Sophia Goldschmidt (866170)

25. Januar 2025

B-BSRAC00XX - Betriebssysteme und Rechnerarchitektur

Inhaltsverzeichnis

1 Einleitung

Die Aufgaben wurden vollständig auf einem macOS-System mit Apple-Silicon-Prozessor (M1) bearbeitet. Für die praktische Umsetzung kam eine in UTM virtualisierte Debian-12-Umgebung zum Einsatz.

Alle beschriebenen Schritte, Konfigurationen und Tests wurden direkt in dieser Linux-Instanz durchgeführt, sodass die Ergebnisse reproduzierbar und unabhängig vom Host-System nachvollziehbar sind.

2 Aufgabe 1

2.1 1a iproute2

Für die Ermittlung der IP-Adresse wurde der Befehl *ip addr show* verwendet. Dieser Befehl gehört zur modernen iproute2-Werkzeugfamilie und gilt heute als Standard unter Linux.

Im Vergleich zu älteren Tools wie *ifconfig* bietet er eine vollständigere und präzisere Darstellung der Netzwerk informationen, einschließlich IPv4-, IPv6-Adressen, Status der Schnittstellen und weiterer relevanter Parameter. Da aktuelle Linux-Distributionen iproute2 standardmäßig einsetzen und *ifconfig* häufig nicht mehr vorinstalliert ist, ist *ip addr show* die zuverlässigere und zukunftssichere Methode.[3]

Die für diese Aufgabe verwendeten Befehle sind in der Datei `history_866170_1a.txt` im Ordner `anhang` dokumentiert.

2.2 1b Benutzererstellung

Zur Einrichtung des Benutzers `mitarbeiter` wurden folgende Schritte durchgeführt:

1. **Anlegen des Benutzers mit Home-Verzeichnis und Bash-Shell**

   Der Benutzer wird mit dem Home-Verzeichnis `/home/wbh` und der Bash als Login-Shell erstellt:

   ```
   sudo useradd -m -d /home/wbh -s /bin/bash mitarbeiter
   ```

2. **Setzen des Passworts**

   Das Passwort wird auf `GeHeim` gesetzt:

   ```
   echo "mitarbeiter:GeHeim" | sudo chpasswd
   ```

3. **Hinzufügen zur Administratorengruppe**

   Der Benutzer wird der sudo-Gruppe hinzugefügt:

   ```
   sudo usermod -aG sudo mitarbeiter
   ```

4. **Überprüfung der Einrichtung**

   Es wird kontrolliert, ob der Benutzer korrekt erstellt wurde, die Gruppen stimmen und das Home-Verzeichnis existiert:

   ```
   grep mitarbeiter /etc/passwd
   groups mitarbeiter
   ls -ld /home/wbh
   ```

Diese Schritte stellen sicher, dass der Benutzer `mitarbeiter` ein korrektes Home-Verzeichnis besitzt, die Bash als Standard-Shell verwendet und über funktionierende sudo-Rechte verfügt.

Die für diese Aufgabe verwendeten Befehle sind in der Datei `history_866170_1b.txt` im Ordner anhang dokumentiert.

2.3 1c Berechtigungsmanagement

Im Rahmen dieser Aufgabe wird ein gemeinsamer Arbeitsbereich eingerichtet, die Passwortänderung für normale Benutzer deaktiviert und eine Datei `chat.txt` so konfiguriert, dass sie über Hardlinks von allen Gruppenmitgliedern gemeinsam genutzt wird.

1. **Numerische Rechtevergabe**

   Numerische Rechte setzen immer den vollständigen Rechtssatz einer Datei oder eines Verzeichnisses. Dadurch sind sie eindeutig und unabhängig vom vorherigen Zustand. Symbolische Angaben verändern dagegen nur einzelne Rechte. Daher werden in dieser Aufgabe möglichst numerische Rechte verwendet.

2. **Gemeinsames Gruppenverzeichnis /home/teamwork**

   Das Verzeichnis `/home/teamwork` dient als gemeinsamer Arbeitsbereich der Gruppe `wbh`. Damit alle Gruppenmitglieder darin arbeiten können und neue Dateien automatisch dieselbe Gruppe erhalten, wird das *setgid*-Bit gesetzt.

   ```
   sudo mkdir /home/teamwork
   sudo groupadd wbh
   sudo chgrp wbh /home/teamwork
   sudo chmod 2770 /home/teamwork
   ```

Das führende 2-Bit im Modus `2770` aktiviert das setgid-Bit. Dadurch erben alle neu erstellten Dateien automatisch die Gruppe `wbh`.

