Dokumentation
Betriebssysteme Projekt 2

Kommandozeile
Option A

Oliver Böhm 210204535
Matthias Stecker 210204441
André Wulf 210204710

Gliederung

1. Einleitung

1.1. Aufgabenstellung

In dem 2. Projekt des Praktikums Betriebssysteme, ist das Ziel eine Kommandozeile anzufertigen. Diese wird hauptsächliche über die Konsole gesteuert. Wichtige Aufgaben in dem Projekt sind, sich über die Prozessverwaltung (Erzeugung, Kontrolle und Terminierung) oder die Interprozesskommunikation (Signale, Sockets und Pipes) zu informieren und dies zu implementieren.
Die einzelnen Aufgaben spiegeln sich in der Kommandozeile wieder. Es soll eine Eingabeaufforderung mit Eingabebearbeitung und Befehlshistorie erstellt werden. Des Weiteren soll das Ausführen von Befehlen mit beliebigen Argumenten erfolgen. Wenn nun mehrere Befehle hintereinander geschrieben werden, soll dies durch trennen von sequenziellen Befehlen mit einem Semikolon („ ; “) geschehen. Die Befehle werden standesgemäß im Vordergrund ausgeführt. Wenn die Befehlseingabe beendet werden soll, soll dies mithilfe eines Und-Zeichen („&“) geschehen. Nun soll der Befehl aber in den Hintergrund wechseln. Durch die Tastenkombination „Strg + C“ sollen die Vordergrundprozesse terminieren. Dies sollte aber keinen Einfluss auf die Kommandozeile und die Hintergrundprozesse haben. Bei der Kommandozeile ist es ebenfalls wichtig, dass Signale abgefangen und mit entsprechenden Nachrichten ausgegeben werden. Wenn nun ein Syntaxfehler in der Eingabe vorliegt, soll außerdem eine aussagekräftige Fehlermeldung erscheinen. Die Eingabe und Ausgabe (I/O) soll umgeleitet werden in eine Datei. Hierbei ist zu beachten, dass dies beim Einlesen mittels Kleiner-als-Zeichen (<) geschehen soll und beim Ausgeben mittels Größer-als-Zeichen (>). Weitere wichtige Befehle in der Kommandozeile sind die Befehle „exit“, dieser soll die Konsole beenden und „cd *Pfad*“, dieser soll das Arbeitsverzeichnis ändern. Die Umgebungsvariablen werden wie folgt definiert: eine Variable wird gesetzt mittels „setenv *Variablen Wert*“, das Löschen einer Variable mittels „unsetenv *Variable*“ und das ersetzen eines Variablenwerts mittels „$ *Variable*“ oder „${*Variable*}.
Des Weiteren ist es verlangt, dass wir eine der zwei Optionen wählen, wir haben uns für die Option A entschieden. Hierbei sollen wir zusätzlich die Möglichkeit geben, dass der Nutzer Befehle in der Konsole durch das Zeichen „|“ verknüpfen kann. Außerdem sollen Pipes im Vorder- und Hintergrund ausgeführt werden.

1.2. Nutzen des Programms

Die fertige Konsole soll sich auf einen Linux Betriebssystem ausführen und bedienen lassen. Sie dient jedoch hauptsächlich als Lernprojekt für uns, damit wir nach und nach in die C-Programmierung unter Linux einsteigen. Hierbei war der Schwerpunkt die Systemkommunikation, wie etwa Prozess und unterschiedliche Methoden der Prozesskommunikation. Die Konsole kann durchaus verwendet werden, jedoch enthält sie wenige Funktionen und kann mit dem Linux Terminal kaum mithalten.

2. Theorie

2.1. Kommandozeile

Die Kommandozeile ist ein Eingabebereich für die Steuerung einer Software. Wenn man von der Kommandozeile spricht, wird diese oft als Konsole oder Terminal bezeichnet. Diese läuft im Allgemeinen in einem Textmodus ab. Die Kommandozeile wird von einer Shell oder einem Kommandozeileninterpreter gesteuert und die entsprechenden Funktionen ausgeführt, welches von beiden in der jeweiligen Software vorkommt ist Betriebssystem abhängig.
Die Befehlen bzw. Kommandos kommen oftmals in einer Zeichenketten-Form vor. Diese werden über die Tastatur eingegeben. Bei vielen Konsolen ist es so, dass die Befehle auf englisch verfasst sind. Die Ausführung der Befehle wird meist direkt aus der Zeile durch zusätzlich angegebene Parameter gesteuert. Heute zu Tage ist es sehr unüblich, dass das Programm den Benutzer interaktiv befragt. Ein Kommandozeilenprogramm läuft somit typischerweise mit den gegebenen Parametern einmal ab, bevor eine erneute Befehlseingabe möglich ist. Ein automatisiertes Abarbeiten mehrerer Kommandos nennt man Stapelverarbeitung.

