Aufgabe 1.1: Einfachauswahl-Fragen (22 Punkte)

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

Wollen Sie eine Multiple-Choice-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) Man unterscheidet die Begriffe *Programm* und *Prozess*. Welche der folgenden Aussagen zu diesem Thema ist richtig? 2 Punkte

- ❏ Der UNIX-Systemaufruf `fork(2)` lädt eine Programmdatei in einen neu erzeugten Prozess.
- ❏ Mit Hilfe von Threads kann ein Prozess mehrere Programme gleichzeitig ausführen.
- ❏ Der C-Präprozessor erzeugt aus mehreren Programmteilen (Modulen) einen Prozess.
- ❏ Ein Programm kann durch mehrere Prozesse gleichzeitig ausgeführt werden.

b) Man unterscheidet Traps und Interrupts. Welche Aussage ist richtig? 2 Punkte

- ❏ Ein Trap wird immer unmittelbar durch die Aktivität des aktuell laufenden Prozesses ausgelöst.
- ❏ Weil das Betriebssystem nicht vorhersagen kann, wann ein Prozess einen Systemaufruf tätigt, sind Systemaufrufe in die Kategorie *Interrupt* einzuordnen.
- ❏ Der Zugriff auf eine physikalische Adresse kann nie zu einem Trap führen.
- ❏ Wenn ein Interrupt einen schwerwiegenden Fehler signalisiert, muss das Betriebssystem den unterbrochenen Prozesses sofort beenden.

c) Welche Aussage zum Thema RAID ist richtig? 2 Punkte

- ❏ RAID 5 kann den gleichzeitigen Ausfall von mehr als einer Platte kompensieren.
- ❏ Bei RAID 4 nutzen sich die im Verbund beteiligten Platten ungleichmäßig stark ab.
- ❏ Der Ausfall einer Platte in einem RAID-0-System führt nicht zu Datenverlust.
- ❏ Der Lesedurchsatz ist bei einem RAID-1-System geringer als bei einem System ohne RAID, weil beide Platten beauftragt werden müssen.

d) Welche Aussage über Einplanungsverfahren ist richtig? 2 Punkte

- ❏ Statisch vorab (*offline*) berechnete, deterministische Ablaufpläne eignen sich für Systeme mit strikten Echtzeitanforderungen.
- ❏ Der Konvoieffekt kann bei kooperativen Einplanungsverfahren wie *First-Come-First-Served* nicht auftreten.
- ❏ *Virtual-Round-Robin* ist für den interaktiven Betrieb ungeeignet.
- ❏ *Round-Robin* bevorzugt E/A-intensive Prozesse zu Gunsten von rechenintensiven Prozessen.

e) Welche Aussage zu Programmbibliotheken ist richtig? 2 Punkte

- ❏ Statische Bibliotheken können nicht in C implementiert werden.
- ❏ Eine statische Bibliothek, die in ein Programm eingebunden wurde, muss zum Ladezeitpunkt dieses Programms im Dateisystem vorhanden sein.
- ❏ Programm-Module, die von mehreren Anwendungen gemeinsam genutzt werden, können in Form einer dynamischen Bibliothek zentral installiert werden, um Speicherplatz zu sparen.
- ❏ Änderungen am Code einer dynamischen Bibliothek (z. B. Bugfixes) erfordern immer das erneute Binden aller Programme, die diese Bibliothek benutzen.

f) Wodurch kann es in einem System zu Nebenläufigkeit kommen? 2 Punkte

- ❏ Durch Traps.
- ❏ Durch Seitenflattern.
- ❏ Durch Multithreading auf einem Monoprozessorsystem.
- ❏ Durch langfristiges Scheduling.

g) Welche Aussage zu Speicherzuteilungsverfahren ist richtig? 2 Punkte

- ❏ Bei allen listenbasierten Zuteilungsverfahren (*First-*, *Next-*, *Best-*, *Worst-Fit*) kann externer Verschnitt auftreten.
- ❏ Interne Fragmentierung ist bei Freispeicherverwaltung mittels einer Bitkarte nicht möglich.
- ❏ Bei *Worst-Fit* ist das Verschmelzen freier Blöcke besonders einfach.
- ❏ Beim *Buddy*-Verfahren können zwei gleich große Freispeicherblöcke, die nebeneinander im Speicher liegen, in jedem Fall verschmolzen werden.

h) Welche Aussage zur Seitenumlagerung in virtuellen Adressräumen ist richtig? 2 Punkte

