Aufgabe 1: Ankreuzfragen (7 Punkte)

1) Einfachauswahlfragen (4 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) Welcher Wert wird von folgendem C Programm zurückgegeben? 2 Punkte

```
#define SUB(a, b) a - b
#define DIV(a, b) a / b

int main() {
    return 5 * DIV ( SUB ( 4, 3), 2);
}
```

- ☐ 2
- ☐ 4
- ☐ 18
- ☐ 19

b) Namensräume dienen u. a. der Organisation von Dateisystemen. Welche Aussage ist richtig? 2 Punkte

- ☐ Flache Namensräume sind besonders einfach implementierbar und damit vor allem für Mehrbenutzersysteme gut geeignet.
- ☐ Der Nachteil von hierarchischen Namensräumen besteht darin, dass das Dateisystem spezielle Funktionen zum Auflösen von Namenskonflikten implementieren muss.
- ☐ Nur bei hierarchischen Namensräumen können symbolische Verweise auf Dateien erzeugt werden.
- ☐ In einem hierarchisch organisierten Namensraum dürfen gleiche Namen in unterschiedlichen Kontexten enthalten sein.

2) Mehrfachauswahlfragen (3 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) Bei der Ausführung von Programmen kann es zu Ausnahmesituationen kommen. Diese werden in die Kategorien *Trap* und *Interrupt* eingeteilt. Welche der folgenden Aussagen sind zutreffend? 3 Punkte

- ○ Ein *Trap* wird immer unmittelbar durch eine 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.
- ○ Unterbrechungssperren ermöglichen es die Ausführung des Unterbrechungshandhabers eines *Traps* zu verzögern.
- ○ Bei der mehrfachen Ausführung eines unveränderten Programms mit gleichen Eingabedaten treten immer die gleichen *Interrupts* auf.
- ○ Normale Rechenoperationen können zu einem *Trap* führen.
- ○ Die CPU sichert automatisch einen Teil des Prozessorzustands, bevor der Unterbrechungshandhaber aktiviert wird.

Aufgabe 2: `sp` (15 Punkte)

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

Schreiben Sie ein Programm `sp`, das mehrere Eingabedateien zu einer einzelnen Datei kombiniert. Dabei erwartet `sp` beliebig viele Dateipfade als Kommandozeilenargumente. Diese Dateien werden von `sp` nach einem unten beschriebenen Verfahren aneinander gereiht und auf dem Standardausgabekanal `stdout` ausgegeben.

Die Eingabedateien werden mit Metadaten versehen die zum Wiederherstellen benötigt werden. Diese beschreibenden Daten umfassen Größe und Name der Datei. Je Eingabedatei beginnt die Ausgabe mit einer dreistelligen Dezimalzahl, welche die Länge des Dateinamens beschreibt – der maximal unterstützte Dateiname ist 999 Zeichen lang. Darauf folgt der Dateiname selbst. Anschließend wird die Dateigröße als siebenstellige Dezimalzahl (maximal 9 999 999 Byte) angegeben. Schließlich folgt der Dateiinhalt. Sie dürfen davon ausgehen, dass keine überlangen Dateinamen vorkommen. Wenn eine Datei zu groß ist, beendet sich `sp` mit einer Fehlermeldung.

Ein Beispielaufruf an `sp` und die Beschreibung der resultierenden Ausgabe können aussehen wie folgt:

```
dust@faui48sp: ./sp loesung/stuffpacker.c
021loesung/stuffpacker.c0002766#include <stdio.h>...
```

Länge Dateiname | Dateiname | Dateigröße | Dateiinhalt

Wenn mehrere Dateien als Eingabe vorliegen werden diese nach dem oben beschriebenem Schema hintereinander auf den Ausgabekanal geschrieben.

```
dust@faui48sp: ./sp abc xyz > gepackt.sp
dust@faui48sp: cat gepackt.sp
003abc0000005hallo003xyz0000004welt
```

abc | xyz

Implementieren Sie für diese Funktionalität folgende Funktionen:

`void pack(FILE *dst, char *file)` Die Funktion erwartet einen Ausgabestrom (`dst`) und einen Dateipfad (`file`). Sie liest die Metadaten der Datei `file` und schreibt, nach dem oben beschriebenen Schema, auf `dst`. Wenn es sich bei `file` um einen symbolische Verknüpfung handelt, soll dieser gefolgt werden.

`int main(int argc, char *argv)` Ruft für alle Kommandozeilenargumente `pack` auf.

`void write_from_buf(FILE *dst, char *src, unsigned len)` Schreibt `len` Bytes von `src` nach `dst`.

