Praktikum 8 - Interprozess-Kommunikation

1 Vorbemerkung

1.1 Informationsaustausch zwischen Prozessen

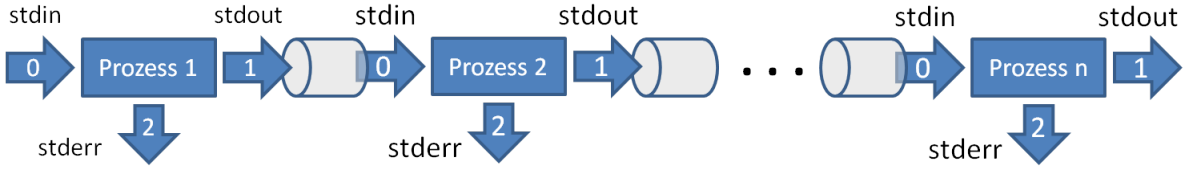
Prozesse sind in der Lage, Daten untereinander auszutauschen bzw. Status-Zustände abzufragen und darauf entsprechend reagieren und ggf. steuernd eingreifen. Als Kommunikationskanäle gibt es u.a. Signale und Semaphore zur Steuerung sowie Shared Memories und Pipes (unbenannt/benannt) zum Datenaustausch.

Das grundsätzliche Ziel von Pipes ist es dabei, die Ausgabe eines Prozesses als Eingabe in einen anderen Prozess zu übernehmen und diese Daten im neuen Prozess weiter zu verarbeiten.

1.2 Unbenannte Pipes (unnamed pipes) zum Datenaustausch zwischen Prozessen

Unbenannte Pipes dienen zum direkten Verknüpfen der Ein- und Ausgabekanälen von zwei Prozessen. Diese Kombination kann auch mehrfach hintereinander folgen, so dass ggf. eine Kette von Prozessen entsteht.

```
user@debian:~$ kommando1 | kommando2 | ... | kommando[n]
```



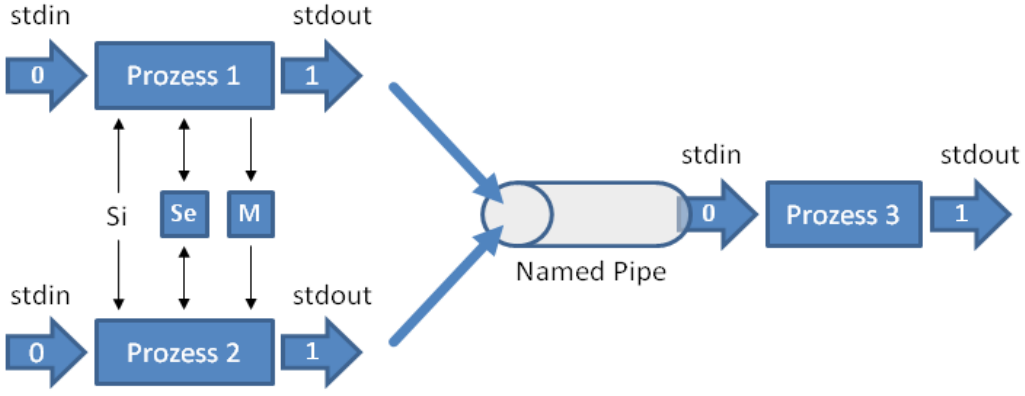
1.3 Benannte Pipes (named pipes) zum Datenaustausch zwischen Prozessen

Sie dienen zum Verknüpfen der Ausgaben von ein oder mehreren Prozessen als Eingabe zum nächsten Prozess. Dabei kommt es ggf. zum gleichzeitigen Zugriff auf die Pipe, wobei die Daten dann nach dem FIFO-Prinzip (first-in first-out) weitergeleitet werden.

Benannte Pipes werden unter Linux als spezielle Gerätedateien im Dateisystem angelegt.

```
user@debian:~$ ls -l [<name>]

-rwxrwxrwx count user group size date name
|
+-------------------------------------- Typ der Datei
                                        (p) pipe (fifo)
```



Legende : Si = Signale / Se = Semaphore / M = Shared Memory

2 Übungsaufgaben

Das **„watch“**-Fenster soll bei allen Aufgaben zur Anzeige von relevanten Daten geöffnet sein.