3. **Passwortänderung für Benutzer deaktivieren**

   Benutzer sollen ihre Passwörter nicht selbst ändern können. Dazu wird das setuid-Bit der Datei `/usr/bin/passwd` entfernt. Ohne setuid kann das Programm nicht länger mit Root-Rechten ausgeführt werden.

   ```
   sudo chmod u-s /usr/bin/passwd
   ```

   Damit kann nur noch der Administrator Passwörter ändern. Alle anderen Benutzer verlieren diese Möglichkeit, wie in der Aufgabe gefordert.

4. **Gemeinsame Datei chat.txt mit identischem Inode**

   Die Datei `chat.txt` soll von mehreren Benutzern gemeinsam genutzt werden. Damit alle auf dieselbe Datei zugreifen, wird sie einmal erstellt und dann für jeden Benutzer über einen Hardlink eingebunden. Ein Hardlink verweist auf denselben Inode und somit auf denselben Dateiinhalt.

   ```
   sudo touch /home/teamwork/chat.txt
   sudo chmod 664 /home/teamwork/chat.txt
   sudo chgrp wbh /home/teamwork/chat.txt


   sudo ln /home/teamwork/chat.txt /home/wbhuser/chat.txt
   ```

   Änderungen am Inhalt wirken sich sofort auf alle Hardlinks aus.

5. **Zusammenfassung**

   Die Konfiguration gewährleistet eine konsistente Zusammenarbeit in der Gruppe `wbh`:

   - numerische Rechte sorgen für klare und reproduzierbare Berechtigungen,
   - das setgid-Verzeichnis regelt die automatische Gruppenerbung,
   - die Deaktivierung des setuid-Bits verhindert eigenständige Passwortänderungen,
   - Hardlinks ermöglichen das gemeinsame Bearbeiten derselben Datei über denselben Inode.

Die für diese Aufgabe verwendeten Befehle sind in der Datei `history_866170_1c.txt` im Ordner `anhang` dokumentiert.

2.4 1d Berechtigungsmanagement

Im Rahmen dieser Aufgabe sollen bestimmte Berechtigungen für die Benutzer konfiguriert werden. Dabei gilt, dass jeder Student ausschließlich auf die Verzeichnisse der von ihm belegten Fächer Vollzugriff (Lesen, Schreiben, Ausführen) erhält. Dateien anderer Studierender dürfen gelesen, jedoch nicht gelöscht oder verändert werden, da das Löschen ausschließlich dem jeweiligen Eigentümer erlaubt ist. Zusätzlich ist sicherzustellen, dass neu angelegte Dateien und Verzeichnisse automatisch dieselben Zugriffsrechte übernehmen, wofür Default-ACLs eingesetzt werden.[5]

1. **Vorbereitung der Verzeichnisse**

   Zunächst werden die erforderlichen Verzeichnisse für die verschiedenen Fachbereiche erstellt. Dazu gehören die Verzeichnisse `/home/fach/betriebssysteme`, `/home/fach/rechnerar` und `/home/fach/softwareentwicklung`. Diese werden mit den entsprechenden Rechten versehen.

   ```
   sudo mkdir -p /home/fach/{betriebssysteme,rechnerarchitektur,softwareentwickl
   ```

2. **Rechtevergabe auf Verzeichnisse**

   Die Fachverzeichnisse werden vollständig dem Systemadministrator zugeordnet und anschließend restriktiv abgesichert. Durch das Setzen des Eigentümers und der Gruppe auf `root` liegt die administrative Kontrolle ausschließlich bei root. Die Rechte `700` stellen sicher, dass nur der Besitzer Lese-, Schreib- und Ausführungsrechte besitzt, während alle anderen Benutzer keinerlei Zugriff haben. Der Zugriff für Studierende wird später gezielt über Access Control Lists (ACLs) gewährt.

   ```
   sudo chown root:root /home/fach/*
   sudo chmod 700 /home/fach/*
   ```

3. **Erstellen und Konfigurieren der Test-Benutzer**

   Die Benutzer `student1` und `student2` werden angelegt und erhalten ein Passwort.

   ```
   sudo useradd -m -s /bin/bash student1
   sudo useradd -m -s /bin/bash student2
   sudo passwd student1
   sudo passwd student2
   ```

4. **Zugriffssteuerung mit ACLs**

   Für die Fachbereiche wird mithilfe von Access Control Lists (ACLs) sichergestellt, dass jeder Student nur auf die Verzeichnisse seiner belegten Fächer Zugriff hat.

