Technische Universität Dortmund

Klausur „Betriebssysteme“

	erreichbare Punkte	erhaltene Punkte				
Aufgabe 1	8	**a**	**b**	**c**	**d**	
Aufgabe 2	10	**a**	**b**	**c**		
Aufgabe 3	9	**a**	**b**			
Aufgabe 4	11	**a**	**b**			
Aufgabe 5	7	**a**	**b**			
Aufgabe 6	15	**a**	**b**			
Summe	60					
Spickzettel vorhanden?						

(Matrikel-Nr.)

(Name) (Vorname)

Durch meine Unterschrift bestätige ich

- den Empfang der vollständigen Klausur (15 Seiten inklusive Deckblatt),
- den Empfang der Manualseiten (ein Blatt mit 3 Manualseiten: clearerr/ferror/feof, fopen/fdopen/fileno und fread/fwrite),
- die Kenntnisnahme der Hinweise auf Seite 2.

Dortmund, 08.09.2022

(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), 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) **Prozesse und Scheduling:** Welche Aussage ist über die Funktionsweise der `exec()`-Familie richtig?

2 Punkte

- ☐ Dem Eltern-Prozess wird die Prozess-ID des Kind-Prozesses zurückgeliefert.
- ☐ Der an `exec()` übergebene Funktionszeiger wird durch einen neuen Thread im aktuellen Prozess ausgeführt.
- ☐ `exec()` erzeugt einen neuen Kind-Prozess und startet darin das angegebene Programm.
- ☐ Falls kein Fehler auftritt, kehrt der Aufruf von `exec()` nicht zurück.

b) **Speicher:** Welche Aussage trifft auf das *Demand-Paging* zu?

2 Punkte

- ☐ Demand-Paging setzt eine segmentierte Speicherverwaltung voraus.
- ☐ Demand-Paging erlaubt es, größere logische Adressräume anzulegen, als Hauptspeicher vorhanden ist. Allerdings muss vorausgesetzt werden, dass ein Prozess nicht alle Seiten des logischen Adressraums tatsächlich anspricht.
- ☐ Demand-Paging benötigt keinerlei Hardware-Unterstützung, da sich alle benötigten Mechanismen auch ohne MMU realisieren lassen.
- ☐ Demand-Paging lädt eine Seite erst dann in den Hauptspeicher, wenn sie tatsächlich angesprochen wird. Nicht benutzte Seiten werden unter Umständen aus dem Hauptspeicher ausgelagert.

c) **Synchronisation und Verklemmungen:** Welche der folgenden Aussagen zum Thema Synchronisation ist richtig?

2 Punkte

- ☐ Bei einem Zyklus im Betriebsmittelbelegungsgraph liegt ein unsicherer Zustand vor.
- ☐ Ein Anwendungsprozess muss bei der Verwendung von Semaphoren Interrupts sperren, um Probleme durch Nebenläufigkeit zu verhindern.
- ☐ Ein Mutex kann ausschließlich für einseitige Synchronisation verwendet werden.
- ☐ Gibt ein Prozess einen Mutex frei, den er selbst zuvor nicht angefordert hatte, stellt dies einen Programmierfehler dar; der fehlerhafte Prozess sollte dann abgebrochen werden.

d) **Sicherheit:** Welche Aussage über Systemsicherheit ist korrekt?

2 Punkte

- ☐ Die Isolation des Adressraums eines Prozesses ist ohne eine MMU nicht möglich.
- ☐ Der Einsatz von Systemaufrufen (system calls) im Anwendungsprogramm stellt eine Verletzung der Privilegseparation dar.
- ☐ Das Betriebssystem überprüft alle Speicherzugriffe und verhindert Pufferüberläufe.
- ☐ Die Privilegseparation auf Hardwareebene (z.B. Schutzringe) gibt dem Betriebssystem die Kontrolle über die laufenden Prozesse des Systems.

Aufgabe 2: Prozesse und Scheduling (10 Punkte)

a) UNIX-Systemaufrufe

4 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.

```
1     int x = 1;
2     void next() {
3         printf("%d,␣", x++);
4     }
5     int main() {
6         next();
7         pid_t pid = fork();
8         if (pid > 0) {
9                 wait(NULL);
10                next();
11                int x = 0;
12                next();
13        } else if (pid == 0) {
14                next();
15        } else {
16                next();
17                printf("Ohje\n");
18        }
19        return 0;
20    }
```

Ausgabe:

---

b) Fehlerbehandlung

2 Punkte

Wir nehmen nun an, dass unmittelbar nach Start des Programms (noch vor dem fork-Aufruf) die maximal erlaubte Prozesszahl des ausführenden Nutzers auf 1 beschränkt wird. Es darf also nur ein einziger Prozess dieses Nutzers gleichzeitig existieren. Geben Sie hier die Ausgabe an, die in diesem Szenario vom Programm ausgegeben wird.

