Betriebssysteme
Aufgabenblatt 2 „Prozesse und Dienste“

Prof. Dr.-Ing. Christian Lins

Version 1.1/2025-11-11

Aufgabe

Mit diesem Praktikumsblatt vertiefen wir den Umgang mit Prozessen und Threads auf Unix-Systemen und schauen uns an, wie wir mit einem Linux-System umgehen und das System administrieren.

Lernziele[1]

- Administrative Aufgaben auf einem Linux-System durchführen können.
- Prozesserzeugung in Unix-/Linux-Systemen verstehen und anwenden.
- Lösungsansätze für das Consumer-Producer-Problem anwenden.

Literaturhinweise

Zur Vorbereitung sollte neben den Vorlesungsunterlagen gelesen werden:

- Kapitel 3.2 „Systemaufrufe“ aus dem Buch *Grundkurs Betriebssysteme* von Peter Mandl.
- Kapitel 4 „Prozesse und Threads“ aus dem Buch *Grundkurs Betriebssysteme* von Peter Mandl.
- Kapitel 6.4 „Synchronisationsmechanismen in Java“ aus dem Buch *Grundkurs Betriebssysteme* von Peter Mandl.

Hinweis zu Shellbefehlen

In den Praktikumsaufgaben gebe ich manchmal Shell-Kommandos an, z. B.

```
$ echo "Hallo Welt"
# apt install gcc
```

Achten Sie hierbei immer auf das erste Zeichen einer Zeile: **#** steht für eine Shell mit root-Rechten, **$** für eine Shell mit Benutzerrechten. Manchmal wird *vor* dem Zeichen noch ein bestimmtes Verzeichnis angegeben, falls dies relevant ist.

1 Debian-Linux einrichten

Im letzten Aufgabenblatt sollten Sie Debian-Linux in einer virtuellen Maschine installieren (oder haben eine VM zur Verfügung gestellt bekommen). Wir wollen das System nun richtig in Betrieb nehmen und kennenlernen.

[1] Kompetenzniveau: Anwenden

1.1 Nutzeraccount einrichten

Möglicherweise wurde für Ihre VM noch kein Standardbenutzer eingerichtet, sondern nur der Adminstrationsbenutzer root. In diesem Fall müssen Sie einen Benutzeraccount einrichten, den Sie für die tägliche Arbeit benutzen können.

Dazu verwenden Sie als root as Programm `adduser`, das für Sie einen Benutzer anlegen kann, nachdem es Schritt für Schritt Informationen abgefragt hat. Nachträglich können Sie das Passwort eines Accounts mit `passwd` setzen. Das sollten Sie für den root-Benutzer auch tun.

Mit `# su - benutzername` können Sie in eine Shell mit dem angelegten Benutzer wechseln und dort arbeiten, z. B. Programme kompilieren. Mit `exit` verlassen Sie die Benutzershell.

1.2 System-Updates und Software installieren

Die meisten Linux- und Unix-Systeme verfügen über sogenannte Paketquellen, über die sich das System und die darauf installierte Software aktuell halten lässt und sich auch Software bequem nachinstallieren lässt. Debian und alle darauf basierenden Linux-Distributionen wie Ubuntu oder Mint verwenden das Werkzeug apt dafür:

```
// Aktualisieren der Paketquellen
# apt update

// Aktualisieren alle installierten Pakete
# apt upgrade

// Installieren eines Paketes mein-paket
# apt install mein-paket
```

Informieren Sie sich in der Manpage von apt über weitere Subkommandos des Programms. Unter `https://packages.debian.org/` können Sie die Paketlisten bequem im Netz durchsuchen, falls der Paketname nicht gleich dem Programmnamen ist (oft ist das aber so).