- ❏ Die Seitenersetzungsstrategie *Second Chance* (*Clock*) ist nur in der Theorie interessant, weil ihre Implementierung komplexe Datenstrukturen erfordert.
- ❏ Unter *Seitenflattern* versteht man das ständige Ein- und Auslagern von Speicherseiten, wenn der physisch vorhandene Hauptspeicher nicht ausreicht.
- ❏ Bei der Seitenersetzungsstrategie *LRU* wird diejenige Seite ausgelagert, auf die in der Vergangenheit am seltensten zugegriffen wurde.
- ❏ Beim Auslagern einer Speicherseite muss der zugehörige Seitendeskriptor angepasst werden. Beim Einlagern ist dies nicht nötig.

i) Welche Aussage über Prozesszustände ist auf einem Monoprozessorsystem mit präemptiver Einplanungsstrategie richtig? 2 Punkte

- ❏ Ein Prozess kann nicht durch eigene Aktivität vom Zustand *laufend* in den Zustand *blockiert* gelangen.
- ❏ Da sich zu jedem Zeitpunkt maximal ein Prozess im Zustand *laufend* befinden kann, sind in mehrfädigen Programmen keinerlei Synchronisationsmaßnahmen erforderlich.
- ❏ Die Monopolisierung der CPU durch einen Prozess ist nicht möglich, da das Betriebssystem den Übergang von *laufend* nach *bereit* erzwingen kann.
- ❏ Ein *blockierter* Prozess kann sich selbst durch Ausführen des Systemaufrufs `exec(2)` in den Zustand *laufend* überführen.

j) Für lokale Variablen, Aufrufparameter usw. einer Funktion wird bei vielen Prozessoren ein Stack-Frame angelegt. Welche Aussage ist richtig? 2 Punkte

- ❏ Es ist nicht möglich auf lokale *automatic*-Variablen zuzugreifen, die sich im Stack-Frame einer anderen Funktion befinden.
- ❏ Bei rekursiven Funktionsaufrufen kann der Speicher des Stack-Frames in jedem Fall wiederverwendet werden, weil die gleiche Funktion aufgerufen wird.
- ❏ Ein Pufferüberlauf eines lokalen Arrays wird immer zu einem *Segmentation Fault* führen und kann somit keine sicherheitskritischen Auswirkungen haben.
- ❏ Wenn in einem UNIX-Prozess mehrere Threads parallel laufen, benötigt jeder von ihnen einen eigenen Stack.

k) Welche Aussage zum Thema Adressraumschutz ist richtig? 2 Punkte

- ❏ Adressraumschutz durch Abteilung eignet sich besonders für Systeme, die von mehreren Nutzern gleichzeitig verwendet werden.
- ❏ Beim Adressraumschutz durch Eingrenzung erfolgt die Umsetzung von Programmadressen zu Hauptspeicheradressen zur Laufzeit durch die MMU.
- ❏ Adressraumschutz durch Segmentierung benötigt einen verschiebenden Lader, um Programmadressen an Arbeitsspeicheradressen zu binden.
- ❏ Bei allen Verfahren des Adressraumschutzes führt jeder Zugriff auf eine ungültige Speicheradresse zu einem Trap.

Aufgabe 1.2: Mehrfachauswahl-Fragen (8 Punkte)

Bei den Multiple-Choice-Fragen in dieser Aufgabe sind jeweils m Aussagen angegeben, n ($0 \leq n \leq m$) Aussagen davon sind richtig. Kreuzen Sie **alle richtigen** Aussagen an. Jede korrekte Antwort in einer Teilaufgabe gibt einen halben Punkt, jede falsche Antwort einen halben Minuspunkt. Eine Teilaufgabe wird minimal mit 0 Punkten gewertet, d. h. falsche Antworten wirken sich nicht auf andere Teilaufgaben aus.

Wollen Sie eine falsch angekreuzte Antwort korrigieren, streichen Sie bitte das Kreuz mit drei waagrechten Strichen durch (☒).

Lesen Sie die Frage genau, bevor Sie antworten!

a) Welche der folgenden Aussagen zum Thema Synchronisation sind richtig? 4 Punkte

- ❍ Der Einsatz von nicht-blockierenden Synchronisationsmechanismen kann nicht zu Verklemmungen führen.
- ❍ Wenn gewöhnliche Anwendungsprozesse Interrupts sperren könnten, wäre kein effektiver Schutz vor CPU-Monopolisierung möglich.
- ❍ Für nicht-blockierende Synchronisationsverfahren ist spezielle Unterstützung durch das Betriebssystem notwendig.
- ❍ Semaphore lassen sich für mehrseitige Synchronisation einsetzen.
- ❍ Semaphore lassen sich nicht für einseitige Synchronisation einsetzen.
- ❍ Zur Synchronisation eines kritischen Abschnitts ist passives Warten immer besser geeignet als aktives Warten.
- ❍ Monitore sind nur für die Synchronisation in Monoprozessorsystemen einsetzbar.
- ❍ Für die Synchronisation zwischen dem Hauptprogramm und einer Signalbehandlungsfunktion sind Schlossvariablen (*Locks*) ungeeignet.

b) Welche der folgenden Aussagen zum Thema Dateispeicherung sind richtig?