   ```
   sudo setfacl -m u:student1:rwx /home/fach/betriebssysteme
   ```

```
sudo setfacl -m u:student1:rwx /home/fach/softwareentwicklung
sudo setfacl -m u:student2:rwx /home/fach/betriebssysteme
```

Kurze Aufschlüsselung des Befehls:

- `sudo`: Ausführung des Befehls mit administrativen Rechten.
- `setfacl`: Werkzeug zum Setzen und Verwalten von Access Control Lists.
- `-m`: *modify*, fügt eine neue ACL hinzu oder ändert eine bestehende.
- `u:student1:rwx`: ACL-Eintrag für den Benutzer (*user*) `student1` mit Lese-, Schreib- und Ausführungsrechten.
- `/home/fach/betriebssysteme`: Zielverzeichnis, auf das sich die ACL bezieht.

5. **Setzen von Default-ACLs und expliziten ACLs**

   Mit den folgenden Befehlen werden sowohl effektive Zugriffsrechte als auch Default-ACLs für das Verzeichnis `/home/fach/betriebssysteme` definiert. Dadurch wird sichergestellt, dass bestehende Zugriffe klar geregelt sind und neu angelegte Dateien und Unterverzeichnisse automatisch die gewünschten Rechte erben.

   ```
   sudo setfacl -m d:u::rwX,d:g::r-X,d:o::--- /home/fach/betriebssysteme
   ```

   Dieser Befehl setzt **Default-ACLs**. Neue Dateien und Verzeichnisse erhalten damit für den Besitzer Lese- und Schreibrechte, für die Gruppe Leserechte und Ausführungsrechte nur bei Verzeichnissen, für andere Benutzer keinerlei Rechte.

   ```
   sudo setfacl -m u::rwx,g::rwx,o::--- /home/fach/betriebssysteme
   ```

   Hier werden die **effektiven ACLs** für das bestehende Verzeichnis festgelegt. Besitzer und Gruppe erhalten volle Zugriffsrechte, während allen anderen Benutzern der Zugriff vollständig verwehrt wird.

   ```
   sudo setfacl -m d:o::r-- /home/fach/betriebssysteme
   ```

   Dieser zusätzliche Default-ACL-Eintrag erweitert die Standardrechte für andere Benutzer auf reinen Lesezugriff bei neu erstellten Dateien, ohne Schreib- oder Ausführungsrechte zu vergeben.

   Alternative, konsistente Vergabe der ACLs

   Besser wäre es gewesen, die Zugriffsrechte von Anfang an in einem einzigen, konsistenten Schritt zu setzen. Dadurch lassen sich nachträgliche Korrekturen vermeiden und sowohl die effektiven Rechte als auch die Default-ACLs klar und nachvollziehbar definieren.

   ```
   sudo setfacl -m u::rwx,g::rwx,o::---,d:u::rwX,d:g::r-X,d:o::--- \
   /home/fach/betriebssysteme
   ```

Mit diesem Befehl erhalten der Besitzer und die Gruppe vollständige Zugriffsrechte auf das bestehende Verzeichnis, während anderen Benutzern der Zugriff verwehrt wird. Gleichzeitig wird über die Default-ACLs sichergestellt, dass neu angelegte Dateien und Unterverzeichnisse automatisch die vorgesehenen Berechtigungen übernehmen.

Bedeutung der einzelnen Bestandteile:

- `sudo`: Ausführung mit administrativen Rechten.
- `setfacl`: Setzt Access Control Lists.
- `-m`: *modify*, ergänzt oder ändert ACL-Einträge.
- `u::rwx`: Vollzugriff für den Besitzer des Verzeichnisses.
- `g::rwx`: Vollzugriff für die zugehörige Gruppe.
- `o::—`: Kein Zugriff für andere Benutzer.
- `d:u::rwX`: Default-ACL für den Besitzer, Schreib- und Leserechte, Ausführen nur bei Verzeichnissen.
- `d:g::r-X`: Default-ACL für die Gruppe, Lesen und Ausführen nur bei Verzeichnissen.
- `d:o::—`: Keine Standardrechte für andere Benutzer.
- `/home/fach/betriebssysteme`: Zielverzeichnis der ACL-Konfiguration.

Zur Kontrolle der vergebenen Zugriffsrechte wird der folgende Befehl verwendet:

```
sudo getfacl /home/fach/betriebssysteme
```

Dieser Befehl zeigt die aktuell gesetzten effektiven ACLs sowie die Default-ACLs des Verzeichnisses an. Anhand der Ausgabe lässt sich überprüfen, ob die gewünschten Berechtigungen korrekt gesetzt wurden und ob neu angelegte Dateien die vorgesehenen Standardrechte erben.

