Aufgabe 1: Ankreuzfragen (28 Punkte)

a) Einfachauswahlfragen (14 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.

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

- ○ Im Fehlerfall wird im Kind-Prozess -1 zurückgeliefert.
- ○ Dem Vater-Prozess wird die Prozess-ID des Kind-Prozesses zurückgeliefert.
- ○ Der Kind-Prozess bekommt die Prozess-ID des Vater-Prozesses.
- ○ Der Rückgabewert ist in jedem Prozess (Kind und Vater) jeweils die eigene Prozess-ID.

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

- ○ Durch dynamische Bibliotheken
- ○ Durch Interrupts
- ○ Durch langfristiges Scheduling
- ○ Durch Traps

III) Welche Aussage ist in einem Einprozessor-Betriebssystem richtig? 2 Punkte

- ○ 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“.
- ○ 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“.

IV) Welche der folgenden Aussagen zum Thema Prozesszustände ist korrekt? 2 Punkte

- ○ Der Planer (*Scheduler*) kann einen Prozess in den Zustand „blockiert“ überführen, indem er einen anderen Prozess einlastet.
- ○ In einem Vierkernsystem können sich maximal vier Prozesse gleichzeitig im Zustand „bereit“ befinden.
- ○ Blockierte Prozesse werden in den Zustand „bereit“ überführt, wenn die benötigten Betriebsmittel zur Verfügung stehen.
- ○ In einem Vierkernsystem gibt es stets genau vier laufende Prozesse.

V) Welche Aussage zu Betriebsmitteln ist richtig? 2 Punkte

- ○ Hardwarebetriebsmittel können Prozessen immer nur exklusiv zugeteilt werden.
- ○ Speicher verliert bei Stromausfall seinen Inhalt. Er gilt deshalb als transientes Betriebsmittel.
- ○ Konsumierbare Betriebsmittel werden zur Laufzeit generiert und verbraucht.
- ○ Der Zugriff auf Betriebsmittel erfordert immer Synchronisation.

VI) Welche der folgenden Aussagen zum Thema Fäden (*Threads*) ist richtig? 2 Punkte

- ○ Bei User-Threads (*Fasern*) ist die Scheduling-Strategie nicht durch das Betriebssystem vorgegeben.
- ○ Kernel-Threads können Multiprozessoren nicht ausnutzen.
- ○ Die Umschaltung zwischen User-Threads ist eine privilegierte Operation und muss deshalb im Systemkern erfolgen.
- ○ Für jeden Kernel-Thread verwaltet das Betriebssystem einen eigenen, geschützten Adressraum.

VII) Welche der folgenden Aussagen zum Thema Adressraumschutz ist richtig? 2 Punkte

- ○ Der Adressraumbelegungsplan (*memory map*) beschreibt für ein Programm, auf welche Speicheradressen es zugreift.
- ○ Ein Nachteil der wortorientierten Segmentierung sind Speicherverluste durch internen Verschnitt.
- ○ Mit dem Einsatz von Segmentierung ist es möglich, dieselbe logische Adresse in unterschiedlichen logischen Adressräumen auf verschiedene physikalische Adressen im realen Adressraum abzubilden.
- ○ In einem segmentierten Adressraum kann zur Laufzeit kein weiterer Speicher mehr dynamisch nachgefordert werden.

b) Mehrfachauswahlfragen (14 Punkte)

Bei den Richtig oder Falsch Fragen in dieser Aufgabe sind jeweils m Aussagen angegeben, davon sind n ($0 \leq n \leq m$) Aussagen richtig. Kreuzen Sie jeweils an, ob die entsprechende Aussage richtig oder falsch ist.
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.