4 Punkte

- ❍ Bei kontinuierlicher Speicherung von Dateien ist es unter Umständen mit großem Aufwand verbunden, eine bestehende Datei zu vergrößern.
- ❍ Bei indizierter Speicherung von Dateien ensteht externer Verschnitt auf der Platte.
- ❍ Bei einem Journaling-Dateisystem werden Änderungen immer zuerst am Dateisystem durchgeführt und anschließend in der Log-Datei protokolliert.
- ❍ Eine Datei in einem Winows-NT-Dateisystem kann nur genau einen Dateinamen haben, da dieser in ihrem Master-File-Table-Eintrag gespeichert ist.
- ❍ Der *Inode* einer Datei wird getrennt von ihrem Inhalt auf der Platte gespeichert.
- ❍ Festplatten eignen sich besser für sequentielle als für wahlfreie Zugriffsmuster.
- ❍ Extents sind ein Konzept in modernen Dateisystemen, das die Vorteile kontinuierlicher Speicherung mit der Flexibilität indizierter Speicherung verbindet.
- ❍ Journaling-Dateisysteme können bei einer Datei einen defekten Datenblock auf der Platte kompensieren, da die Informationen aus der Log-Datei rekonstruiert werden können.

Aufgabe 2: pinboard (60 Punkte)

Sie dürfen diese Seite zur besseren Übersicht bei der Programmierung heraustrennen!

Schreiben Sie ein Programm `pinboard`, das eine Pinnwand als Netzwerkdienst auf dem TCP/IPv6-Port 1952 (`LISTEN_PORT`) anbietet. Nutzer können die an der Pinnwand hängenden Nachrichten lesen oder eine neue Nachricht hinterlassen. Jede Nachricht besteht aus einem Titel und einem Text und wird als Datei im Verzeichnis `/var/lib/pinboard` (Makro `MESSAGE_DIR`) abgespeichert, wobei der Dateiname dem Titel und der Dateiinhalt dem Nachrichtentext entspricht.

Ein Client sendet pro Verbindung genau eine Befehlszeile mit einer Maximallänge von 1024 (`MAX_LINE_LEN`) Zeichen. Der Server unterstützt zwei Befehle:

- `show` gibt für alle gespeicherten Nachrichten deren Titel und Nachrichtentext gemäß dem folgenden Beispiel aus (Titel der Beispielnachricht ist `"SP"`):

```
== SP ==
Diese Nachricht enthält zwei Zeilen.
Viel Erfolg in der Klausur! :-)
```

- `pin` *<msgTitle>* hängt eine neue Nachricht mit dem Titel *msgTitle* an die Pinnwand. Der Nachrichtentext wird vom Client im Anschluss an die Anfragezeile gesendet (beliebig viele Zeilen, bis EOF). Ein gültiger Nachrichtentitel darf **nicht** mit einem `'.'` beginnen und **keinen** `'/'` enthalten.

Das Programm soll folgendermaßen strukturiert sein:

- Das Hauptprogramm nimmt auf einem Socket Verbindungen von außen an. Für jede eingehende Verbindung wird ein eigener Prozess gestartet, der die Funktion `serve()` ausführt. Hierbei sollen nicht mehr als 32 (`MAX_CLIENTS`) Kindprozesse gleichzeitig aktiv sein. Ist die Maximalzahl bereits erreicht und eine neue Verbindung wird akzeptiert, wird passiv gewartet, bis ein Kindprozess sich beendet.
- Funktion `void serve(int clientSock)`: Liest die Anfragezeile vom Client ein, wertet sie aus und ruft zur Abarbeitung die passende Funktion (`show()` oder `pin()`) auf. Eine leere Zeile ist ohne Fehlermeldung zu ignorieren. Falls die Anfragezeile keinen gültigen Befehl enthält oder die Abarbeitung des Befehls fehlgeschlagen ist, erhält der Client die Antwortzeile `"Failed!"`. Nach erfolgreich bearbeiteter Anfrage wird am Ende die Zeile `"OK"` gesendet.
- Funktion `int show(FILE *tx)`: Durchsucht das Nachrichtenverzeichnis (nicht-rekursiv) nach regulären Dateien und sendet deren Namen (= Nachrichtentitel) und Inhalt gemäß dem oben angeführten Schema an den Client. Dateien, deren Name mit einem `'.'` beginnt, werden ignoriert. Tritt ein Fehler auf, so soll die Verarbeitung nach Möglichkeit fortgesetzt werden.
  Im Erfolgsfall liefert die Funktion 0 zurück, im Fehlerfall -1.
- Funktion `int pin(FILE *rx, const char msgTitle[])`: Überprüft zunächst den Nachrichtentitel `msgTitle` auf Gültigkeit (siehe oben; **Tipp**: Benutzen Sie die Funktion **`strchr(3)`**, deren Man-Page mitgeliefert ist). Anschließend wird im Nachrichtenverzeichnis eine Datei erstellt, deren Name dem Nachrichtentitel entspricht. In diese Datei wird der komplette Nachrichtentext geschrieben, den der Client sendet.
  Im Erfolgsfall liefert `pin()` den Wert 0 zurück, bei Fehler oder ungültigem Nachrichtentitel -1.

Auf den folgenden Seiten finden Sie ein Gerüst für das beschriebene Programm. In den Kommentaren sind nur die wesentlichen Aufgaben der einzelnen zu ergänzenden Programmteile beschrieben, um Ihnen eine gewisse Leitlinie zu geben. Es ist überall sehr großzügig Platz gelassen, damit Sie auch weitere notwendige Anweisungen entsprechend Ihrer Programmierung einfügen können.

Einige wichtige Manual-Seiten liegen bei - es kann aber durchaus sein, dass Sie bei Ihrer Lösung nicht alle diese Funktionen oder gegebenenfalls auch weitere Funktionen benötigen.

```
#include <dirent.h>
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>

// Konstanten, Hilfsfunktionen
#define LISTEN_PORT  1952
#define MAX_CLIENTS  32
#define MAX_LINE_LEN 1024
#define MESSAGE_DIR  "/var/lib/pinboard"

static void die(const char message[]) {
    perror(message);
    exit(EXIT_FAILURE);
}

// Funktionsdeklarationen, globale Variablen usw.
```

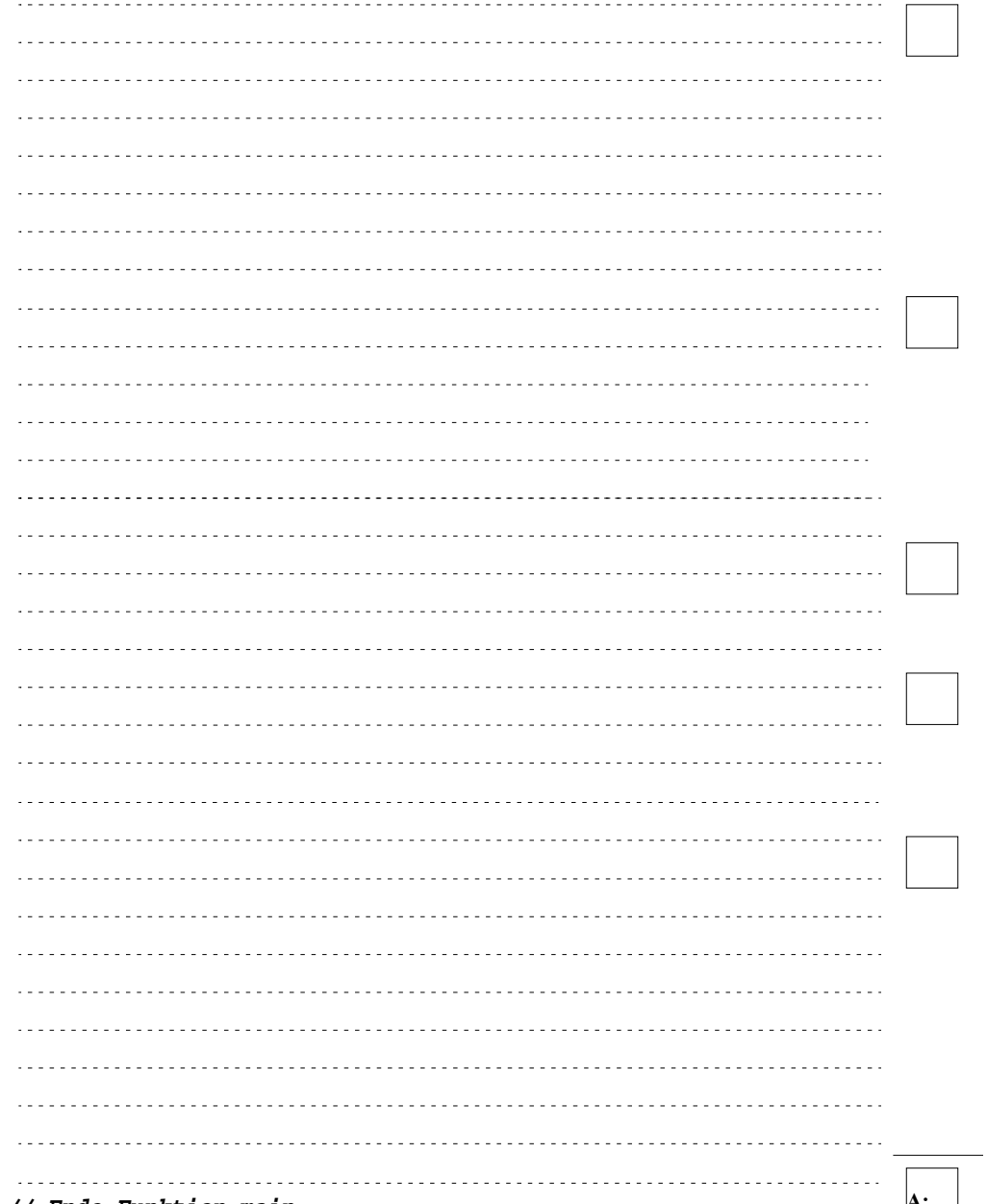


```
// Funktion main()
```

```
    // Socket erstellen und auf Verbindungsannahme vorbereiten
```

