocal);
34:     }

35:     return 0;
36: }
```

```
37: int main() {
38:     ERROR(sem_init(&data_requested, 1, 1))
39:     ERROR(sem_init(&data_available, 1, 0))

40:     ERROR(pthread_create(&producer_handle, 0, producer, 0))
41:     ERROR(pthread_create(&consumer_handle, 0, consumer, 0))

42:     int ret_value = 0;
43:     ERROR(pthread_join(producer_handle, (void*)&ret_value))
44:     ERROR(pthread_join(consumer_handle, (void*)&ret_value))

45:     ERROR(sem_destroy(&data_requested))
46:     ERROR(sem_destroy(&data_available))

47:     return 0;
48: }
```

Aufgabe: Entscheiden Sie **ob** es bei der Ausführung des Programms zu einer **Verklemmung** kommen kann. Antworten Sie **entweder** *Ja*, wenn Sie glauben, dass eine Verklemmung möglich ist **oder** antworten Sie *Nein*, falls Sie vom Gegenteil ausgehen.

Falls Sie *Ja* antworten, füllen Sie zudem die gegebene Tabelle mit einem beispielhaften Ablauf aus, an dessen Ende das Programm vollständig verklemmt ist. Das heißt, wenn alle Threads blockiert sind. Die ersten Einträge des Ablaufs sind als Beispiel, wie die Tabelle ausgefüllt werden soll, gegeben. In der Spalte *data* soll immer der Wert stehen, den die Variable *data* nach Ausführung der Codezeile in der Spalte *Zeile* hat.

Antwort:

Thread	Zeile	data	Thread	Zeile	data
main	38	100			
main	39	100			
main	40	100			
main	41	100			
main	42	100			
producer	16	100			
consumer	28	100			

c) Der Babier von Semaphoria

6 Punkte

Ein Babier (barber) betreibt einen Salon mit n Wartestühlen und 1 Friseurstuhl. Immer wieder betreten Kunden (customers) den Laden und tun folgendes:

- Wenn noch mindestens 1 Stuhl frei ist, nehmen sie Platz und warten auf den Babier,
- andernfalls verlassen sie den Laden direkt wieder (leave).

Gleichzeitig arbeitet der Babier wartende Kunden ab, indem er ihnen die Haare schneidet (cut_hair). Immer wenn der Babier cut_hair ausführt, werden einem Kunden die Haare geschnitten, das heißt es darf und muss für jede cut_hair Aktion des Babiers genau 1 Kunde get_hair_cut ausführen.

Aufgabe

Tragen Sie im folgenden **Pseudo-Code**-Programm auf den gestrichelten Linien (_ _) die korrekten **Ausdrücke ein**, sodass die Synchronisation alle oben genannten Regeln sicherstellt.

Gehen Sie davon aus, dass der Salon gerade erst öffnet.

```
/* gemeinsamer Speicher */
int n = <some_fixed_value_greater_zero>;
sem_t mutex = __;
sem_t barber = __;
sem_t customer = __;
int n_customers = 0;
```

```
/* Barber */
sem_wait(_______________);
sem_post(_______________);
cut_hair();
```

```
/* Customer */
sem_wait(_______________);
if(n_customers == n+1) {
  sem_post(_______________);
  leave();
}
n_customers++;
sem_post(_______________);

sem_post(_______________);
sem_wait(_______________);
get_hair_cut();

