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?

- ○ 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?

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

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

- ○ 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?

- ○ 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?

- ○ 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?

- ○ 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?

- ○ 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, 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:

Richtig	Falsch	
☐	☐	Das Betriebssystem erweitert konzeptionell den Befehlssatz des Prozessors.
☐	☐	Multiplexing und Isolation der Hardwareresourcen sind Kernaufgaben eines Betriebssystems.
☐	☐	Durch Ausführen eines Programms als Administrator gelangt man in den 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:

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.
☐	☐	Vor dem Beenden eines Prozesses muss man alle geöffneten Dateien mit `close()` schließen, da sonst die dazugehörigen Resourcen veloren gehen.

III) Bewerten Sie die folgenden Aussagen zum Thema Signale:

Richtig	Falsch	
☐	☐	Alle mit `kill()` versendete Signale führen zum Beenden des Prozesses.
☐	☐	Durch Signale können Nebenläufigkeitsprobleme in grundsätzlich nicht-parallelen Programmen entstehen.
☐	☐	Es gibt synchrone und asynchrone Signale.
☐	☐	Signale sind das UNIX-Pendant zu Traps/Interrupts auf Hardwareebene.

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

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:

```
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 undefinierten 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 – josh (23 Punkte)

Schreiben Sie ein Programm `josh` (**jo**b-counting **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.
- Empfängt die Shell das Signal `SIGUSR1`, so gibt sie einen Statusbericht über die Anzahl der gestarteten und beendeten Jobs aus. Dafür müssen die Zähler `started_jobs` und `finished_jobs` vom Hauptprogramm geführt werden.

Die **bereitgestellte** Funktion `int getNextCommand()` 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`. Das Element `invalid` signalisiert, ob die gelesene Zeile valide für die Ausführung ist oder übersprungen werden soll. Geben Sie den von `getNextCommand()` allokierten Speicher mittels `freeCommand()` wieder frei, sobald dieser nicht mehr benötigt wird.

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 bereitgestellte Funktion `die(char *msg)`. Die weiteren Funktionen sind von Ihnen zu implementieren:

`child()` soll das Hauptprogramm des Kindes enthalten.

`register_status_handler()` soll den entsprechenden Handler registrieren, um auf das Signal `SIGUSR1` zu reagieren. `sigusr1_handler()` ist der (gegebene) Handler, der beim Eintreffen des Signals aktiviert werden soll.

`bury()` soll auf das Ende eines Kindprozesses warten. Tritt beim Warten 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 und die Zähler aktualisiert. Sie können davon ausgehen, dass alle Kinder sich normal beenden (nicht durch Signale).

Zur Veranschaulichung dient die hier abgebildete Beispielsitzung mit der `josh` (Eingabezeilen sind **fett** markiert; Ihre Ausgabeformatierung darf abweichen):

```
$ ./josh
echo 1
Started [echo 1] pid=24159
1
Exited pid=24159 exitstatus=0
sleep 15
Started [sleep 15] pid=24160
           << Signal SIGUSR1 wird von außen an josh gesendet >>
josh status: started=1 finished=1
Exited pid=24160 exitstatus=1
           << Signal SIGUSR1 wird von außen an josh gesendet >>
josh status: started=0 finished=2
```



```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>

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;

int started_jobs=0;             // statistics counter
int finished_jobs=0;

// PROVIDED FUNCTIONS

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

//free allocated memory
void freeCommand(cmd_t *cmd);

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

// the SIGUSR1 handler printing the status
void sigusr1_handler(int signum);

// TO IMPLEMENT

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

// register the SIGUSR1 handler
void register_status_handler(void);

// child implementation
void child(cmd_t *cur_cmd);
```

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

  while (getNextCommand(&cur_cmd)) {

  } // end while
```

M:


```
void child(cmd_t *cur_cmd) {

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

CB:

```
void sigusr1_handler(int signum);// der Handler; gegeben
```

```
void register_status_handler(void){
```

L:

d) Theoriefragen zum Arbeiten mit Signalen (4 Punkte)

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

I) Welche Klasse von Problemen muss man zusätzlich betrachten, wenn Signale Teil der Programmlogik werden?

II) Was ist bei der Verwendung von (Bibliotheks-) Funktionen in Signalhandlern zu beachten? Nennen Sie ein Beispiel.

Aufgabe 3: Programmieraufgabe – chocolate factory (15 Punkte)

Vervollständigen Sie das Programm `chocolate_factory`, welches eine Simulation eines berühmten Schokoladenfabrikanten übernimmt.

Die Fabrik stellt für die Produktion Arbeiter ein, die die Schokoladenkühe melken, bis diese keinen Ertrag mehr bringen. Um Protesten der Belegschaft wegen Massenbeschäftigung entgegenzuwirken dürfen maximal 12 Arbeiter zeitgleich angeheuert werden. Implementieren sie die einzelnen Arbeitsbereiche des Fabrikanten Willy, des Beobachters Charlie und der als Arbeiter eingestellten Oompa Loompas.

Der Fabrikant Willy übernimmt die folgenden Aufgaben:

- Er initialisiert alle notwendigen Variablen und Zustände der Fabrik.
- Er lädt Charlie ein, die Fortschritte zu beobachten.
- Solange es neue Kühe gibt, heuert er Oompa Loompas als Arbeiter an (max 12 gleichzeitig) und gibt jedem eine Kuh zum Melken.
- Er wartet darauf, dass die verbleibenden Arbeiter fertig werden.
- Er sagt Charlie Bescheid, dass seine Aufgabe beendet ist.
- Er wartet, dass Charlie seine Aufgabe beendet hat und beendet dann die Simulation.

Der Beobachter Charlie übernimmt folgende Aufgabe:

- Er wartet darauf, dass ein Oompa Loompa Fortschritt gemacht hat.
- Er liest den Zustand aus.
- Er ruft den Zustand und die Beschäftigungssituation aus (`printf`).
- Wenn Willy nicht das Ende verkündet, wiederholt Charlie sein Vorgehen.
- Er macht eine letzte Zustandsmeldung und sagt Willy, dass er auch fertig ist.

Die Oompa Loompas übernehmen folgende Aufgabe:

- Solange die zugewiesene Kuh Schokolade liefert, wird diese gemolken.
- Jeder Teilertrag wird verzeichnet und Charlie gemeldet.
- Liefert die Kuh keinen Ertrag mehr, so kündigt der Oompa Loompa.

Ergänzen sie den C-Code so, dass das beschriebene Verhalten erreicht wird. Hierfür müssen Sie die gegebenen Semaphoren so verwenden, dass die Arbeitsabläufe passend synchronisiert werden. Hinweise:

- Achten Sie auf eine korrekte Fehlerbehandlung der verwendeten Funktionen.
- Die Ausgabe von Fehlern soll auf dem stderr-Kanal erfolgen.
- Ergänzen Sie das folgende Codegerüst so, dass ein vollständig übersetzbares Programm entsteht.
- `hire_oompa_loompa()` und `invite_charlie()` starten jeweils neue Threads, die die Funktionen `oompa_loompa()` und `charlie()` aufrufen.

```
#define MAX_WORKERS 12

bool charlie_done = False;
sem_t available_jobs, finished_oompas, charlie_finished;
struct global_statistics{
  long long int chocolate;
  sem_t mutex;
  sem_t change;
} statistics;

void hire_oompa_loompa(cow * cow);
void willy(void);
void invite_charlie(void);

int main(int argc, char *argv[]) {
  //init the factory
  sem_init(&available_jobs,     0, [          ]);
  sem_init(&finished_oompas,    0, [          ]);
  sem_init(&charlie_finished,   0, [          ]);
  sem_init(&statistics.mutex,   0, [          ]);
  sem_init(&statistics.change,  0, [          ]);
  willy();
}

void oompa_loompa(cow *cow) {
  int chocolate_output;
  while ((chocolate_output = milk_cow(cow) != 0)) {
      [                                      ];
    statistics.chocolate += chocolate_output;
      [                                      ];
      [                                      ];
  }
  release_cow(cow);
    [                                        ];
    [                                        ];
}
```



