

HOWTO zu UNIX-Prozess-Schnittstellen

(C) 2007-2008 T.Birnthaler/H.Gottschalk <howtos(at)ostc.de>

OSTC GmbH, <http://www.ostc.de>

\$Id: unix-process-interfaces-HOWTO.txt,v 1.16 2011-04-04 15:29:24 tsbirn Exp \$

Dieses Dokument beschreibt die Schnittstellen von UNIX-Prozessen.

Inhaltsverzeichnis

- 1) Übersicht
- 2.1) Programmname
- 2.2) Aufrufargumente
- 2.3) Umgebungsvariablen
- 2.4) Konfigurationsdateien
- 2.5) Signale
- 2.6) Standard-Ein/Ausgabekanäle (STDIN/STDOUT/STDERR)
- 2.7) Exitstatus
- 2.8) Dateien
- 2.9) Weitere Schnittstellen

1) Übersicht

Ein UNIX-Prozess kennt folgende Schnittstellen, über die er Daten mit seiner Umgebung (das ist sein Elternprozess, seine Kindprozesse sowie alle anderen Prozesse) austauschen kann (IN=Eingabe, OUT=Ausgabe, "/"=entweder-oder, "+"=gleichzeitig):

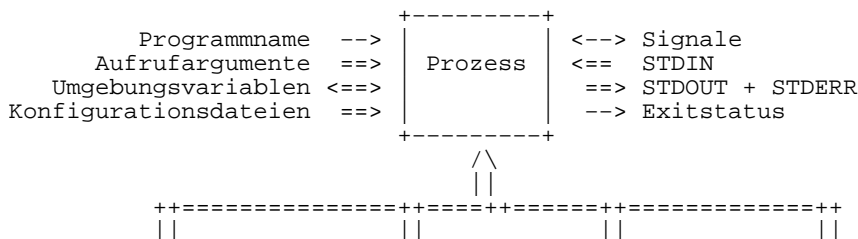
Name	Art	Datentyp
1) Programmname	IN	Ein Wort
2) Aufrufargumente	IN/OUT	Liste von Worten
3) Umgebungsvariablen	IN/OUT	Paare der Form VAR=TEXT
4) Konfigurationsdateien	IN	Byte-Strom (Text/Daten)
5) Signale	IN+OUT	Eine Zahl (0-31/63)
6a) Standard-Eingabekanal (STDIN)	IN	Byte-Strom (Text/Daten)
6b) Standard-Ausgabekanal (STDOUT)	OUT	Byte-Strom (Text/Daten)
6c) Standard-Fehlerkanal (STDERR)	OUT	Byte-Strom (Text/Daten)
7) Exitstatus	IN/OUT	Eine Zahl (0-255)
8) Dateien (Geräte, Named Pipe) Lockdateien	IN+OUT IN+OUT	Byte-Strom (Text/Daten) Flag
9) Netzwerk (Socket)	IN+OUT	Byte-Strom (Text/Daten)
10) IPC (Inter Process Communication)	IN+OUT	Byte-Strom (Text/Daten)
Named Pipe	IN+OUT	Byte-Strom (Text/Daten)
Semaphore (Vektor)	IN+OUT	Zahlen (Text/Daten)
Message	IN+OUT	Byte-Strom (Text/Daten)
Shared Memory	IN+OUT	Byte-Strom (Text/Daten)
Socket	IN+OUT	Byte-Strom (Text/Daten)
Stream	IN+OUT	Byte-Strom (Text/Daten)
11) Datenbank (Socket, Named Pipe)	IN+OUT	Byte-Strom (Text/Daten)

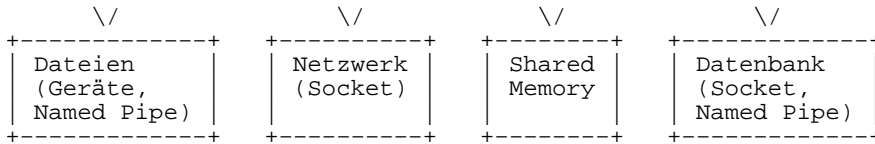
Der obere Teil umfasst die AUTOMATISCH vorhandenen Schnittstellen, die von den Prozessen ohne expliziten Aufbau einer Verbindung benutzt werden können. Oft werden dies Schnittstellen sogar "stillschweigend" benutzt, ohne dass dies großartig auffällt.

Der untere Teil der Schnittstellen muss explizit GEÖFFNET werden, d.h. im Prozess muss durch einen "open"- oder "connect"-Aufruf explizit eine Verbindung dazu hergestellt werden.

Einige Signale werden automatisch zwischen Eltern- und Kindprozessen ausgetauscht (z.B. SIGCHLD, SIGPIPE) bzw. bei bestimmten Ereignissen automatisch vom Betriebssystem an Prozesse verschickt.