2.2. Shell

Die Shell ist eine Software, die den Benutzer in einer bestimmten Form mit dem Computer verbindet. Die typischen Aufgaben sind die Bereitstellung von Kerneldiensten oder das Anbieten von Oberflächen für Systemkomponenten. Es existieren 2 Arten von Shells, die Kommandozeile und die grafische Benutzeroberfläche.

2.3. Rolle in Prozessverwaltung

Zunächst sollten wir klären was die Prozessverwaltung ist, um danach auf die spezielle Rolle einzugeben und ihre Bedeutung.
Die Prozessverwaltung ist ein Bestandteil von Betriebssystemen. Diese ist für die ordnungsgemäße Verwaltung von mehreren Prozessen zuständig, die auf einem Computer ausgeführt werden. Man kann eine grobe Aufteilung in der Prozessverwaltung feststellen. Zum einen haben wir das “unterbrechende“ und „nicht unterbrechende“ Steuerprogramm. Nicht unterbrechende Steuerprogramme lassen einen Prozess, nachdem ihm die CPU einmal zugeteilt wurde, solange laufen, bis dieser diese von sich aus wieder freigibt oder bis er blockiert. Unterbrechende Steuerprogramme teilen die CPU von vornherein nur für eine bestimmte Zeitspanne zu und entziehen dem Prozess diese daraufhin wieder.
Das heißt die Prozessverwaltung spielt eine ganz wichtige Rolle. Denn ohne sie wäre kein vernünftiges arbeiten mit mehreren Prozessen möglich, was auch einen größeren Zeitaufwand bedeuten würde.

2.4. Interprozesskommunikation

Bei der Interprozesskommunikation (IPC) sind Methoden zum Informationsaustausch, informatisch gesprochen Datenübertragung, von nebenläufigen Prozessen oder Threads vorhanden. Im engeren Sinne versteht man unter IPC die Kommunikation zwischen Prozessen auf demselben Computer. Die Speicher ist hierbei aber strikt getrennt voneinander. Im weiteren Sinne bezeichnet IPC aber jeden Datenaustausch in Verteilten Systemen, angefangen bei Threads, die sich ein Laufzeitsystem teilen, bis hin zu Programmen, die auf unterschiedlichen Rechnern laufen und über ein Netzwerk kommunizieren.

2.5. Pipes und ihre Implementierung

In der Informatik beschreibt der Begriff „Pipe“ einen gepuffert uni- oder bidirektionalen Datenstrom zwischen zwei Prozessen nach dem First-In-First-Out Prinzip. Dies bedeutet, dass der Output des ersten Prozesses sofort als Input des zweiten dient. Falls mehrere Pipes

hintereinander stehen, wird dieses Muster fortgeführt: Der Output des zweiten Prozesses, welcher den Output des ersten als Input verwendete, wird als Input des dritten verwendet und so weiter.
Für die Implementierung wird dies mithilfe eines Filedescriptors gemacht. Standardmäßig ist für jedes Programm die Eingabe „stdin“ und die Ausgabe „stdout“. Diese müssen entsprechend umgeleitet werden. Der Filedescriptor gibt an, wohin die Ein- und Ausgabe geleitet wird. Unter Unix ist der Filedescriptor ein Array aus zwei Integer-Werten. Der eine Wert ist die Adresse der Eingabe und der andere die entsprechende Ausgabeadresse. Für das Programm muss entsprechend die eine Seite geschlossen werden, sowie die andere Seite mit dem Filedescriptor der Standardausgabe verbunden werden. Für das sequentiell nächste Programm muss entsprechend die Eingabeseite von der Standardeingabe auf den Filedescriptor gelegt werden.

2.6. Vorder- und Hintergrundprozesse

Der wesentliche Unterschied zwischen Vor- und Hintergrundprozessen besteht darin, dass bei Hintergrundprozessen der Nutzer die Konsole noch nutzen kann, um anderweitige Befehle auszuführen.
Wird die „execl()“ Funktion aufgerufen, wird der Prozess im Vordergrund ausgeführt.
Um die Funktion jedoch im Hintergrund auszuführen, muss ein neuer „Thread“ gestartet werden. Dies geschieht mittels „fork()“. Bei dem Funktionsaufruf wird der aktuelle Prozess gespiegelt. Dabei werden alle Argumente übergeben.
Das einzige was sich ändert ist die Prozess-ID, die für den Kindprozess erzeugt wird, welcher somit als eigenständiger Prozess vom Betriebssystem geführt wird. Durch den Rückgabewert der fork()-Methode kann daraufhin entschieden werden, welche Zweige im Eltern- und welche im Kindprozess ausgeführt werden.
So können die Background-Prozesse ausgeführt werden und das Hauptprogramm trotzdem weiterlaufen. Die Ausgabe, dass ein Background-Prozess fertig ist, erfolgt ebenfalls in diesen Zweigen. Die Kindprozesse müssen dann mittels der exit()-Funktion beendet werden, da die Prozesse sonst als sogenannte „Daemon-Prozesse“ im Hintergrund weiterlaufen.

