Ein Bild, das Logo enthält.

Automatisch generierte Beschreibung

Werkstück A

im Modul

Betriebssysteme und Rechnernetze

im Studiengang Wirtschaftsinformatik

Dokumentation

**Alternative 2**

vorgelegt von:

Tristan Buls 1440643

Yannis Körner 1432965

Daniel Schor 1435234

Benjamin Macholz 1459728

Prüfer: Christian Baun

Abgabetermin: 25.06.2023

Inhaltsverzeichnis

[Aufgabenbeschreibung 3](#_Toc138195759)

[Analyse 3](#_Toc138195760)

[**Programmiersprache C++** 3](#_Toc138195761)

[**Konzept** 3](#_Toc138195762)

[**Verwendung von /proc und Filedeskriptoren:** 3](#_Toc138195763)

[**Visualisierung der Prozesserzeugung und -verzweigung:** 4](#_Toc138195764)

[**Dateiverwaltung:** 4](#_Toc138195765)

[**Freigabe von Betriebssystemressourcen:** 4](#_Toc138195766)

[**Nutzung von C++-Bibliotheken:** 4](#_Toc138195767)

[Implementierung 5](#_Toc138195768)

[**int child()** 5](#_Toc138195769)

[**void date()** 5](#_Toc138195770)

[**void exec()** 7](#_Toc138195771)

[**void releasesResources()** 8](#_Toc138195772)

[**enum Optionen** 9](#_Toc138195773)

[**string OptionenToString()** 9](#_Toc138195774)

[**void menu()** 10](#_Toc138195775)

[**int input()** 10](#_Toc138195776)

[**string readFile()** 11](#_Toc138195777)

[**string dateTime()** 11](#_Toc138195778)

[**int writeFile()** 12](#_Toc138195779)

[**string procReq()** 13](#_Toc138195780)

[**vector<vector><string>> getStatData()** 14](#_Toc138195781)

[**vector<vector<string>> getStatmData()** 15](#_Toc138195782)

[**vector<vector<string>> getMapsData()** 16](#_Toc138195783)

[**string processInfoToString()** 17](#_Toc138195784)

[**void visualizeRelationship()** 18](#_Toc138195785)

[**int main()** 19](#_Toc138195786)

[Demonstration 21](#_Toc138195787)

[Fazit 23](#_Toc138195795)

# Aufgabenbeschreibung

Unsere Aufgabe besteht darin, ein Programm zu entwickeln, das Prozesse in Linux erzeugt und visualisiert. Wir dürfen die Programmiersprachen C/C++ nutzen. Dabei sollen Informationen wie Prozess ID, Rechte, UID, GID und RAM-Ausnutzung angezeigt und gespeichert werden. Das Programm soll zudem die Systemaufrufe fork() (um Prozesse zu erzeugen) und exec() (um Prozesse zu verzweigen) beinhalten. Bestimmte Grundfunktionen wie eine Menüauswahl, CLI-Ein-/Ausgabe, Dateispeicherung sowie eine Ressourcenfreigabe soll das Programm verfügen.

# Analyse

## **Programmiersprache C++**

C++ wurde als Programmiersprache gewählt, da sie objektorientierte Programmierung ermöglicht, über eine umfangreiche Standardbibliothek verfügt. Zudem haben wir im vorherigen Semester das Modul OOP absolviert, in diesem Modul haben wir ausführlich mit der Programmiersprache Java befasst die großen Ähnlichkeiten zu C++ aufweist. Darüber hinaus haben wir C++ gegenüber C bevorzugt, da C++ moderner ist.

Wir hatten keine vorherigen Erfahrungen mit der Programmiersprache C++. Die Tatsache, dass C++ eine große Entwicklergemeinschaft hat, war sehr hilfreich für uns, da wir dadurch auf eine Vielzahl von Lernressourcen zugreifen konnten. Viele Fragen, die wir im Verlauf dieses Projekts hatten, wurden bereits vielfach im Internet beantwortet oder behandelt.

## **Konzept**

### **Verwendung von /proc und Filedeskriptoren:**

Das Verzeichnis /proc wird genutzt, um auf Prozessinformationen des Betriebssystems zuzugreifen.

Filedeskriptoren wie /proc/$pid/stat werden verwendet, um spezifische Informationen über Prozesse abzurufen.

Die Informationen werden in entsprechenden Datenstrukturen gespeichert, um sie später verarbeiten und anzeigen zu können.

### **Visualisierung der Prozesserzeugung und -verzweigung:**

Die Beziehung zwischen fork() und exec() wird textuell dargestellt, um die Prozesserzeugung und -verzweigung zu visualisieren.