I) Bewerten Sie die folgenden Aussagen zum Thema Betriebssysteme: 3 Punkte

Richtig	Falsch	
☐	☐	Das Betriebssystem abstrahiert vollständig vom Befehlssatz des Prozessors.
☐	☐	Multiplexing und Isolation der Hardwareresourcen sind Kernaufgaben eines Betriebssystems.
☐	☐	Betriebssystemdienste laufen zwingend im priviligierten Modus des Prozessors.
☐	☐	Mikrokernbetriebssysteme bieten nur einen stark reduzierten Funktionsumfang.
☐	☐	Die Unterstützung des Mehrprogrammbetriebs durch das Betriebssystem erfordert mehr als einen Prozessor.
☐	☐	Ein UNIX-Prozess ist ein virtueller Computer.

II) Bewerten Sie die folgenden Aussagen zu UNIX/Linux-Dateideskriptoren: 3 Punkte

Richtig	Falsch	
☐	☐	Ein Dateideskriptor ist eine Integerzahl, die über gemeinsamen Speicher an einen anderen Prozess übergeben werden kann und von letzterem zum Zugriff auf eine geöffnete Datei verwendet werden kann.
☐	☐	Ein Dateideskriptor ist eine prozesslokale Integerzahl, die der Prozess zum Zugriff auf eine geöffnete Datei benutzen kann.
☐	☐	Bei mehrfachem Öffnen ein- und derselben Datei erhält der Prozess jeweils die gleiche Integerzahl als Dateideskriptor zurück.
☐	☐	Beim Aufruf von `fork()` werden zuvor geöffnete Dateideskriptoren in den Kindprozess vererbt.
☐	☐	Der Aufruf `newfd = dup(fd)` erzeugt eine Kopie der dem Dateideskriptor `fd` zugrundeliegenden Datei; `newfd` enthält einen Dateideskriptor auf die neu erzeugte Datei.
☐	☐	Nur Dateideskriptoren mit dem Flag `FD_CLOEXEC` werden beim Prozessende mit `exit()` vom Betriebssystem geschlossen.

III) Bewerten Sie die folgenden Aussagen zum Thema Signale: 2 Punkte

Richtig	Falsch	
☐	☐	Signale Koordinieren den Zugriff auf exklusive Betriebsmittel.
☐	☐	Durch Signale können Nebenläufigkeitsprobleme in grundsätzlich nichtparallelen Programmen entstehen.
☐	☐	Signale sind immer asynchrone Ereignisse.
☐	☐	Signale sind das UNIX-Pendant zu Traps/Interrupts auf Hardwareebene.

IV) Man unterscheidet Traps und Interrupts (*Unterbrechungen*). Bewerten Sie die folgenden Aussagen: 3 Punkte

Richtig	Falsch	
☐	☐	Der Zeitgeber (Systemuhr) unterbricht die Programmbearbeitung in regelmäßigen Abständen. Die genaue Stelle der Unterbrechungen ist damit vorhersagbar. Somit sind solche Unterbrechungen in die Kategorie Trap einzuordnen.
☐	☐	Normale Rechenoperationen können zu einem Trap führen.
☐	☐	Ein Trap wird immer unmittelbar durch eine Aktivität des aktuell laufenden Prozesses ausgelöst.
☐	☐	Systemaufrufe sind jederzeit möglich und deshalb in die Kategorie Interrupt einzuordnen.
☐	☐	Ein Trap führt zwingend zum Abbruch des Prozesses, da er einen schwerwiegenden Fehler darstellt.
☐	☐	Ein Zugriff auf eine Speicheradresse kann zu einem Trap führen.

V) Betrachten Sie folgendes Programmfragment und bewerten Sie die Aussagen: 3 Punkte

```
static int a = 0x20190308;
int* foo(int x) {
  char b[] = "Hello ";
  static int c;
  int *e = malloc(800*sizeof(int));
  strcat(b, "Paul");
      return e;
}
```

Richtig	Falsch	
☐	☐	`a` wird von Compiler und Linker in der `.bss`-Sektion abgelegt.
☐	☐	`c` ist uninitialisiert und enthält einen zufälligen Wert.
☐	☐	`b` verliert beim Rücksprung aus `foo` seine Gültigkeit.
☐	☐	`e` liegt auf dem Stack.
☐	☐	`strcat` erzeugt einen Pufferüberlauf und damit undefiniertes Verhalten.
☐	☐	Die Rückgabe von `e` ist ein Fehler, da der Speicher außerhalb der Funktion nicht mehr zugreifbar ist.