Kommando	Auf-/Ausgabe des Kommandos bzw. Skripts
`mkdir -m 777 /tmp/pr08`	
`pr08 watch`	Permanente Ausgabe von relevanten Status-Informationen
`pr08 close_all`	Ggf. relevante Prozesse beenden und **„watch“**-Fester zurücksetzen

2.1 Kommunikation zwischen Prozessen mit unbenannten Pipes

Bsp.: Ausgabe von einem Prozess nach bestimmten Zeichen filtern bzw. ausblenden.

`pr08 2.1`	Auf-/Ausgabe des Kommandos bzw. Skripts
`p1.sh`	
`p1.sh > /tmp/pr08/d1`	
`grep 0 /tmp/pr08/d1 > /tmp/pr08/d2`	
`grep -v 2 /tmp/pr08/d2`	
`p1.sh | grep 0 | grep -v 2`	**Alternative Lösung mit unbenannten Pipes**

2.2 Kommunikation zwischen Prozessen mit benannten Pipes

Es wird eine benannte Pipe erstellt, wobei Sie anhand der angezeigten Dateiinformationen ermitteln sollen, wie man eine benannte Pipe im Dateisystem erkennt. Prozess P3 wird gestartet, um diese Pipe kontinuierlich auszulesen und die Daten auszugeben.

`pr08 2.2.1`	Aufgabe des Kommandos
`mknod -m 777 /tmp/pr08/pipe p`	
`p3.sh /tmp/pr08/pipe`	Pipe kontinuierlich auslesen und Daten anzeigen
`echo "Hello" > /tmp/pr08/pipe`	Beispielhaftes Senden von Daten in die Pipe

Danach werden die Prozesse P1 und P2 gestartet, wobei Sie im ersten Schritt ermitteln sollen, in welcher Reihenfolge die Daten durch den Prozess P3 ausgegeben werden.

`pr08 2.2.2`	Aufgabe des Kommandos
`p1.sh /tmp/pr08/pipe`	Prozess P1 zum Senden der Daten in die Pipe
`p2.sh /tmp/pr08/pipe`	Prozess P2 zum Senden der Daten in die Pipe

Mit dem Signal **„`SIGTRAP`“** können die Prozesse P1 und P2 immer wieder umgeschaltet werden, dass sie entweder im Einzelschritt-Modus (jeweils mit ENTER) oder im Automatik-Modus arbeiten. Beim Umschalten vom Einzelschritt- zum Automatik-Modus muss ggf. noch der letzte Einzelschritt-Befehl mit „ENTER“ bestätigt werden, damit der Prozess wieder automatisch läuft.

Mit dem Signal **„`SIGTERM`“** können alle Prozesse normal beendet werden.

Kommando	Aufgabe des Kommandos
	Schritt-/Automatikmodus für P1 wechseln
	Prozesse P1, P2 und P3 beenden

2.3 Steuerung von Prozessen mit Signalen

In diesem Szenario sollen sich die Prozesse P1 und P2 über Signale (Parameter **„-si“**) gegenseitig steuern, damit jeder Prozess seine Daten zusammenhängend an den Prozess P3 senden kann. Die grundsätzliche Verwendung von Signale war bereits Bestandteil des vorherigen Praktikums, deren Funktionsweise an dieser Stelle weiter exemplarisch vertieft werden soll.

Verschaffen Sie sich einen Überblick, wie diese Steuerung zwischen P1 und P2 prinzipiell arbeitet. Achten Sie dabei auch auf den aktuellen Status der Prozesse im Überwachungsfenster.

Ermitteln Sie im Schritt-Modus, wie sich diese Prozesse P1 und P2 gegenseitig steuern.

Senden Sie abschließend entsprechende Signale, damit alle Prozesse beendet werden.

`pr08 2.3`	**Aufgabe des Kommandos**
`p3.sh /tmp/pr08/pipe`	Prozess P3 => kontinuierliches Auslesen der Pipe
`p1.sh -si /tmp/pr08/pipe`	Prozess P1 => signal-gesteuertes Senden in die Pipe
`p2.sh -si /tmp/pr08/pipe`	Prozess P2 => signal-gesteuertes Senden in die Pipe
	Umschalten Schritt-/Automatikmodus für P1/P2
	Prozesse P1, P2 und P3 beenden

Fassen Sie erklärend zusammen, wie in diesem Szenario sichergestellt wird, dass die beiden Prozesse P1 und P2 ihre Daten ohne Unterbrechung an den Prozess P3 senden können.

