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) Welche Aussage über den Rückgabewert von fork() ist richtig? 2 Punkte

- ☐ Der Kind-Prozess bekommt die Prozess-ID des Vater-Prozesses.
- ☐ Dem Vater-Prozess wird die Prozess-ID des Kind-Prozesses zurückgeliefert.
- ☐ Bei erfolgreicher Ausführung kehrt fork() nicht zum Vater-Prozess zurück.
- ☐ Der Rückgabewert ist in jedem Prozess (Kind und Vater) jeweils die eigene Prozess-ID.

b) Welche Aussage ist in einem Monoprozessor-Betriebssystem richtig? 2 Punkte

- ☐ In den Zustand blockiert gelangen Prozesse nur aus dem Zustand bereit.
- ☐ Ist zu einem Zeitpunkt kein Prozess im Zustand bereit, so ist auch kein Prozess im Zustand laufend.
- ☐ Ein Prozess im Zustand blockiert muss warten, bis der laufende Prozess den Prozessor abgibt und kann dann in den Zustand laufend überführt werden.
- ☐ Es befindet sich zu einem Zeitpunkt maximal ein Prozess im Zustand laufend.

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) Ausnahmesituationen bei einer Programmausführung werden in die beiden Kategorien Trap und Interrupt unterteilt. Welche der folgenden Aussagen sind zutreffend?

3 Punkte

- ○ Ein durch einen Interrupt unterbrochener Prozess darf je nach der Interruptursache entweder abgebrochen oder fortgesetzt werden.
- ○ Ein Systemaufruf im Anwendungsprogramm ist der Kategorie Interrupt zuzuordnen.
- ○ Die Ausführung einer Ganzzahl-Rechenoperation (z. B. Addition, Division) kann zu einem Trap führen.
- ○ Bei einem Trap wird der gerade in Bearbeitung befindliche Maschinenbefehl immer noch vollständig zu Ende bearbeitet, bevor mit der Trapbehandlung begonnen wird.
- ○ Nach der Behandlung eines Traps kann das unterbrochene Programm gegebenenfalls fortgesetzt werden.
- ○ Die CPU sichert bei einem Interrupt einen Teil des Prozessorzustands.

Aufgabe 2: `cc` (15 Punkte)

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

Schreiben Sie das Konsolenprogramm ***c**arbon **c**ounter* (`cc`), mit dessen Hilfe man einen Überblick über verursachte CO2-Emmissionen erhalten kann. Das Programm nimmt keine Parameter auf der Kommandozeile entgegen und liest im Anschluss eine Reihe von `add` oder `log`-Kommandos von der Standardeingabe. Mit `add <activity> <emissions>` kann der Nutzer eine neue Aktivität und die damit (pro Einheit) verursachten Emmissionen zur Liste bekannter Aktivitäten hinzufügen. Wurde eine Aktivität hinzugefügt, kann der Nutzer durch `log <activity> <quantity>` seinen Emmissionszähler um den entsprechenden, als `quantity * emmissions` berechneten, Wert erhöhen und ausgeben lassen. Ungültige Benutzereingaben werden ohne Warnung ignoriert, bei allen anderen Fehlern beendet sich das Programm mit einer Fehlermeldung. Wird das Ende der Standardeingabe erreicht, beendet sich das Programm im Erfolgsfall mit `EXIT_SUCCESS`.

Ein Beispiel für einen erfolgreichen Durchlauf könnte folgendes sein:

```
$ ./cc
add flight 270
add car 160
log flight 1000
Your carbon count: 270000
log car 100
Your carbon count: 286000
```

Implementieren Sie `cc` mit den folgenden Funktionen:

`int main(int argc, char **argv)`
: Iteriert über die Zeilen, die von `stdin` eingelesen werden. Gehen Sie davon aus, dass jede Zeile aus höchstens 256 Nutzzeichen, inklusive `\n`, besteht. Jede Zeile wird in durch Leerzeichen getrennte Token (Kommando, Aktivität, und Zahl) aufgeteilt und die Zahl wird mithilfe der vorgegebenen Funktion `parse_positive_int()` in `int` konvertiert. Ist das Kommando `add`, wird ein neuer Eintrag zum Ende des `map`-Arrays dynamisch hinzugefügt. Gehen Sie davon aus, dass `add` nie mehr als einmal mit dem gleichen Aktivitätsnamen aufgerufen wird. Ist das Kommando `log`, wird `calc_carbon()` aufgerufen und der um den Rückgabewert erhöhte `carbon_counter` wird auf der Standardausgabe ausgegeben (Teil der Grundfunktionalität des Programms).

`int calc_carbon(struct mapping *map, size_t mapsize, char *a, int q)`
: Iteriert über `map` und sucht den für die Aktivität `a` gespeicherten `emission`-Wert. Dieser wird dann mit Quantität `q` multipliziert und zurückgegeben. Wird kein Eintrag für die Aktivität gefunden, wird das Problem ohne Warnung ignoriert und 0 zurückgegeben.

```c
#include <ctype.h>
#include <dirent.h>
#include <errno.h>
#include <fnmatch.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <unistd.h>

// Gehen Sie davon aus, dass jede Zeile aus maximal 256 Nutzzeichen
// (inklusive \n) besteht.
#define MAX_LINE_LENGTH 256

static int parse_positive_int(char *str) {
  /* Vorgegeben. Gibt einen positiven int zurück, im Fehlerfall -1. */
}

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

struct mapping {
  char activity[MAX_LINE_LENGTH + 1];
  int emission;
};

static int calc_carbon(struct mapping *, size_t, char *, int);
```

```
int main(int argc, char **argv) {
  if (argc != 1) {
    fprintf(stderr, "USAGE: %s\n", argv[0]);
    return EXIT_FAILURE;
  }

  struct mapping *map = NULL;
  size_t mapsize = 0;
  int carbon = 0;


  // Über Zeilen iterieren:


    // Zeile in Token aufteilen:









    // Zahl in int konvertieren:


    // add-Kommando:
```

```
    // log-Kommando:
```

```
  // Beenden und Ressourcen freigeben:
```

```
  int fflush_err = fflush(stdout);
  if (fflush_err) {
    die("fflush");
  }

  return EXIT_SUCCESS;
}
```

M:

```
static int calc_carbon(struct mapping *map,
                       size_t mapsize, char *a, int q) {
```

```
}
```

C:

Aufgabe 3: Sockets und Addressräume (8 Punkte)

1) Es soll eine TCP-Verbindung zu einem Server hergestellt werden, dessen Netzwerkverbindung jedoch kurzfristig unterbrochen wurde. Dem Domain Name System (DNS) ist dies nicht bekannt und es ist kein Ersatz-Server eingetragen.

Welche der drei Systemfunktionen zum Verbindungsaufbau (`socket(2)`, `getaddrinfo(3)`, `connect(2)`) schlägt hierbei fehl? Durch welchen Fehlercode (`EACCESS` "permission denied", `EBADF` "not a valid open file descriptor", oder `ECONNREFUSED` "no one listening on the remote address") könnte das Problem angezeigt werden? (1 Punkt)

2) Beschreiben Sie die drei Begriffe *realer*, *logischer* und *virtueller* Adressraum. (7 Punkte)