Prüfung Betriebssysteme 1 (WS 23/24)

Hinweise:
Die Klausur umfasst 8 Aufgaben auf 6 Seiten. Die Arbeitszeit beträgt **90 Minuten**. Bitte notieren Sie auf jedem abgegebenen Blatt Ihren Namen, Ihr Nutzerkennzeichen („s-Nummer“) und Ihre Matrikelnummer! Zur Bearbeitung der Aufgaben sind **keine Hilfsmittel** zugelassen. Bitte geben Sie alle Aufgabenblätter mit ab und vermerken Sie bitte die **Anzahl abgegebener Blätter** (ohne die Aufgabenblätter) auf dem Deckblatt! Mit Ihrer Unterschrift bestätigen Sie die Bereitschaft zur Prüfung sowie die Belehrung über die Prüfungsbedingungen. Bei versuchter oder erfolgter Täuschung gilt die Prüfung als nicht bestanden.

Name: _____ Mat.-Nr.: _____ Unterschrift: _____

NKZ (s-Nr.): _____ abgegebene Blätter: _____

Aufgabe	1	2	3	4	5	6	7	8	$\sum$
Punkte	4	13	13	11	7	4	4	0	56

1. Gegeben sind zwei Klassifikationskriterien für Betriebssysteme. Nennen Sie die zugehörigen Kategorien und geben Sie jeweils ein passendes Beispiel eines Betriebssytems an (insgesamt 4). 4

Nutzer-Anzahl	Single-User-Betriebssystem	
	z. B.	z. B.
Anzahl gleichzeitiger Aktivitäten		
	z. B.	z. B.

- Single-User-OS (MS-DOS) vs. Multi-User-OS (Linux, Windows NT und später) (*) auf Kategorien, (*) auf 2 Beispiele
- Single-Tasking-OS (MS-DOS) vs. Multi-Tasking-OS (Linux, Windows NT)

⌀3.8/4 Punkte. Omaaufgabe. 'Nuff said.

2. Gegeben sind die folgenden einfachen regulären Ausdrücke. Beschreiben Sie so präzise wie möglich, welche Zeichenketten selektiert werden und geben Sie jeweils ein *konkretes* Beispiel für diese Zeichenketten an.

   (a) `\<P[aeiou]\+l\>` 2

   alle Worte, die mit 'P' beginnen, dann folgen beliebig viele Vokale (mindestens einer) und dann ein 'l'. Beispiel: „Paul“ (*)

   Einige haben hier „`+1`“ gelesen, aber einen solchen Ausdruck gibt es in RegExp nicht.

   (b) `\<[Ss]at\?uriert\?\>` 2

alle Worte, die mit einem großen oder kleinen 'S' beginnen, danach ein 'a', danach kann ein 't' stehen, es folgen 'urier' und dann kann noch ein 't' stehen. Beispiel: `saturiert`, `Saurier`.

Hier kannten manche nicht die Bedeutung des Fragezeichens.

(c) `^#include.*>$` 2

alle *Zeilen*, die mit einem `#include`-Statement beginnen (*) und mit einem > enden (*); Beispiel: `#include <stdio.h>` (*) (d. h., Systemincludes, keine eigenen)

Hier wurde gern übersehen, dass die Zeile mit einem > enden muss. Folglich werden auch nicht alle Includes selektiert, sondern nur die Systemincludes (aber z. B. nicht `#include "foo.h"`). Manche übersahen des weiteren, dass ganze Zeilen gesucht werden, keine Worte.

(d) `\<\([0-9]\{1,3\}\.\)\{3\}[0-9]\{1,3\}\>` 2

IP-Adressen *ohne* Berücksichtigung der Oktettgröße (d. h., „999“ würde als Oktett akzeptiert)

Meist richtig beantwortet.