S:

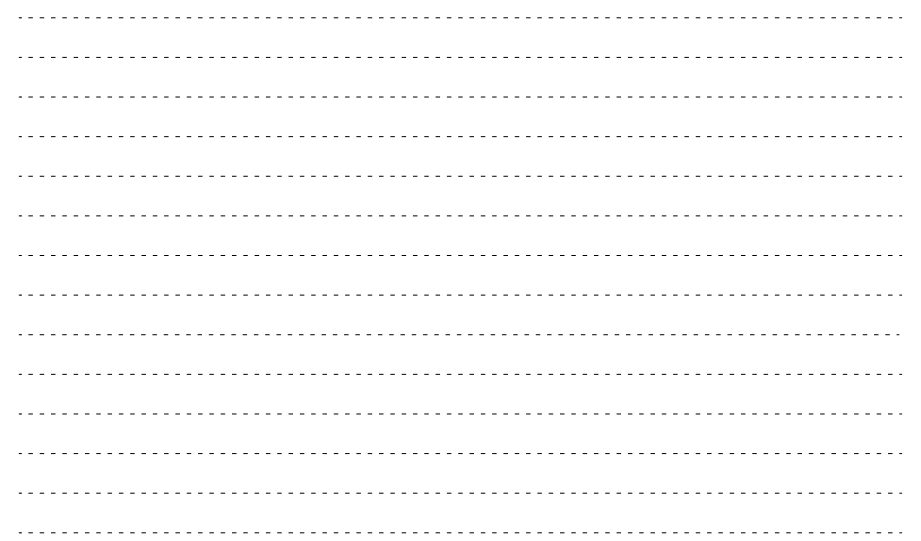
```
// Verbindungen annehmen und Kindprozesse starten
```



```
// Ende Funktion main
```

A:

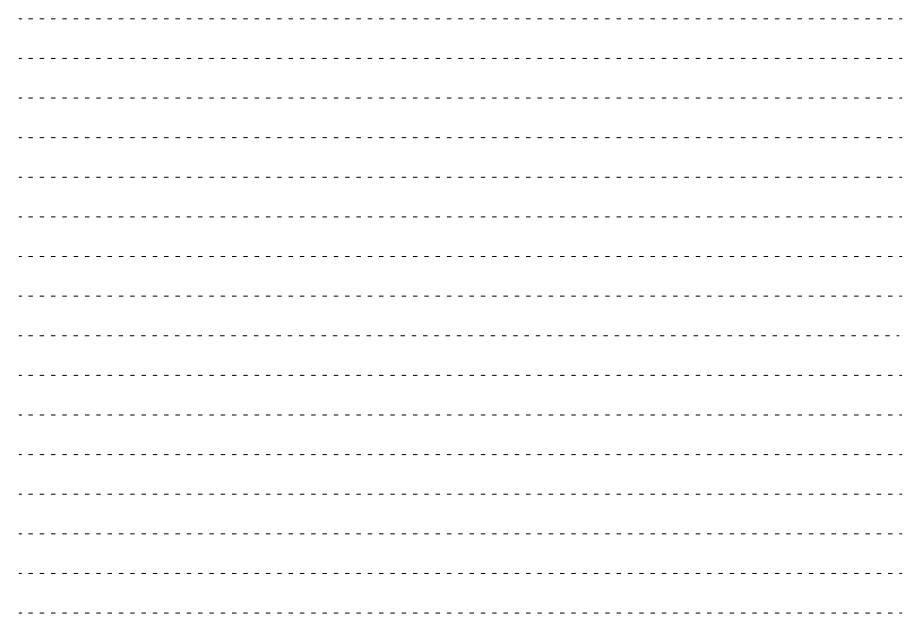

```
// Signalbehandlung für SIGCHLD
```