Ausgabe:

---

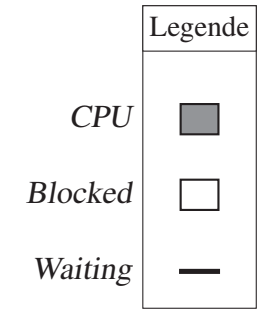
c) Scheduling-Verfahren

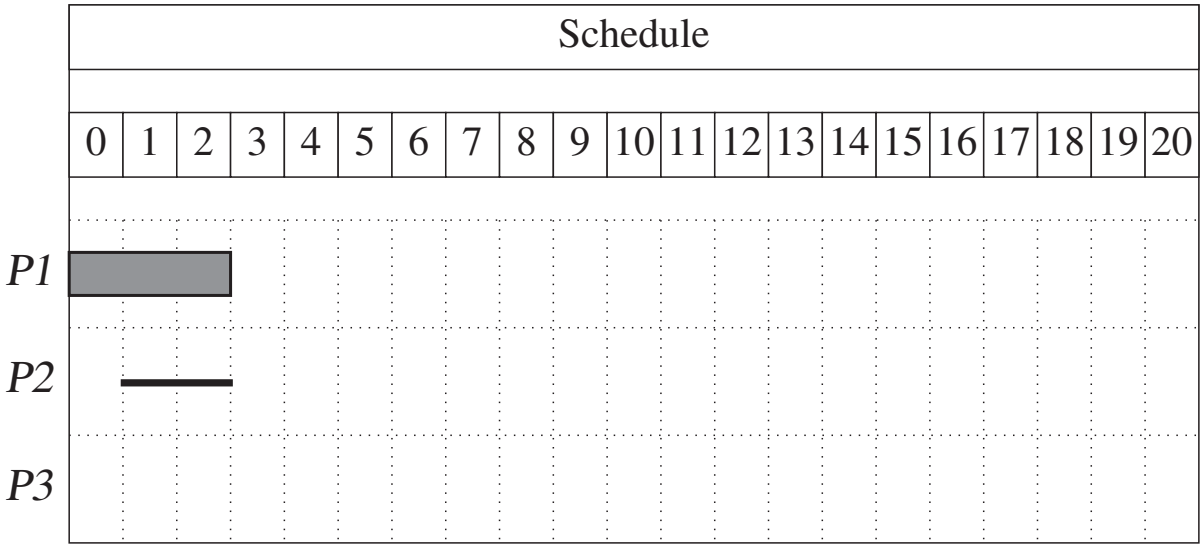
4 Punkte

Ein Betriebssystem verwaltet **drei zyklisch** arbeitende Prozesse P1, P2 und P3. Jeder Prozess führt zunächst Berechnungen auf der CPU aus. Sobald diese vollständig abgeschlossen sind, folgt ein E/A-Stoß, anschließend ist der Prozess wieder rechenbereit. Die Prozesse treffen zum Zeitpunkt der in der Tabelle angegebenen Ankunftszeit ein. In der folgenden Tabelle sind die Zeitangaben für die Ankunftszeit und Dauer von CPU- und E/A-Stößen jeweils in ms angegeben.

Prozess	Ankunftszeit	CPU-Zeit	E/A-Zeit
P1	0	4	3
P2	1	6	5
P3	4	3	4

Zeichnen Sie in das folgende Gantt-Diagramm ein, wie die drei Prozesse P1, P2 und P3 abgearbeitet werden, wenn das Scheduling nach der **Virtual-Round-Robin-Strategie** mit einer Zeitscheibendauer von **3 ms** vorgenommen wird. Die Prozessumschaltzeit kann vernachlässigt werden. E/A-Vorgänge können parallel ausgeführt werden. Markieren Sie in dem folgenden Diagramm die Prozesszustände entsprechend der Legende. Sie müssen die Kasten für *CPU* nicht vollständig ausfüllen. Es genügt, wenn Sie die Fläche schraffieren.
Hinweis: Die ersten drei Zeiteinheiten sind bereits fertig ausgefüllt.