(e) `\<[:alnum:][:alnum:]/[:alnum:][:alnum:]/[:alnum:][:alnum:]\>` 2

3 Paare von beliebigen Buchstaben oder Ziffern, jeweils getrennt durch einen Slash; Beispiel: `0x/GG/28`

Manche vermuteten hier eine Datumsangabe, aber es sind auch jegliche Buchstaben in den Zeichenpaaren erlaubt.

(f) `\<\([0-9a-fA-F]\{2\}:\)\{5\}[0-9a-fA-F]\{2\}\>` 3

6 zweistellige Hexzahlen (*), getrennt jeweils durch einen Doppelpunkt (*); d. h., alle MAC-Adressen, in hexadezimaler Schreibweise Beispiel „28:d2:44:e4:e9:45“ (*)

Viele vermuteten (fälschlich), dass es sich hier um eine IPv6-Adresse handelt. Diese hat aber 8 Komponenten! Es ist in Wirklichkeit eine MAC-Adresse; das war aber gar nicht gefragt. Wichtig ist, dass es hexadezimale Zahlen sind, keine beliebigen Buchstaben/Zahlen-Kombinationen.

Insgesamt mit ⌀9.2/13 ganz gut beantwortet; Sie haben sich offenbar ordentlich auf RegExp vorbereitet – oder ich habe zu lasch bepunktet. Einige haben im Eifer des Gefechts die Angabe der Beispiele komplett vergessen, was natürlich erhebliche Punkteinbußen nach sich zieht. Binsenweisheit: Aufgabenstellung sorgfältig lesen!

3. Notieren Sie UNIX-Kommandos bzw. -Kommandofolgen, die die folgenden Anforderungen erfüllen. Achten Sie auf die angegebenen Verzeichnisse und eventuell notwendige Umleitungen.

(a) Kopieren aller Dateien, deren Name auf `.txt` endet, aus dem *aktuellen* Verzeichnis in das *übergeordnete* Verzeichnis 2

```
cp *.txt ..
```

Meist korrekt beantwortet. Ist allerdings auch was für die Oma.

(b) Die Dateinamen des gesamten Verzeichnisbaums soll in Langform in eine Datei namens `ls-lR` geschrieben werden, die im aktuellen Verzeichnis entstehen soll. Fehlermeldungen sollen unterdrückt werden. 3

```
ls -lR / >./ls-lR 2>/dev/null
```

Manche vergaßen das Wurzelverzeichnis (es sollte ja der *gesamte* Verzeichnisbaum ausgegeben werden), relativ viele vergaßen die Umleitung von `stderr`.

(c) Eine Datei `ganzwichtig.org`, die sich im *Wurzelverzeichnis* des aktuellen Rechners befindet, soll auf den entfernten Rechner `ilpro122.informatik.htw-dresden.de` in das `Homeverzeichnis` des Nutzers `robge` kopiert werden. 3

```
scp /ganzwichtig.org robge@ilpro122.informatik.htw-dresden.de:~
```

Der Slash vor `ganzwichtig.org` ist ganz wichtig – er wurde gern vergessen. Viele wussten nicht, wie man den Nutzer beim entfernten Ziel angibt und wollten dann nach `/home/robge` (o. ä.) kopieren. Das ignoriert aber die Tatsache, dass sie sich dann als der gleiche Nutzer wie

lokal anmelden, und dann normalerweise keine Zugriffsrechte für das Homeverzeichnis eines anderen Nutzers haben.

(d) Es soll der Prozess mit der PID 32168 abgebrochen werden (der Prozess gehört dem aufrufenden Nutzer). 2

```
kill -SIGKILL 32168
```

Ziemlich viele gaben hier als Lösung identisch „`kill(SIGKILL,32168)`“ an. Das ist aber ein Syscall und kein Kommando; somit habe ich einen Punkt abgezogen. Außerdem deutet das natürlich auf die Existenz eines nichtautorisierten Kommunikationskanals während der Prüfung hin. Schade. Andere Signale habe ich auch als richtig gewertet, sofern diese den Abbruch als Defaultaktion aufweisen.