### **Dateiverwaltung:**

Das Programm ermöglicht das Schreiben von Prozessinformationen in einer Datei. Der Dateiname wird automatisch ermittelt aus dem aktuellen Datum und Uhrzeit. Zudem bietet es die Möglichkeit, Informationen aus Dateien zu lesen, indem der Dateiname eingegeben wird.

### **Freigabe von Betriebssystemressourcen:**

Bevor das Programm beendet wird, werden alle verwendeten Betriebssystemressourcen ordnungsgemäß freigegeben, um eine Überlastung des Systems zu vermeiden. Dazu durchläuft das Programm eine Liste von Prozess-IDs. Die Funktion wartet kurz, ob die Prozesse von selbst beendet werden. Falls dies der Fall ist, gibt die Funktion eine Meldung aus, dass der Prozess ordnungsgemäß beendet wurde, zusammen mit dem Exit-Status. Wenn jedoch ein Prozess sich nach einer kurzen Wartezeit nicht beendet, terminiert die Funktion den Prozess durch ein Signal und gibt dies im Terminal aus.

### **Nutzung von C++-Bibliotheken:**

Die Anwendung macht Gebrauch von relevanten C++-Bibliotheken, um z.B. auf Prozessinformationen zuzugreifen und Systemaufrufe wie fork() und exec() durchzuführen. Hierzu fanden wir die folgenden Bibliotheken sehr hilfreich:

1. <iostream>: Die <iostream>-Bibliothek ermöglicht die Ein- und Ausgabe von Daten über die Konsole.
2. <vector>: Der <vector> ist ein dynamisches Array in C++, das verwendet wird, um Elemente in variabler Größe zu speichern und zu verwalten.
3. <unistd.h>: <unistd.h> ist eine Header-Datei, die Funktionen für Betriebssystemaufrufe wie `fork()` und `exec()` enthält. Sie ermöglicht das Erzeugen neuer Prozesse und das Ausführen von Programmen.
4. <limits>: Die Bibliothek <limits> bietet Funktionen zur Behandlung von Fehlereingaben und Grenzwerten. Sie kann verwendet werden, um Eingaben auf bestimmte Bereiche zu beschränken oder auf Fehler zu überprüfen.
5. <sys/wait.h>: Die Header-Datei <sys/wait.h> enthält Funktionen zur Steuerung von Prozessen und zur Behandlung von Statusänderungen von Kindprozessen. Sie ermöglicht das Warten auf die Beendigung von Kindprozessen und das Abrufen von Informationen über deren Status.
6. <sys/syscall.h>: Die Header-Datei <sys/syscall.h> enthält Definitionen für Systemaufrufe auf der Plattformebene. Sie ermöglicht die Verwendung spezifischer Systemaufrufe im Code.
7. <fstream>: Die Bibliothek <fstream> ermöglicht das Lesen und Schreiben von Dateien. Sie wird verwendet, um Dateien zu öffnen, zu lesen oder zu schreiben.
8. <sstream>: Die Bibliothek <sstream> ermöglicht die Verarbeitung von Zeichenketten als Datenströme. Sie wird verwendet, um Zeichenketten in Arrays oder anderen Datentypen umzuwandeln.

# Implementierung

Der Code definiert zwei Konstanten LOG\_FOLDER und OPTION\_FOLDER, die den Namen der Ordner angeben, in denen Protokolldateien gespeichert werden. Das Programm nutzt auch Funktionen aus dem Verzeichnis “options“, dies ist z.B. wichtig bei der Ausgabe von “Hello World!“. Man kann dieses Verzeichnis auch noch beliebig erweitern mit anderen Funktionen.



## **int child()**

Die Funktion int child() erstellt einen neuen Kindprozess und fügt seine PID zum vector<pid\_t> prozesse hinzu. Der Rückgabewert gibt an, ob es sich um den Kindprozess oder den Elternprozess handelt.



## **void date()**

Diese Wrapper-Funktion void date() ersetzt den aktuellen Prozess durch das Programm "date" mithilfe der Funktion execl(). Dabei wird als erstes Argument der Pfad zum Ausführen des Programms angegeben. Das zweite Argument "date" ist der Name des Programms, das ausgeführt werden soll. Das dritte Argument "-u" gibt an, dass das Programm die Zeit in der koordinierten Weltzeit anzeigen soll. Das vierte und letzte Argument, NULL, dient als Marker, der das Ende der Argumentliste kennzeichnet.



