Übungsblatt für die 11. und 12.Übung

Entwickeln Sie einen Teil eines Echtzeitsystems, das aus vier Prozessen besteht:

1. **Conv**. Dieser Prozess liest Messwerte von A/D-Konvertern (Analog/Digital) ein. Er prüft die Messwerte auf Plausabilität, konvertiert sie gegebenfalls und schreibt sie in einen Speicherbereich Mess. Wir lassen Conv in Ermangelung eines physischen A/D-Konverters Zufallszahlen erzeugen. Diese müssen in einem bestimmten Bereich liegen, um einen A/D-Konverter zu simulieren.
2. **Log**. Dieser Prozess liest die Messwerte des A/D-Konverters (Conv) aus und schreibt sie in eine Datei auf der Festplatte.
3. **Stat**. Dieser Prozess liest die Messwerte des A/D-Konverters (Conv) aus und berechnet statistische Daten, unter anderem Mittelwert und Summe.
4. **Report**. Dieser Prozess greift auf die Ergebnisse von Stat zu und gibt die statistischen Daten in der Shell aus.

Bezüglich der Daten in den gemeinsamen Speicherbereichen gelten als Synchronisationsbedingungen:

- **Conv** muss erst in den gemeinsamen Speicherbereich Mess die Werte eintragen, bevor **Log** und **Stat** die Messwerte auslesen können.
- **Stat** muss erst Statistikdaten in Statistik eintragen, bevor **Report** die Daten aus Statistik lesen kann.

Entwerfen und implementieren Sie das vorgestellte Realzeitproblem in C mit den entsprechenden Systemaufrufen und realisieren Sie den Datenaustausch zwischen den vier Prozessen einmal mit **Pipes**, **Message Queues** und **Shared Memory mit Semaphore**. Am Ende der praktischen Übung müssen drei Implementierungsvarianten des Programms existieren. Der Quellcode soll durch Kommentare verständlich sein.

Die Funktionalität der Programme muss in der Übung demonstriert werden!

1 Vorgehensweise

Die Prozesse Conv, Log, Stat, und Report sind parallele Endlosprozesse, das bedeutet, sie werden als Endlosprozesse realisiert. Schreiben Sie ein Gerüst zum Start der Endlosprozesse mit dem Systemaufruf `fork`. Beobachten und überwachen Sie mit geeigneten Programmen wie `top`, `ps` und `pstree` Ihre parallelen Prozesse und stellen Sie die Vater-Kindbeziehungen fest.

Das Programm kann mit der Tastenkombination `Ctrl-C` abgebrochen werden. Dazu müssen Sie einen Signalhandler für das Signal `SIGINT` implementieren. Beachten

Sie bitte, dass Sie beim Abbruch des Programmes alle von den Prozessen belegten Betriebsmittel (Message Queues, Shared Memory, Semaphoren) freigegeben werden.

Entwickeln und implementieren Sie die folgenden frei Varianten:

- Implementierungsvariante **Pipes**: Benutzen Sie zum Datenaustausch zwischen den Prozessen Verbindungskanäle (Pipes). Ein Prozess (Produzent) kann in eine Pipe Daten ablegen und ein anderer Prozess (Konsument) kann aus der Pipe Daten entnehmen.
- Implementierungsvariante **Message Queues**: Benutzen Sie für die gemeinsamen Daten eine Nachrichtenwarteschlange (Message Queue) in die die Produzenten-Prozesse Nachrichten ablegen und aus der die Konsumenten-Prozesse Nachrichten entnehmen.
- Implementierungsvariante **Shared Memory mit Semaphore**: Legen Sie die gemeinsamen Daten in gemeinsame Speichererbereiche, auf den die Prozesse zugreifen. Damit es beim gleichzeitigen Zugriff von zwei Prozessen auf einen gemeinsamen Speicher nicht zu Inkonsistenzen kommt, implementieren Sie die Synchronisationsbedingungen mit Hilfe von Semaphoren.

Überwachen Sie die Message Queues, Shared Memory Bereiche und Semaphoren mit dem Programm `ipcs`. Mit `ipcs` können Sie auch Message Queues, Shared Memory Bereiche und Semaphoren wieder freigeben, wenn Ihr Programm dieses bei einer inkorrekten Beendigung versäumt hat.