Aufgabe 1: Ankreuzfragen (30 Punkte)

1) Einfachauswahlfragen (20 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) Wozu dient die CAS (Compare-And-Swap) Instruktion? 2 Punkte

- ☐ Zur Realisierung einer effizienten Verdrängungssteuerung bei einseitiger Synchronisation.
- ☐ Um in einem System mit Seitennummerierung (Paging) Speicherseiten in die Auslagerungspartition (swap area) schreiben zu können.
- ☐ Um auf einem Multiprozessorsystem einfache Modifikationen an Variablen ohne Sperren implementieren zu können.
- ☐ Um bei der Implementierung von Schlossvariablen (Locks) aktives Warten zu vermeiden.

b) Wie wird erkannt, dass eine Seite eines virtuellen Adressraums gerade ausgelagert ist? 2 Punkte

- ☐ Bei ausgelagerten Seiten ist im Seitendeskriptor das „present bit“ nicht gesetzt. Das Betriebssystem erkennt dies und löst bei einer Adressauflösung für solch eine Seite einen Trap aus.
- ☐ Im Seitendeskriptor steht bei ausgelagerten Seiten eine Adresse des Hintergrundspeichers und der Speichercontroller leitet den Zugriff auf den Hintergrundspeicher um.
- ☐ Die MMU erkennt bei der Adressumsetzung, dass die physikalische Adresse ungültig ist und löst einen Trap aus.
- ☐ Im Seitendeskriptor wird ein spezielles Bit geführt, das der MMU zeigt, ob eine Seite eingelagert ist oder nicht. Falls die Seite nicht eingelagert ist, löst die MMU einen Trap aus.

c) Welche der folgenden Aussagen über Einplanungsverfahren ist richtig? 2 Punkte

- ☐ Beim Einsatz präemptiver Einplanungsverfahren kann laufenden Prozessen die CPU nicht entzogen werden.
- ☐ Probabilistische Einplanungsverfahren müssen die exakten CPU-Stoßlängen aller im System vorhandenen Prozesse kennen.
- ☐ Bei kooperativer Einplanung kann es zur Monopolisierung der CPU kommen.
- ☐ Asymmetrische Einplanungsverfahren können ausschließlich auf asymmetrischen Multiprozessor-Systemen zum Einsatz kommen.

d) Welche der folgenden Aussagen zum Thema Seitenfehler (page fault) ist richtig? 2 Punkte

- ☐ Ein Seitenfehler zieht eine Ausnahmebehandlung nach sich. Diese wird dadurch ausgelöst, dass die MMU das Signal SIGSEGV an den aktuell laufenden Prozess schickt.
- ☐ Ein Seitenfehler wird ausgelöst, wenn der Versatz in einer logischen Adresse größer als die Länge der Seite ist.
- ☐ Seitenfehler können auch auftreten, obwohl die entsprechende Seite gerade im physikalischen Speicher vorhanden ist.
- ☐ Wenn der gleiche Seitenrahmen in zwei verschiedenen Seitendeskriptoren eingetragen wird, löst dies einen Seitenfehler aus (Gefahr von Zugriffskonflikten!).

e) Was versteht man unter RAID 0? 2 Punkte

- ☐ Ein auf Flash-Speicher basierendes, extrem schnelles Speicherverfahren.
- ☐ Auf Platte 0 wird Parity-Information der Datenblöcke der Platten 1 - 4 gespeichert.
- ☐ Datenblöcke eines Dateisystems werden über mehrere Platten verteilt gespeichert.
- ☐ Datenblöcke werden über mehrere Platten verteilt und repliziert gespeichert.

f) Welche der folgenden Aussagen zum Thema Adressräume ist richtig? 2 Punkte

- ☐ Virtuelle Adressräume sind Voraussetzung für die Realisierung logischer Adressräume.
- ☐ Der virtuelle Adressraum kann nie größer sein als der im Rechner vorhandene Hauptspeicher.
- ☐ Die maximale Größe des virtuellen Adressraums kann unabhängig von der verwendeten Hardware frei gewählt werden.
- ☐ Der physikalische Adressraum ist durch die gegebene Hardwarekonfiguration definiert.

g) Welche Aussage zu Zeigern in C-Programmen ist richtig? 2 Punkte

- ☐ Zeiger können verwendet werden, um in C eine call-by-reference-Übergabesemantik nachzubilden.
- ☐ Zeiger vom Typ `void *` existieren in C nicht, da solche Zeiger auf „Nichts“ keinen sinnvollen Einsatzzweck hätten.
- ☐ Ein Zeiger kann zur Manipulation von Daten in schreibgeschützten Speicherbereichen verwendet werden.
- ☐ Die Übergabesemantik für Zeiger als Funktionsparameter ist call-by-reference.

h) Welche Aussage zu Programmbibliotheken ist richtig?