(e) Aus allen Dateien, deren Name auf `.c` endet, und die sich im Verzeichnis `john`, das im Verzeichnis `src` liegt, welches seinerseits im Homeverzeichnis des Nutzers liegt, (die Dateien liegen also zwei Ebenen unterhalb des Homeverzeichnisses) befinden, sollen alle C-Kommentare (und nur diese) alphabetisch sortiert ausgegeben werden. 3
Hinweis: C-Kommentare haben die Form: `/* <beliebige Zeichen> */`

```
cat ~/src/john/*.c | grep -o "/\*.*\*/" | sort
```

Hier gab es einige wilde Vorschläge, die zeigen, dass doch nicht alle verstanden haben, wie die Pipe auf Bash-Ebene funktioniert. Manche vergaßen den `-o`-Switch bei `grep` (und selektierten damit die ganze Zeile), andere berücksichtigten nur Kommentare, die allein auf einer Zeile stehen. Auch der RegExp war nicht immer einwandfrei. Auch ein `uniq` war eigentlich nicht gefordert. Einige haben selbstständig gleich noch nach C++-Kommentaren (`//`) gesucht, was ich ausnahmsweise nicht bestraft habe, obwohl es strenggenommen keine korrekte Lösung ist.

Insgesamt wurde die Aufgabe einigermaßen akzeptabel beantwortet (mit ⌀8.8/13 Punkten). Totalausfälle gab es nur vereinzelt; die Mehrheit der Teilnehmer wies zumindest brauchbare Kenntnisse einiger UNIX-Kommandos nach.

4. Analysieren Sie das folgende Shellskript. Beantworten Sie dazu die folgenden Fragen.

```
1  #!/bin/bash
2
3  if [ $# -ne 2 ]; then
4      echo "Wrong number of parameters"
5      exit 1
6  fi
7
8  if [ -d $2 ]; then
9      echo "Directory $2 already exists."
10 else
11     mkdir $2
12     if [ $? -ne 0 ]; then
13         echo "Cannot create directory $2."
14         exit 2
15     fi
16 fi
17
18 ls -1 ./*.$1 &>/dev/null
19 if [ $? -ne 0 ]; then
20     echo "No files found"
21     exit 3
22 fi
23
24 let sum=0
25 let cnt=0
26 for file in ./*.$1
27 do
28     cp $file $2
29     let cnt++
30     size=$(cat $file | wc -c)
31     ((sum=$sum+$size))
32 done
33
34 echo "Copied: $sum bytes in $cnt files."
```

```
35  exit 0
```

(a) Wozu dient das Statement in Zeile 1? 1

Das ist die Angabe, welcher Interpreter zur Abarbeitung des Shellskripts gestartet werden soll.

Das wussten die meisten; ich habe auch „Magic Word“ zur Erkennung von Shellskripten als richtig anerkannt. Einige schrieben aber selbstreferentielle Sachen wie „Das ist der She-Bang, er steht in der ersten Zeile eines Shellskriptes.“ – das beantwortet die Frage nicht.

(b) Wozu dient das Skript? Was wird ausgegeben? 4

Es kopiert alle Dateien mit der als Kommandozeilenparameter übergebenen Dateinamensendung (*) in das Verzeichnis, das als zweiter Parameter (*) übergeben wird. Die Anzahl kopierter Dateien sowie die Summe deren Größe in Bytes wird nach stdout geschrieben (*). Falls ein Fehler auftritt, wird stattdessen eine Fehlermeldung ausgegeben (*).

Die meisten Teilnehmer beantworteten diese Teilaufgabe ordentlich (sie ist infolge der `echo`-Statements allerdings ziemlich einfach). Manche schrieben fälschlich, dass *eine* Datei kopiert würde. Wer hier komplett patzte, hat seinen Beleg garantiert plagiiert. Shame on you!

