# Betriebssysteme

## Prozesskommunikation

G. Richter

25. April 2017

# Kommunikation von Prozessen

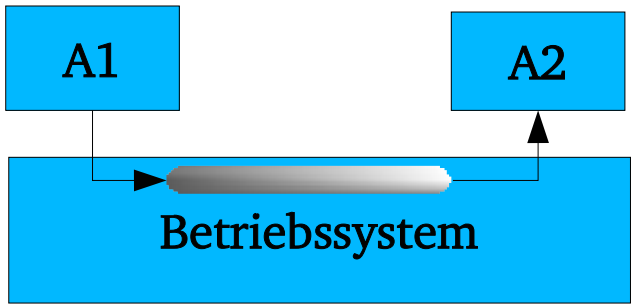
Ein Nachteil der strikten Trennung von Prozessen ist, dass Prozesse mit einfachen Mitteln keine Daten austauschen können. Sie können im eigenen Adressraum Daten lesen und schreiben, die Adressräume anderer Prozesse sind für sie unerreichbar.
Andererseits ist der Datenaustausch zwischen Prozessen, die kooperieren müssen, unabdingbar.
Es müssen aber nicht nur Daten transferiert oder gemeinsam genutzt werden, sondern Prozesse müssen auch synchronisiert werden.

# Erzeuger-Verbraucher-Problem

- Ein Programm produziert Daten, die von einem anderen Programm weiterverarbeitet werden. Arbeiten beide gleich schnell, gibt es keine Probleme. Dies ist meist nicht so.
- 2 Möglichkeiten:
  1. Erzeuger produziert schneller als Verbraucher verarbeiten kann
  2. Erzeuger produziert langsamer als Verbraucher verarbeitet
- Lösung:
  1. Erzeuger muss angehalten werden, wenn Verbraucher nichts mehr aufnehmen kann
  2. Verbraucher muss angehalten werden, wenn keine Daten verfügbar sind

# Pipes

- Pipes sind Kommunikationsverbindungen zwischen zwei Prozessen.
- sie werden wie Dateien angesprochen (Systemaufrufe read, write)
- Sie besitzen eine Schreib- und eine Leseseite
- Der schreibende Prozess kann die Pipe bis zu einem gewissen Füllstand (~64kB, parametrierbar) beschreiben. Versucht er nach Erreichen des max. Füllstands weiter zu schreiben, wird er blockiert (wartend gesetzt).
- Wir die Pipe wieder geleert, wird der Prozess fortgesetzt und kann weiter schreiben.
- Der Leser kann die Pipe auslesen, bis sie leer ist, dann wird er ebenfalls blockiert.
- Treffen Daten ein, wird er fortgesetzt.



# Arten von Pipes

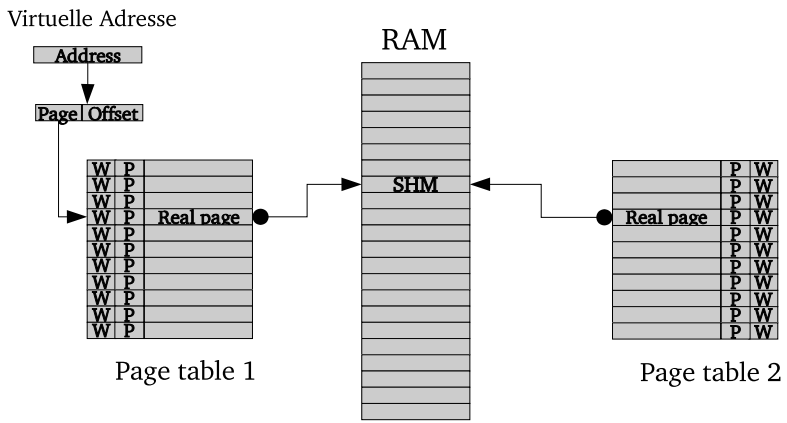
- Es gibt „Named“ Pipes und „Unnamed“ Pipes
- Unnamed Pipes sind nur von eng verwandten Prozessen benutzbar (Prozesse, die aus einem Fork hervorgehen, erben alle Ressourcen des Erzeugers, also auch die erzeugten Pipes).
- Named Pipes werden über einen Systemaufruf „mkfifo()“ oder das Systemprogramm „mkfifo <NAME>“ erzeugt.
- FIFO können von allen Prozessen genutzt werden, welche die Rechte dazu haben.
- FIFO sind im Dateisystem sichtbar, es „special files“.