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

Aufgabe 3: Synchronisation und Verklemmungen (9 Punkte)

a) Provozierte Verklemmungen

5 Punkte

Die beiden **Semaphore x** und **y** repräsentieren jeweils eine unteilbare System-Ressource und sind **mit 1 initialisiert**. Der rechte und linke Programmausschnitt wird jeweils von verschiedenen Prozessen ausgeführt. Vor dem Aufruf von `work_with_x_y()` sollen beide Ressourcen belegt, danach wieder freigegeben werden. Sorgen Sie durch Einfügen von Semaphor-Operationen (`p(&x), p(&y), v(&x), v(&y)`) in beiden Ausschnitten dafür, dass beim Belegen der Ressourcen eine Verklemmung möglich (aber nicht zwingend!) wird. Erhöhen Sie die Wahrscheinlichkeit für eine Verklemmung, indem Sie an geeigneten Stellen `sleep(1)` (1s Warten; ein Aufruf pro Programm) einfügen.

```
/* Ausschnitt von Programm 1 */      /* Ausschnitt von Programm 2 */
```

```
    work_with_x_y();                    work_with_x_y();
```

---

b) Verklemmungen

4 Punkte

Erklären Sie den Unterschied zwischen der Verklemmungsvorbeugung (*deadlock prevention*) und der Verklemmungsvermeidung (*deadlock avoidance*). Warum ist die Verklemmungsvermeidung in der Praxis kaum einzusetzen?

Aufgabe 4: Speicherverwaltung und virtueller Speicher (11 Punkte)

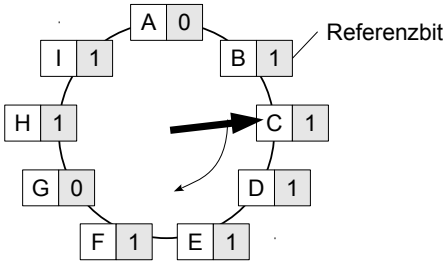
a) Seitenersetzungsstrategien (8 Punkte)

I) *Second Chance*

3 Punkte

In diesem Szenario soll die Seitenersetzungsstrategie *Second Chance* angewendet werden, welche mit Hilfe eines Umlaufzeigers und einem Referenzbit pro Seitenrahmen (Speicherkachel) umgesetzt wird. Gegeben sei der unten gezeigte Initialzustand. In den nächsten **drei** Schritten sollen nun neue, noch nicht vorhandene Seiten eingelagert werden. Geben Sie an, welche Seiten (A bis I) bei diesen drei Einlagerungsvorgängen ersetzt werden.

Anmerkung: Während des Vorgangs finden keine Anfragen der vorhandenen Seiten statt. Achten Sie auf die richtige Reihenfolge der Ersetzungen.



1. ersetzte Seite	
2. ersetzte Seite	
3. ersetzte Seite	

II) Theorie

1 Punkt

Zu welcher in der Realität nur schwer zu implementierenden Ersetzungsstrategie stellt *Second Chance* eine Annäherung dar?

III) *Least Recently Used*

4 Punkte

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

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

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		5	1	2	3	4	3	1	5	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!)

b) Verschnitt

3 Punkte

Erklären Sie den Unterschied zwischen *internem* und *externem* Verschnitt.

Aufgabe 5: E/A und Dateisysteme (7 Punkte)

a) E/A-Scheduling

4 Punkte

Gegeben sei ein Plattenspeicher mit 16 Spuren. Der E/A-Scheduler bekommt immer wieder Aufträge für eine bestimmte Spur. Die Leseaufträge in L_0 sind dem E/A-Scheduler bereits bekannt. Nach einem bearbeiteten Auftrag erhält er die Aufträge in L_1. Nach vier weiteren (d.h. nach insgesamt fünf) bearbeiteten Aufträgen erhält er die Aufträge in L_5. Zu Beginn befindet sich der Schreib-/Lesekopf über Spur 0.

$L_0 = \{3, 7, 9, 15\},\ L_1 = \{2, 13\},\ L_5 = \{0, 7, 8, 12\}$

Tragen Sie hier die Reihenfolge der gelesenen Spuren für einen E/A-Scheduler ein, der nach der **Fahrstuhlstrategie (Elevator)** arbeitet.

Ersatztabelle (weitere auf dem Reserveblatt):

b) Geräteklassen

3 Punkte

Nennen Sie die aus der Vorlesung bekannten Geräteklassen von E/A-Geräten und erläutern Sie kurz den Unterschied zwischen diesen.