Obige Schnittstellen sind in folgender Grafik nochmal übersichtlich dargestellt. "Dünne" Schnittstellen (nur EIN Wert übergebbar) sind durch dünne Pfeile -->, "dicke" Schnittstellen (mehr als ein Wert übergebbar) sind durch dicke Pfeile ==> gekennzeichnet.

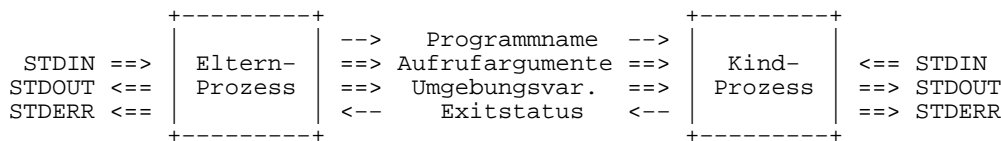




Folgende Schnittstellen bieten eine Datenübergabe nur in EINER Richtung und sind beschränkt auf bestimmte Prozess-Zusammenhänge:

Schnittstelle	Beschränkung
Programmname	Elternprozess ---> Kindprozess
Aufrufargumente	Elternprozess ==> Kindprozess
Umgebungsvariablen	Elternprozess ==> Kindprozess
Exitstatus	Elternprozess <--- Kindprozess
STDIN	Elternprozess + Kindprozesse (oder Geschwister)
STDOUT	Elternprozess + Kindprozesse (oder Geschwister)
STDERR	Elternprozess + Kindprozesse (oder Geschwister)

In grafischer Form:



2.1) Programmname

Jedem Prozess ist der Programmname bekannt, über den er gestartet wurde. Dieser Programmname ist Teil der Aufrufargumente und zwar das 0-te Argument. Normalerweise würde man erwarten, dass der Name eines Programms nur durch Umbenennen veränderbar ist (was wenig sinnvoll erscheint). Unter UNIX können allerdings in Form von Hardlinks und Symbolischen Links beliebig viele weitere Namen für ein Programm vergeben werden ("ln" bzw. "ln -s"). Der Programmstart über einen Link führt dazu, dass der Programmname nicht mehr dem eigentlichen Programmnamen entspricht, sondern dem Linknamen.

Über diese "Argument" wird häufig das Grundverhalten des Kommandos gesteuert. Beispielsweise kennt das Programm "bzip2" die zusätzlichen Namen "bunzip2" und "bzcat". Ein weiteres typisches Beispiel ist das LVM-System (Logical Volume Management), das ein einziges Hauptprogramm namens "lvm" und 40 Unterkommandos (namens "pv/vg/lv...") kennt, die alle als symbolische Links auf dieses Hauptprogramm realisiert sind.

2.2) Aufrufargumente

Umfassen alle beim Aufruf eines Programms hinter dem Programmnamen auf der Kommandozeile angegebenen Optionen ("-x" bzw. "--xxx") und Argumente. Die erlaubte Maximallänge dieser Liste ist sehr groß, aber nicht unbeschränkt (etwa 100.000-2 Mio Zeichen). Der Programmname und die einzelnen Aufrufargumente werden durch "Whitespace" getrennt (-> Quotierung). Beispiel sind die vielen Optionen des Befehls "ls" und der Quell- und Zieldateiname des Befehls "mv".

Trennung durch Leerzeichen bei der Shell -> Vorsicht!

2.3) Umgebungsvariablen

Jeder Prozess hat einen Umgebungsbereich (Environment), in dem er Variablen + ihre Werte hinzufügen, abfragen, ändern und löschen kann. Diese Variablen werden meistens GROSS geschrieben. Die Größe dieses Bereiches ist unbeschränkt, typischerweise stehen darin etwa 100-200 Variablen.

Diese Liste von Umgebungsvariablen mit ihren Werten wird von jedem Prozess beim Start eines Kindprozesses an diesen vererbt, indem zu diesem Zeitpunkt eine KOPIE der Liste erstellt und dem Kindprozess zugeordnet wird.

Jeder Prozess hat nur auf seinen eigenen Umgebungsbereich Zugriff, er kann keine Variablen+Werte im Umgebungsbereich anderer Prozesse ändern. Mit dem Ende eines Prozesses verschwindet auch sein Umgebungsbereich.

Beispiel ist die Variable \$HOME, die den Befehl "cd" beeinflusst, die Variable "\$PATH" zur Steuerung der Suche nach Programmen und die Variable "\$LANGUAGE", mit der die Spracheinstellung vieler Programm festlegt wird.

