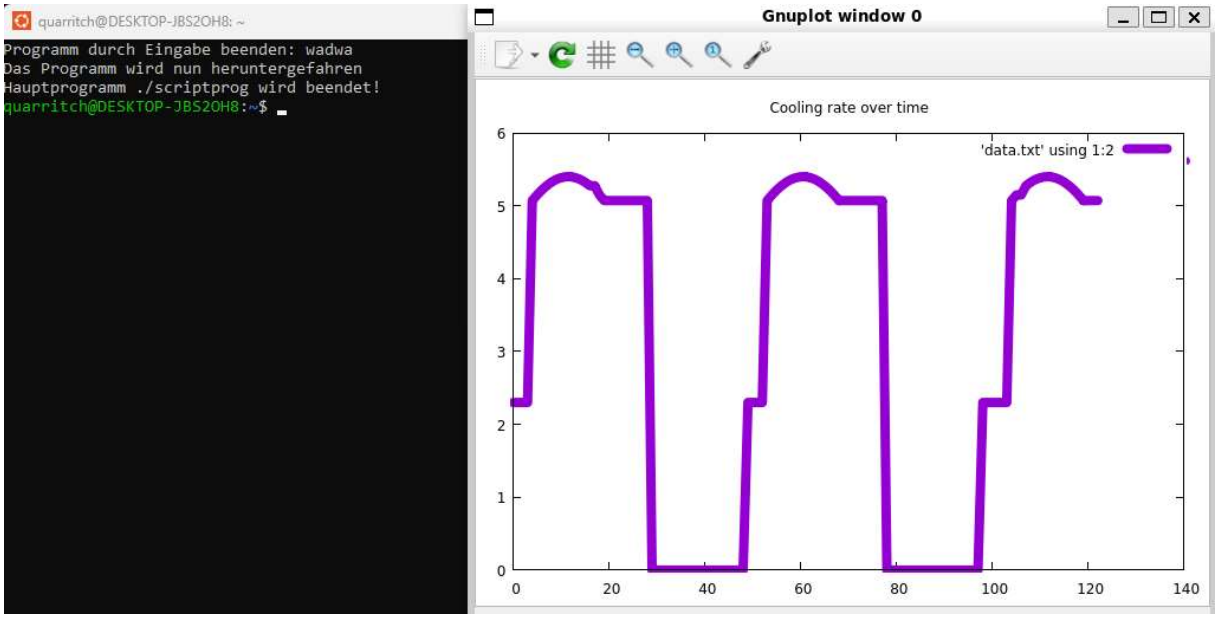
Dokumentation der Arbeitsmappe in Systemprogrammierung Vorlesung



Ein Projekt zur Simulation eines mechatronischen Systems unter Berücksichtigung von Prozessen und deren Kommunikation

Die Ausarbeitung erfolgte durch:

Dirk Frerichs

Timo Heider

Arne Nürnberg

Martin Pfeiffer

Alheshan Odai

Unter Betreuung von:

Prof. Dr. Ammar Memari

Inhalt

1. Einleitung

Im Rahmen der Vorlesung „Systemprogrammierung“ wurde uns die Aufgabe zugewiesen, ein robotisches System zu entwerfen, welches die Eingabe eines Sensors verarbeitet und mittels einer Steuerung einen Motor in seinen Eigenschaften steuert und kontrolliert. Da der Aufbau eines solchen Systems in der Realität den zeitlichen Rahmen übersteigt, wurde eine Simulation des Systems erarbeitet. Hierzu wurde ein bash-Skript zur Hand genommen, dass nach den Anforderungen die jeweiligen Prozesse aufruft und überwacht. Die Prozesse müssen korrekt implementiert sein sowie über eine effiziente Interprozesskommunikation verfügen. Auf dieser Grundlage wurde eine Klimaanlage simuliert, die je nach bestimmter Temperatur einen anderen Kühlungsgrad erzielt und den Raum abkühlt. Das System kann beliebig erweitert oder ergänzt werden.

