Prüfung Betriebssysteme 1

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	5	10	13	8	8	7	9	0	60
erreicht									

1. Die folgende Tabelle soll Klassifikationskriterien für Betriebssysteme, die entsprechenden Ausprägungen und Beispiele für die Ausprägungen enthalten. Füllen Sie alle leeren Felder aus. [5]

Klassifikationskriterium	**Nutzeranzahl**	
Kategorien	Single-User-BS	Multi-User-BS
Beispiele	MS-DOS	Linux

Klassifikationskriterium	**Anzahl paralleler Aktivitäten**	
Kategorien	Single-Tasking-BS	Multi-Tasking-BS
Beispiele	MS-DOS	Windows

Mehrheitlich sehr gut beantwortet (⌀4.6 Punkte), aber natürlich sehr, sehr leicht. Manche haben vergessen, das zweite Klassifikationskriterium auszuformulieren. Manche schreiben als Beispiel für ein Single-Tasking-BS „Linux“.

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) `\<[AaEeIiOoUu][aeiou]\+\>` [2]

   groß- und kleingeschriebene Worte, die nur aus Vokalen bestehen, mindestens 2 Buchstaben lang (*), z. B. „Aeiou“ (*).
   Manche verwechseln Selbstlaute mit Umlauten.

   (b) `0x[0-9a-fA-F]\{4\}:[0-9a-fA-F]\{4\}` [2]

   ZK, beginnend mit „0x“, dann 4stellige Hexzahl, ‘:’ und weitere 4stellige Hexzahl (z. B. „0xFFFF:E000“)

(c) `#.*$` 2

alle ZK, die mit einem '#' beginnen bis zum Ende der Zeile (alle Bash-Kommentare)

(d) `\<M\(is\{2\}\)\{2\}ip\{2\}i\>` 2

„Mississippi" (nichts weiter)

(e) `\<0\>\|[-+][1-9][0-9]*\>` 2

eine einzelne '0' oder beliebige Ganzzahlen, mit erzwungenem Vorzeichen, z. B. '+1000'

Mehrheitlich recht ordentlich beantwortet (⌀6.7 Punkte). Manche vergaßen konkrete Beispiele (Aufgabenstellung nicht bis zu Ende gelesen?). Aufgabe c) und „Mississippi' bereiteten Mühe.

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 ..
```
Manche kennen die Syntax für das übergeordnete Vz. nicht.

(b) Anzeige aller verschiedenen Worte aus der Datei `bibel.txt`, die sich im *Home*verzeichnis befindet. 2

```
grep -o "\<[[:alpha:]]\{2,\}\>" ~/bibel.txt | sort | uniq
```
Viele vergessen, die einzelnen Worte zu selektieren, oder machen kein `sort` vor `uniq`.

(c) Ausgabe aller Zeilen der Datei `foo.txt`, die mit einer zweistelligen Zahl beginnen. 2

```
grep "^[0-9]\{2\}" foo.txt
```
Meist richtig beantwortet; einige kennen das Symbol für den Zeilenanfang nicht.

(d) Auf dem (entfernten) Rechner ilux150 soll der Prozess mit der PID 32168 abgebrochen werden (der Prozess gehört dem aufrufenden Nutzer). 2

```
ssh ilux150 "kill -SIGKILL 32168"
```
Viele vergessen hier, dass *entfernt* zu arbeiten ist, nur sehr wenige haben das richtige Signal verwendet.

(e) Die Gesamtanzahl Zeilen aller Dateien im aktuellen Verzeichnis, die auf '.txt' enden, soll ausgegeben werden (nichts weiter). 2

```
cat *.txt | wc -l
```
Fast immer richtig beantwortet; manche rufen irrigerweise `ls` anstelle `cat`.

(f) 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
```
Typische Fehler: Das Wurzelverzeichnis '/' als Ausgangspunkt nicht angegeben, manche kennen die Umleitungssymbole nicht.

Aufgabe wurde sehr heterogen gelöst (⌀7.5 Punkte), zwischen 0 und 13 Punkten war alles dabei.

4. Schreiben Sie ein Bash-Shellskript, das das folgende leistet: 8

- Übernahme von genau *einem* Kommandozeilenparameter, einer IP-Adresse,
- bei fehlerhafter Parameteranzahl Ausgabe einer Meldung und Beendigung des Skriptes,

- syntaktische Prüfung der IP-Adresse *ohne* Prüfung des Zahlenbereichs (d. h. 900.1.1.1 wäre erlaubt); ggf. Fehlermeldung und Beendigung des Skriptes,
- Test, ob Rechner erreichbar ist; falls nicht, Fehlermeldung und Beendigung des Skriptes,
- Kopieren der Datei `auth.log` als aktueller Nutzer aus dem Verzeichnis `/var/log` des entfernten Rechners in das aktuelle lokale Verzeichnis mit Fehlerprüfung (dies würde real fehlschlagen, weil Sie dafür root-Rechte benötigen; dies ignorieren wir),
- Prüfung, ob Kopiervorgang erfolgreich, Ausgabe einer Fehlermeldung und Ende des Skriptes, falls nicht,
- Ermittlung und Ausgabe der Anzahl fehlerhafter Einloggversuche, erkennbar an der Zeichenkette „`Failed password`“ in der Datei
- Falls Zeichenkette nicht gefunden, Ausgabe von „Keine fehlerhaften Einloggversuche.“