2.4) Konfigurationsdateien

Die meisten Programme lesen beim Start (und evtl. auch beim Empfang des Signals

SIGHUP=1) einen fest definierten Satz von Konfigurationsdateien ein. Diese Konfigurationsdateien heißen in der Regel so wie das Programm selbst und liegen zentral im "/etc"-Verz. bzw. in den Heimatverz. der Benutzer in Form von "versteckten" Dateien und/oder Verz. (der Datei/Verz.name beginnt mit ".").

Zuerst werden für alle Benutzer die zentralen (globalen) Konfigurationsdateien, dann wird pro Benutzer dessen benutzerspezifischen (lokalen) aus seinem Heimatverz. eingelesen. Mit letzteren werden die zentralen Einstellungen überschrieben. Hier als Beispiel die Konfigurationsdateien der Bourne-Shell "sh":

```
/etc/profile # zentral
~/.profile  # benutzerspezifisch
```

Oder für den Editor "emacs":

```
/etc/emacs/* # zentral
~/.emacs.d/* # benutzerspezifisch
```

Oder für den SSH-Client "ssh":

```
/etc/ssh/ssh_config
~/.ssh/ssh_config
```

Fehlen Konfigurationsdateien, so wird dies in der Regel stillschweigend übergangen und das im Programm eingebaute Standardverhalten aktiviert.

Werden Einstellungen in einer Konfigurationsdateien geändert, so ist dies dem laufenden Prozess per Signal SIGHUP (1) mitzuteilen oder er ist zu anzuhalten und neu zu starten.

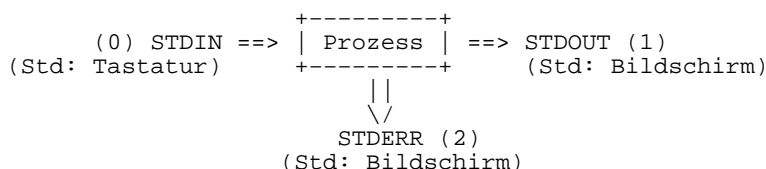
2.5) Signale

Ein Prozess kann einen festen Satz von Signalen (Nachricht bestehend aus einer Nummer von 0-31/63) von anderen Prozessen empfangen bzw. an andere Prozesse senden ("kill"). Über die (effektive) UID/GID von Sende- und Empfangsprozesse und über den "Signal-Handler" des Empfangsprozesses wird geregelt, ob ein Prozess darauf reagiert oder nicht. Ein Signal kann im empfangenden Prozess abhängig von der Signalnummer bestimmte Reaktionen auslösen (z.B. liest SIGHUP oft die Konfigurationsdatei neu ein), ihn per abbrechen (z.B. per SIGKILL), das Signal kann aber auch ignoriert werden. Es gibt 32 bzw. 64 Signale, die wichtigsten sind:

Name	Nr	Bedeutung
SIGEXIT	0	"Ende des Skripts" in der Bash
SIGHUP	1	Konfigurationsdatei erneut einlesen (Daemon) [hangup]
SIGINT	2	Abbrechen durch <Strg-C> [interrupt]
SIGKILL	9	Bedingungsloser Prozessabbruch
SIGTERM	15	Prozess beenden (Standard-Signal von "kill") [terminate]
SIGCONT	18	Ausgabe weiterlaufen lassen durch <Strg-Q>
SIGSTOP	19	Ausgabe anhalten durch <Strg-S>
SIGTSTP	20	In Hintergrund stellen durch <Strg-Z>
SIGQUIT	3	Prozessende erreicht
SIGILL	4	Nicht erlaubte Anweisung gefunden [illegal]
SIGBUS	7	Bus-Zugriffsverletzung (Alignment)
SIGFPE	8	Mathematik-Fehler [floating point exception]
SIGSEGV	11	Fehlerhafter Speicherzugriff [segment violation]
SIGPIPE	13	Prozess in Pipeline nicht mehr da [broken pipe]
SIGALRM	14	Timeout hat stattgefunden [alarm]
SIGCHLD	17	Ein Kindprozess hat sich beendet [child]

2.6) Standard-Ein/Ausgabekanäle (STDIN/STDOUT/STDERR)

Über die Standard-Ein/Ausgabekanäle kann jeder Prozess Daten mit anderen Prozessen austauschen (meist mit der Shell). In der Regel handelt es sich dabei um zeilenorientierte ASCII-Texte, aber auch Binärdaten können übertragen werden. Sie eignen sich auch für die effiziente Übergabe sehr großer Datenmengen:

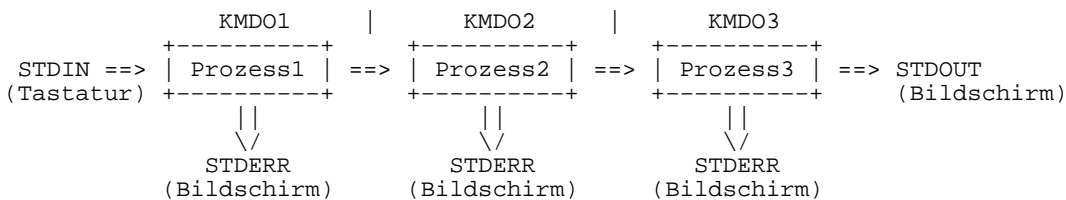


Ein Prozess behandelt die drei Kanäle wie bereits geöffnete sequentielle Dateien, aus denen er ausschließlich lesen oder in die er ausschließlich schreiben kann. D.h. darin neu positionieren ist nicht möglich, bereits gelesene Daten können nicht erneut gelesen und bereits geschriebene Daten können nachträglich nicht mehr verändert werden.

Diese Kanäle bilden die Grundlage der Dateiumlenkung und des Pipemechanismus unter UNIX. Standardmäßig ist STDIN mit der Tastatur verbunden und STDOUT sowie STDERR mit dem Bildschirm. Die aktuelle Belegung der drei Kanäle wird von einem Elternprozess (meist die Shell) an seine Kindprozesse vererbt. Sie sind aber EINMAL beim Start eines Prozesses auch auf andere Dateien/Geräte umlenkbar. In der Shell geschieht diese Umlenkung durch folgende Anweisungen:

```
KMDO < DATEI      # Eingaben von STDIN von DATEI lesen
KMDO > DATEI      # Ausgaben auf STDOUT auf DATEI schreiben
KMDO 2> DATEI     # Ausgaben auf STDERR auf DATEI schreiben
KMDO1 | KMDO2    # STDOUT von KMDO1 mit STDIN von KMDO2 verbinden
```

Mehrere Kommandos lassen sich über ihre Standard-Ein/Ausgabekanäle leicht zu einer "Filterkette" oder "Pipeline" zusammensetzen, indem per Pipe-Symbol "|" die STDOUT- und die STDIN-Kanäle der Kommandos miteinander verbunden werden.



Die Fehlerkanäle STDERR der verbundenen Prozesse werden dabei nicht mit in den Datenstrom eingeschleust, sondern bleiben getrennt pro Prozess und werden normalerweise auf dem Bildschirm ausgegeben. Sollen sämtliche Ein/Ausgabedaten und die Fehlermeldungen von/auf Datei gelesen/geschrieben werden, so sind pro Kommando noch Dateiumlenkungen anzugeben:

```
KMDO1 < INPUT 2> ERR1 | KMDO2 2> ERR2 | KMDO3 2> ERR3 > OUTPUT
```

2.7) Exitstatus

Wird von jedem Prozess am Ende seiner Ausführung an seinen Elternprozess zurückgegeben. Der Wert "0" bedeutet, dass der Prozess erfolgreich ausgeführt wurde. Ein Wert ungleich "0" bedeutet, dass ein Fehler auftrat. Diese Fehlernummern sind für jedes Programm unterschiedlich definiert (siehe zugehörige man-Page).

Der Exitstatus kann vom aufrufenden Programm (Elternprozess, meist die Shell) abgefragt und abhängig davon unterschiedlich reagiert werden. Existiert sein Elternprozess nicht mehr, dann bleibt ein Kindprozess so lange als "Zombie" am Leben, bis er vom Betriebssystem mit dem "init"-Prozess verbunden wird und dort seinen Exitstatus abliefern kann.

2.8) Dateien

Ein Prozess kann (fast) beliebig viele Dateien öffnen, auf die er je nach Dateityp, Zugriffsmodus und Zugriffsrechten (auf Basis der effektive UID/GID von Prozess und Datei) sequentiellen/wahlfreien lesenden und/oder schreibenden Zugriff hat. Als "Dateien" sind auch "Named Pipes" und "Geräte" (Hardware) möglich.

2.9) Weitere Schnittstellen

Weitere Schnittstellen von UNIX-Programmen:

- * Dateien (Geräte, Named Pipe)
 - + Lockdateien
- * IPC (Inter Process Communication)
 - + Named Pipe
 - + Semaphore
 - + Message
 - + Shared Memory
 - + Stream
- * Datenbank (Socket, Named pipe)
- * Netzwerk
 - + Socket

Diese Schnittstellen müssen explizit GEÖFFNET werden, d.h. es muss im Programm explizit eine Verbindung dazu hergestellt werden. Im Gegensatz dazu sind alle vorher beschriebenen Schnittstellen AUTOMATISCH vorhanden und können sofort benutzt werden, ohne sich um den Aufbau einer Verbindung zu kümmern.