Aufgabe 1: (16 Punkte)

Bei den Multiple-Choice-Fragen ist jeweils nur **eine** richtige Antwort eindeutig anzukreuzen. Auf die richtige Antwort gibt es die angegebene Punktzahl.

Wollen Sie eine Multiple-Choice-Antwort korrigieren, kreisen Sie bitte die falsche Antwort ein und kreuzen die richtige an.

Lesen Sie die Frage genau, bevor Sie antworten.

a) Welche Angaben enthält ein Eintrag eines Katalogs (Verzeichnis) in einem Standard-UNIX-Dateisystem? 2 Punkte

- ❏ Blocknummer des Inode-Plattenblocks und Dateiname
- ❏ Inode-Nummer und Dateiname
- ❏ Dateiname, Dateigröße, Eigentümer und Zugriffsrechte
- ❏ nur Inode-Nummer

b) Ein Hauptprogramm und eine Interruptbehandlung greifen nebenläufig auf die Variable `uint16_t foo` zu. Das Hauptprogramm verwendet foo in der Anweisung `uint16_t bar = foo;` der Interrupthandler verwendet foo im Vergleich `if(foo == 5)`. Welches Nebenläufigkeitsproblem kann auftreten? 3 Punkte

- ❏ Lost-Update
- ❏ Lost-Wakeup
- ❏ keines
- ❏ Das Hauptprogramm könnte einen inkonsistenten Wert lesen, da foo aus 2 Bytes besteht und nicht mit einer Instruktion geladen werden kann.

c) In Betriebssystemen wie Linux oder Windows unterscheidet man die Begriffe Programm und Prozess. Welche Aussage ist richtig? 2 Punkte

- ❏ Programme sind Anwendungen des Benutzers, während Prozesse Aktivitäten des Betriebssystems sind.
- ❏ Programme sind C-Quellcode-Dateien, die durch einen C-Compiler in einen lauffähigen Prozess übersetzt werden können.
- ❏ Ein Prozess hat einen eigenen virtuellen Adressraum. Daten des Prozesses sind vor direktem Zugriff durch andere Prozesse geschützt.
- ❏ Ein Programm ist ein Prozess in Ausführung.

d) Welche Aussage zum Thema virtueller Adressraum ist richtig? 3 Punkte

- ❏ Die Umrechnung von virtuellen zu physischen Adressen erfolgt beim Übersetzen durch den Compiler.
- ❏ Dieselbe virtuelle Adresse kann in verschiedenen Prozessen auf unterschiedliche physische Adressen abgebildet werden.
- ❏ Virtuelle Adressen entsprechen Variablennamen in einem C-Programm. In Zeigern werden dagegen physische Adressen gespeichert, mit denen man die Abbildung umgehen kann.
- ❏ Die Abbildung von virtuellen auf physische Adressen erfolgt während der Programmlaufzeit durch eine spezielle Softwarekomponente.

e) Welche Aussage zu Zeigern ist richtig? 2 Punkte

- ❏ Die Speicherstelle, auf die ein Zeiger verweist, kann niemals selbst einen Zeiger enthalten.
- ❏ Beim Rechnen mit Zeigern muss immer der Typ des Zeigers beachtet werden.
- ❏ Ein Zeiger kann zur Manipulation von schreibgeschützten Datenbereichen verwendet werden.
- ❏ Zeiger vom Typ `void*` benötigen weniger Speicher als andere Zeiger, da bei anderen Zeigertypen zusätzlich die Größe gespeichert werden muss.

f) Welche Aussage zum Thema Interrupts und Traps ist richtig? 2 Punkte

- ❏ Bei einem abgelaufenen Timer handelt es sich um einen Trap, wenn der Timer durch den Prozess selbst aktiviert wurde.
- ❏ Interrupts sind Hardwaresignale, die durch das Betriebssystem behandelt werden müssen, wohingegen Traps durch die Aktivität eines Prozesses ausgelöst werden und auch von einem solchen behandelt werden müssen.
- ❏ Beim Ausführen von Programmcode wird vom Prozessor eine Division durch 0 erkannt und ein Hardwaresignal generiert. In diesem Fall handelt es sich um einen Trap.
- ❏ Interrupts bezeichnen ein asynchrones Signalisierungskonzept auf Mikroprozessoren, Traps das Äquivalent in einer Betriebssystemumgebung.

g) Welche Aussage zum Thema Prozesse/Threads ist **falsch**? 2 Punkte

- ❑ Die Umschaltung von User-Level-Threads ist effizienter als die Umschaltung von Kernel-Level-Threads, da hierzu keine Betriebssystemaufrufe vonnöten sind.
- ❑ Das Umschalten zwischen Threads ist eine privilegierte Operation und kann daher nur vom Betriebssystem durchgeführt werden.
- ❑ Threads teilen sich einen Adressraum.
- ❑ Kernel-Level-Threads können auf einem Multiprozessorsystem echt parallel ausgeführt werden.