Hinweise:

- Die Kommandoausgaben müssen, anders als im Beleg, nicht unterdrückt werden.
- Denken Sie an definierte Rückgabewerte und an die erste Zeile eines Shellskriptes.

```
#!/bin/bash

if [ $# -ne 1 ]; then
    echo "Usage: foo <ipaddress>"
    exit 1
fi
echo "$1" | grep "\<\([0-9]\{1,3\}\.\)\{3\}[0-9]\{1,3\}\>"
if [ $? -ne 0 ]; then
    echo "No IP address given"
    exit 2
fi
ping -c1 $1
if [ $? -ne 0 ]; then
    echo "Host not reachable"
    exit 3
fi
scp $1:/var/log/auth.log .
if [ $? -ne 0 ]; then
    echo "Copying /var/log/auth.log failed."
    exit 4
fi
failed=$(cat auth.log | grep "Failed password" | wc -l)
if [ $failed -eq 0 ]; then
    echo "No failed login attempts"
else
    echo "$failed login attempts"
fi
exit 0
```

Bewertung: 1 auf Shebang, 1 auf `$#`-Test, 1 auf IP-Adress-Test, 1 auf `ping` 1 auf `scp` 1 auf Zählen aller fehlgeschlagenen Logins, 1 auf korrekte `exits`, 1 auf allgemeine Richtigkeit des Skriptes

Wiederum sehr heterogene Punktverteilung; man sieht sehr genau, wer seinen Beleg selbst programmiert hat und wer nicht (⌀4.6 Punkte). Typische Fehler:

- RegEx für IP-Adresse falsch
- fälschlich `wget` anstelle `scp`, bei `scp` den entfernten Rechner nicht angegeben
- Anzahl Einloggversuche zwar ermittelt, aber nicht in Variable abgelegt, nicht ausgegeben
- mindestens die Hälfte der Teilnehmer liefert im Fehlerfall negative Rückgabewerte zurück
- mehrere Teilnehmer programmieren den Syntaxtest der IP-Adresse so:
  `if [ $1 == '<IP-Adresse>' ]; then ...`

5. Analysieren Sie das folgende C-Programm, und beantworten Sie danach die folgenden Fragen:
   (a) Welche Systemrufe werden durch wen in welcher Reihenfolge ausgeführt? 2
   (b) Steht die Reihenfolge der Ausgaben fest? Geben Sie eine gültige Ausgabereihenfolge an. 2
   (c) Wozu dient die Funktion `func()`? Wann wird sie aufgerufen (wenn überhaupt)? 2
   (d) In welcher Reihenfolge werden die Prozesse beendet? Begründen Sie Ihre Antwort. 2

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

void func(int nr)
{
  printf("Eins\n");
  signal(SIGUSR1, SIG_DFL);
  return;
}

int main(int argc, char* argv[])
{
  pid_t pid;
  int ret;

  pid = fork();
  if (pid == -1) {
    perror("fork");
    exit(1);
  }

  if (pid == 0) {
    ret = (int) signal(SIGUSR1, &func);
    if (ret == SIG_ERR) {
      printf("Vier\n");
      exit(1);
    }
    while(1)
      ;
    printf("Zwei\n");
  }