sem_wait(_______________);
n_customers--;
sem_post(_______________);
```

Aufgabe 4: Speicherverwaltung und virtueller Speicher (10 Punkte)

a) *Platzierungsstrategien*

4 Punkte

Die folgenden beiden Tabellen zeigen die Belegung eines 32 MiB Speichers an, der in 2 MiB große Blöcke eingeteilt ist.

Anfragen, die nicht in das Muster passen werden auf die nächsten 2 MiB aufgerundet.

Im Folgenden werden zwei Anfragen A und B nacheinander Speicher anfragen.

Vervollständigen Sie die jeweiligen Tabellen so, dass die nachfolgenden Zeilen der Tabelle die Speicherbelegung nach der Anfrage von A und nach der Anfrage von B repräsentieren.

Verwenden Sie dabei jeweils die angegebene Platzierungsstrategie. Kann eine Anfrage nicht beantwortet werden, kennzeichnen Sie die entsprechende Zeile der Tabelle mit einem **E** in der ersten Spalte und lassen den Rest leer.

Worst Fit-Verfahren:

Aktion	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
Start	W		X	X	X				Y	Y	Y	Y			Z	
A res. 8 KiB																
B res. 4,1 MiB																

Best Fit-Verfahren:

Aktion	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
Start	W		X	X	X				Y	Y	Y	Y			Z	
A res. 8 KiB																
B res. 4,1 MiB																

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

b) Buddy-Verfahren

3 Punkte

Erklären Sie **stichpunktartig** die Funktionsweise des Buddy-Verfahrens. Beschränken Sie sich dabei auf das Speichern von Daten. **Nennen** Sie zudem eine Datenstruktur, die sich für den Einsatz des Buddy-Verfahrens anbietet.

c) Segmentbasierte Addressberechnung

3 Punkte

Geben Sie für die logischen Adressen $0000\ 102E_{16}$ und $FE45\ 111A_{16}$ 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, berechnen Sie die theoretische physikalische Adresse aber dennoch.

Segmenttabelle:

	Startadresse	Länge
00_{16}	$9802\ 3E21_{16}$	$00\ 020F_{16}$
01_{16}	$B70B\ 1761_{16}$	$00\ AFFE_{16}$
02_{16}	$0A1A\ 62E1_{16}$	$00\ F0BA_{16}$
03_{16}	$E04D\ 0846_{16}$	$00\ 0911_{16}$
...		
FE_{16}	$0012\ 3011_{16}$	$A0\ 0000_{16}$
FF_{16}	$5B65\ 8A50_{16}$	$00\ 2304_{16}$

logische Adresse: $0000\ 102E_{16}$

→ physikalische Adresse:

logische Adresse: $FE45\ 111A_{16}$

→ physikalische Adresse:

Aufgabe 5: Dateisysteme (10 Punkte)

a) Journaled File Systems

6 Punkte

Erläutern Sie kurz die grundsätzliche Funktion von *Journaled File Systems* (JFS). Beantworten Sie hierfür die folgenden Punkte:

- **Warum** werden JFS verwendet?
- **Wie** unterscheidet sich der **Ablauf** einer Änderung am Dateisystem im Vergleich zu Systemen ohne JFS?
- **Was** passiert beim Bootvorgang?
- **Welche** Möglichkeiten bieten JFS zur Fehlererholung?

b) Speicherung von Dateien

4 Punkte

Im Bezug auf **kontinuierliche Speicherung**:

- **Erklären** Sie kurz die Funktionsweise,
- **Benennen** Sie *zwei* übliche Probleme,
- **Benennen** Sie *ein* Anwendungsszenario in welchem der Einsatz sinnvoll ist und
- **Benennen** Sie *eine* alternative Methode zur Speicherung von Dateien.