Nötige Pakete Installieren Sie nun den gcc (Paket: build-essential) und die Java-Laufzeitumgebung (Paket: openjdk-21-jdk-headless) sowie Git (Paket: git). Falls Sie auf der Konsole auch Quelltext oder andere Textdateien bearbeiten können, benötigen Sie einen Editor, z. B. neovim oder nano, die sich auch über apt installieren lassen. Damit sollten Sie das nötige Werkzeug für die anderen Aufgaben auf der VM installiert haben.

Optionale Pakete Falls Sie bereits über eine VM mit grafischer Oberfläche verfügen, können Sie den Editor `kate` nachinstallieren. Eine grafische Oberfläche lässt sich über Pakete nachinstallieren (lxde, xorg und lightdm), zumindest bei den VMs, die Sie lokale auf Ihrem Rechner starten. Bei den VMs, zu denen wir nur per SSH Kontakt aufnehmen, können wir diese Pakete zwar installieren, sehen können wir die Oberfläche dann aber nicht.

2 Systemdienst ClipboardService

2.1 Allgemein

Wie Sie aus dem Seminar wissen, startet der Betriebssystem-Kern nach dem Bootvorgang, einen ersten Prozess (Init-Prozess), der die weitere Konfiguration des Betriebssystems übernimmt. In allen großen Linux-Distributionen[2] ist dieser erste Prozess der *System Daemon*, kurz systemd. Systemd startet insbesondere Systemdienste für die einzelnen Aufgaben, u. a. zur Verwaltung der Hardware wie Drucker oder des Netzwerks.

Wir können dann, wenn wir das System verwalten wollen, über Kommandos mit systemd interagieren. Alles was mit System-Diensten zu tun hat, können wir über das Programm systemctl erledigen:

[2] Systemd ist entgegen der verbreiteten Unix-Philosophie ein recht komplexes Stück Software mit vielen Aufgaben, das wegen der engen Verzahnung mit dem Kernel auch nur unter Linux lauffähig ist. Unix-System wie FreeBSD verwenden noch Varianten des klassischen Init-Systems.

```
// Einen Dienst aktivieren
# systemctl enable dienstname

// Einen Dienst starten
# systemctl start dienstname
# systemctl stop dienstname
```

Aufgabe: Finden Sie mit systemctl heraus wieviele Systemdienste (Tipp: units) auf Ihrer VM laufen.

Mit dem Kommando `journalctl` können wir uns die Ausgaben der Systemdienste anschauen, da systemd auch das Logging verwaltet.

2.2 Ein eigener Systemdienst

Einen eigenen Systemdienst zu erstellen, ist nicht weiter schwer. Letzlich ist es nur ein Prozess, der auf Anfragen (allg. Aufgaben) von Clienten wartet. Oft wird die Kommunikation über Sockets geregelt. Da dies bei Ihnen erst in einem nächsten Semester ansteht, habe ich Ihnen einen kleinen Systemdienst vorbereitet. Sie finden den Code unter `https://codeberg.org/chrlns/haw_bs_clipboard`.

Der bereitgestellte Systemdienst ClipboardService stellt auf dem Terminal eine Art Zwischenablage bereit, er kann also Texte entgegen nehmen und speichern und auf Anfrage wieder zurückliefern. Diesen Dienst wollen wir nun im System einbinden.

Checken Sie den Quelltext mit Git aus und übersetzen Sie ihn:

```
~$ git clone https://codeberg.org/chrlns/haw_bs_clipboard.git
~$ cd haw_bs_clipboard
~/haw_bs_clipboard$ javac *.java
```

2.3 Den Dienst im System einbinden

Damit das System unseren Dienst kennt, müssen wir ihn über eine Service-Datei bekannt machen. Eine Vorlage dafür liegt dem ClipboardService bei, für die Standard-Debian-Systeme muss hier nichts angepasst werden (schauen Sie dennoch einmal hinein). Sie können das Verzeichnis daher einfach ins passende Systemverzeichnis kopieren. Unter Linux/Unix liegen Konfigurationsdateien für alle System-weit installierte Software in `/etc` und den entsprechenden Unterverzeichnissen. Kopieren Sie die clipboard@.service-Datei in /etc mit (als root/sudo):

```
# cp clipboard@.service /etc/systemd/system/
```

