# Inhalt

Interprozesskommunikation I

- Signale
- unbenannte Pipes
- benannte Pipes (FIFOs)

# Interprozesskommunikation

(**IPC**: inter-process-communication)

Unix-Systeme bieten folgende Möglichkeiten, Daten zwischen Prozessen auszutauschen und Prozesse zu synchronisieren:

- signals
- pipes
- FIFOs (named pipes)
- message queues
- shared memory
- named semaphores
- [lock-files]
- [sockets und RPC]

[]: wird hier nicht behandelt

# Signale (asynchronous events)

- Signale werden an Prozesse gesendet.
- Signale werden mit event-handlern bearbeitet.
- Der Empfang von Signalen kann **ignoriert** werden (nicht alle Signale).
- Der Empfang von Signalen kann zunächst **blockiert** und später bearbeitet werden (unblocking blockierter Signale).
- Es gibt **21 POSIX required signals** (`SIGABRT, SIGTERM, ...`)
- **19** dieser Signale haben **default-handler.**
- Die meisten default-handler können **überschrieben** werden (nicht **`SIGKILL`** und **`SIGSTOP`**).
- **`SIGUSR1`** und **`SIGUSR2`** sind Benutzersignale, für die es keine default-handler gibt.
- Die handler können **selbst unterbrochen** werden
  ➔ während dieser Zeit evtl. andere Signale blockieren.

- **`kill -l`** zeigt alle verfügbaren Signale an (z.Zt. 64 in Ubuntu)
- **`kill -SIGxxx pid`** sendet ein Signal von der shell an einen Prozess, z.B.:

```
$ kill -SIGSTOP pid      → anhalten
$ kill -SIGCONT pid      → fortsetzen
$ kill -SIGTERM pid      → beenden
```

- **`stty -a`** zeigt die „*signal generating characters*“ (z.B. **`Strg-C → SIGINT`**)
- Programme/Prozesse können:
  - Signale an andere Prozesse senden: **`kill(pid, sigxxx)`**
  - Signale an sich selbst senden: **`raise(sigxxx)`**
  - auf beliebiges Signal warten: **`pause()`**
  - auf bestimmte Signale warten: **`sigsuspend(maske)`**

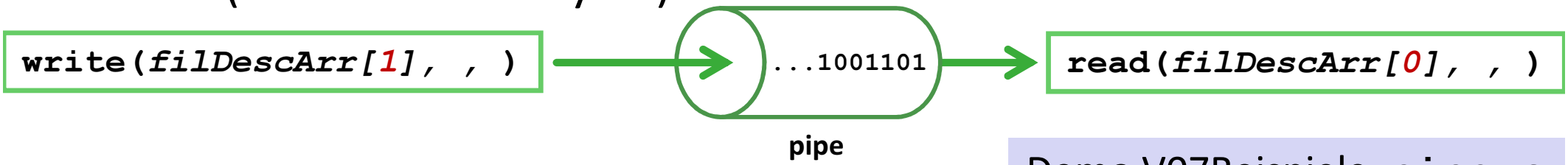
Demo V07Beispiele: **`signals.c`**

# Unbenannte Pipes

- Pipes sind *strombasierte* Kanäle (Bitstrom).
- Unbenannte pipes können nur zur Kommunikation **zwischen parent- und child-Prozessen** verwendet werden, da kein global verfügbarer Name existiert.
- Historisch gibt es verschiedene Unix-pipes.
- POSIX-pipes sind **unidirektional** (bidirektional → unvorhersehbares Verhalten)
- Die Funktion **`pipe(`*`filDescArr`*`)`** erzeugt eine pipe und liefert ein array mit den Deskriptoren der beiden Enden:

  ***`filDescArr[0]`*** → read-Ende
  ***`filDescArr[1]`*** → write-Ende

- Eine **read**-Operation:
  - **blockiert** den Prozess bei einer leeren pipe, bis am anderen Ende etwas hinein geschrieben wird
  - kehrt zurück, wenn die pipe nicht leer ist
  - **kehrt mit 0** zurück, wenn die pipe leer ist und das andere Ende von keinem Prozess geöffnet wurde
- Mit der Funktion *file control* **`fcntl(...)`** kann das blocking-Verhalten mit flags beeinflusst werden.
- Zum Lesen und Schreiben werden normalerweise die **unbuffered-I/O** file-functions **read** und **write** verwendet.
- **`write`**-Operationen auf pipes sind **atomisch**, solange nicht mehr als **`PIPE_BUF`**-bytes geschrieben werden (Ubuntu: 4096 bytes)




Demo V07Beispiele: **`pipe.c`**

# Benannte Pipes (FIFOs)

- FIFOs existieren als **special-files** im Filesystem
  ➔ globaler **Name** verfügbar als *filename*

Die Daten werden nicht in die Datei geschrieben!

- Die Funktion **`mkfifo()`** erzeugt eine FIFO-pipe.
- FIFOs sind **persistent** – sie existieren auch nach Prozessende und können weiter verwendet werden.
- FIFOs werden mit:
  - **`open(...)`** und **`close(...)`** geöffnet/geschlossen
  - **`unlink(...)`** aus dem Filesystem entfernt (aus der shell auch mit **`rm`** ***`fifoname`***). Es wird nur der Name gelöscht, geöffnete Verbindungen bleiben bestehen.
- FIFOs können von mehr als zwei Prozessen verwendet werden:

named pipe (FIFO)

– POSIX-FIFOs:

- Zuerst zum Lesen öffnen (noch kein Prozess hat zum Schreiben geöffnet) ➔ Prozess **blockiert.**
- Zum Schreiben öffnen bricht mit einem Fehler ab, wenn noch kein Prozess zum Lesen geöffnet hat (*).
- Schreibt ein Prozess in einen FIFO, dessen Lese-Ende geschlossen ist, bekommt er das Signal **`SIGPIPE`.**

– Linux FIFOs:

- können lesend und schreibend geöffnet werden, im blocking- und nonblocking-mode (http://manpages.ubuntu.com/manpages/lucid/de/man4/fifo.4.html)
- können – im Gegensatz zu * auch zuerst schreibend geöffnet werden:
  ➔ Prozess **blockiert**, bis ein Leseprozess die pipe öffnet.

Demo V07Beispiele: **`fifoA.c`** und **`fifoB.c`**