# Message Queues

- Message Queues werden über einen Systemaufruf erzeugt
  msgget()
- Messages werden darin als Objekte mit Typ gespeichert
- Messages sind nur als Ganzes schreib- und lesbar
  msgsnd(), msgrcv()
- Beim Lesen können Messages eines bestimmten Typs selektiert werden
  msgrcv(„msgtype„)

# Shared Memory

- Mit shared memory wird Speicher zwischen Prozessen geteilt
- ein Prozess richtet den Speicher innerhalb seines Adressraums ein.
- Ein oder mehrere andere Prozesse blenden das shared memory in Ihren Adressraum ein
- Das Betriebssystem realisiert den gemeinsamen Speicher dadurch, dass ein oder mehrere Seiten in den PT der beiden Prozess auf dieselben reellen Seiten im Hauptspeicher zeigen.
- Da die Zugriffssteuerung über die PTs seitengranular ist, können immer nur ganze Seiten im SHM verwaltet werden.



# Semaphore

- Semaphore sind Zählvariable, die über Systemaufrufe modifiziert werden und an die Zustände von Prozessen gekoppelt werden.
- Ein Semaphor wird bei der Erzeugung mit einem Wert $> 0$ initialisiert: sem_init(..).
- Ein Prozess kann den Semaphor belegen. Dabei wird der Semaphor dekrementiert: sem_wait(). Wenn der Semaphor den Wert Null hat, wird der Prozess blockiert. Andernfalls wird der Prozess fortgesetzt.
- Wird der Semaphor von einem anderen Prozess inkrementiert: sem_post(), wird der Prozess, der blockiert wurde, wieder fortgesetzt und der Semaphor wieder dekrementiert.
- Ein Semaphor, der mit dem Wert „1“ initialisiert wurde kann zum gegenseitigen Ausschluss von konkurrierenden Threads oder Prozessen genutzt werden. (mutex = mutual exclusion)

# Signale

- Signale sind asynchrone oder synchrone Ereignisse, die Prozessen mitgeteilt werden können

# Signale

- Signale sind asynchrone oder synchrone Ereignisse, die Prozessen mitgeteilt werden können
- Vergleichbar mit Interrupts auf Hardwareebene

# Signale

- Signale sind asynchrone oder synchrone Ereignisse, die Prozessen mitgeteilt werden können
- Vergleichbar mit Interrupts auf Hardwareebene
- Synchron: SIGFPE bei Division durch Null - Tritt synchron zum Divisionsbefehl auf

# Signale

- Signale sind asynchrone oder synchrone Ereignisse, die Prozessen mitgeteilt werden können
- Vergleichbar mit Interrupts auf Hardwareebene
- Synchron: SIGFPE bei Division durch Null - Tritt synchron zum Divisionsbefehl auf
- Asynchron: SIGTERM kann einem Prozess jederzeit gesendet werden

# Signale

- Signale sind asynchrone oder synchrone Ereignisse, die Prozessen mitgeteilt werden können
- Vergleichbar mit Interrupts auf Hardwareebene
- Synchron: SIGFPE bei Division durch Null - Tritt synchron zum Divisionsbefehl auf
- Asynchron: SIGTERM kann einem Prozess jederzeit gesendet werden
- Es gibt abfangbare Signale (SIGTERM, SIGFPE). Hier kann der Prozess eigenen Routinen zur Signalbehandlung implementieren oder das Signal maskieren

# Signale

- Signale sind asynchrone oder synchrone Ereignisse, die Prozessen mitgeteilt werden können
- Vergleichbar mit Interrupts auf Hardwareebene
- Synchron: SIGFPE bei Division durch Null - Tritt synchron zum Divisionsbefehl auf
- Asynchron: SIGTERM kann einem Prozess jederzeit gesendet werden
- Es gibt abfangbare Signale (SIGTERM, SIGFPE). Hier kann der Prozess eigenen Routinen zur Signalbehandlung implementieren oder das Signal maskieren
- Nicht abfangbare Signale (SIGKILL, SIGSTOP) verwenden immer die Default-Reaktion: exit() und können auch nicht maskiert werden.