2. Systemarchitektur

Die Systemarchitektur wurde durch die Anforderung der Aufgabenstellung aufgebaut. Sie beinhaltet eine Interprozesskommunikation, die Werte zwischen Prozessen übermittelt. Weiterhin müssen die Prozesse ihre Ausgaben loggen, sodass verständlich wird, was passiert. Die allgemeine Aufgabe des Projekts bestand darin, ein robotisches System mit einem Eingang, einem Ausgang und einem Controller zu simulieren. In unserer Gruppe haben wir folgende Systemarchitektur entworfen, um die Abläufe bestmöglich zu beschreiben:

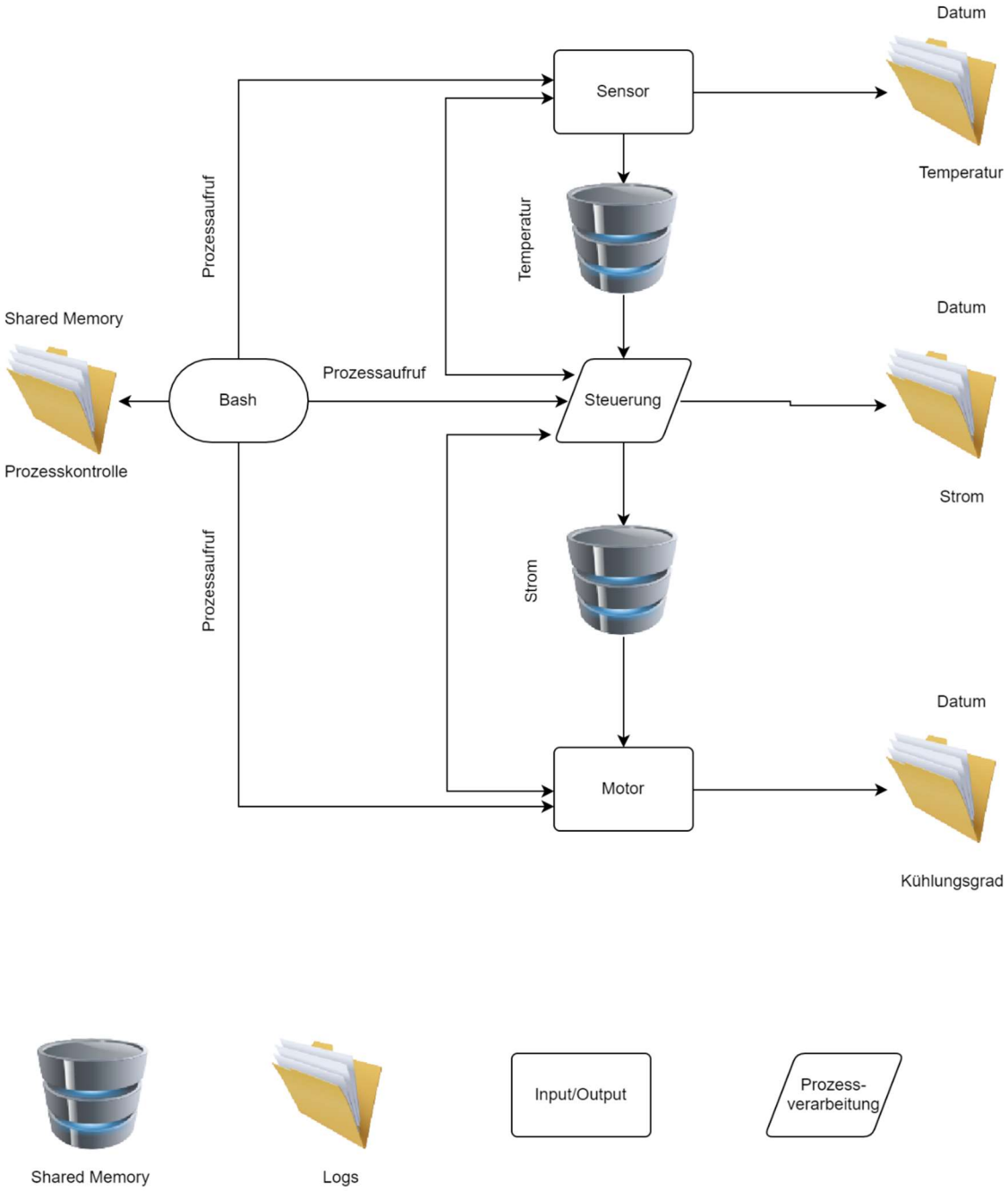


Abbildung 1: Systemarchitektur

Demnach werden die definierten C-Programme (Sensor, Steuerung, Motor) durch ein Bash-Programm aufgerufen und das Laufen der Prozesse kontrolliert. Falls diese nicht laufen sollten, werden sie erneut aufgerufen. Zuerst wird die Steuerung gestartet, welche die spezifischen Shared Memories erzeugt. Auf allen sind sowohl Schreib-, als auch Leserechte vorhanden, welche allerdings nicht zwangsweise genutzt werden. Näheres dazu ist dem Kapitel 3 zu entnehmen. Nach der Steuerung werden der Sensor und der Motor gestartet. Um Werte verarbeiten zu können, schreibt der Sensor Temperaturwerte in eine Shared Memory, welche dann von der Steuerung ausgelesen werden kann. Gleichzeitig schreibt der Sensor seine Werte mit einem Datum versehen in eine Logdatei. Die Steuerung verarbeitet die Temperatur zu einem Stromwert, welcher dann sowohl in eine weitere

Shared Memory als auch zusammen mit einem Datum in eine Logdatei geschrieben wird. Schlussendlich liest der Motor den Stromwert und berechnet daraus einen Kühlungsgrad. Der Kühlungsgrad wird genauso wie die anderen Werte zusammen mit einer Temperatur geloggt. Die Anwendung entspricht in diesem Fall einer Klimaanlage, allerdings kann sie beliebig verändert oder erweitert werden. Um deutlich zu machen, wie sich die Klimaanlage verhält, wurde ein Gnuplot implementiert, der den Kühlungsgrad bezogen auf die Zeit graphisch ausgibt.