Setzen des Sticky-Bits

Mit dem folgenden Befehl wird auf dem Verzeichnis `/home/fach/betriebssysteme` das sogenannte *Sticky-Bit* gesetzt:

```
sudo chmod +t /home/fach/betriebssysteme
```

Das Sticky-Bit bewirkt, dass Dateien innerhalb dieses Verzeichnisses nur noch von ihrem jeweiligen Eigentümer, vom Verzeichniseigentümer (root) oder vom Systemadministrator gelöscht oder umbenannt werden können. Dadurch wird verhindert, dass Benutzer fremde Dateien entfernen, obwohl sie Schreibrechte auf das Verzeichnis besitzen. Dies erhöht die Datensicherheit in gemeinsam genutzten Arbeitsverzeichnissen.[1, 4]

Beispiel Ausgabe:

```
# file: home/fach/betriebssysteme
# owner: root
# group: root
# flags: --t
user::rwx
user:student1:rwx
user:student2:rwx
user:student3:rwx
group::rwx
mask::rwx
other::---
default:user::rwx
default:group::r-x
default:other::r--
```

6. **Test der gesetzten Berechtigungen**

   Zur Überprüfung der konfigurierten Zugriffsrechte wurden gezielt Aktionen mit unterschiedlichen Benutzern durchgeführt. Dabei wurde geprüft, ob das Erstellen, Lesen, Ändern und Löschen von Dateien gemäß den definierten ACLs und dem gesetzten Sticky-Bit korrekt eingeschränkt ist.

   ```
   sudo -u student1 touch /home/fach/betriebssysteme/datei1
   sudo -u student2 rm /home/fach/betriebssysteme/datei1
   sudo -u student1 bash -c 'echo "test" > /home/fach/betriebssysteme/datei1'
   sudo -u student2 bash -c 'echo "test" > /home/fach/betriebssysteme/datei1'
   sudo -u student2 cat /home/fach/betriebssysteme/datei1
   sudo -u student1 rm /home/fach/betriebssysteme/datei1
   ```

   Die Tests zeigen, dass `student1` als berechtigter Benutzer Dateien anlegen, verändern und löschen kann. `student2` darf die Datei lesen, ist jedoch aufgrund des Sticky-Bits nicht berechtigt, fremde Dateien zu löschen oder zu verändern. Damit verhalten sich die Zugriffsrechte wie vorgesehen.

Die für diese Aufgabe verwendeten Befehle sind in der Datei `history_866170_1d.txt` im Ordner anhang dokumentiert.

3 Aufgabe 2

3.1 Code

Das Script für diese Aufgebe befindet in dem Pfad `anhang\code`, als `kursverwaltung.sh`.

3.2 Test

Die für diese Aufgabe verwendeten Befehle zum testen und Screenshots der Ergebnisse sind in diesen Dateien, im Ordner `anhang` dokumentiert.

`history_866170_2a.txt`

`history_866170_2b.txt`

`history_866170_2c.txt`

`history_866170_2d.txt`

`history_866170_2d_berechtigungsprüfung.txt`

`screen_866170_2d_user_test.png`

`screen_866170_2e.png`

`screen_866170_2f.png`

4 Aufgabe 3

4.1 arm64 (Mac M1)

Beispiel Ausgabe, für Ausführung auf MacOS, mit arm64:

```
Architektur erkannt: arm64
Passende Datei: script_arm64.sh
Lade herunter: script_arm64.sh
Quelle: http://192.168.64.3:80/script_arm64.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    72  100    72    0     0   6421      0 --:--:-- --:--:-- --:--:--  6545
Download abgeschlossen: script_arm64.sh
Fuehre aus: script_arm64.sh
Dieses Skript gehoert zur ARM64-Architektur.
Geloescht: script_arm64.sh
```

Die Scripte für diese Aufgebe befindet sich in dem Pfad

`anhang\code\aufgabe3`.

5 Aufgabe 4

5.1 Warum lässt sich die Prozesstabelle mit Bash nicht millisekundengenau ausführen und warum ist ein herkömmliches Linux nicht in der Lage, dies zu lösen?