2.4 Steuerung vom Zugriff auf Systemressourcen mit Semaphoren

In diesem Szenario sollen die Prozesse P1 und P2 konkurrierend auf die Pipe zugreifen, wobei über einen Semaphor (Parameter **„-se“**) gesteuert werden soll, dass die jeweiligen Prozesse ihre Daten zusammenhängend an den Prozess P3 senden können.

Erarbeiten Sie sich im Vorfeld einen theoretischen Überblick über den Einsatz von Semaphoren zur Steuerung von Zugriffen auf Systemressourcen.

Verschaffen Sie sich mit der folgenden Aufgabe einen Überblick, wie eine solche Zugriffssteuerung exemplarisch zwischen P1 und P2 arbeitet. Achten Sie dabei auch auf den aktuellen Status vom Semaphor im Überwachungsfenster.

Ermitteln Sie im Schritt-Modus detailliert, wie die beiden Prozesse P1 und P2 den Zugriff auf die gemeinsam benötigte Systemressource steuern.

Senden Sie abschließend entsprechende Signale, damit alle Prozesse beendet werden.

`pr08 2.4`	**Aufgabe des Kommandos**
`p3.sh /tmp/pr08/pipe`	Prozess P3 => kontinuierliches Auslesen der Pipe
`p1.sh -se /tmp/pr08/pipe`	Prozess P1 => semaphor-gesteuertes Senden in die Pipe
`p2.sh -se /tmp/pr08/pipe`	Prozess P2 => semaphor-gesteuertes Senden in die Pipe
	Umschalten Schritt-/Automatikmodus für P1/P2
	Prozesse P1, P2 und P3 beenden

Fassen Sie erklärend zusammen, wie in diesem Szenario sichergestellt wird, dass die beiden Prozesse P1 und P2 ihre Daten ohne Unterbrechung an den Prozess P3 senden können.

2.5 Datenaustausch über Shared Memory

In diesem Szenario soll der Prozess P1 durch den Parameter **„-m“** periodisch zuvor ermittelte Daten in einem definierten Speicherbereich ablegen. Anschließend soll der Prozess P2 durch den Parameter **„-m“** diese Daten in undefinierten Abständen aus dem festgelegtem Speicherbereich auslesen und danach über die Pipe an den Prozess P3 weiterleiten.

Starten Sie zuerst den Prozess P1 und verschaffen Sie sich dabei einen Überblick, wie der gemeinsame Speicherbereich beschrieben wird.

Erarbeiten Sie sich im Schritt-Modus von Prozess P1 die Details der Arbeitsweise und stoppen Sie danach am Anfang eines neuen Durchlaufes.

Starten Sie danach den Prozess P2 und ermitteln Sie, wie die Daten aus dem Datenspeicher an den Prozess P3 gesendet werden. Nutzen Sie dazu ggf. auch den Schritt-Modus analog zu Prozess P1.

Lassen Sie danach beide Prozesse im Automatik-Modus bzw. auch im Schritt-Modus laufen und erklären Sie, wie in diesem Beispiel das Lesen bzw. Senden von unvollständigen Datensätzen an den Prozess P3 verhindert wird.

Senden Sie abschließend entsprechende Signale an P1, P2 und P3, um diese Prozesse zu beenden.

`pr08 2.5`	**Aufgabe des Kommandos**
`p3.sh /tmp/pr08/pipe`	Prozess P3 => kontinuierliches Auslesen der Pipe
`p1.sh -m`	Prozess P1 => Ablegen von Daten in den Speicher
`p2.sh -m /tmp/pr08/pipe`	Prozess P2 => Senden der abgelegten Daten in die Pipe
	Umschalten Schritt-/Automatikmodus für P1
	Umschalten Schritt-/Automatikmodus für P2
	Prozesse P1, P2 und P3 beenden

Fassen Sie erklärend zusammen, wie in diesem Szenario sichergestellt wird, wie die beiden Prozesse P1 und P2 ihre Daten ohne Unterbrechung austauschen und anschließend an den Prozess P3 senden können.