3. Interprozesskommunikation

Zur Interprozesskommunikation dient die Shared Memory, welche in der Lage ist, permanent den aktuellen Wert des Temperatursensors zu übertragen. Dies ist relevant, da bei einer FIFO-Struktur eventuell neue Werte deutlich zu spät ausgelesen werden. In dieser Applikation ist für die Prozessverarbeitung des Inputs nur der in der Realität vorliegende Wert relevant, welche die Shared Memory bietet. Nachteil dieser Variante ist, dass im Falle eines Schreibens mehrerer Prozesse auf ein Shared Memory Segment nur der letzte geschriebene Wert vorhanden ist. Allerdings muss hierzu das Schreibkommando aufgerufen werden. Um das Programm einfacher zu halten, erhalten alle Prozesse alle Rechte, denn solange das Schreibkommando für ein Segment nicht von mehr als einem Prozess aufgerufen wird, kann kein Fehler entstehen.

Die Prozesse kommunizieren Untereinander nur über die Shared Memory, allerdings gibt das Hauptprogramm ein Signal zum Beenden aller Prozesse durch die Erstellung einer Datei namens „finish.log“, welche von den C-Programmen wahrgenommen wird. Sofern dieses Skripts vorhanden ist, schließt der Controller die Shared Memory Segmente und der Speicherraum wird freigegeben.

4. Realisierung

Nach der Systemarchitektur werden Temperaturwerte erzeugt. In diesem Programm kann zwischen einer Sinus- und einer Rechteckfunktion von simulierten Werten unterschieden werden. Deren Änderungsgeschwindigkeit und Amplitude kann im Skript eingestellt werden. Allerdings muss das Programm danach erneut kompiliert werden. Das Kompilieren kann mit folgender Zeile durchgeführt werden:

gcc -o [meinskript] [meinskript.c] [-lm]

„meinskript“ muss durch den Namen der C-Datei ersetzt werden. Das Argument „-lm“ dient dazu, die Bibliothek „math.h“ zu verlinken. Dies ist bei dem Sensor notwendig, sonst ist keine Kompilierung möglich.