Tabelle 1: Prozessablaufplan *(eigene Darstellung)*

Prozess	Laufzeit [ms]	Ankunftszeit [ms]
A	4000	0
B	3000	2000
C	3000	4000
D	5000	7000
E	5000	9000

Mit Bash ist eine millisekundengenaue Ausführung nicht möglich, weil Bash kein Echtzeit-System ist, keine deterministische Zeitplanung besitzt und die Ausführung von Prozessen vollständig vom Linux-Scheduler abhängt.

Der Linux-Scheduler ist dafür zuständig, auf einer CPU mehrere Prozesse scheinbar parallel auszuführen. Er arbeitet präemptiv, das heißt, ein laufender Prozess kann jederzeit unterbrochen werden, wenn der Scheduler entscheidet, dass ein anderer Prozess die CPU erhalten soll.

Der Scheduler vergibt die CPU in Zeitscheiben. Diese Zeitscheiben sind nicht fest auf exakt 1 ms definiert, sondern hängen vom Kernel, vom internen Zeitgeber und vom aktuellen Scheduling-Algorithmus ab. Moderne Linux-Systeme verwenden den Completely Fair Scheduler (CFS)[2]. Dieser versucht Fairness, nicht Zeitgenauigkeit, zu erreichen: Jeder Prozess soll proportional zu seiner Gewichtung (Nice-Wert) CPU-Zeit bekommen.

Der CFS misst dafür die virtuelle Laufzeit eines Prozesses und bevorzugt jeweils den Prozess, der relativ gesehen am wenigsten CPU erhalten hat. Dadurch entstehen dynamische Umschaltzeitpunkte, keine festen Start- oder Endzeiten.

Zusätzlich kommen unvermeidbare Effekte hinzu, wie Timer-Interrupts, die den Scheduler aufrufen oder Kontextwechsel, bei denen Register und Speicherzustände gesichert und restauriert werden.

Diese Vorgänge kosten Zeit und verschieben reale Ausführungszeitpunkte. Deshalb enden Prozesse nicht exakt bei 4000 ms, sondern z. B. bei 4018 ms. Die Abweichung ist systembedingt und nicht kontrollierbar.

5.2 Welcher Betriebssystem-Typ wäre in der Lage, die Echtzeitanforderung zu erfüllen?

Ein RTOS garantiert deterministische Reaktionszeiten und zeitlich vorhersagbares Scheduling. Prozesse erhalten die CPU strikt nach Priorität oder festen zeitlichen Vorgaben, wodurch maximale Latenzen bekannt und begrenzt sind.

Beispiele für geeignete Betriebssysteme sind:

- RIOT[6]
- Rodos[7]
- Zephyr[8]

5.3 Erweiterung der Tabelle 1: Prozessablaufplan

Vergleich der Soll und Ist Zeiten:

```
[server.sh] A | 4000 | 0000 | 4021 | 0038
[server.sh] B | 3000 | 2000 | 3024 | 2028
[server.sh] C | 3000 | 4000 | 3024 | 4030
[server.sh] D | 5000 | 7000 | 5025 | 7024
[server.sh] E | 5000 | 9000 | 5023 | 9028
```

Die gesammte Ausgabe für diese Aufgabe, ist in der Datei `out_866170_4d.txt` im Ordner anhang dokumentiert.

Die Scripte für diese Aufgebe befindet sich in dem Pfad

anhang\code, als `server.sh`

und

anhang\code, als `prozess.sh`.

Literatur

[1] Ankit Jain. *Linux Sticky Bit*, 2020. baeldung.com, abgerufen am 09.12.2025.

[2] Linux Kernel Documentation. *Scheduling – CFS Scheduler Design*, 2024. docs.kernel.org, abgerufen am 25.01.2025.

[3] Ralf Spenneberg. Schwarm auf der flucht – iproute2 ersetzt die alten net-tools, 2006. linux-magazin.de, abgerufen am 08.12.2025.

[4] Tyler Carrigan. *Understanding SUID, SGID, and the Sticky Bit*, 2024. redhat.com, abgerufen am 09.12.2025.

[5] Vito_Leone / ubuntuusers Wiki. *Access Control Lists (ACL)*, 2025. wiki.ubuntuusers.de, abgerufen am 09.12.2025.

[6] Wikipedia-Autoren. *RIOT (Betriebssystem)*, 2025. wikipedia.org, abgerufen am 25.01.2025.

[7] Wikipedia-Autoren. *RODOS (Betriebssystem)*, 2025. wikipedia.org, abgerufen am 25.01.2025.

[8] Wikipedia-Autoren. *Zephyr (Betriebssystem)*, 2025. wikipedia.org, abgerufen am 25.01.2025.