2.6 Blockierung von Prozessen durch belegte Ressourcen (deadlock)

In diesem Szenario sollen sich die Prozesse P1 und P2 über einen Semaphor (Parameter **„-se“**) gegenseitig steuern, damit sie ihre Daten zusammenhängend an den Prozess P3 senden können.

Senden Sie anschließend Signale, die P3 wechselweise stoppen (**nicht beenden!)** bzw. weiterlaufen lassen und damit das Auslesen bzw. Leeren der Pipe beeinflussen.

`pr08 2.6`	**Ergebnis des Kommandos**
`p3.sh /tmp/pr08/pipe`	Prozess P3 => kontinuierliches Auslesen der Pipe
`p1.sh -se /tmp/pr08/pipe`	Prozess P1 => semaphor-gesteuertes Senden in die Pipe
`p2.sh -se /tmp/pr08/pipe`	Prozess P2 => semaphor-gesteuertes Senden in die Pipe
	Stoppen vom Prozess P3
	Fortsetzen vom Prozess P3
	Prozesse P1, P2 und P3 beenden

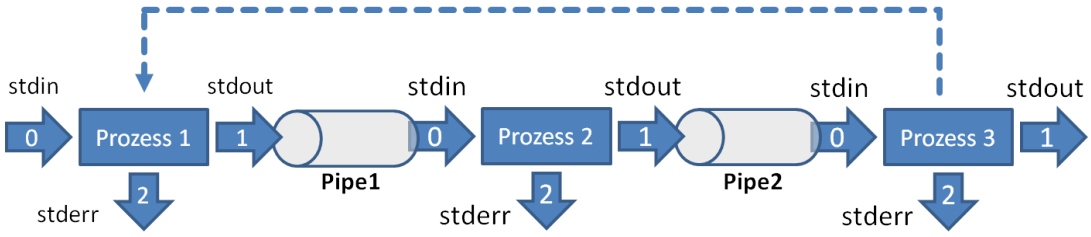
Fassen Sie erklärend zusammen, welchen Einfluss der Status von Prozess P3 auf die Prozesse P1 und P2 hat.

Name: Nils Hezrig **Studiengang:** SMSB **Gruppe:** 1

3 Praxisaufgabe (**elektronisch (!)** ausgefüllt auf Ilias abgeben / **für MIMEB optional**)

3.1 Prozesse, Prozesssteuerung und Interprozess-Kommunikation

Prozess P1 soll mit den Rechten vom Benutzer „user“ in einer Konsole laufen und mit Prozess P2 über eine benannte Pipe "pipe1" im Verzeichnis "/home"“ kommunizieren.
Die Prozesse P2 + P3 sollen mit den Rechten vom Benutzer „root“ zusammen in einer zweiten Konsole über eine unbenannte Pipe miteinander kommunizieren, wobei Prozess P3 in Abhängigkeit von bestimmten Werten mit Signalen steuernd auf Prozess P1 einwirkt (s. Abbildung).



Prozess P1	**Prozess P2**	**Prozess P3**
-Hintergrundprozess -Zählerstand mit PID ausgegeben -Zählerstand in Pipe1 umleiten -PID in „/tmp/p1.pid“ sichern	-Pipe1 auslesen -Zählerstände mit Zeichenkette 00, 25, 50 oder 75 in Pipe2 weiterleiten -frei wählbare Zeichenkette aus der Weiterleitung herausfiltern	-Zähler = 200: P1 für 10s anhalten -Zähler = 300: P1 für 10s anhalten -Zähler = 400: P1 neu starten -Zähler = 500: P1 normal beenden -Zähler > 600: P1 abbrechen
pr01.sh	**pr02.sh**	**pr03.sh**

```
[ $# != 2 ] && exit

#Signalverarbeitung
trap 'c=20' 1
trap 'c=10' 2 15

echo $$ > $1

out=/dev/stdout
[ -p "$2" ] && out=$2

for((c=20;c>15;c++))
do
  sleep 0.1
  echo "$$ send: $c"
  echo $c > $out
done

echo "normal exit ..."
```

```
[ -z "$2" ] && exit

if [ -p "$1" ]
then

  a="-E 00|25|50|75"
  b="$2"

  loop=1
  while [ $loop -eq 1 ]
  do
    grep $a $1 | grep -v $b
  done

else

  echo "no pipe ..."

fi
```

```
[ ! -e "$1" ] && exit

while read c
do
  P1=$(cat $1)
  echo "$$ receive from $P1: $c"
  case $c in
    200|300)
         kill -STOP $P1
         sleep 10
         kill -CONT $P1 ;;
    400) kill -1 $P1 ;;
    500) kill $P1 ;;
      *) echo "$c = no match" ;;
  esac
  if [ $c -gt 600 ]
  then
    kill -9 $P1
  fi
done
```