2 Punkte

- ☐ Eine statische Bibliothek, die in ein Programm eingebunden wurde, muss zum Ladezeitpunkt dieses Programms im Dateisystem vorhanden sein.
- ☐ Beim Binden mit einer statischen Bibliothek werden in einem Programm nur Verweise auf verwendete Symbole der Bibliothek angelegt.
- ☐ Programm-Module, die von mehreren Anwendungen gemeinsam genutzt werden, können in Form einer dynamischen Bibliothek zentral installiert werden, um Speicherplatz zu sparen.
- ☐ Eine Änderung am Code einer statischen Bibliothek (z. B. Bugfixes) erfordert kein erneutes Binden der Programme, die diese Bibliothek benutzen.

i) Welche Aussage zu den Eigenschaften eines Journaling-Dateisystem ist richtig?

2 Punkte

- ☐ Alle Änderungen am Dateisystem werden in Form von Transaktionen in einer Log-Datei mitprotokolliert. Wird nach einem Systemabsturz festgestellt, dass eine Transaktion in der Log-Datei unvollständig ist, muss die betroffene Datei gelöscht werden.
- ☐ Je größer eine Platte ist, desto länger dauert der Reparaturvorgang eines Journaling-Dateisystem nach einem Systemabsturz.
- ☐ Es wird immer zuerst die Änderung im Dateisystem auf der Platte duchgeführt und anschließend zur Absicherung ein entsprechender Eintrag in die Log-Datei geschrieben.
- ☐ Die Einträge in der Protokolldatei müssen immer auch Informationen zu einem „Undo“ und „Redo“ der Transaktion enthalten.

j) Welche Aussage zum Thema „Aktives Warten“ ist richtig?

2 Punkte

- ☐ Bei verdrängenden Scheduling-Strategien verzögert aktives Warten nur den betroffenen Prozess, behindert aber nicht andere.
- ☐ Auf Mehrprozessorsystemen ist aktives Warten unproblematisch und deshalb dem passiven Warten immer vorzuziehen.
- ☐ Aktives Warten vergeudet gegenüber passivem Warten immer CPU-Zeit
- ☐ Zur Implementierung einer Schlossvariable mit aktivem Warten ist keine Unterstützung durch das Betriebssystem notwendig.

2) Mehrfachauswahlfragen (10 Punkte)

Bei den Mehrfachauswahlfragen in dieser Aufgabe sind jeweils m Aussagen angegeben, davon sind n ($0 \leq n \leq m$) Aussagen richtig. Kreuzen Sie alle richtigen Aussagen an.
Jede korrekte Antwort in einer Teilaufgabe gibt einen Punkt, jede falsche Antwort einen 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 persistenter Datenspeicherung sind richtig?

3 Punkte

- ○ Mit sequenziellem Zugriffsmuster erreicht man bei magnetischen Festplatten einen besseren Datendurchsatz als bei wahlfreiem Zugriff.
- ○ Journaling-Dateisysteme sind immun gegen defekte Plattenblöcke.
- ○ Journaling-Dateisysteme garantieren, dass auch nach einem Systemausfall alle Metadaten wieder in einen konsistenten Zustand gebracht werden können.
- ○ Im Vergleich zu den anderen Verfahren ist bei indizierter Speicherung die Positionierzeit des Festplatten-Armes beim Zugriff auf alle Datenblöcke einer Datei minimal.
- ○ Bei verketteter Speicherung dauert der wahlfreie Zugriff auf eine bestimmte Dateiposition immer gleich lang, wenn Cachingeffekte außer Acht gelassen werden.
- ○ Bei indizierter Speicherung von Dateien müssen unter Umständen mehrere Blöcke geladen werden, bevor der Dateiinhalt gelesen werden kann.