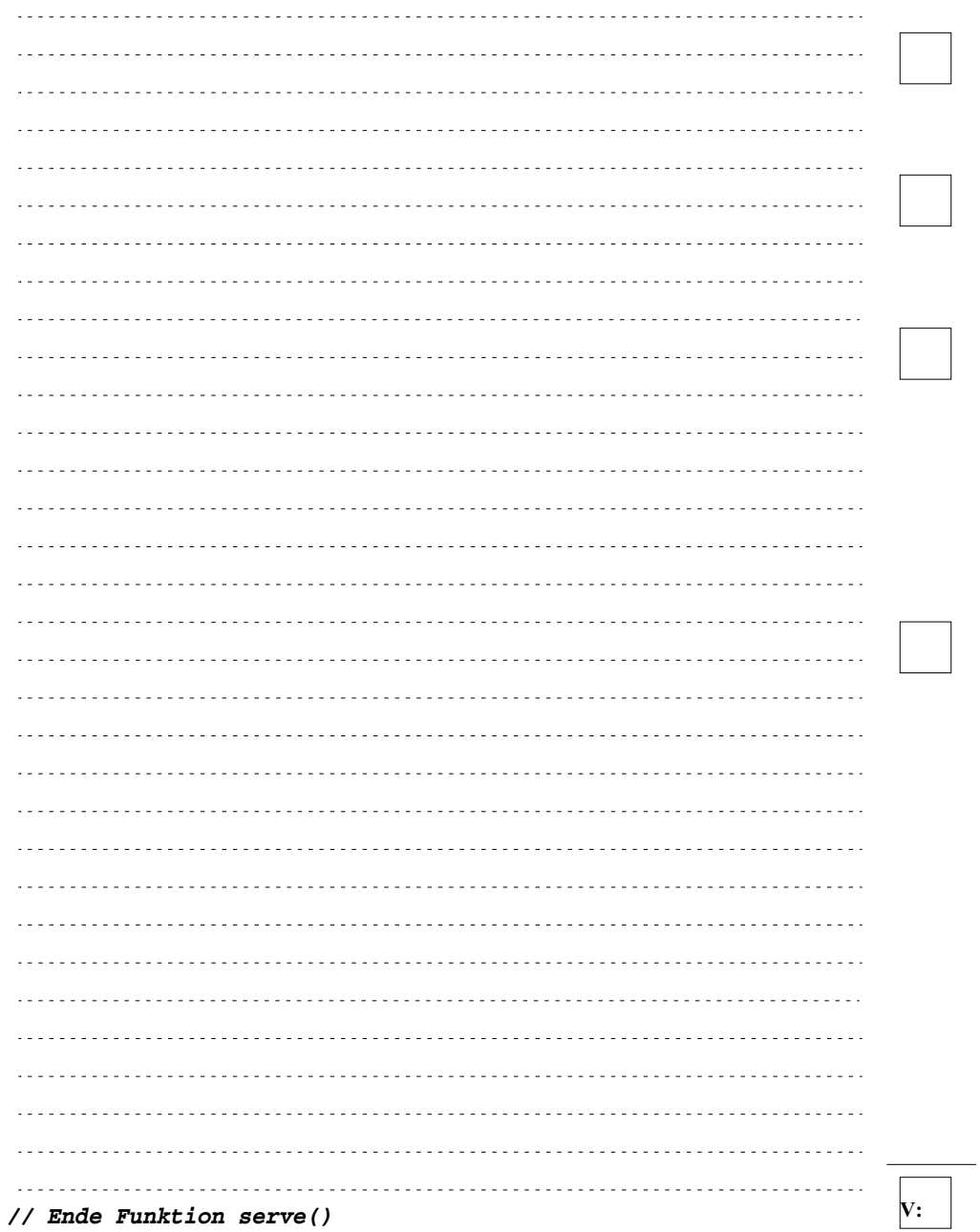


C:

```
// Ende Signalbehandlung

// Funktion serve()
```

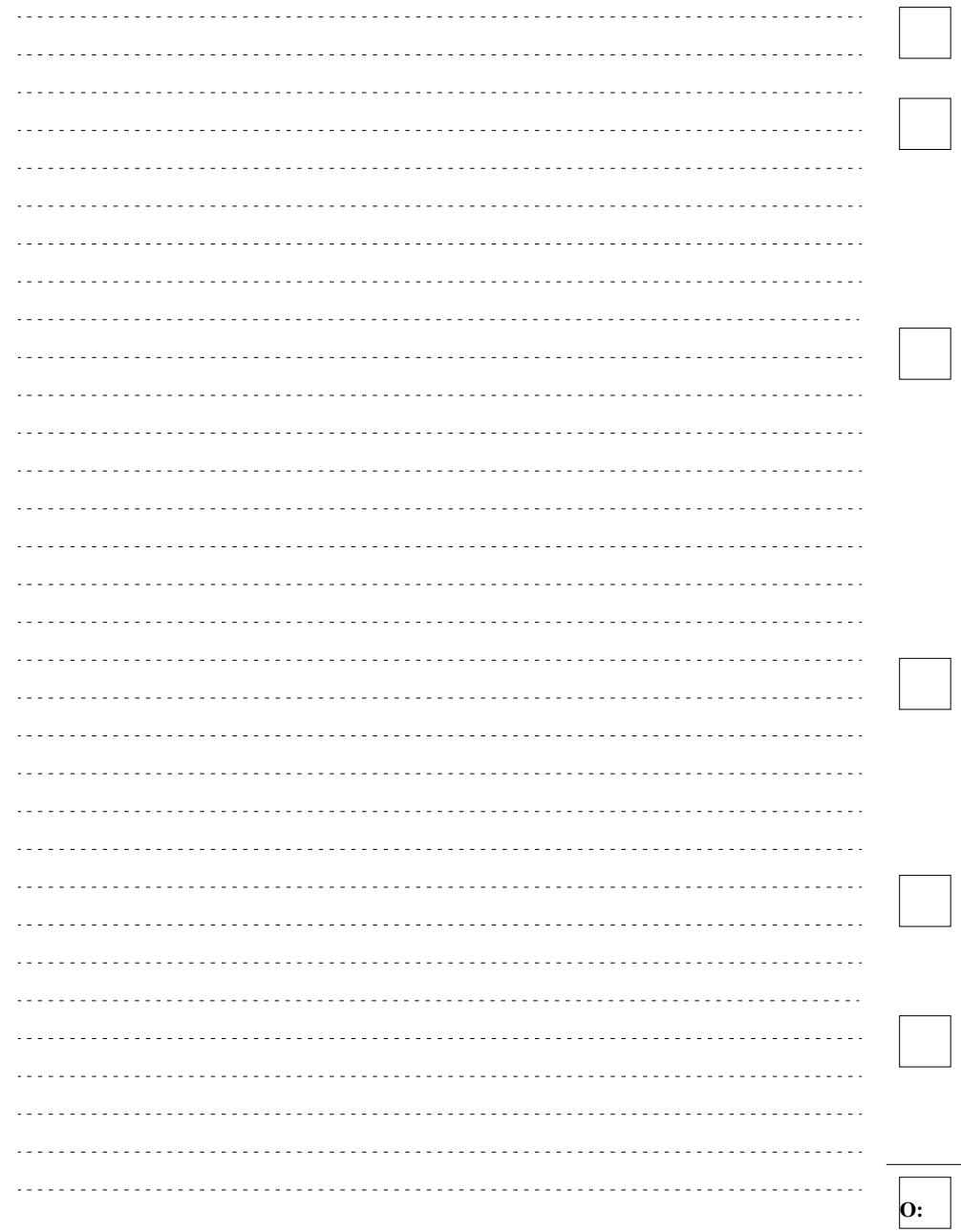




```
// Ende Funktion serve()
```

V:


```
// Funktion show()
```



O:

```
// Ende Funktion show()

// Funktion pin()

// Ende Funktion pin()
```

P:

Aufgabe 3: (21 Punkte)