Aufgabe 2: Programmieraufgabe – elsh (31 Punkte)

Schreiben Sie ein Programm `elsh` (**e**rror-**l**ogging **sh**ell), welches als Kommandozeileninterpreter (Shell) fungiert und folgende Funktionalität implementiert:

- Die Shell nimmt Befehle von der Standardeingabe entgegen und führt diese in einem Kindprozess aus. Sie gibt beim Starten eines Kindes dessen Befehlszeile und die Prozess ID auf der Standardfehlerausgabe (`stderr`) aus.
- Während der Ausführung des Kindes nimmt die Shell keine weiteren Eingaben entgegen.
- Ausgaben des Kindprozesses auf die Standardfehlerausgabe werden neben der normalen Weiterleitung der Shell zusätzlich der Datei `error.log` im aktuellen Verzeichniss angehängt.

Die **bereitgestellte** Funktion `int getNextCommand(cmd_t * cmd)` übernimmt dabei für Sie das Einlesen und Parsen der Befehlszeilen. Der Rückgabewert signalisiert, ob das Ende der Eingabe (0) erreicht wurde oder eine Zeile erfolgreich gelesen wurde (1). Die übergebene `cmd`-Struktur enthält nach dem Aufruf die gelesene Zeile in `line` und ihre Aufteilung in einzelne Bestandteile im Array `argv`. Der Eintrag `invalid` signalisiert, ob die gelesene Zeile valide für die Ausführung ist oder übersprungen werden soll.

Die weiteren Funktionen sind von Ihnen zu implementieren. Beachten Sie die beigefügten Man-Pages. Diese geben Hinweise auf die Verwendung von Bibliotheksfunktionen und deren Verhalten. Achten Sie weiterhin darauf, dass Sie auf eventuelle Fehlersituationen der Bibliotheksfunktionen reagieren. Im Fehlerfall beenden sie das Programm mit einer Fehlermeldung über die bereitgestellten Funktion `die(char* msg)`:

`child()` enthält das Hauptprogramm des Kindes.

`prepare_log()` öffnet die Logdatei `error.log` zum Schreiben (Anhängen) im aktuellen Verzeichnis und gibt den Dateideskriptor der geöffnete Datei zurück. Falls die Datei nicht existiert, soll sie mit den Schreib- und Leserechten für den Eigentümer erstellt werden.

`log_stderr()` liest die Fehlerausgabe des Kindes und schreibt diese sowohl in die Datei, wie auch in die Fehlerausgabe der Shell. Beendet das Kind die Fehlerausgabe, so wartet die Shell auf das Terminieren des Kindes.

`bury()` wartet auf das Ende eines Kindprozesses. Tritt hier ein Fehler auf, so wird die Shell nicht beendet sondern nur eine Fehlermeldung ausgegeben. Für jeden „begrabenen“ Kindprozess wird eine Zeile mit der Prozess ID und dem Exitstatus auf der Standardfehlerausgabe ausgegeben. Sie können davon ausgehen, dass alle Kinder sich normal beenden (nicht durch Signale o.ä.).