```
void charlie(void) {
  while (!charlie_done) {
      [                                      ];
      [                                      ];
    long long int choc = statistics.chocolate;
      [                                      ];
    int worker_slots;
      [                                      ];
    printf("Chocolate = %lld, workers = %d\n", choc, MAX_WORKERS - worker_slots);
  }
  printf("Well done, we produced %lld choco units.\n", statistics.chocolate);
    [                                      ];
}

void willy(void) {
  cow* cur_cow;
  int job_counter = 0;
  invite_charlie();
  while ((cur_cow = buy_cow()) != NULL) {
      [                                      ];
      [                                      ];
      [                                      ];
  }
  // Wait for Oompa Loompas to finish
    [                                      ] {
      [                                      ];
  }
  //Notify Charlie to terminate
  charlie_done = True;
    [                                      ];

  //wait for Charlie to terminate
    [                                      ];
  printf("Everything done. Bye\n");
}
```

Aufgabe 4: Textaufgabe (12 Punkte)

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

a) Beschreiben Sie die Begriffe **Prozess**, **Programm** und **Datei** im Kontext von Betriebssystemen. Setzen Sie diese in Verbindung zueinander.

b) Im Mehrprogrammbetrieb ist es nicht möglich, dass alle Programme beim Laden an die Speicheradresse, die beim Linken angenommen wurde, platziert werden können.

Wie geht das Betriebssystem damit um?
Nennen Sie hierzu zwei mögliche Varianten der Implementierung.

c) Bei Schedulingverfahren wird in Online- und Offlinescheduling unterschieden. Nennen Sie den entscheidenden Unterschied und jeweils einen Vorteil.

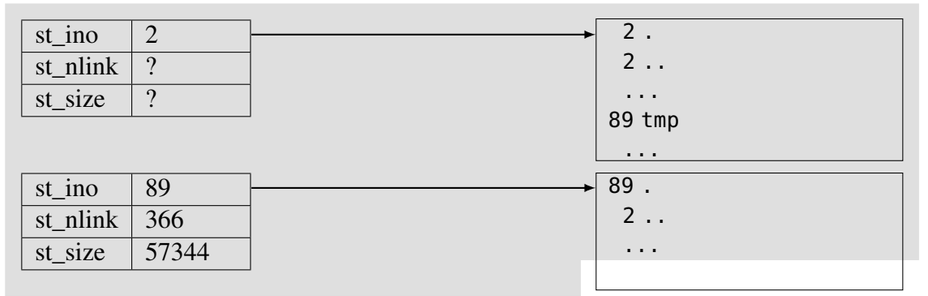
Aufgabe 5: Dateisystem (12 Punkte)

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 124
64 drwxr-xr-x   4 student 36864 Aug 23 17:13 .
89 drwxrwxrwt 366 root    57344 Aug 23 17:05 ..
66 drwxr-xr-x   3 student 20480 Aug 23 17:14 bar
68 -rw-r--r--   1 student     6 Aug 23 16:03 file
65 drwxr-xr-x   2 student  4096 Aug 23 16:02 foo
69 lrwxrwxrwx   2 student    11 Aug 23 16:04 other -> kekse/sonne
69 lrwxrwxrwx   2 student    11 Aug 23 16:04 stuff -> kekse/sonne

/tmp/gbs/bar:
total 60
66 drwxr-xr-x 3 student 20480 Aug 23 17:14 .
64 drwxr-xr-x 4 student 36864 Aug 23 17:13 ..
67 drwxr-xr-x 2 student  4096 Aug 23 17:18 stuff

/tmp/gbs/bar/stuff:
total 28
67 drwxr-xr-x 2 student  4096 Aug 23 17:18 .
66 drwxr-xr-x 3 student 20480 Aug 23 17:14 ..
70 -rw-r--r-- 1 student    89 Aug 23 17:18 solution

/tmp/gbs/foo:
total 40
65 drwxr-xr-x 2 student  4096 Aug 23 16:02 .
64 drwxr-xr-x 4 student 36864 Aug 23 17:13 ..
```

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	