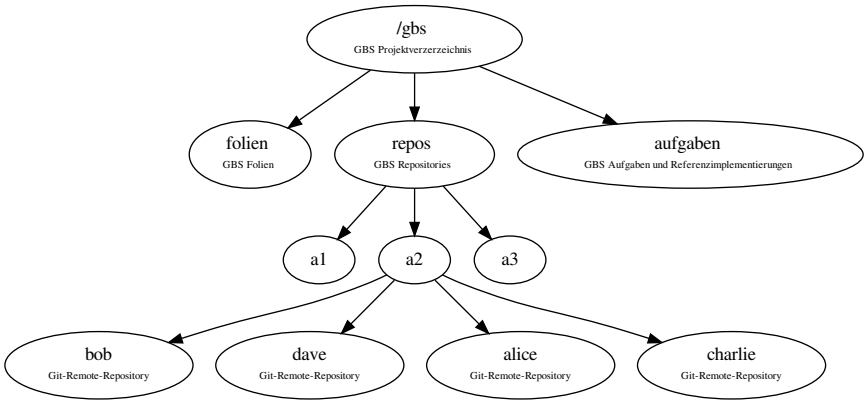
Aufgabe 1: Verkettete Listen

Wichtig: Lesen Sie die "**Hinweise**" auf diesem Aufgabenblatt; Spezifikationen in diesem Teil sind ebenfalls einzuhalten!

Abgabetermin Gruppe A: 26.10.2023 23:59
Abgabetermin Gruppe B: 02.11.2023 23:59

Allgemeine Hinweise zu den GBS Übungen

- Die Aufgaben sind in Zweiergruppen zu bearbeiten (siehe Aufgabenstellung). Bei Gruppenarbeit sind Lösungsweg und Programmierung gemeinsam zu erarbeiten.
- Zur Versionsverwaltung wird jedem Teilnehmer für jede Aufgabe ein Git-Repository zur Verfügung gestellt, welches auch zur Abgabe verwendet wird. Zum Zugriff auf das Repository muss jeder Teilnehmer zunächst SSH Zugang über die Weboberfläche unter `https://gbs.sra.uni-hannover.de` einrichten.
- Für jede Aufgabe ist die Bearbeitungszeit auf zwei Wochen festgelegt. Die Abgabe ist jeweils am Tag vor der nächsten Tafelübung bis 23:59 durch einen Git push zu tätigen.
- Die Übungsaufgaben müssen spätestens bis zum jeweiligen Abgabetermin abgegeben werden. In darauffolgenden Rechnerübungen werden einzelne abgegebene Lösungen besprochen – jeder Übungsteilnehmer muss dabei in der Lage sein, die gesamte Lösung seiner Gruppe zu erläutern. Kann jemand seine Lösung auf Anforderung nicht erklären, wird für ihn die Aufgabe als nicht abgegeben bewertet (im Zweifelsfall kann hierzu ein Gespräch außerhalb der Rechnerübung stattfinden).
- Die abgegebenen Programme werden automatisch auf Ähnlichkeit mit anderen Programmen desselben Semesters und früherer Semester überprüft. Werden starke Übereinstimmungen festgestellt, wird die Aufgabe als nicht abgegeben bewertet.
- Jeder Benutzer erhält für GBS ein spezielles Projektverzeichnis mit dem Namen `/gbs/repos/<aufgabe>/<username>`. Die Projektverzeichnisse werden für alle angemeldeten Teilnehmer automatisch erstellt. In diesem Verzeichnis befinden sich die Git Remotes.



- Die Aufgaben sind durch rechtzeitiges pushen in das jeweilige Repository abzugeben. Eine Aufgabe kann beliebig oft abgegeben werden; es gilt, soweit nicht anders vereinbart, die letzte rechtzeitige Abgabe.
- Sollten Sie aus triftigem Grund eine fristgerechte Abgabe versäumen, so ist eine verspätete Abgabe weiterhin möglich. Damit eine solche Abgabe bei der Wertung berücksichtigt wird, ist jedoch in jedem Fall eine Rücksprache mit dem Übungsleiter erforderlich. Eine vorhandene rechtzeitige Abgabe wird durch eine verspätete Abgabe nicht gelöscht.