2.7. Prozesswechsel

Bei einem System mit mehreren Prozessen konkurrieren diese miteinander. Bei der Prozessauswahl werden mehrere Aspekte berücksichtigt. Dies sind die wichtigsten:
1. Es sind Prozesse in eine Prioritätsliste eingetragen. Der Prozess mit der Besten Priorität bekommt den Prozessor zugesichert. Falls es jedoch mehrere derartige Prozesse gibt, kommt der dran, der am längsten auf der Prioritätsliste wartet.
2. Es gibt Restfristen für die Prozessantworten und diese sind abbildbar auf Prioritäten.
Bei der Durchführung des Prozesswechsels stellen sich mehrere Fragen: An welcher Stelle im Programmcode befindet sich die Ausführung? Wie ist die aktuelle Unterprogrammaufruffolge? Welche Werte haben die lokalen und globalen Variablen? Sind all diese Fragen beantwortet kann ein Prozesswechsel durchgeführt werden.

3. Nutzerhandbuch

Unsere Konsole ist jetzt in der Lage jegliche integrierte Befehle auszuführen.
Diese werden grundsätzlich in der Anfangszeile der Konsole eingegeben und mit der Entertaste bestätigt und damit ausgeführt.
Falls ein Befehl nicht korrekt eingegeben worden ist, erscheint eine Fehlermeldung. Mit `STRG+C` beenden Sie einen laufenden Prozess.
Mit Pfeiltaste-oben und -unten können Sie zwischen zuvor eingegeben Befehlen wechseln, da die Konsole für jede Eingabe eine sogenannte History anlegt (auch wenn die Eingabe nicht korrekt war). Eingaben innerhalb eines Befehls, die Sie mit Apostrophs eingrenzen oder einen Backslash voraus stellen (also zum Beispiel `'test'` oder `\test`), werden geschützt und nicht weiter ausgeführt. Ein Rautesymbol (`#`) erstellt einen Kommentar, was bedeutet, dass hinter diesem Zeichen stehende Eingaben bis zum Ende der Zeile ignoriert werden.
Außerdem können Sie mit Hilfe von Semikolons verschiedene Befehle hintereinander ausführen (also zum Beispiel `befehl1 ; befehl2`).

Auflistung der Befehle:

`exit`
Beendet die Konsole. Sie können diese erneut starten, indem Sie den erzeugten Makefile wieder ausführen.

***`cd`** (Change Directory)*
Wechselt das derzeit ausgewählte Verzeichnis. Standardmäßig ist dieses /home/USERNAME. Geben Sie ein gültiges Verzeichnis (getrennt von `cd` mit einer Leerzeile) ein. Stellen Sie sicher, dass die Verzeichnisse vor dem Wechseln existieren. Bei falscher Eingabe erscheint eine Fehlermeldung. `cd ..` wechselt in das übergeordnete Verzeichnis.

`setenv X Y / unsetenv X`
`Setenv` definiert eine Umgebungsvariable mit der in künftigen Prozessen gearbeitet werden kann. `X` ist hierbei der Name der Variabeln und `Y` ihr Wert. `unsetenv` entfernt den Wert wieder aus dieser Variable.

`&`-Befehl
Mit Hilfe eines `&`-Zeichens können Sie Prozesse, die bei einem normalen Start im Vordergrund laufen, im Hintergrund starten. Dies bedeutet, dass ein Prozess im Hintergrund weiterläuft, sie aber dennoch einen neuen Befehl ausführen können. Beachten Sie, dass `STRG+C` nur Vordergrundprozesse beenden kann.

`< datei`
Durch Verwendung von < mit Nachstellung einer Leerzeile und eines gültigen Textdateinamens innerhalb eines Befehls, wird der Input dieses Befehls aus dem angegebenen Textdokument gelesen.

`> datei`
Hier erfolgt die Verwendung anders herum, als obige Beschreibung: Das Ergebnis des Befehls wird in ein angegebenes Textdokument hineingeschrieben.

Pipes (|)
Sogenannte Pipes lassen sich mit Hilfe des Mittelstrichs (`|`) angeben. Hierbei handelt es sich um an eine Aneinanderreihung verschiedener Befehle. Das Ergebnis des ersten Befehls dient als Eingabe des zweiten, das des Zweiten wiederum als Eingabe des Dritten und so weiter. Der letzte Befehl gibt das Endergebnis aus.
Beispiel: (`befehl1 | befehl2 | befehl3`)