Zur Veranschaulichung eine Beispielsitzung mit der `elsh` (Eingabezeilen sind **fett** markiert; Ihre Ausgabeformatierung darf abweichen):

```
$ ./elsh
echo 1
Started [echo 1] pid=24159
1
Exited pid=24159 exitstatus=0
cat GBS_solution
Started [cat GBS_solution] pid=24160
cat: GBS_solution: No such file or directory
Exited pid=24160 exitstatus=1
```



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

#define READ 0
#define WRITE 1
#define ERROR 2

typedef struct cmd {
  int invalid;                  // 0=valid, 1=skip this line
  char * line;                  // the full line as read from stdin
  size_t line_length;           // size of the mem behind line
  char ** argv;                 // argument array
} cmd_t;

// PROVIDED FUNCTIONS

// read a line, store in cmd
int getNextCommand(cmd_t *cmd);

// print error message, terminate
void die(char * msg) {
  perror(msg);
  exit(EXIT_FAILURE);
}

// TO IMPLEMENT

// wait for child to terminate
void bury(void);

// open log file
int prepare_log(void);

// log child stderr output to file and own stderr
void log_stderr(int pipe_ends[], int outfile);

// child implementation
void child(int pipe_ends[], cmd_t *cur_cmd);
```

```
int main(int argc, char* argv[]) {
  cmd_t cur_cmd;

  while (getNextCommand(&cur_cmd)) {

  } // end while
```

M:


```
void child(int pipe_ends[], cmd_t *cur_cmd) {
```

```
void bury(void) { // Wait for child termination
  pid_t child;
  int status;
```

CB:

```
int prepare_log(void){
```

```
void log_stderr(int pipe_ends[], int outfile) {
```

L:

Aufgabe 3: Programmieraufgabe – submit (11 Punkte)

Vervollständigen Sie das Programm `submit`, welches eine Abgabe von Programmieraufgaben übernimmt. Der Aufgabenname wird hierbei als Parameter an das Programm übergeben:

```
./submit aufgabeX
```

Das Programm soll im Detail wie folgt funktionieren:

- Das Programm submit prüft zu Beginn, ob ein Aufgabenname als Parameter übergeben wurde. Sollte dies nicht der Fall sein, gibt es eine entsprechende Fehlermeldung aus und beendet sich.
- Wurde das Programm korrekt aufgerufen, durchsucht es den aktuellen Ordner nach Dateien mit der Endung „`.c`“. Jede gefundene C-Datei wird anschließend mit Hilfe der bereits existierenden Funktion `void submit(const char *exercise, const char *filename);` an den Abgabeserver übertragen.
- Wurde mindestens eine C-Datei an den Abgabeserver übertragen, gibt das Programm die Meldung `Abgabe von <ANZAHL> Dateien erfolgreich aus` und beendet sich mit dem Exit-Status `0`.
- Wurde keine C-Datei übertragen, gibt das Programm die Fehlermeldung `Keine Datei gefunden!` aus und beendet sich mit dem Exit-Status `1`.

Hinweise:

- Achten Sie auf eine korrekte Fehlerbehandlung der verwendeten Funktionen.
- Die Ausgabe von Fehlern soll auf dem stderr-Kanal erfolgen.
- Es kann davon ausgegangen werden, dass jeder Eintrag der auf „.c“ endet auch eine reguläre Datei ist.
- Ergänzen Sie das folgende Codegerüst so, dass ein vollständig übersetzbares Programm entsteht.

```
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <dirent.h>
#include <unistd.h>
#include <string.h>
#include <fnmatch.h>

void submit(const char * exercise, const char *filename);
!//newpage
```

```
int main(int argc, char * argv[]) {
  if ( [          ] ) {
    fprintf(stderr, "Usage: %s aufgabeX\n", [          ]);
    [          ];
  }
  char *aufgabe = [          ];
  DIR* d = [          ];
  if ( [          ] ) {
    perror("opendir");
    exit(EXIT_FAILURE);
  }
  // Search folder for *.c files and submit them
  [          ];
  [          ];
  while (errno = 0, [          ]) {
    if ( [          ] ) {
      [          ];
    }
  }
  // error handling, print summary
  if ( [          ] ) {
    [          ];
    exit(EXIT_FAILURE);
  }
  [          ];
  if ( [          ] ) {
    printf("Abgabe von %d Dateien erfolgreich\n", [          ]);
    exit(EXIT_SUCCESS);
  } else {
    fprintf( [          ] , "Keine Dateien gefunden!\n");
    exit(EXIT_FAILURE);
  }
}
```


Aufgabe 4: Textaufgabe (8 Punkte)

Die folgenden Beschreibungen sollen kurz und prägnant erfolgen (Stichworte, kurze Sätze).

a) Wie lauten die Voraussetzungen für einen Deadlock und kann es bei dem Folgenden Programmbeispielauf einem POSIX System dazu kommen? Wenn ja, warum? Wie könnte man das Problem hier lösen?

6 Punkte