Nun müssen wir systemd mitteilen, dass wir etwas an der Konfiguration geändert haben:

```
# systemctl daemon-reload
```

Die Service-Datei trägt ein @ am Ende des Namens, daher weiß systemd, dass es sich um ein Template für einen User-Dienst handelt. Jeder User des Systems kann diesen Dienst nun für sich aktivieren und erhält eine eigene Instanz des Dienstes (also in unserem Fall eine eigene Zwischenablage).

Aktivieren Sie nun den Dienst für Ihren normalen Benutzer und starten Sie ihn:

```
$ systemctl enable clipboard@$(whoami).service
$ systemctl start clipboard@$(whoami).service
```

Prüfen Sie mit `systemctl status ...` ob der Dienst korrekt läuft.

Aufgabe: Finden Sie heraus unter welcher PID der Dienst läuft.

Aufgabe: Beenden Sie den Dienstprozess ohne systemctl. Wie ist der Status des Dienstes nach 1 Sek. und nach etwa 6 Sek.?

2.4 Den Dienst verwenden

Wenn nun der ClipboardService im Hintergrund läuft, können wir den Client nutzen, um Strings in der Zwischenablage zu speichern und wieder abzurufen:

```
$ java ClipboardClient --store "Hallo Welt"
OK: Text gespeichert.
$ java ClipboardClient --get
Hallo Welt
```

Aufgabe: Nutzen Sie Ihre Kenntnisse über die Shell und Shellskripte vom Aufgabenblatt 1, um die Nutzung des Dienstes wie folgt zu ermöglichen:

```
$ put "Hallo Welt"
$ get
Hallo Welt
```

Aufgabe: Denken Sie noch etwas weiter und überlegen Sie wie man eine solche Zwischenablage vielleicht ganz ohne Systemdienst nur mit Shellmitteln realisieren könnte.

3 Prozesserzeugung in UNIX (C-Programmierung)

3.1 Erstellen der HAW-Shell

Schreiben Sie ein C-Programm `hawsh`, das die Funktionalität einer (stark eingeschränkten) Shell besitzt. Die HAW-Shell soll dabei folgende Eigenschaften aufweisen:

1. Die Shell gibt einen Prompt-String aus, in dem das aktuelle Arbeitsverzeichnis und der Name des aktuellen Benutzers enthalten sind.
2. Der Benutzer kann danach den Namen eines in die Shell eingebauten Befehls (Liste s.u.) oder den Namen einer beliebigen Programmdatei eingeben (ohne Optionen und ohne Argumente, wer mag, kann das implementieren)
3. Die Shell interpretiert anschließend den angegebenen Befehl und führt ihn aus. Falls kein eingebauter Befehl erkannt wurde, erzeugt die Shell einen neuen Prozess und veranlasst das Laden der Programmdatei anhand des übergebenen Namens.
4. Falls das letzte Zeichen eines (Nicht-Builtin-)Befehls ein „&“ ist (ohne Leerzeichen als Zwischenraum!), wartet die Shell nicht auf die Beendigung des Befehls, sondern meldet sich sofort zurück (d. h. der Befehl wird im Hintergrund ausgeführt).
5. Weiter bei Punkt 1.

Es ist dafür zu sorgen, dass bei Fehlersituationen der Benutzer ausreichend informiert und ein stabiler Zustand erreicht wird.

Folgende eingebaute Befehle soll die Shell selbst bereitstellen:

Name des Befehls	Wirkung des Befehls
quit	Beenden der HAW-Shell
/ [Pfadname]	Wechsel des aktuellen Arbeitsverzeichnisses (analog zu cd). Es muss immer ein kompletter Pfadname eingegeben werden.