- Die Lösungen der Aufgaben müssen, sofern es Programmieraufgaben sind, auf den GBS Systemen lauffähig sein. Für die Bearbeitung kann auf einem beliebigen System gearbeitet werden. Wir empfehlen die Arbeit auf dem Abgabesystem oder der bereitgestellten VM um Kompatibilität und Interoperabilität zu gewährleisten.
- Die GBS_VM ist unter `https://sra.uni-hannover.de/GBS_VM.ova` abrufbar.
- Das Abgabesystem ist via `HTTP` und `SSH` unter `gbs.sra.uni-hannover.de` erreichbar.

Aufgabe 1: Verkettete Listen

Diese Aufgabe dient dem Einstieg in die Übungen der Veranstaltung Grundlagen der Betriebssysteme. Inhalt dieser Aufgabenstellung ist die Verwendung des Abgabesystems, der Umgang mit einem UNIX System und ein Einstieg in C Programmierung für Systementwicklung.
Für die Bearbeitung der Aufgaben werden jeweils Vorlagen als Git Repository bereitgestellt. Ihre Ergebnisse geben Sie ab, indem Sie ihre Resultate in ihr Repository commiten und den Stand auf unsere Server pushen. Der Zugang zum Abgabeserver ist via SSH möglich. Der Server ist über gbs.sra.uni-hannover.de erreichbar. Benutzername und Passwort erhalten Sie ebenfalls über diesen Server. Das Webportal ist erreichbar unter `https://gbs.sra.uni-hannover.de`.

1.1 Abgabesystem

Für die Abgabe der Übungsaufgaben gibt es ein Online-Abgabesystem. Dieses besteht aus mehreren Komponenten. Um dieses nutzen zu können, müssen Sie sich ihre Zugangsdaten abholen. Dies ist unter `https://gbs.sra.uni-hannover.de` möglich. Anschließend können Sie sich mittels ssh auf den Abgabeserver zugreifen.

1.2 Versionsverwaltung mit Git

Für jede Aufgabe erhalten Sie ein Git-Repository, in das die Abgaben eingetragen werden sollen. Dieses ist jeweils an folgendem Ort zu finden: `/gbs/repos/a1/<benutzername>`. Clonen Sie dieses (Remote-) Repository in ein Arbeitsverzeichnis. Ihr Repository für diese Aufgabe kann mit folgendem Aufruf geclont werden:
`git clone ssh://<username>@gbs.sra.uni-hannover.de:/gbs/repos/a1/<username> gbs_a1`

Jetzt könnte mit dem Bearbeiten der Aufgabe begonnen werden. Alle Ergebnisse werden anschließend in das Repository commited. Um die Aufgabe abzugeben muss der Zustand in das Abgabe-Repository übertragen werden. Dies kann beliebig oft geschehen. Gewertet wird stets die letzte rechtzeitige Abgabe.
`git push origin master`

1.3 Partnerwahl

Suchen sie ich einen Partner für die Abgabe dieser Aufgabe. Einigen sie sich darauf wer welche Rolle von Alice und Bob übernimmt. Alice sollte jetzt den Benutzernamen von Bob in die Datei `PARTNER` schreiben und dies an den Server senden, damit Bob Zugriff erhält. Darauf kann Bob das Repo clonen und seine ersten Ergebnisse eintragen und an den Server senden, damit der Handshake abgeschlossen wird. Haben Alice und Bob alle schritte erfüllt, so ist die Partnerwahl unumkehrbar abgeschlossen.
Eine detailierte Beschreibung des Protokolls ist in den Übungsfolien und in `alice_und_bob.pdf` aufgeführt.

1.4 Theoriefragen (4P)

Bitte geben Sie diesen Aufgabenteil in der Datei `antworten.txt` ab. Die Antworten sind in eigenen Worten zu formulieren.

1. Was sind die Aufgaben eines Betriebssystems? **(1P)**
2. Machen Sie sich mit dem UNIX-Kommando **man(1)** vertraut. **(3P)**
   (a) Erklären Sie die Ausgabe des Kommandos `ls | wc -l`.
   (b) Erklären Sie welche Rolle die Umgebungsvariable PATH spielt.
   (c) Geben Sie eine Möglichkeit an, um auf der UNIX-Konsole die aktuelle Wochennummer (Hinweis: Teil des Datums) anzuzeigen.