Aufgabe 6: Programmierung (15 Punkte)

Implementieren Sie in zwei Teilen das Programm `countbytes`, welches mit einer beliebigen Anzahl an Dateien, mindestens jedoch einer, aufgerufen wird. Für jede angegebene Datei soll die Anzahl der darin enthaltenden Bytes bestimmt werden. Dazu müssen die Systemaufrufe `fopen()`, `fread()` und `fclose()` verwendet werden.

Schnittstelle: **`./countbytes`** [Datei1] [Datei2] [Datei3] ...

Dieser Beispielaufruf „**`./countbytes`** a.txt programm.c“würde die nachfolgende Ausgabe erzeugen:

```
Die Datei 'a.txt' enthält 42 Bytes.
Die Datei 'programm.c' enthält 32 Bytes.
```

Funktion `main`

- Zunächst soll die Argumentenliste auf den korrekten Aufruf hin überprüft werden.
- Anschließend soll Ihr Programm für jedes Argument die Funktion `int counts_bytes(char *filename)` aufrufen.
- Für jeden Aufruf soll der Rückgabewert in einer Variablen zwischengespeichert werden.
- Zum Abschluss soll die untersuchte Datei sowie die Anzahl der darin enthaltenen Bytes ausgegeben werden. Die Ausgabe kann z. B. so aussehen:
  `Die Datei 'a.txt' enthält 42 Bytes.“`

Funktion `count_bytes`

- Öffnen Sie die übergebene Datei zum Lesen.
- Nun soll die Datei mittels `fread` in Stücken eingelesen werden. Verwenden Sie hierzu einen Buffer der Größe `BUF_SIZE`.
- Die Anzahl der eingelesenen Bytes soll aufsummiert werden.
- Zum Schluss muss die Datei geschlossen und die Summe der Bytes zurückgegeben werden.

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: `fopen/fclose`, `fread`, `clearerr/ferror/feof`

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

a) `main`-Funktion (5 Punkte)

```
#include <stdio.h>
#include <stdlib.h>
#define BUF_SIZE 100

static size_t count_bytes(const char *filename);

int main(int argc, const char *argv[]) {
}
```

a:

b) Funktion `count_bytes` (10 Punkte)

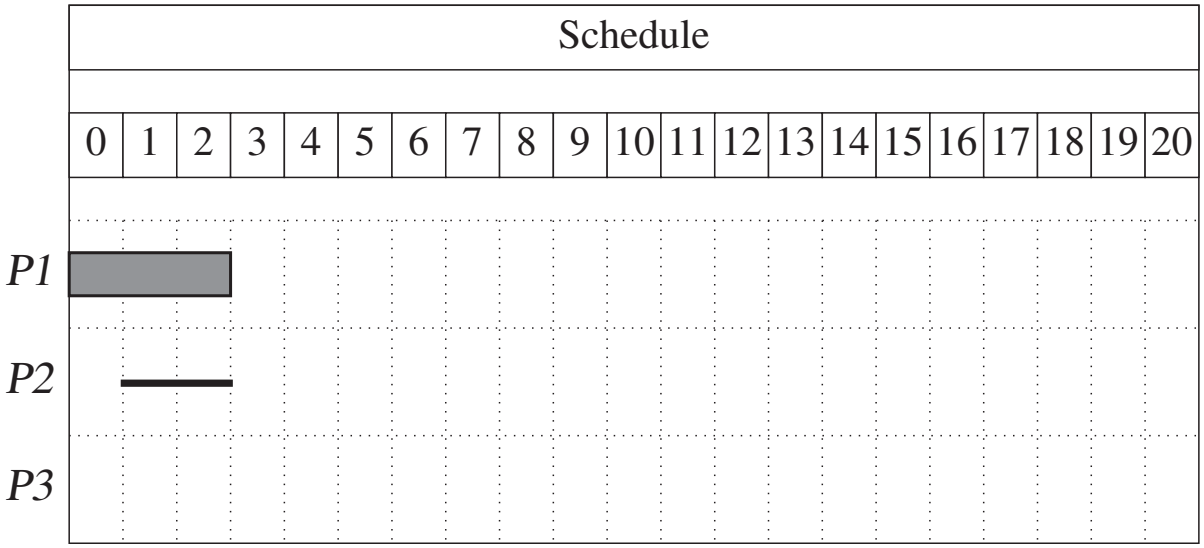
```
static size_t count_bytes(const char *filename) {

}
```

b:

Reserveblatt

Ersatzganttchart für Aufgabe 2



Ersatztabelle für Aufgabe 4

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

Ersatztabelle für Aufgabe 5