Aufgabe 2a: Tankkontrolle (30 Punkte)

Schreiben Sie ein AVR-Mikrokontroller-Programm zur Kontrolle eines Wassertanks mit einem Fassungsvermögen von 60000 Litern. Der Zufluss wird über ein Ventil geregelt, welches sich im entsperrten Zustand bei eintreffendem Wasser automatisch öffnet und bei versiegendem Wasser wieder schließt. Öffnen (fallende Flanke) und Schließen (steigende Flanke) des Ventils werden dem Mikrokontroller hierbei über eine Interruptleitung signalisiert. Beim Erreichen eines vollen Tanks sperrt die Steuerung das Zuflussventil. Die Wasserentnahme erfolgt durch eine externe Steuerung, welche an einer Interruptleitung jeweils nach jedem entnommenen Liter eine fallende Flanke generiert.

Das Programm soll im Einzelnen wie folgt funktionieren:

- Zu Beginn ist der Tank leer und es fließt kein Wasser (Ventil geschlossen).
- Die main-Funktion ruft zunächst die Funktion `void init();` auf, welche die Initialisierung der I/O-Ports und Interruptquellen durchführt. Hierbei dürfen keine Annahmen über den initialen Zustand der Register gemacht werden.
- Die Steuerung soll den Prozessor in den Standardstromsparmodus versetzen, wenn gerade nichts zu tun ist.
- Während eines Zuflusses überwacht die Steuerung die in den Tank strömende Wassermenge. Hierzu können Sie eine Schleife verwenden und vereinfachend annehmen, dass zu jedem Schleifendurchlauf ein Liter Wasser in den Tank strömt (unabhängig vom Inhalt der Schleife). Beim Erreichen der maximalen Tankfüllung sperrt die Steuerung das Ventil (Ausgangspegel: 0=entsperrt, 1=gesperrt).
- Eine Wasserentnahme ist jederzeit möglich und entsprechende Signale sollen so zuverlässig wie möglich verarbeitet werden. Sobald wieder Platz im Tank frei ist wird das Ventil wieder entsperrt.

Information über die Hardware

Ventilsignal: **PORTD**, **Pin 2**
- externe Interruptquelle **0**, ISR-Vektor-Makro: **INT0_vect**
- Aktivierung der Interruptquelle erfolgt durch Setzen des **INT0**-Bits im Register **GICR**

Ventilsperre: **PORTD**, **Pin 0**
- Pin als Ausgang konfigurieren: entsprechendes Bit in **DDRD**-Reg. auf 1

Entnahmesignal: **PORTD**, **Pin 3**
- externe Interruptquelle **1**, ISR-Vektor-Makro: **INT1_vect**
- Aktivierung der Interruptquelle erfolgt durch Setzen des **INT1**-Bits im Register **GICR**

Externe Interruptquelle **0** bzw. **1** konfigurieren:
- Pin als Eingang konfigurieren: entsprechendes Bit in **DDRD**-Reg. auf 0
- angeschlossenes Gerät verbindet den Pin mit Masse, es muss der interne Pullup-Widerstand verwendet werden (entspr. Bit in **PORTD**-Reg. auf 1 setzen).
- Konfiguration der externen Interruptquellen (Bits in Register **MCUCR**):

externe Int.quelle 0		**externe Int.quelle 1**		**Beschreibung**
ISC01	**ISC00**	**ISC11**	**ISC10**	
0	0	0	0	Interrupt bei low Pegel
0	1	0	1	Interrupt bei beliebiger Flanke
1	0	1	0	Interrupt bei fallender Flanke
1	1	1	1	Interrupt bei steigender Flanke

Ergänzen Sie das folgende Codegerüst so, dass ein vollständig übersetzbares Programm entsteht.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#define FILL_MAX 60000

/* Funktiondeklarationen, globale Variablen, etc. */
```

.....

.....

.....

.....

.....

.....

.....