1.5 Hello World (1P)

Einfache Ausgabe
In dieser Aufgabe soll ein minimales C-Programm geschrieben werden.
Schreiben Sie ein C Programm, das folgende Ausgabe erzeugt:

```
Hallo Welt!
```

Erste Betriebssysteminteraktion
Auf einem UNIX-System wird jeder Benutzer mit seiner Nutzer-ID (UID) identifiziert. Ihm ist zusätzlich eine primäre Gruppe (GID) zugeordnet. Erweitern Sie ihr Programm, sodass es zusätzlich auch die UID und GID ausgibt:

```
Hallo Welt!
user: 1000
group: 1234
```

1.6 Leben mit Linux (2P)

Ermitteln Sie die folgendem Informationen mittels Kommandozeilenaufrufen. Schreiben Sie die benötigten Befehle in die Datei `befehle.txt`. Speichern Sie die erhaltenen Informationen jeweils in der angegebenen Textdatei und geben Sie diese ab.

`prozesse.txt` Liste aller gerade laufenden Prozess auf dem System. Mit vollständiger Ausgabe der Kommandozeile. Sortiert nach Benutzername

`dateien.txt` Liste aller Dateien in `/gbs/aufgaben/a1/testdir` (auf dem gbs-Server) mit ihren Dateiattributen und Modifikationszeiten.

`print.txt` Liste aller Zeilen (mit Dateinamen und Zeilennummer), die *printf* enthalten. Gesucht wird in allen Dateien in `/gbs/aufgaben/a1/testdir`

`kill.txt` Starten Sie das Shell-Skript `/gbs/aufgaben/a1/endlos_stern.sh` im Hintergrund, finden Sie dessen PID heraus und beenden Sie es. Die Textdatei soll den Verlauf der Kommandozeile beinhalten.

Hinweise zur Aufgabe:

- Erforderliche Dateien: PARTNER, antworten.txt, hello.c, befehle.txt, prozesse.txt, dateien.txt, print.txt, kill.txt
- Hilfreiche *Manual-Pages*: **getuid(2)**, **getgid(2)**
- Hilfreiche Tools: **script(1)**, **ps(1)**, **ls(1)**, **grep(1)**, **kill(1)**, **bash(1)**
- Das C-Programm ist in der angegebenen C-Datei abzulegen. Es muss dem ANSI-C11-Standard entsprechen und mit dem GNU-C-Compiler auf dem Linux-Abgaberechner kompilieren. Dazu ist der Compiler mit folgenden Parametern aufzurufen.
  `gcc -std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror -o hello hello.c`
- Das Makefile bietet jeweils die passenden Compileraufrufe
- Sollte bei der Ausführung einer verwendeten Funktion (z.B. `getuid(2)`) ein Fehler auftreten, sind keine Fehlermeldungen auszugeben – der Fehler muss aber gemäß der Aufgabenstellung sinnvoll behandelt werden.
- Unterprogramme und globale Variablendefinitionen sind ausreichend zu kommentieren. Achten Sie bitte außerdem auf saubere Gliederung des Quellcodes!

1.7 Verkettete Liste (lilo) (4P)

Implementieren Sie eine einfach verkettete Liste, welche nicht-negative Ganzzahlen verwaltet. Auf die Liste soll mit den folgenden Funktionen zugegriffen werden:

- `int list_append(list_t *list, int value)`: Fügt einen Wert in die Liste ein, wenn dieser noch nicht vorhanden ist. Im Erfolgsfall gibt die Funktion den eingefügten Wert zurück, ansonsten den Wert -1.
- `int list_pop(list_t * list)`: Entnimmt den ältesten Wert aus der Liste und gibt diesen zurück. Ist kein Wert in der Liste vorhanden, wird -1 zurückgeliefert.

Im Repository finden Sie eine Vorlage für die Quelldatei. Das darin enthaltende Hauptprogramm (Funktion `main()`) fügt einige Werte in die Liste ein und entnimmt diese wieder. Die Codesequenz aus der Vorlage soll folgende Ausgabe erzeugen:

```
insert 47: 47
insert 11: 11
insert 23: 23
insert 11: -1
remove: 47
remove: 11
```

Hinweise zur Lilo:

- Hilfreiche *Manual-Pages*: **malloc(3)**, **free(3)**
- Implementieren Sie keine Listenfunktionalität in der Funktion `main()`. Allerdings können Sie die Funktion `main()` erweitern, um Ihre Implementierung zu testen.
- Der Versuch, eine negative Zahl in die Liste einzufügen, soll unterbunden und als Fehler gewertet werden.
- Sollte bei der Ausführung einer verwendeten Funktion (z.B. **malloc(3)**) ein Fehler auftreten, sind keine Fehlermeldungen auszugeben – der Fehler muss aber gemäß der Aufgabenstellung sinnvoll behandelt werden.