Die Steuerung liest die Werte und verarbeitet sie zu einem Strom, um den Motor anzutreiben. In der Steuerung selbst kann nicht sonderlich viel editiert werden. Hier geht es um die Bereitstellung der Shared Memory und die Kontrolle über die Aktionen der Prozesse.

Der Motor übersetzt den Strom in einen Kühlungsgrad, welcher dann ausgegeben wird. In den Logdateien (siehe Kapitel 4.3) sowie in der graphischen Ausgabe sind die erzeugten Werte nachzuvollziehen.

Insgesamt muss die Programmstruktur von einem Bash-Skript verwaltet werden, welches die Programme aufruft und die Prozesse kontrolliert. In diesem Bash-Skript besteht weiterhin die Anforderung nach einem sauberen Herunterfahren der Prozesse und die Verwaltung der Logdateien bei Neustart.

4.1 Aufzurufende Programme und deren Kontrolle

Die Realisierung wurde anhand der Systemarchitektur durchgeführt. Das Bash-Skript ruft alle C-Skripte in folgender Reihenfolge auf:

1. Control
2. Sensor
3. Motor

Damit wird sichergestellt, dass der erste Prozess die Shared Memory erstellen und der Sensor in seinem Skript direkt darauf zugreifen kann. Genauso kann der Motor im Anschluss auf die Shared Memory zugreifen.

Die Prozesse dürfen dabei aber nicht ausfallen, da sonst keine neuen Werte geschrieben werden oder gar die Kontrolle über die Shared Memory verloren geht. Daher ist es notwendig, zu kontrollieren, ob alle Prozesse am Laufen sind oder neu gestartet werden müssen. Im Bash-Skript wird daher nach dem Prozessnamen gesucht und falls dieser in der Prozessübersicht nicht vorhanden ist, muss das jeweilige Programm neu gestartet werden. Wenn das Programm beendet werden soll, kann auf dem Kommandofenster eine Eingabe eingegeben werden, dann erzeugen die einzelnen Unterprozesse jeweils eine Datei mit der Endung *„finish“.* Wichtig ist, dass der Prozess „control“ nicht vor den anderen Prozessen beendet wird, da sonst Fehler in der Shared Memory entstehen können. Daher ist hier eine Abfrage über den Lauf der anderen Prozesse implementiert. Sofern diese nicht mehr laufen, kann auch „control“ beendet werden. Sofern alle Prozesse regulär beendet worden sind, fährt auch das Hauptprogramm herunter. Beim Neustart werden dann alle bestehenden Logdateien gelöscht.

Das Hauptprogramm überwacht in einer Funktion die genutzten Shared Memory Segmente und die Systemaufrufe. Das bash-Skript speichert sie nach der Struktur aus Kapitel 4.3 ab.

4.2 Fehlerbehandlung

Im Bereich der Fehlerbehandlung werden bestimmte Aktionen abgefragt und falls diese nicht existieren, wird der Prozess mit einer Errormessage beendet. Dieses Verhalten wird auch bei dem Erstellen der Shared Memory oder beim Übergeben der Argumente in die Logbibliothek verwendet. Damit kann der User direkt herausfinden, welche Prozesse fehlerhaft sind. Allerdings ist darauf zu achten, dass das Hauptprogramm in bash einen fehlerhaften Prozess eventuell neu startet. Daher lässt das Hauptprogramm einen Neustart der Programme insgesamt vier Mal zu. Danach wird das gesamte Skript beendet und eine dementsprechende Fehlermeldung ausgegeben.

Es ist ebenfalls anzumerken, dass am Anfang des Hauptprogramms in bash die Zeile *set -euo pipefail* angegeben wurde, welche dafür sorgt, dass nach einem Error im Hauptprogramm ein Stoppen erfolgt. Genauso ist das System so vor Pipefails und dem Löschen des Rootverzeichnisses mit *rm -rf „nicht-vorhandene-Variable“* geschützt.