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

3 Punkte

- ○ Bei Adressraumschutz durch Abgrenzung werden die Betriebssystemdaten vor Zugriffen aus dem Anwendungsprogramm geschützt.
- ○ Ein Speicherbereich, der mit Hilfe der Funktion `free(3)` freigegeben wurde, kann im logischen Adressraum des zugehörigen Prozesses verbleiben.
- ○ Ein Adressraumwechsel ist eine privilegierte Operation und kann daher nur durch das Betriebssystem vorgenommen werden.
- ○ Da das Laufzeitsystem auf die Betriebssystemschnittstelle zur Speicherverwaltung zurückgreift, ist die Granularität der von `malloc(3)` zurückgegebenen Speicherblöcke vom Betriebssystem vorgegeben.
- ○ Mit `malloc(3)` angeforderter Speicher, welcher vor Programmende nicht freigegeben wurde, kann vom Betriebssystem nicht mehr an andere Prozesse herausgegeben werden und ist damit bis zum Neustart des Systems verloren.
- ○ Bei einer seitenorientierten Adressraumorganisation muss die Größe jeder Seite in der Seitentabelle gespeichert werden.

c) Welche der folgenden Aussagen zu UNIX-Dateisystemen sind richtig?

4 Punkte

- ○ Auf eine Datei in einem Dateisystem verweisen immer mindestens zwei „hard links“.
- ○ Innerhalb eines UNIX-Dateisystembaumes können die Inhalte mehrerer Festplatten eingebunden sein.
- ○ Ein Pfadname, der nicht mit einem ’/’-Zeichen beginnt, wird relativ zum Home-Verzeichnis des Benutzers interpretiert.
- ○ Wird eine Datei gelöscht, so werden auch alle symbolische Verknüpfungen gelöscht, die auf diese Datei verweisen.
- ○ Zur Anzeige des Inhaltes einer Datei ist es notwendig, das Leserecht auf dem übergeordneten Verzeichnis zu besitzen.
- ○ Der selbe Inode kann im Dateisystem im selben Verzeichnis mehrfach über verschiedene Namen referenziert werden.
- ○ Beim lesenden Zugriff auf eine Datei über eine symbolische Verknüpfung kann ein Prozess den Fehler „Permission denied“ erhalten, obwohl er das Leserecht auf der symbolischen Verknüpfung besitzt.
- ○ Für das Löschen einer Datei is es nicht erforderlich Schreibrechte auf diese zu haben.

Aufgabe 2: `narr` (60 Punkte)

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

Schreiben Sie das Programm `narr` (**N**etzwerk **A**nonymisierungs **R**echne**r**), das zur Anonymisierung von HTTP-Anfragen genutzt werden kann. `narr` ist ein Proxy, der im Auftrag eines Klienten eine HTTP-Anfrage durchführt und die Antwort des Servers an den Klienten zurücksendet, um so die Identität des Klienten zu verschleiern. Des Weiteren werden immer **genau** 5 **Anfragen** mit Hilfe eines Thread-Pools parallel abgearbeitet. Angenommene Verbindungen werden über einen Ringpuffer an die Threads weitergeleitet. Die Bearbeitung einer Anfrage durch einen Thread lässt sich in 3 Schritte untergliedern: 1) Anfrage empfangen 2) Kommunikation mit dem Ziel 3) Antwort zurückgeben. Um es Beobachtern des Datenverkehrs zu erschweren, Anfragen und Ziele in Verbindung zu bringen, werden die Schritte 2 und 3 der parallelen Anfragen synchron zueinander ausgeführt. Die Synchronisation erfolgt mit Hilfe von speziellen Semaphoren. Die Semaphor-Funktionen `P()` und `V()` (die Sie im zweiten Teil dieser Aufgabe selbst implementieren dürfen) nehmen einen zusätzlichen Parameter entgegen, der es erlaubt, mehrere Plätze auf einmal mit einem Aufruf atomar zu belegen, bzw. freizugeben.

Implementieren Sie folgenden Funktionen:

`int main(void)` Initialisiert das Programm, setzt einen Socket für alle IPv6-Adressen auf Port 8080 auf und wartet auf eingehende Verbindungen. Angenommene Verbindungen werden via `bbPut` über den Puffer an den Thread-Pool weitergereicht. Nachdem 5 Anfragen weitergeleitet wurden, sorgt `main` für die weitere Synchronisation der Abarbeitung durch die Arbeiter-Threads. Hierzu wartet `main` auf die Beendigung eines Schritts durch die 5 Threads und signalisiert diesen anschließend die Weiterarbeit über den Semaphor des jeweils nächsten Schritts. Erst nachdem der dritte Schritt freigegeben wurde, beginnt `main` wieder mit der Annahme von neuen Verbindungen. Verbindungsabbrüche dürfen das Programm nicht beenden. Auftretende Fehler sollen in allen Fällen auf `stderr` protokolliert werden und nach Möglichkeit nicht zur Beendigung von `narr` führen. Nur wenn eine weitere ordnungsgemäße Funktionalität ausgeschlossen werden kann, soll sich `narr` beenden!