  else {
    sleep(5);
    kill(pid, SIGUSR1);
    sleep(5);
    kill(pid, SIGUSR1);
    wait(NULL);
    printf("Drei\n");
    exit(0);
  }
}
```

Hinweise:

- Die Fehlerbehandlung wurde bewusst teilweise vernachlässigt.
- Es könnte ggf. Anweisungen geben, die nicht (!) erreicht werden.

(a) Vater: `fork()`, `sleep()`, `kill()`, `sleep()`, `kill()`, `wait()`, `exit()`
Sohn: `signal()`, `signal()`

(b) Ja, steht fest. `Eins`, `Drei` (`Zwei` wird nicht ausgegeben, da Sohn vorher abgebrochen wird). Auch richtig: „steht nur fest, wenn `signal()` keinen Fehler liefert“

(c) `func()` ist ein Signalhandler. Er wird aufgerufen, wenn der Vater dem Sohn **das erste Mal** mittels `kill()` das Signal `SIGUSR1` schickt.

(d) Der Sohn wird zuerst beendet, da der Vater ihn aktiv mit `kill()` tötet, und *zusätzlich* auf sein Ende mittels `wait()` wartet.
Verheerend! Geschätzt 10% der Teilnehmer haben verstanden, wie der Signalmechanismus funktioniert (⌀3.4 Punkte). Teilaufgabe d) wurde zumeist noch richtig beantwortet. Typische Fehler:

- `signal()` als „Signal wird gesendet“ aufgefasst (erinnert sich jemand an meine Warnung aus dem Praktikum?)
- „`signal()` springt zum Handler `func()`“
- `SIG_DFL` und/oder `SIG_ERR` als Signale angesehen
- Unendliche Schleife als `printf()` beinhaltend angesehen (Dabei habe ich sogar in die Hinweise geschrieben, dass es nichterreichbare Anweisungen geben könnte!)
- nicht verstanden, dass nach `fork()` zwei unabhängige Prozesse existieren
- fälschlich angenommen, Vater schicke sich selbst das Signal

6. Ein Prozess möchte einem anderen Prozess mit Hilfe einer Datei (unidirektional) Daten übermitteln.
   (a) Nennen Sie die benötigten Systemrufe im Sender- und im Empfängerprozess! [4]
   (b) Handelt es sich um einen persistenten IPC-Mechanismus? Begründen Sie Ihre Antwort! [1]
   (c) Was gibt der Dateipositionszeiger an? Nennen Sie zwei verschiedene Möglichkeiten, ihn zu bewegen. [2]

(a) Sender: `open()`, `write()`, `close()`; Empfänger: `open()`, `read()`, `close()`

(b) ja, weil die Datei ihren Erzeuger ggf. überlebt.

(c) Die Position in einer eröffneten Datei, von der als nächstes gelesen bzw. geschrieben wird. Versetzung mittels `read()`/`write()` bzw. explizit mittels Seek-Operation.

Die Aufgabe war als einfachster Punktelieferant gedacht, nur sind die Dateirufe schon ziemlich am Anfang drangewesen. Das Ergebnis fällt sehr schlecht aus: ⌀ 2.7 Punkte. Mir ist unklar, wo die Ursache liegt. Immerhin 9 Studenten haben die Pipe anstelle der Datei behandelt. Mehrere Teilnehmer beweisen, dass es sich um einen IPC-Mechanismus handelt (es war gefragt, ob und warum dieser *persistent* ist!) Ursache dürfte ein ungenügendes Lesen der Aufgabenstellung sein.

7. Beantworten Sie die folgenden Fragen knapp, aber präzise.
   (a) Was versteht man bei UNIX unter einem Filter? Warum ist es günstig, Werkzeuge als Filter zu implementieren? [2]

Programm, das von `stdin` liest und nach `stdout` schreibt. Filter können per Pipe an der Kommandozeile miteinander verkettet werden.

   (b) Wie wird eine Pipe in der Programmiersprache C angelegt, wie wird sie gelöscht? [2]

Anlegen: Syscall `pipe()`, Löschen durch Schließen aller offenen Dateideskriptoren (Vater und Söhne; `close()`)

   (c) Interpretieren Sie die Rechte der Datei `shellskript-04.sh` (Wer darf was?)! [3]

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

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

   (d) Welche (drei) Parameter beeinflussen die Reaktionszeit eines Prozesses beim Round-Robin-Scheduling? [2]

Anzahl Prozesse n, Dauer Quantum t_q, Dauer des Context Switch t_{cs}

Mittelmäßig beantwortet: viele wissen die Parameter beim RR-Scheduling und kennen die UNIX-Dateirechte. Deutlich weniger wissen, wie eine Pipe programmiert wird und kaum einer weiß (zwei mickrige Teilnehmer, IIRC) was ein Filter ist (⌀ 4.3 Punkte). Reines Auswendigwissen, kein Verständnis nötig. Eher enttäuschend.

8. **Bonus:** Beschreiben Sie, was die folgende Bash-Kommandozeile tut: 2 (B)

```
read n; s=0; while [ $n -gt 0 ]; do ((s=$s+$n)); ((n--)); done; echo $s
```

Liest eine Zahl n ein und ermittelt $s = \sum_{i=1}^{n} i$.

Von ziemlich vielen Teilnehmern bearbeitet worden und dies gar nicht mal so schlecht (⌀ 1.2 Punkte). Hauptfehler: nur textuell die einzelnen Statements wiedergegeben, aber nicht erkannt, dass einfach die Summe von 1 bis n gebildet wird, fälschlich $n!$ (Fakultät), Fibonacci-Zahl oder Eulersche Zahl als Ergebnis vermutet, „ließt“ anstelle „liest“ geschrieben (grauenhaft!)

Alles in allem ist das Ergebnis dieser Klausur sehr schlecht. Eine Durchfallquote von 33% (Vorjahr: 20%) ist katastrophal. Hauptprobleme sind das mangelhafte Verstehen des `fork()`- und des Signalmechanismus sowie das Beherrschen der Arbeit mit Dateien in C. Die Linuxkommandos und das Shellskript wurden hingegen einigermaßen akzeptabel beherrscht. Hinzu kommen offensichtliche Probleme, die Aufgabenstellung und Hilfestellungen *komplett* zu erfassen (viele beantworten die gestellten Fragen nicht und schreiben *irgendwas* aus dem Kontext).