4.3 Verwaltung von Logs

Um zu überprüfen, welche Werte wo geschrieben worden sind, können Logdateien gelesen werden. Im Betrieb wird eine Dateistruktur erzeuget, aus welcher die Dateien eingesehen werden können (siehe Abbildung 2):

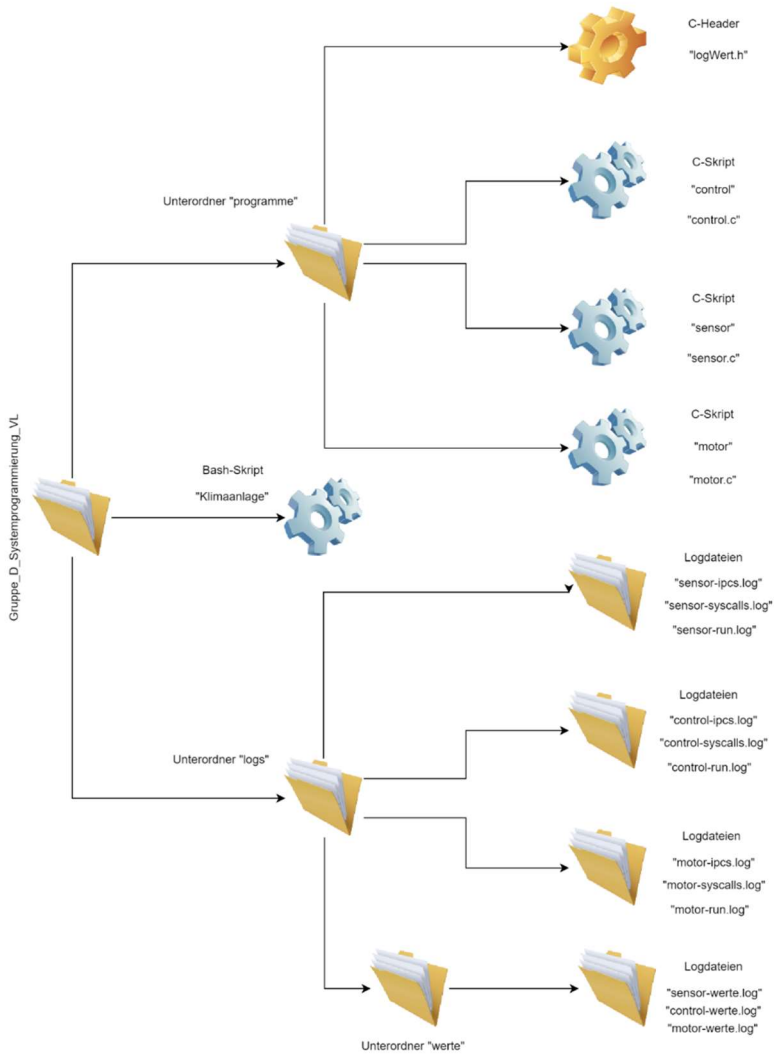


Abbildung 2: Dateistruktur im Betrieb

Alle Logs mit den Endungen „ipcs.log“ enthalten die Zuweisung der aktuellen Shared Memory Segmente. Ebenso stehen die Systemaufrufe eines Programms in den Logdateien mit der Endung „syscalls.log“. Die Logdateien mit der Endung „run.log“ zeigen an, ob ein Prozess noch läuft oder nicht. Alle diese Dateien werden mit dem nächsten Laufschritt überschrieben, es wird also immer nur der aktuelle Logeintrag angezeigt, um das System nicht zu „bloaten“. Der Unterordner „werte“ beinhaltet die auf die Shared Memory geschrieben Werte der einzelnen C-Skripte. Sie sind unterteilt in [Datum] [Wert], wie auch der ersten Zeile der Logdateien zu entnehmen. Diese Logdateien schreiben fortlaufend über die Dauer des gesamten Programms. Bei einem Neustart des Bash-Skripts werden sie gelöscht und dann neu geschrieben.