Beispiel-Dialog (Systemrückmeldungen sind nur Beispiele):

```
$ hawsh
/home/Franz - Was willst du, Franz? ps
4 7 0 2 ... bash
4 7 1 8 ... hawsh

/home/Franz - Was willst du, Franz? ps&
```

```
/home/Franz - Was willst du, Franz?
4702 ..... bash
4718 ..... hawsh
/home/Franz - Was willst du, Franz? /home/Franz/BS
Neues Arbeitsverzeichnis: /home/Franz/BS
/home/Franz/BS - Was willst du, Franz? quit
... und tschüss!
$
```

3.2 Testen der HAW-Shell

Testen Sie Ihr hawsh-Programm bzgl. aller eingebauten Befehle, der UNIX-Befehle date, ls und env sowie eines nicht existierenden Befehls (z.B. abcde) und zwar jeweils mit Ausführung im Vordergrund und im Hintergrund (&)!

Tipps:

- Zur Analyse und Fehlersuche können Sie auch geeignete Linux-Befehle vom Aufgabenblatt 1 anwenden.
- Verwenden Sie den Beispielcode aus der Vorlesungsfolien als Vorlage!
- Folgende Bibliotheksfunktionen sind zur Lösung hilfreich (Dokumentation über man-Befehl): getenv, setenv, chdir, getcwd, strcmp, strlen, fork, waitpid, execlp, exit (system darf nicht benutzt werden!)
- Prüfen Sie immer die Rückgabewerte der Systemfunktionen!

4 Threads mit Java

In dieser Übung implementieren Sie ein **Druckerspooler-System**, welches das **Producer-Consumer-Problem** demonstriert. Mehrere **Druckaufträge** werden an eine Warteschlange (Spooler) mit der maximalen Kapazität 5 gesendet, während ein oder mehrere **Drucker** diese Aufträge abarbeiten. Die Warteschlange hat eine begrenzte Kapazität und die Drucker dürfen nur dann Aufträge entnehmen, wenn welche vorhanden sind. Es dürfen nur dann dann neue Aufträge hinzufügt werden, wenn die Warteschlange nicht voll ist. Eine Überwachungskomponenten (PrinterMonitor) soll alle paar Sekunden den Füllstand der Warteschlange ausgeben.

4.1 Aufgabe

Bilden Sie das oben skizzierte Szenario auf das Producer-Consumer-Problem ab. Welche Komponenten sind Konsumenten, welche Produzenten?

Implementieren Sie die Komponenten in Java in ein komplettes Programm. Welche Komponenten müssen als Threads laufen?

Achten Sie auf die saubere Synchronisation zwischen den Komponenten.

Beispielausgabe

```
[PrintJobGenerator 1] Fügt hinzu: Druckauftrag 1 (Dokument1.pdf, 3 Seiten)
[PrinterSpooler] Warteschlange: 1/5
[Printer 1] Druckt: Druckauftrag 1 (Dokument1.pdf, 3 Seiten)
[PrinterSpooler] Warteschlange: 0/5
[PrintJobGenerator 2] Fügt hinzu: Druckauftrag 2 (Dokument2.pdf, 5 Seiten)
[PrinterSpooler] Warteschlange: 1/5
[SpoolerMonitor] Aktuelle Warteschlange: 1/5
[Printer 1] Druckt: Druckauftrag 2 (Dokument2.pdf, 5 Seiten)
[PrinterSpooler] Warteschlange: 0/5
...
```

4.2 Implementierungshinweise

- Nutzen Sie `synchronized`-Blöcke.
- Verwenden Sie `wait()` und `notifyAll()` für die Synchronisation zwischen Producern und Consumern.
- Simulieren Sie die Druckzeit z. B. mit `Thread.sleep(random.nextInt(3000) + 2000)` (2–5 Sekunden).
- Das Programm soll nach **20 Sekunden** automatisch beendet werden.
- Beenden Sie alle Threads sauber mit `interrupt()`.