(c) Welche potentiellen Fehler werden durch das Skript abgefangen? 4

- falsche Anzahl ($\neq 2$) an Kommandozeilenparametern
- versehentliches Anlegen des Zielverzeichnisses, wenn dieses bereits existiert
- fehlschlagendes Anlegen des Zielverzeichnisses, sofern dieses noch nicht existiert
- keine Dateien mit der angegebenen Dateinamensendung im aktuellen Verzeichnis vorhanden

Wurde von der übergroßen Mehrzahl der Teilnehmer korrekt beantwortet.

(d) Wozu dient die Zahl hinter den `exit`-Statements in den Zeilen 5, 14, 21 und 35? Welche Werte sind an dieser Stelle prinzipiell möglich? 2

Das ist der Resultatwert des Skriptes, der in der rufenden Umgebung ausgewertet werden kann. Der zulässige Wertebereich ist 0…255 (1 Byte).

Hier musste ich relativ häufig Punkte abziehen. Ziemlich viele antworteten auf die zweite Frage: „0, 1, 2 und 3“, „alle Integerwerte“ oder erklärten, dass 0 bei Erfolg und $\neq 0$ bei Mißerfolg ausgegeben werden muss. Das war aber nicht gefragt. Ich bin für meinen Geschmack ziemlich draufrumgeritten, dass genau 1 Byte an dieser Stelle übergeben werden kann, obwohl die Bash beliebige Integerwerte akzeptiert.

Insgesamt ⌀8.4/11 Punkten, also ein einigermaßen akzeptables Ergebnis.

5. Beschreiben Sie knapp, wie zwei Prozesse mittels einer Pipe Daten austauschen können! Welche Systemrufe sind dafür in welcher Reihenfolge nötig? Gehen Sie auf die Vorbereitung, den eigentlichen Datenaustausch und das „Aufräumen“ ein. 7

- `pipe()` aufrufen, legt Kommunikationskanal an (*)
- `fork()` dupliziert Prozess (*)
- jeder der beiden Prozesse muss einen Filedeskriptor der Pipe schließen mittels `close` (*)
- Datenaustausch (unidirektional) mittels `read()` und `write()` (**) über den noch offenen Deskriptoren
- nach Abschluss rufen beide Prozesse `close()` über dem benutzten Deskriptor auf → bei letztem `close()` wird Pipe gelöscht (*)
- Einen weiteren Punkt (*) für saubere Lösung, Fehlerbehandlung, `wait()`.

Hier schlägt's richtig 'rein. Neben vielen vollständig korrekten Lösungen gibt es bei dieser Aufgabe auch jede Menge Totalausfälle (dazwischen ist eigentlich nichts). Ich erinnere mich an die wenigen Teilnehmer, als die Pipe in der Vorlesung dran war. Und im Praktikum waren wohl zu viele mit ihrem Beleg beschäftigt – ich habe aber mehrfach auf die Prüfungsrelevanz hingewiesen. ⌀4/7 Punkte ☹

6. In einem Rechensystem mögen 4 Prozesse per *Round Robin* geplant werden. Die Länge des Quantums betrage $t_q = 100$ms, die Dauer des Kontextwechsels betrage $t_{cs} = 5$ms.

   (a) Ermitteln Sie die Dauer zwischen 2 Aktivierungen ein- und desselben Prozesses unter der Voraussetzung, dass alle Prozesse ihr Quantum vollständig ausnutzen. [2]

   (b) Angenommen, die Dauer des Quantums wird halbiert. Verbessern oder verschlechtern sich dadurch die Kenngrößen des Systems *mittlere Reaktionszeit* und *Verwaltungsaufwand*? Begründen Sie Ihre Antwort. [2]

   (a) 4x100ms+4x5ms=420ms (einen, wenn 4x gerechnet wird)

   (b) Die mittlere Reaktionszeit verbessert sich, da die Prozesse häufiger drankommen (*). Der Verwaltungsaufwand verschlechtert sich, weil anteilig mehr Umschaltaufwand nötig ist (*).