## **void exec()**

Die Funktionen void exec(const char \*path) und void exec(string path) dienen als Wrapper-Funktionen für die “exec“-Familie von Funktionen. Sie ermöglichen das Starten einer ausführbaren Datei, wobei der Pfad entweder als const char\* oder als string Argument übergeben werden kann. Durch diese Überladung wird die Funktion flexibler, da sie es uns ermöglicht einen string zu erstellen/bauen ohne dass wir den genauen const char \*path benötigen.



## **void releasesResources()**

Die Funktion void releaseResources() beendet bzw. wartet auf die Beendigung aller laufenden Prozesse, die im vector<pid\_t> prozesse gespeichert sind. Sie gibt auch Informationen über den Beendigungsstatus der Prozesse aus.



## **enum Optionen**

Wir haben eine Enumeration namens "Optionen" eingeführt, die verschiedene Auswahlmöglichkeiten für das Benutzermenü definiert, wie z.B. "Datum ausgeben" oder "Prozessinformationen anzeigen". Diese Entscheidung basiert auf der Modularität, die eine Enumeration im Vergleich zu statischen Zeichenketten bietet. Durch die Verwendung einer Enumeration wird der Code lesbarer, sauberer, leichter verständlich, besser nachvollziehbar und ist leicht zu erweitern. Darüber hinaus ermöglicht die Kombination der Enumeration mit einem Switch-Case-Block eine einfache Handhabung und Organisation des Codes.



## **string OptionenToString()**

Die Funktion string OptionenToString() konvertiert eine Option in einen String, der im Benutzermenü angezeigt werden kann.

## **void menu()**

Diese Funktion wird zum Start des Programmes immer ausgeführt, um das Benutzermenü darzustellen. Falls das Terminal zu voll oder unübersichtlich wird, besteht die Möglichkeit, die Funktion void menu() im Benutzermenü aufzurufen. Diese Funktion löscht das Terminal zuerst mit dem Betriebssystem-Kommando "clear" und fügt dann Zeile für Zeile das Benutzermenü wieder ein. Dadurch entsteht erneut eine klar strukturierte Auswahlmöglichkeit wie beim Starten des Benutzermenüs.



## **int input()**

Die Funktion int input() erfasst die Eingabe des Benutzers für die gewünschte Funktion des Programms als Ganzzahl. Falls der Benutzer eine falsche Angabe tätigt von dem Optionsmenü wird die Eingabe gelöscht, ignoriert und der Wert auf 0 gesetzt.



## **string readFile()**

Die Funktion string readFile() liest den Inhalt einer Datei anhand des angegebenen Dateipfades. Zunächst wird geprüft, ob die Datei erfolgreich geöffnet werden konnte. Falls nicht, wird eine Fehlermeldung zurückgegeben. Anschließend wird die Datei zeilenweise ausgelesen und der Inhalt in einem String gespeichert. Am Ende wird die Datei geschlossen und der Inhalt zurückgegeben.



## **string dateTime()**

Die Funktion string dateTime() gibt das aktuelle Datum und die Uhrzeit als formatierten String zurück. Diese Funktion wird hauptsächlich verwendet, um den Namen der Logdateien zu generieren.



## **int writeFile()**

Die Funktion int writeFile() schreibt den angegebenen Text in eine Datei. Wenn kein Dateipfad angegeben ist, wird eine Protokolldatei mit der Bezeichnung die in der string dateTime() generiert wird, im Ordner LOG\_FOLDER erstellt. Der Modus “a“ (append) fügt den Text an die Datei an, während der Modus “w“ (write) die Datei überschreibt.



## **string procReq()**

Die Funktion string procReq() ist eine Wrapper-Funktion, die die Abfrage von Infromationen aus den /proc/[pid]/ -Dateien in Linux vereinfacht. Die Funktion nimmt als Argumente die Prozess-ID und den Namen der Datei und liest anhand dieser Informationen die Datei aus mit der string readFile() Funktion und gibt als Ergebnis eine Zeichenkette zurück, die den Inhalt der Abfrage des Programmes darstellt.



## **vector<vector><string>> getStatData()**

Die Funktion vector<vector<string>> getStatData() liest die Informationen von stat aller Prozesse ein und speichert sie in einem Vector. Das Ergebnis ist ein zweidimensionaler vector, wobei der Äußere vector beliebig viele innere vector speichert, welche Prozessinformationen für den jeweiligen Prozess enthalten.

Die Funktion getStatData() verwendet procReq(), um die Datei "stat" für jeden Prozess im Verzeichnis /proc abzurufen. Die Informationen werden dann in separate Wörter aufgeteilt und im inneren vektor gespeichert. Jeder vektor mit den Prozessinformationen wird dem äußeren vector stats hinzugefügt. Somit hat man zu jedem Prozess die ganzen Prozessinformationen gespeichert.