Aufgabe 6: Programmieraufgabe: Standardeingaben in Datei kopieren (31 Punkte)

Vervollständigen Sie das Programm `tee_plus`, welches kontinuierlich die Standardeingabe liest, Eingaben in die Standardausgabe und weitere Dateien schreibt, und auf ein Signal hin Statistiken ausgibt.

Beispielnutzung: `producer | tee_plus kopie1 kopie2 | consumer`

Aufgabenstellung:

Das Programm soll folgende Anforderungen erfüllen:

- Die Standardeingabe (stdin) wir regelmäßig abschnittsweise gelesen, bis das Ende des Stroms erreicht ist.
- Jeder Abschnitt der Eingabe wird direkt nach dem Lesen in alle Dateien, welche als Argumente über die Kommandozeile übergeben werden, geschrieben.
- Die Dateien in den Kommandozeilenargumenten werden dabei neu angelegt, bzw. überschrieben, falls sie schon existieren.
- Die Eingaben werden ebenfalls in die eigene Standardausgabe (stdout) geschrieben.
- Auf das Signal `SIGUSR1` hin, wird die Anzahl an übertragenen Bytes und die Zeit seit Start den Programms auf die Standardfehlerausgabe (stderr) geschrieben.
- Diese Ausgabe der Statistiken erfolgt im regulären Programmfluss, nachdem eine Unterbrechung durch das Signal festgestellt wurde. Eine entsprechende Prüfung erfolgt regelmäßig.
- Blockierende Lese- oder Schreiboperation werden ggf. durch das Signal unterbrochen und nach der Ausgabe der Statistiken fortgesetzt.

Setzen Sie diese Anforderungen innerhalb der folgenden Teilaufgaben um:

a) Definieren Sie eine Handler-Funktion, die den Zeitpunkt des Signals über eine Variable signalisiert. Registrieren Sie den Handler für das Signal `SIGUSR1`.

b) Definieren Sie eine Hilfsfunktion, welche die insgesamt übertragene Datenmenge und Laufzeit ausgibt, wenn der Handler ausgeführt wurde.

c) Definieren Sie Lese- und Schreiboperationen, welche gegen Unterbrechungen durch Signale abgesichert sind. Im Falle einer Unterbrechung werden ggf. Statistiken ausgeben und die jeweilige Operation wiederholt.

d) Bereiten Sie die Ausführung des Hauptprogramms vor, indem Sie die Startzeit bestimmen und die Dateideskriptoren vorbereiten.

e) Implementieren Sie in einer Schleife das Lesens und Schreibens der Eingaben, bis die Eingabe endet.

f) Geben Sie alle Ressourcen frei und beenden Sie das Programm ordnungsgemäß.

Relevante Systemaufrufe:

- Beigefügten Handbuchseiten: `sigaction`, `open`, `read`, `write`, `close`, `malloc`, `time`

Wichtige Hinweise:

- *Belegte Ressourcen* (d.h. geöffnete Dateien, allozierter Speicher) müssen im Normalfall *freigegeben* werden. Ausnahme: Fehlerfall!
- Achten Sie in Ihrer Implementierung auf *korrekte Fehlerbehandlung* aller Systemaufrufe und Hilfsfunktionen. Nutzen Sie `error("Name des Aufrufs")`, um Schreibarbeit zu sparen (Achtung: ggf. auf `errno` achten).
- Einfache syntaktische Fehler (z.B. vergessene Strichpunkte) führen nicht zu Punktabzug, es geht um die semantische Umsetzung.
- Beachten Sie den Unterschied zwischen Dateideskriptoren und Ein-/Ausgabeströmen.

Code:

```
// Gehen Sie davon aus, dass alle notwendigen Header eingebunden sind.
// Vorgaben {
// Format für die Statistik
#define STAT_FORMAT "Bytes␣übertragen:␣%zd␣in␣%.1ld\n"
// Makros für Standard-Eingabe/-Ausgabe/-Fehler Filehandles
#define STD_IN STDIN_FILENO
#define STD_OUT STDOUT_FILENO
#define STD_ERR STDERR_FILENO
// Kurzform für Fehlerbehandlung (Programmabbruch)
void error(char* msg) {
    perror(msg);
    exit(errno);
}
// Verarbeitungs-Zeitpunkt des Signals
static volatile time_t sigusr1 = 0;
// Start-Zeitpunkt des Programms
static time_t start;
// Gesamtzahl der gelesenen Bytes
static size_t bytes_total = 0;
// Vorgaben }
```

a) Handler-Funktion implementieren & registrieren (6.5 Punkte)

```
// Handler-Funktion {
// HIER IHREN CODE EINFÜGEN

// Handler-Funktion }
static void register_handler() {
    // Handler-Registrierung {
    // HIER IHREN CODE EINFÜGEN

    // Handler-Registrierung }
}
```

b) Statistik ausgeben (4 Punkte)

```
void print_statistics() {
    // Statistikausgabe {
    // HIER IHREN CODE EINFÜGEN

    // Statistikausgabe }
}
```

c) Sicheres Lesen und Schreiben (4.5 Punkte)

```
// do_write: 1 falls save write gewünscht; sonst save read
ssize_t save_rw(
    int fd, void* buf, size_t count, int do_write
) {
    ssize_t n = -1;
    while (n < 0) {
        // Sicheres Lesen und Schreiben {
        // HIER IHREN CODE EINFÜGEN
        // Sicheres Lesen und Schreiben }
    }
    return n;
}
```

d) Programmstart (7.5 Punkte)

```
int main(int argc, char *argv[])
{
    register_handler();
    char buf[4096];
    // Startzeit bestimmen {
    // HIER IHREN CODE EINFÜGEN



    // Startzeit bestimmen }
    // Dateideskriptoren vorbereiten {
    // HIER IHREN CODE EINFÜGEN



    // Dateideskriptoren vorbereiten }
```

e) Hauptprogrammschleife (6.5 Punkte)

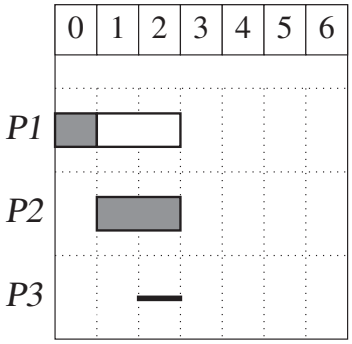
```
    for (;;) {
        print_statistics();
        // Lesen der Eingabe {
        // HIER IHREN CODE EINFÜGEN



        // Lesen der Eingabe }
        // Schreiben des Gelesenen {
        // HIER IHREN CODE EINFÜGEN



        // Schreiben des Gelesenen }
    }
```

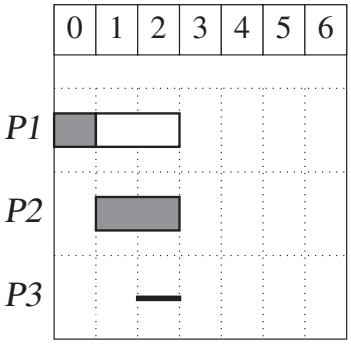
f) Programmende (2 Punkte)

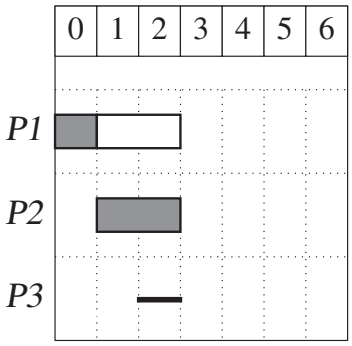
```
    // Aufräumen {
    // HIER IHREN CODE EINFÜGEN



    // Aufräumen }
    return EXIT_SUCCESS;
}
```

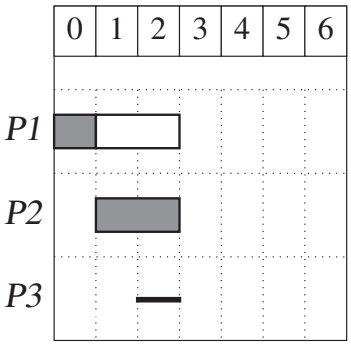
Reserveblatt

Bitte die Strategie mit angeben!









Bitte das Verfahren angeben:

Aktion	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
Start	W		X	X	X				Y	Y	Y	Y			Z	
A res. 8 KiB																
B res. 4,1 MiB																

Bitte das Verfahren angeben:

Aktion	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
Start	W		X	X	X				Y	Y	Y	Y			Z	
A res. 8 KiB																
B res. 4,1 MiB																