`void copy_file(FILE *dst, FILE *src)` Schreibt den von `src` lesbaren Inhalt nach `dst`.

Ihnen wird die Funktion **`void uint_to_str(unsigned num, char buf[], unsigned len)`** zur Verfügung gestellt, welche die Zahl `num` in dezimaler Darstellung nach `buf` schreibt und dabei die führenden Nullen hinzufügt. Die Größe von `buf` wird der Funktion mittels `len` mitgeteilt. Es wird kein `'\0'` Byte an das Ende von `buf` geschrieben.

Hinweise:

- Im Fehlerfall soll sich `sp` mit einer aussagekräftigen Fehlermeldung beenden
- Gehen Sie davon aus, dass sich die Dateigröße zwischen Öffnen und Abfragen nicht ändert
- `sp` soll auch Binärdateien als Eingabe unterstützen

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
#include <unistd.h>

#define MAX_FILE_LEN 9999999
#define FILE_LEN_CHARS 7
#define MAX_FILE_NAME 999
#define FILE_NAME_CHARS 3

static void
uint_to_str(unsigned num, char buffer[], unsigned len)
{
  // Implementierung vorgegeben
}

static void die(const char *msg)
{
  perror(msg);
  exit(EXIT_FAILURE);
}

static void copy_file(FILE *, FILE *);
static void write_from_buf(FILE *, char *, unsigned);
static void pack(FILE *, char *);
```

```
void pack(FILE *dst, char *file) {
  // Lokale Variablen

  // Metadaten auslesen

  // Metadaten + Dateiinhalt schreiben

}
```

T:

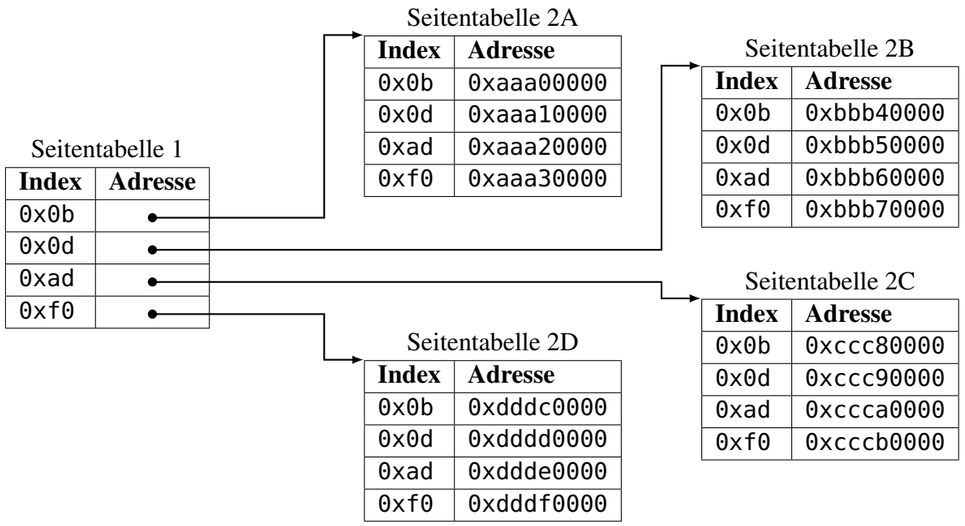

```
void write_from_buf(FILE *dst, char *src, unsigned len) {

}

void copy_file(FILE *dst, FILE *src) {
  // Dateiinhalt schreiben

}

// Hauptfunktion (main)
int main(int argc, char **argv) {
  // Aufruf von pack für alle Argumente

}
```

H:

Aufgabe 3: Offene Fragen (8 Punkte)

1) Virtueller Speicher (4 Punkte)

Gegeben sei die nachfolgend dargestellte Hierarchie zweistufiger Seitentabellen. Die Adresslänge des genutzten Systems sei 32 Bit, die Größe einer Seite 64 kByte. Für die Indizierung der zweistufigen Abbildung werden pro Stufe 8 Bit genutzt.



Auf welche **reale Adresse** wird die logische Adresse **0x0badf00d** abgebildet? Listen Sie stichpunktartig die zur Bestimmung notwendigen Zwischenschritte sowie die daraus resultierenden Zwischenwerte auf!

2) Korrektes Programmieren (4 Punkte)

Folgendes Programm enthält Programmierfehler. Beschreiben Sie diese und deren Auswirkungen.

```
int main(void) {
    int *p = malloc(10 * sizeof(int));
    for (int i = 0; i <= 10; ++i)
        p[i] = i;

    return 0;
}
```