```
/* Unterbrechungsbehandlungsfunktionen */
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

I:


```
/* Funktion main */
```

.....

.....

```
  /* Initialisierung */
```

.....

.....

```
  /* Warten auf Ereignisse */
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
  /* Zufluss überwachen */
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
/* Ende der Funktion main */
```

M:

```
/* Funktion init */
```

Aufgabe 2b: (15 Punkte)

Schreiben Sie ein Backup-Programm, das alle regulären Dateien im aktuellen Verzeichnis durch Aufruf eines externen Programms archiviert.

Sämtliche Fehlermeldungen sollen auf den Standardfehlerkanal stderr ausgegeben werden, nicht auf die Standardausgabe.

Das Programm soll wie folgt funktionieren:

- Es wird das aktuelle Verzeichnis durchsucht. Verzeichniseinträge, deren Dateiname mit einem Punkt ’.’ beginnt oder bei denen es sich nicht um eine reguläre Datei handelt, werden ignoriert.
- Für jede reguläre Datei wird ein Kindprozess erzeugt und in diesem das Programm `/usr/bin/backup` mit dem Namen der zu sichernden Datei als Argument gestartet.
- Das Hauptprogramm wartet auf die Terminierung des Kindprozesses und fährt dann mit dem Durchsuchen des Verzeichnisses fort.
- Fehler beim Sichern einzelner Dateien sollen nicht zum Abbruch des Programms führen.
- Das Hauptprogramm terminiert, nachdem das Verzeichnis vollständig abgearbeitet wurde.

N:

Ergänzen Sie das folgende Codegerüst so, dass ein vollständig übersetzbares Modul entsteht.

```
#include <dirent.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/stat.h>
#include <unistd.h>
#include <sys/wait.h>

/* Funktion main */

.....
.....
.....
.....
.....
.....

/* Verzeichnis oeffnen */

.....
.....
.....
.....
.....
.....
```



```
  /* Verzeichnis durchsuchen und Dateien sichern */

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

  /* Ressourcen freigeben */

.....
.....
.....
/* Ende der Funktion main */
```

0,5

B:

Aufgabe 3: (18 Punkte)

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

a) In Bezug auf die Semantik bei der Übergabe von Parametern an Funktionen unterscheidet man generell die Konzepte *call by value* und *call by reference.* Beschreiben Sie die beiden Konzepte! (4 Punkte)

b) Wie wird die Parameterübergabe über den Stack (z. B. auf einer Intel-Prozessorarchitektur) realisiert. Beschreiben Sie, wie die aktuellen Parameter abgelegt werden und wie der Zugriff über die formalen Parameter erfolgt (ggf. durch Skizze veranschaulichen!) (6 Punkte)

c) Die Programmiersprache C unterscheidet vier Kategorien in Bezug auf die Gültigkeit (d. h. die Sichtbarkeit) von Variablen. Nennen Sie die verschiedenen Gültigkeitsbereiche und beschreiben Sie, wie bzw. wo man eine Variable dafür jeweils anlegen muss. (4 Punkte)

d) Welche Bedeutung haben die Gültigkeitsbereiche in Bezug auf Nebenläufigkeit bei Interrupts (d. h. wie sind Variablen der verschiedenen Kategorien von Nebenläufigkeitsproblemen betroffen und warum ist dies so)? (4 Punkte)

Aufgabe 4: (11 Punkte)

a) Beschreiben Sie das Synchronisationskonzept "Semaphor". Welche Operationen bieten Semaphore an und wie funktionieren diese Operationen? (7 Punkte)

b) Gegeben sind zwei Threads T1 und T2. T1 führt die Operationen X1 und X2 aus, T2 die Operationen Y1 und Y2. Ergänzen Sie die Skizze in programmiersprachenähnlicher Form, wie man mit Hilfe eines Semaphors S die beiden Threads so synchronisiert, dass T1 vor der Ausführung der Operation X2 darauf wartet, dass Thread 2 die Operation Y1 ausgeführt hat (Barrier-Synchronisation). (4 Punkte)

T1:	**T2:**
`X1;`	`Y1;`
`X2;`	`Y2;`