```
//thread 1
do {
  sem_wait(&sem1);
  sem_wait(&sem2);
  counter++;
  sem_post(&sem2);
  sem_post(&sem1);
} while (1);
```

```
//thread 2
do {
  sem_wait(&sem2);
  sem_wait(&sem1);
  counter--;
  sem_post(&sem1);
  sem_post(&sem2);
} while(1);
```

b) Worin besteht der Unterschied zwischen einem Symlink und einem Hardlink auf Ebene des Dateisystems?

2 Punkte

Aufgabe 5: Dateisystem (12 Punkte)

Gegeben ist die folgende Ausgabe des Kommandos `ls -aoRi /tmp/gbs/` (rekursiv absteigende Ausgabe aller Dateien und Verzeichnisse unter `/tmp/gbs/` mit Angabe der Inode-Nummer, des Referenzzählers und der Dateigröße) auf einem Linux-System.

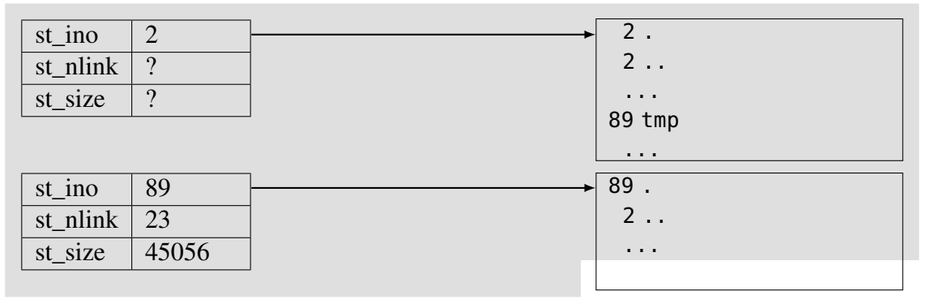
```
student@gbs:~$ ls -aoRi /tmp/gbs/
/tmp/gbs/:
total 60
13 drwxr-xr-x  4 student  4096 Mär  7 00:31 .
89 drwxrwxrwt 23 root    45056 Mär  7 00:24 ..
14 drwxr-xr-x  2 student  4096 Mär  7 00:32 folder
15 drwxr-xr-x  2 student  8317 Mär  7 00:25 folder2
16 lrwxrwxrwx  5 student    12 Mär  7 00:31 important -> folder/file3

/tmp/gbs/folder:
total 8
14 drwxr-xr-x 2 student 4096 Mär  7 00:32 .
13 drwxr-xr-x 4 student 4096 Mär  7 00:31 ..
17 -rw-r--r-- 2 student   42 Mär  7 00:25 file1
18 lrwxrwxrwx 1 student   18 Mär  7 00:32 file2 -> ../folder2/thefile

/tmp/gbs/folder2:
total 10
15 drwxr-xr-x 2 student 8317 Mär  7 00:25 .
13 drwxr-xr-x 4 student 4096 Mär  7 00:31 ..
17 -rw-r--r-- 2 student   42 Mär  7 00:25 fileX
19 -rw-r--r-- 1 student  765 Mär  7 00:25 thefile
```

Ergänzen Sie im weißen Bereich die auf der folgenden Seite im grauen Bereich bereits angefangene Skizze der Inodes und Datenblöcke des Linux-Dateisystems um alle entsprechenden Informationen, die aus obiger Ausgabe entnommen werden können.

Inodes



st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

Diese Seite darf herausgetrennt werden.