Protokoll zu Betriebssysteme und Netzwerke

Übung: Message Queue, Kindprozesse und Aufgabenverteilung in pfusch.c File
Datum: 10.04.2025
Gruppe: 2
Teilnehmer: Alissa Lehner

Inhaltsverzeichnis

1 Protokoll – Message Queue, Kindprozesse und Aufgabenverteilung in C 3
1.1 Aufgabenerklärung 3
1.2 Adressierte Themengebiete 3
1.3 Fortschrittserkennung 3
1.4 Gelernte Erkenntnisse 3
1.5 Anwendung im Assignment 3
1.6 Zusätzliche Funktionen erarbeiten 4
1.7 Was war interessant? 4

1 Protokoll – Message Queue, Kindprozesse und Aufgabenverteilung in C

1.1 Aufgabenerklärung

In der Datei pfusch.c wurde ein Programm entwickelt, welches mit mehreren Prozessen arbeiten. Das Ziel war es, mehrere Kindprozesse zu erzeugen, die jeweils Aufgaben abgearbeitet haben. Die Aufgaben wurden vom Hauptprozess erzeugt und über eine POSIX-Message-Queue an die Kinder verteilt. Jedes Kind wartet auf die Aufgabe, bearbeitet sie ab und beendet sie anschließend.

1.2 Adressierte Themengebiete

Es wurden viele wichtige Themen abgearbeitet, die für die Prozesskommunikation wichtig sind:

- Erzeugung von Kindprozessen mit fork()
- Wartemechanismen mit wait() und WEXITSTATUS
- Arbeiten mit POSIX-Message-Queues (mq_open, mq_send, mq_receive, mq_close, mq_unlink)
- Nutzung von getopt() zur Argumentverarbeitung
- Arbeiten mit Zufallswerten und sleep() zur realistischen Simulation von Aufgabenbearbeitung

1.3 Fortschrittserkennung

Aus der Übungseinheit wurde mitgenommen, wie man mehrere Prozesse startet und über Queues die Aufgaben zuteilt. Schwierig meinerseits war das Verständnis und der Umgang mit mq_send und mq_receive. Diese ermöglichen eine Kommunikation zwischen Eltern- und der Kindprozessen. Wichtig war auch, dass die Queue Größe angepasst wurde, mq_maxmsg, sodass keine Blockierung enstehen konnte.

1.4 Gelernte Erkenntnisse

Erkannt wurde, dass ein paralleles Arbeiten mit Prozessen nicht gar so kompliziert ist, sobald verstanden wurde, worum es geht und wie es angewendet wird. Damit der Code übersichtlich blieb, wurden die Bereiche ordentlich getrennt, sodass erkannt wurde, welcher Abschnitt für die Aufgabenerstellung und welche für die Kindprozesse verantwortlich ist. Ebenso ist eine klare Fehlerbehandlung wichtig, sodass man direkt erkennen kann wo im Code ein Problem vorliegt, so wird die Zeit nach dem Suchen gespart und es ist ebenso effektiver zu arbeiten.

1.5 Anwendung im Assignment

Passend zur Übungseinheit, wurde eine Aufgabe „Task Ventilator“ mitgegeben, wo genau diese Prozesse behandelt und angewendet werden. Bei der Aufgabe werden genauso Aufgaben verteilt, bearbeitet und Ergebnisse zurückgeliefert. Durch das Verständnis des Codes im File pfusch.c ist es eine gute Grundlage für die Umsetzung solcher Symste in der neuen Aufgabenstellung.

1.6 Zusätzliche Funktionen erarbeiten

- Die genaue Verwendung von mq_attr und wie man die maximale Nachrichtenanzahl sinnvoll einstellt.
- Wie man Fehler bei mq_open, mq_send oder mq_receive erkennt und sauber behandelt.
- Welche Header und Flags benötigt werden, damit mq_open und Co. überhaupt funktionieren.
- Wie man die Ausgaben der Prozesse verständlich gestaltet, obwohl alle gleichzeitig schreiben.

1.7 Was war interessant?

Es war spannend zusehen, wie aus wenig Codezeilen ein funktionierdes Programm entsteht, welches mit mehreren Prozessen gleichzeitig arbeitet. Ebenso, dass die Prozesse auch unabhängig voneinander arbeiten konnten und sich somit nicht gegenseitig gestört haben. Auch das Thema, dass aus wenigen Kindern, ganz schnell mehrere Kinder programmiert werden konnten, mit jeweils mehreren Aufgaben.