`void *worker(void *)` Dient als Einstiegsfunktion des Thread-Pools. Sie entnimmt mittels `bbGet` eine Anfrage aus dem Puffer und bearbeitet diese in drei, mit den anderen Arbeitern synchronisierten Schritten:

1. Einlesen aller Zeilen und Auswerten der ersten Zeile (`connection_setup` und `request_parse` sind vorgegeben und nicht zu implementieren!)
2. Anfrage senden (exklusive der ersten Zeile) und Antwort abwarten (`forward_request` ist vorgegeben)
3. Weiterleiten der Antwort an den Klienten

Zwischen jedem dieser Schritte sollen sich die Threads mit Hilfe von Semaphoren synchronisieren. Hierzu signalisieren sie `main` jeweils den Abschluss eines Schritts und warten darauf, dass der Beginn des folgenden Schritts freigegeben wird. Gehen Sie davon aus, dass auftretende Fehler bereits in den aufgerufenen Funktionen protokolliert werden. In jedem Fall muss an der Synchronisation aller Schritte teilgenommen werden.

`char **receive_lines(FILE *rx)` Liest von `rx` Zeile für Zeile ein und liefert einen `NULL`-terminierten Vektor mit Zeigern auf die gelesenen Zeilen. Im Fehlerfall soll `NULL` zurückgegeben werden und der Fehlergrund auf `stderr` ausgegeben. Sie dürfen davon ausgehen, dass ein Zeile maximal 8192 Zeichen lang ist.

Hinweise:

- Die Funktion `request_parse` speichert lediglich Zeiger in den Eingabepuffer
- Falls nicht anders angegeben, setzen vorgegebene Funktionen im Fehlerfall die errno. Erfolg wird über einen gültigen Zeiger oder den `int` Wert 0 mitgeteilt.
- Vorausdeklarationen von Funktionen dürfen Sie außer Acht lassen.

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

```c
#include <sys/socket.h>
#include <netdb.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <pthread.h>
#include <signal.h>
#include "jbuffer.h"
#include "sem.h"

// globale Variablen
static SEM *sig_step2, *sig_step3, *sig_main;
static BNDBUF *bb;

// Datenstrukturen
struct connection { FILE *rx, *tx; };
struct request { char *domain, *path; };

// Vorgegebene Funktionen:

// Terminiert das laufende Program mit einer errno-basierten Fehlerausgabe
static void die(const char *msg);

// Funktion zum Freigeben eines NULL-terminierten Arrays "lines"
static void clear_lines(char **lines);
// Übermittelt alle Zeilen des NULL-terminierten Arrays "lines" an "fp"
static int send_lines(FILE *fp, char **lines);
// Wandelt "fd" in die FILE Streams von "c" um. Schließt "fd" immer.
static int connection_setup(struct connection *c, int fd);
// Zerlegt "line" in die für "rq" relevanten Teile
static int request_parse(struct request *rq, char *line);
// Übermittelt die in "lines" stehenden Zeilen den in "rq" beschriebenen Server
static char **forward_request(const struct request *rq, char **lines);

// Erstellt einen Ringpuffer
BNDBUF *bbCreate(size_t size);
// Legt den Wert "value" in "bb"; blockiert falls dieser gerade voll ist
void bbPut(BNDBUF *bb, int value);
// Liest einen Wert aus "bb"; blockiert falls dieser gerade leer ist
int bbGet(BNDBUF *bb);

// Erstellt eine Semaphore
SEM *semCreate(int initVal);
// Blockiert bis "c" Plätze in der Semaphore allokiert werden konnten
void P(SEM *sem, unsigned c);
// Gibt "c" Plätze in der Semaphore frei
void V(SEM *sem, unsigned c);
```



```c
int main(void) {
  // Initialisieren des globalen Zustands


  // Erstellen des passiven Sockets
```

```
  // Erstellen des Thread-Pools

  // Annehmen der Verbindungen

} // Ende von main
```

M:


```
// Funktion für den Arbeiterfaden
```

W:


```
// Funktion zum Einlesen aller Zeilen eines FILE-Streams
```

Z:

Semaphore (10 Punkte)