   Eigentlich eine Omafrage. Viele verrechnen sich und beziehen den Prozess selbst nicht mit ein (falsches Ergebnis: 315ms). Häufig werden die Begründungen „vergessen", geschätzt 10% wissen gar nicht, wovon ich hier rede. ⌀2.4/4 Punkten

7. Beantworten Sie die folgenden Fragen knapp, aber präzise.

   (a) Nennen Sie zwei verschiedene Möglichkeiten, einen Prozess zu beenden. [1]

   Signal zustellen, `exit()` aufrufen, `return` in `main()`

   (b) Interpretieren Sie die Rechte der Datei `shellskript-04.sh` (Wer darf was?)! [2]

   ```
   -rwxr-x--x 1 robge fi    563 Feb  5 19:42 shellskript-04.sh
   ```

   Der Eigentümer (`robge`) darf Lesen, Schreiben und Ausführen; Mitglieder der Gruppe (`fi`) dürfen die Datei lesen und ausführen, alle anderen Nutzer dürfen die Datei nur ausführen.

   (c) Was versteht man unter präemptiven Multitasking? [1]

   Jede Aktivität kann jederzeit durch den Scheduler unterbrochen werden.

   Die meisten konnten a) und b) problemlos bearbeiten; große Probleme gab es aber bei c) – mit diesem Begriff konnte die große Mehrheit nichts anfangen, was schlecht ist, denn es ist ein äußerst wichtiger Begriff. ⌀2.8/4 Punkte

8. **Bonus:** Beschreiben Sie knapp, was der folgende (unsinnige) Code tut. Wievielmal wird `Huhu!` ausgegeben? Begründen Sie Ihre Antwort. Die `#include`-Anweisungen wurden der Übersichtlichkeit halber fortgelassen. [3 (B)]

```c
void sighandler(int sig)
{
  return;
}

int main(void)
{
  int ret;

    ret = fork();
    if (ret == 0) {
      signal(SIGINT, &sighandler);
    }
    else {
      ret = kill(ret, SIGINT);
    }
    printf("Huhu!\n");
```

```
        exit(0);
    }
```

Vater erzeugt einen Sohn. Der Sohn installiert für `SIGINT` einen Handler, der nur zurückkehrt, während der Vater konkurrierend dem Sohn `SIGINT` schickt. Ist letzterer schneller, dann wird der Sohn abgebrochen, und es wird nur ein Mal `Huhu!` ausgegeben. Ist hingegen der Sohn schneller, dann wehrt er gewissermaßen das Signal ab und es wird in der Folge zwei Mal `Huhu!` ausgegeben.

Das war ein bißchen schwieriger. Vollständig haben das nur 5.5 Teilnehmer (einer hat die korrekte Lösung wieder durchgestrichen . . .) beanwortet. Eine ganze Menge haben aber verstanden, dass der Sohn einen Handler installiert und der Vater SIGINT schickt; auch dies lieferte Punkte. Manche schrieben allerdings auch teilweise hanebüchene Erklärungen. ⌀1.1/3 Punkten

Fazit:
Insgesamt gab es diese Notenverteilung:

Note	1.0	1.3	1.7	2.0	2.3	2.7	3.0	3.3	3.7	4.0	5.0
Anzahl	5	8	11	1	3	5	3	3	5	5	10

Das arithmetische Mittel beträgt 2.78, der Median ist 2.7. Einerseits gibt es diesmal ungewöhnlich viele sehr gute und gute Noten. Andererseits ist die Durchfallquote von 17% ganz schön happig (aber ein kleines bißchen besser als im letzten Jahr).

Der sprachliche Ausdruck ließ manchmal arg zu wünschen übrig; klassische Orthografiefehler sind „ließt“ (als Vergangenheitsform von „lesen“) und „Vorraussetzung“. Die Handschriften waren in diesem Jahr einigermaßen gut zu entziffern, immerhin. Ein Teilnehmer nutzte silberne Tinte, was unter direktem Lampenlicht prima metallisch reflektiert und einen beim Lesen schier wahnsinnig macht. Die Verwendung solcherlei Schreibgeräts ist ab sofort strengstens verboten!