Die Größe der inneren Vektoren beträgt 44, da die “stat“-Datei 44 Spalten enthält.



## **vector<vector<string>> getStatmData()**

Der Ablauf dieser Funktion ist der gleiche wie bei der vector<vector><string>> getStatData(). Nur mit dem Unterschied das die Informationen über die RAM-Nutzung der Prozesse eingelesen werden und die Größe der inneren Vektoren nur Sieben beträgt.



## **vector<vector<string>> getMapsData()**

Der Ablauf dieser Funktion ist der gleiche wie bei der vector<vector<string>> getStatData(). Nur mit dem Unterschied das die Informationen über die Speicheradresse und Rechte aller Prozesse eingelesen werden und die Größe der inneren Vektoren nur Sechs beträgt.



## **string processInfoToString()**

Die Funktion string processInfoToString() erstellt eine formatierte Tabelle mit allen vorherigen Informationen aus den Funktionen getStatData(), getStatmData() und getMapsData().

In einer Schleife werden dann die einzelnen Zeilen der Tabelle erstellt, indem die relevanten Informationen aus den Vektoren extrahiert und in die Ausgabezeichenkette “output“ eingefügt werden. Am Ende wird die fertige Ausgabe zurückgegeben.



## **void visualizeRelationship()**

Die Funktion void visualizeRelationship() zeigt die Beziehung zwischen Eltern- und Kindprozessen an. Zunächst wird die Prozess-ID des Elternprozesses abgerufen. Dann wird über den vector<pid\_t> prozesse iteriert und für jede Prozess-ID überprüft, ob sie mit der Eltern-Prozess-ID übereinstimmt. Wenn ja, wird sie als Elternprozess ausgegeben, andernfalls als Kindprozess. Dadurch wird die Beziehung zwischen den Prozessen visualisiert.



## **int main()**

Die int main() des Programmes initialisiert den vector<pid\_t> prozesse mit der PID des Hauptprozesses und enthält eine Schleife, die das Benutzermenü anzeigt, die Benutzereingabe liest und je nach ausgewählter Option die entsprechende Funktionalität ausführt.

Zum Beispiel werden die Funktionen int child() und void exec() verwendet, um Kinderprozesse zu erzeugen und zu verzweigen. Die Funktionen int writeFile() und int readFile() werden verwendet, um Prozessinformationen in Dateien zu schreiben bzw. aus Dateien zu lesen. Funktionen wie void releaseResources() und void visualizeReltionship() dienen der Bereinigung nach dem Beenden der Prozesse und der Ausgabe von den Beziehungen der Prozesse. 



# Demonstration



## Datum Ausgeben



## Prozessinformationen ausgeben

Ausgabe Prozessinformationen von allen Prozessen (Einschließlich main)



## "Hello World!" ausgeben



## Prozesshierarchie anzeigen



## Logdatei lesen

Zuerst wird man aufgefordert den Namen der zu lesenden Datei anzugeben



Dann werden die gespeicherten Informationen ausgegeben



## Konsole bereinigen



## Beenden und Systemressourcen freigeben



# Fazit

In dieser Dokumentation wurde ein C++-Programm erklärt/beschrieben, das eine interaktive Benutzeroberfläche bereitstellt, um verschiedene Funktionen auszuführen. Es handelt sich um eine menübasierte Anwendung, die Optionen wie das Ausgeben des Datums, das Anzeigen von Prozessinformationen, das Ausführen eines “Hello World!“-Programms (Was zur Demonstration gilt, dass das Programm leicht erweiterbar ist) und das Lesen und Erstellen von Logdateien bietet. Das Programm ermöglicht auch eine Visualisierung der Beziehung zwischen Eltern- und Kindprozessen und bereinigt die Ressourcen beim Beenden.

Die Implementierung des Programms erfolgt in modularen Funktionen, die spezifische Aufgaben erledigen, wie das Lesen von Dateien oder das Ausgeben von Prozessinformationen. Dies macht den Code insgesamt wiederverwendbarer, lesbarer, wartbarer und ermöglicht auch isolierte Tests einzelner Funktionen unabhängig vom restlichen Code.

Abschließend kann festgestellt werden, dass das Programm die vorgegebenen Anforderungen erfüllt, aber in seinem derzeitigen Zustand nicht direkt nützlich für den Benutzer ist. Allerdings ist das Programm durch die Verwendung von void exec(), child(), enum, switch und dem Options Ordner leicht erweiterbar und kann somit auch für andere Benutzer, die keine Verwendung für die restlichen Funktionen haben, nützlich sein.