Praktische Übung in Betriebssysteme (BTS) SS2007

Aufgabenstellung

Entwickeln Sie einen Teil eines größeren 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 aus dem Speicherbereich Mess und schreibt sie in eine Datei auf der Festplatte.
3. **Stat**. Dieser Prozess liest die Messwerte des A/D-Konverters aus dem Speicherbereich Mess und berechnet statistische Daten, unter anderem Mittelwert und Summe. Die statistischen Daten werden in einen Speicherbereich Statistik geschrieben.
4. **Report**. Dieser Prozess greift auf den Speicherbereich Statistik zu und gibt die statistischen Daten aus. Eigentlich sollte dies auf einem Drucker geschehen. Wir geben die Daten der Einfachheit wegen auf dem Monitor, also in der Shell aus.

Bezüglich der gemeinsamen Daten im Speicherbereich Mess gelten als Synchronisationsbedingungen: Conv muss erst in den gemeinsamen Speicherbereich Mess die Werte eintragen, bevor Log und Stat die Messwerte auslesen können.

Bezüglich der gemeinsamen Daten im Speicherbereich Statistik gelten als Synchronisationsbedingungen: 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 und Semaphore**. Am Ende der praktischen Übung müssen drei Implementierungen des Programms in der Programmiersprache C mit den entsprechenden Systemaufrufen existieren.

Anzufertigen ist eine schriftliche Ausarbeitung mit einer Beschreibung der Aufgabe und Lösung sowie der Quellcode der drei Implementierungsvarianten und eine Diskussion der Vor- und Nachteile der drei Implementierungsvarianten.

Die Funktionalität der Programme muss in der Übung demonstriert werden!

Bitte beachten Sie, dass Sie die praktische Übung nur alleine ausfertigen und abgeben können!

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.

Entwerfen Sie die Speicherbereiche zwischen den Prozessen und die Synchronisationsbedingungen zum Zugriff auf die Speicherbereiche in folgenden frei Variationen:

- Implementierungsvariante **Pipes**: Benutzen Sie zum Datenaustausch zwischen den Prozessen Pipes. Pipes sind Verbindungskanäle zwischen zwei Prozessen und funktionieren nach dem FIFO-Prinzip. 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 Botschaftenwarteschlange (Message Queue) in die die Produzenten-Prozesse mit `send(mgssnd)` Nachrichten ablegen und die Konsumenten-Prozesse mit `receive(msgrcv)` Nachrichten entnehmen.
- Implementierungsvariante **Shared Memory und Semaphore**: Legen Sie die gemeinsamen Daten von zwei Prozessen in einen gemeinsamen Speicher, 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.

Literatur

- **Betriebssysteme**, *Albrecht Achilles*, Springer-Verlag (2006)
- **Betriebssysteme**, *Erich Ehses, Lutz Köhler, Petra Riemer, Horst Stenzel, Frank Victor*, Pearson Studium (2005)