5. Benutzeranwendung

Das Grundverzeichnis sollte im Rohzustand folgende Sachen enthalten:

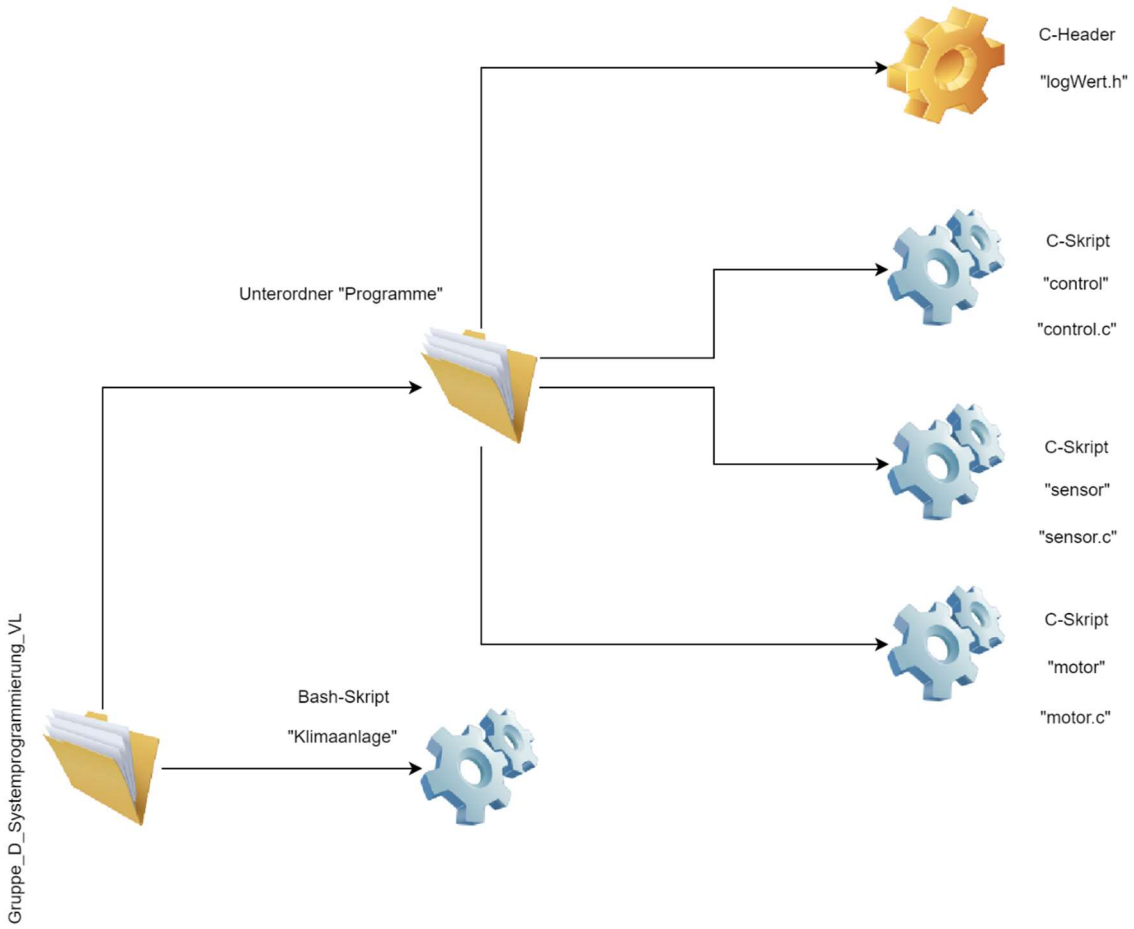


Abbildung 3: Dateistruktur vor Erstbetrieb

Für den Nutzer ist nur relevant, den korrekten Input als Argument zu übergeben. Zum Starten des Programms kann folgende Kommandozeile (im Verzeichnis) verwendet werden:

./Klimaanlage [sinus,rechteck]

Eines der beiden Argumente kann übergeben werden, um für ein bestimmtes Eingangssignal zu sorgen. Das Programm ist in der Lage, ein Sinus- und ein Rechtecksignal zu erzeugen. Danach können Ausgangswerte sowohl aus den Logdateien als auch aus dem Gnuplot entnommen werden.

Zum Beenden des Programms kann auf dem Kommandofenster eine Eingabe eingegeben werden.