

Übungsblatt 10

Aufgabe 1 (Kommunikation von Prozessen)

1. Was ist bei Interprozesskommunikation über gemeinsame Speichersegmente (Shared Memory) zu beachten?
2. Welche Aufgabe hat die Shared Memory Tabelle im Linux-Kernel?
3. Kreuzen Sie an, welche Auswirkungen ein Neustart (Reboot) des Betriebssystems auf die bestehenden gemeinsamen Speichersegmente (Shared Memory) hat.
(Nur eine Antwort ist korrekt!)
 - Die gemeinsamen Speichersegmente werden beim Neustart erneut angelegt und die Inhalte werden wieder hergestellt.
 - Die gemeinsamen Speichersegmente werden beim Neustart erneut angelegt, bleiben aber leer. Nur die Inhalte sind also verloren.
 - Die gemeinsamen Speichersegmente und deren Inhalte sind verloren.
 - Nur die gemeinsamen Speichersegmente sind verloren. Die Inhalte speichert das Betriebssystem in temporären Dateien im Ordner `\tmp`.
4. Nach welchem Prinzip arbeiten Nachrichtenwarteschlangen (Message Queues)?
(Nur eine Antwort ist korrekt!)
 - Round Robin
 - LIFO
 - FIFO
 - SJF
 - LJF
5. Wie viele Prozesse können über eine Pipe miteinander kommunizieren?
6. Was passiert, wenn ein Prozess in eine volle Pipe schreiben will?
7. Was passiert, wenn ein Prozess aus einer leeren Pipe lesen will?
8. Welche zwei Arten Pipes existieren?
9. Welche zwei Arten Sockets existieren?
10. Kommunikation via Pipes funktioniert...
(Nur eine Antwort ist korrekt!)
 - speicherbasiert
 - datenstrombasiert
 - objektbasiert
 - nachrichtenbasiert
11. Kommunikation via Nachrichtenwarteschlangen funktioniert...
(Nur eine Antwort ist korrekt!)
 - speicherbasiert
 - datenstrombasiert
 - objektbasiert
 - nachrichtenbasiert

12. Kommunikation via gemeinsamen Speichersegmenten funktioniert...
(Nur eine Antwort ist korrekt!)
- speicherbasiert datenstrombasiert
 objektbasiert nachrichtenbasiert
13. Kommunikation via Sockets funktioniert...
(Nur eine Antwort ist korrekt!)
- speicherbasiert datenstrombasiert
 objektbasiert nachrichtenbasiert
14. Welche zwei Formen der Interprozesskommunikation funktionieren bidirektional?
- Gemeinsame Speichersegmente Nachrichtenwarteschlangen
 Anonyme Pipes Benannte Pipes
 Sockets
15. Welche Form der Interprozesskommunikation funktioniert nur zwischen Prozessen die eng verwandt sind?
- Gemeinsame Speichersegmente Nachrichtenwarteschlangen
 Anonyme Pipes Benannte Pipes
 Sockets
16. Welche Form der Interprozesskommunikation funktioniert über Rechnergrenzen?
- Gemeinsame Speichersegmente Nachrichtenwarteschlangen
 Anonyme Pipes Benannte Pipes
 Sockets
17. Bei welchen Formen der Interprozesskommunikation bleiben die Daten auch ohne gebundenen Prozess erhalten?
- Gemeinsame Speichersegmente Nachrichtenwarteschlangen
 Anonyme Pipes Benannte Pipes
 Sockets
18. Bei welcher Form der Interprozesskommunikation garantiert das Betriebssystem nicht die Synchronisierung?
- Gemeinsame Speichersegmente Nachrichtenwarteschlangen
 Anonyme Pipes Benannte Pipes
 Sockets

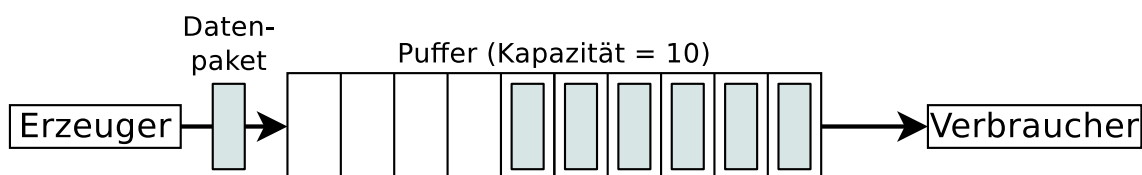
Aufgabe 2 (Kooperation von Prozessen)

1. Was ist eine Semaphore und was ist ihr Einsatzzweck?

2. Welche beiden Operationen werden bei Semaphoren verwendet?
Gesucht sind die Bezeichnungen und eine (kurze) Beschreibung der Funktionsweise.
3. Was ist der Unterschied zwischen Semaphoren und Blockieren (Sperren und Freigeben)?
4. Was ist eine binäre Semaphore?
5. Was ist eine starke Semaphore?
6. Was ist eine schwache Semaphore?
7. Was ist ein Mutex und was ist sein Einsatzzweck?
8. Welche Form der Semaphoren hat die gleiche Funktionalität wie der Mutex?
9. Welche Zustände kann ein Mutex annehmen?
10. Welches Linux/UNIX-Kommando liefert Informationen zu bestehenden gemeinsamen Speichersegmenten, Nachrichtenwarteschlangen und Semaphoren?
11. Welches Linux/UNIX-Kommando ermöglicht es, bestehende gemeinsame Speichersegmente, Nachrichtenwarteschlangen und Semaphoren zu löschen?