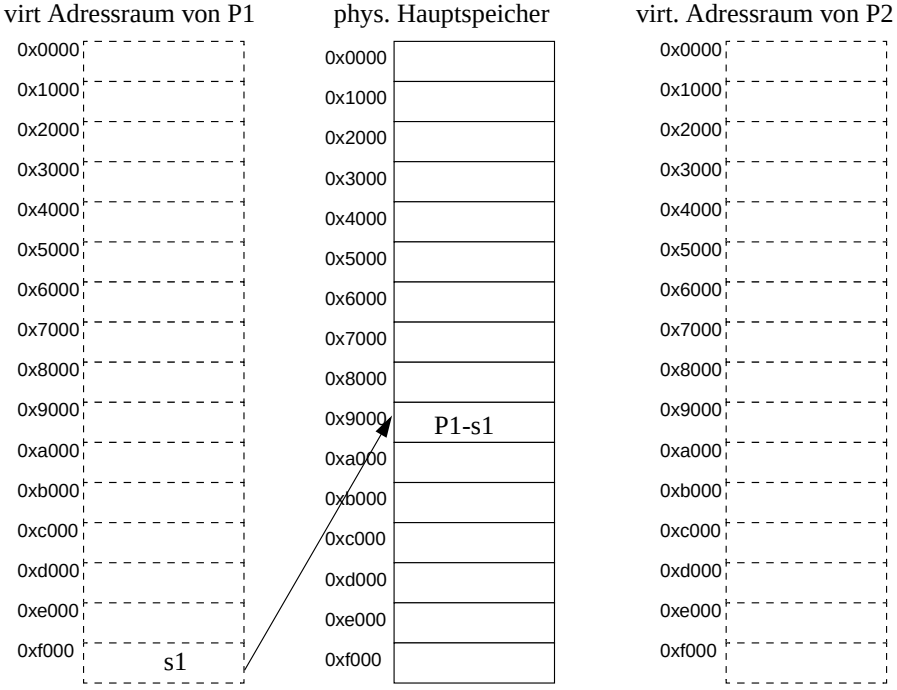
In einem System mit Seitenadressierung (paged address space), Adresslänge = 16 Bit, Seitengröße = 4 KiBi Bytes, Hauptspeichergröße = 64 KiBi Bytes wird ein Programm durch zwei Prozesse P1 und P2 ausgeführt. *(zur Erinnerung :-) : $4096_{10} = 1000_{16}$)*

Die erste Seite eines virtuellen Adressraums wird grundsätzlich nicht genutzt. Das Textsegment des Programms umfasst eine Seite (t1) und wird von den Prozessen gemeinsam genutzt, das Datensegment umfasst ebenfalls eine Seite (d1) direkt im Anschluss daran. Das Stacksegment umfasst eine Seite (s1) ganz am Ende des virtuellen Adressraums.

Den Seiten der Prozesse P1 und P2 sind folgende Seitenrahmen im Hauptspeicher zugewiesen:
P1-t1: 0x6000, P1-d1: 0xd000, P1-s1:0x9000, P2-d1: 0x5000, P2-s1: 0xb000.

Das (sehr kleine) Betriebssystem (BS) belegt die ersten 4 Seitenrahmen.

a) Tragen Sie die den Aufbau der virtuellen Adressräume (wo liegen welche Seiten), die bekannte Belegung der Seitenrahmen des physikalischen Hauptspeichers und die Abbildung (Pfeile) analog zu dem Beispiel von Seite s1 des Prozesses P1 in der nachfolgenden Skizze ein. (5 Punkte)



b) Skizzieren Sie die Datenstrukturen, die die MMU für die Umsetzung des virtuellen Adressraums von P1 benötigt, tragen Sie sinnvolle Zugriffsrechte für die einzelnen Seiten ein und veranschaulichen Sie in Ihrer Skizze, wie die logische Adresse 0x2140 umgesetzt wird. (10 Punkte)

c) Aus welchem Grund wird die erste Seite des virtuellen Adressraums nicht genutzt? (1 Punkt)

d) In Prozess P1 wird folgende Anweisung ausgeführt:

```
char *p = malloc(4000);
```

Erläutern Sie, was das Betriebssystem an dieser Stelle tun muss, und tragen Sie in den Skizzen der Teilaufgaben a) und b) ein, was sich ändert. Wenn erforderlich, wählen Sie selbst geeignete Adressen. Markieren Sie die Eintragungen jeweils mit einem ⓓ . (5 Punkte)

Aufgabe 4: (9 Punkte)

a) Man unterscheidet verschiedene Gewichtsklassen von Prozessen/Threads. Welche sind dies und wodurch entsteht das unterschiedliche "Gewicht"? (6 Punkte)

b) Welche Eigenschaften haben die die unterschiedlichen Prozess/Thread-Arten im Hinblick auf Multiprozessoren bzw. Mehrkernprozessoren? (3 Punkte)