Analysieren Sie den Skript-Code der obigen Prozesse unter Berücksichtigung der Aufgabenstellung und ermitteln Sie die korrekten Aufrufe unter Beachtung der jeweiligen Benutzer. Die frei wählbare Zeichenkette im Aufruf vom Prozess P2 soll dabei verhindern, dass Prozess P3 den Prozess P1 kurzzeitig anhält oder neu startet. Testen Sie dazu ihre Lösungen ggf. in kleineren Schritten.

Kommando	**Benutzer**	**Aufruf der Kommandozeile**
Pipe erstellen	root	mkfifo -m 777 ~/pipe1
Aufruf P1	user	./pr01.sh /tmp/p1.pid ~/pipe1 &
Aufruf P2+P3	root	./pr02.sh ~/pipe1 "-E 200|300|400" | ./pr03.sh /tmp/p1.pid

3.2 Lösungs- und Fehleranmerkungen zur Praktikumsaufgabe

Anpassung der Testumgebung für Praktikum 9

Für dieses Praktikum ist es notwendig, dass die Testumgebung in der virtuellen Maschine etwas angepasst bzw. erweitert werden muss.

1. Einrichtung einer separaten Festplatte als Basis für die unterschiedlichen Dateisysteme.

Bei einer Testumgebung auf der Basis von einem „Live-Debian“ werden standardmäßig keine Festplatten benötigt und waren deshalb bisher auch nicht vorgesehen.

In diesem Praktikum sollen sich jedoch mit Dateisystemen beschäftigt werden, die logischerweise auch entsprechende Datenträger benötigt. Um diese nutzen zu können, muss vor dem Start der virtuellen Maschine eine virtuelle Festplatte hinzugefügt werden.

Das wird bei den Einstellungen „Massenspeicher“ beim ersten Controller eingerichtet, indem dort über das Icon „Festplatte hinzufügen“ und dann in dem neuen Fester das Icon „Erstellen“ gewählt wird.

Die weiteren Vorgaben können einfach übernommen werden, bei der Größe der Festplatte reicht ein Wert von 5 GB, über den Button „Erzeugen“ wird dann die Festplatte angelegt.

2. Anpassung Arbeitsspeicher

Für dieses Praktikum ist es empfohlen, den Arbeitsspeicher der virtuellen Maschine auf mindestens 2,5 GB (2.560 MB)zu erhöhen.

3. Besonderheiten bei installierten Testsystemen

Prinzipiell ist dieses Praktikum auch auf installierten Testsystemen lauffähig, wenn die benötigten Programme manuell nachinstalliert werden und man auch ansonsten genau weiß, was man tut. Denn bei diesem Praktikum kann es vorkommen, dass bei Fehlern in der Durchführung ggf. Daten verloren gehen können.

Auch wenn bei installierten Testsystemen schon eine Festplatte mit verfügbarem Platz für neue Partitionen vorhanden ist, wird der Einsatz einer separaten Festplatte für das Anlegen der entsprechenden Partitionen empfohlen, da beim Praktikum die Festplatte ggf. neu partitioniert und die Partitionen neu formatiert werden.

Wenn an dieser Stelle eine falsche Gerätedatei für die zu verwendende Festplatte angegeben wird, kann es zu Datenverlusten kommen.

Von daher wird empfohlen, sich für dieses Praktikum eine neue Testumgebung auf der Basis von einem „Live-Debian“ mit einer separaten virtuellen Festplatte anzulegen zu arbeiten.

Da es bei diesem Praktikum ggf. zu Datenverlusten bei bereits vorhandenen Datenträgern kommen kann, wird die vollständige Versuchsanleitung erst im Labor nach den einleitenden Ausführungen zu diesem Praktikum zur Verfügung gestellt.