In dieser Teilaufgabe sollen Sie die erweiterten Operationen `P()` und `V()` eines Semaphors implementieren, so wie diese von `narr` benötigt werden. Abweichend von einem normalen Semaphor, bei dem der Wert des Semaphors jeweils immer um 1 erniedrigt bzw. erhöht wird, nehmen `P()` und `V()` in dieser Variante neben dem Zeiger auf den Semaphor noch einen weiteren Parameter `cnt` entgegen. Dieser gibt an, um welchen Wert der Semaphor auf einmal erhöht bzw. erniedrigt wird. Hierbei ist wichtig, dass der Semaphor nur erniedrigt werden darf, wenn er dabei nicht kleiner `0` wird, sonst blockiert die Operation (übertragen auf eine Puffer-Synchronisation bedeutet dies, dass man entweder alle angeforderten Pufferplätze auf einmal bekommt oder gar keinen nimmt).

```
#include "sem.h"
#include <pthread.h>

struct SEM {
  unsigned value;
  pthread_mutex_t mutex;
  pthread_cond_t  cond;
};
typedef struct SEM SEM;

void P(SEM *sem, unsigned cnt) {
```

```
}

void V(SEM *sem, unsigned cnt) {
```

```
}
```

S:

Aufgabe 3: Freispeicherverwaltung (14 Punkte)

Zur Verwaltung von freiem Speicher (z.B. zur feingranularen Verwaltung in Funktionen wie `malloc(3)` und `free(3)`) gibt es verschiedene Strategien zur Herausgabe des Speichers.

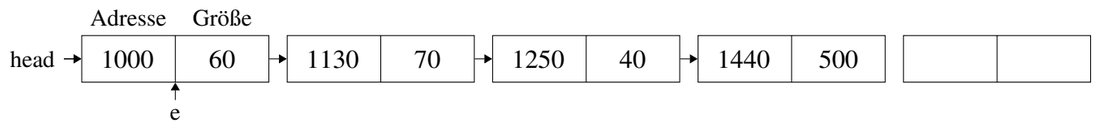
1) Vervollständigen Sie den Zustand der Verwaltungsdatenstrukturen für die untenstehende Folge von `malloc(3)`- und `free(3)`-Aufrufen. Im Rahmen dieser Aufgabe soll dafür das **next-fit**-Verfahren mit Verschmelzung von freien Blöcken angewandt werden. Zeichnen Sie pro Schritt den Einsprungpunkt **e** und den Zustand der Freispeicherliste in die untenstehende Abbildung ein.

Der Einsprungpunkt **e** wird zur Suche des nächsten Blocks genutzt und zeigt auf den zuletzt geteilten Block. Wird ein Block vollständig entnommen, so wird **e** auf dessen Nachfolger gesetzt.

Vermerken Sie zudem, welche Adresse der jeweilige `malloc(3)`-Aufruf zurückliefert. In dem Fall, dass ein Speicherblock aufgeteilt wird, soll der hintere Teil (entspricht dem Speicherbereich mit der höheren Adresse) an den Aufruf Aufrufer zurückgegeben werden.

⓪ kennzeichnet den initialen Zustand der Speicherverwaltung nach der Allokation p_0. (9 Punkte)

⓪ p_0 `= malloc(50); // = 1060` ← Rückgabewert des Aufrufs von malloc



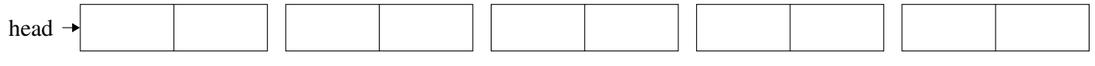
① p_1 `= malloc(70);`

② p_2 `= malloc(20);`

③ p_3 `= malloc(500);`

④ `free(`p_1`);`

⑤ `free(`p_0`);`



2) Nennen Sie je einen Vorteil und einen Nachteil von **next-fit** gegenüber **best-fit**. (1 Punkt)

3) Erläutern Sie den Unterschied zwischen interner und externer Fragmentierung. (2 Punkte)

4) Im Hinblick auf Adressraumkonzepte gibt es bei interner Fragmentierung einen Nebeneffekt in Bezug auf Programmfehler (vor allem im Zusammenhang mit Zeigern). Beschreiben Sie diesen Effekt. (2 Punkte)

Aufgabe 4: Prozesszustände & Scheduling (16 Punkte)

1) Beschreiben Sie die Prozesszustände **bei kurz- und mittelfristiger** Einplanung sowie die Ereignisse, die jeweils zu den Zustandsübergängen führen (Skizze mit kurzer Erläuterung der Zustände und Übergänge). (10 Punkte)

2) Man kann Scheduling-Verfahren nach verschiedenen Kriterien in Kategorien einteilen. Beschreiben Sie für die Kategorisierung kooperatives vs. präemptives Scheduling jeweils die Eigenschaften entsprechender Scheduling-Strategien, nennen Sie konkrete Beispiele für solche Strategien und geben Sie an, in welchen Anwendungssituationen entsprechende Strategien sinnvoll eingesetzt werden. (6 Punkte)