Aufgabe 3 (Erzeuger/Verbraucher-Szenario)

Ein Erzeuger soll Daten an einen Verbraucher schicken. Ein endlicher Zwischenspeicher (Puffer) soll die Wartezeiten des Verbrauchers minimieren. Daten werden vom Erzeuger in den Puffer gelegt und vom Verbraucher aus diesem entfernt. Gegenseitiger Ausschluss ist nötig, um Inkonsistenzen zu vermeiden. Ist der Puffer voll, muss der Erzeuger blockieren. Ist der Puffer leer, muss der Verbraucher blockieren.



Synchronisieren Sie die beiden Prozesse, indem Sie die nötigen Semaphoren erzeugen, diese mit Startwerten versehen und Semaphor-Operationen einfügen.

```
typedef int semaphore;

void erzeuger (void) {
    int daten;
    while (TRUE) {                // Endlosschleife
        erzeugeDatenpaket(daten); // erzeuge Datenpaket

        einfuegenDatenpaket(daten); // Datenpaket in Puffer schreiben

    }
}

void verbraucher (void) {
    int daten;
    while (TRUE) {                // Endlosschleife

        entferneDatenpaket(daten); // Datenpaket aus Puffer holen

        verbraucheDatenpaket(daten); // Datenpaket nutzen
    }
}
```

Aufgabe 4 (Semaphoren)

In einer Lagerhalle werden ständig Pakete von einem Lieferanten angeliefert und von zwei Auslieferern abgeholt. Der Lieferant und die Auslieferer müssen dafür ein Tor durchfahren. Das Tor kann immer nur von einer Person durchfahren werden. Der Lieferant bringt mit jeder Lieferung 3 Pakete zum Wareneingang. An der Ausgabe holt ein Auslieferer jeweils 2 Pakete ab, der andere Auslieferer 1 Paket.

```
Lieferant          Auslieferer_X      Auslieferer_Y
{
  while (TRUE)
  {
    <Tor durchfahren>;

    <Wareneingang betreten>;

    <3 Pakete entladen>;

    <Wareneingang verlassen>;

    <Tor durchfahren>;
  }
}

{
  while (TRUE)
  {
    <Tor durchfahren>;

    <Warenausgabe betreten>;

    <2 Pakete aufladen>;

    <Warenausgabe verlassen>;

    <Tor durchfahren>;
  }
}

{
  while (TRUE)
  {
    <Tor durchfahren>;

    <Warenausgabe betreten>;

    <1 Paket aufladen>;

    <Warenausgabe verlassen>;

    <Tor durchfahren>;
  }
}
```

Es existiert genau ein Prozess `Lieferant`, ein Prozess `Auslieferer_X` und ein Prozess `Auslieferer_Y`.

Synchronisieren Sie die beiden Prozesse, indem Sie die nötigen Semaphoren erzeugen, diese mit Startwerten versehen und Semaphore-Operationen einfügen.

Folgende Bedingungen müssen erfüllt sein:

- Es darf immer nur ein Prozess das Tor durchfahren.
- Es darf immer nur einer der beiden Auslieferer die Warenausgabe betreten.
- Es soll möglich sein, dass der Lieferant und ein Auslieferer gleichzeitig Waren entladen bzw. aufladen.
- Die Lagerhalle kann maximal 10 Pakete aufnehmen.
- Es dürfen keine Verklemmungen auftreten.
- Zu Beginn sind keine Pakete in der Lagerhalle vorrätig und das Tor, der Wareneingang und die Warenausgabe sind frei.

Quelle: TU-München, Übungen zur Einführung in die Informatik III, WS01/02

Aufgabe 5 (Interprozesskommunikation)

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 Plausibilität und konvertiert sie gegebenenfalls. 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 lokale Datei.
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 Messwerte schreiben, bevor **Log** und **Stat** Messwerte auslesen können.
- **Stat** muss erst Statistikdaten schreiben, bevor **Report** Statistikdaten auslesen kann.

Entwerfen und implementieren Sie das Echtzeitsystem 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.

Vorgehensweise

Die Prozesse Conv, Log, Stat, und Report sind parallele Endlosprozesse. Schreiben Sie ein Gerüst zum Start der Endlosprozesse mit dem Systemaufruf `fork`. Überwachen Sie mit geeigneten Kommandos 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 beim Abbruch des Programms alle von den Prozessen belegten Betriebsmittel (Pipes, Message Queues, gemeinsame Speicherbereiche, Semaphore) freigegeben werden.

Entwickeln und implementieren Sie die drei Varianten, bei denen der Datenaustausch zwischen den vier Prozessen einmal mit **Pipes**, **Message Queues** und **Shared Memory mit Semaphore** funktioniert.

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