

Hilfsblatt zu Praktikumsaufgabe 5

Einfache Prozesskommunikation: unidirektionale, anonyme Pipes.

Dieses Hilfsblatt beschreibt einige Grundlagen zur Prozesskommunikation unter POSIX.
Es wird ausschließlich auf Kommunikation über Pipes eingegangen!
Schauen Sie sich die benutzten Systemaufrufe zusätzlich in der Dokumentation an!

Für die Praktikumsaufgabe müssen die beiden hier vorgestellten Mechanismen kombiniert werden!

Pipes:

Eine Pipe stellt eine einfache Kommunikationsschnittstelle dar. Sie besteht aus zwei Dateideskriptoren.

Die Daten werden in das eine Ende der Pipe geschrieben und können aus dem anderen Ende nach dem FIFO Prinzip ausgelesen werden.

Da ein Kindprozess die geöffneten Dateien vom Elternprozess erbt, können somit Informationen zwischen Eltern und Kindprozess ausgetauscht werden bzw. zwischen mehreren Kindern, die alle die Pipe erben können.

Syntax:

```
#include <unistd.h>

int pipe(int fildes[2]);
```

Parameter:

`fildes[2]` : Ein Zeiger auf ein Feld von zwei Dateideskriptoren.

Es werden zwei Dateien geöffnet. Werden Daten in `fildes[1]` geschrieben, können sie aus `fildes[0]` gelesen werden.

In einigen Systemen ist die Pipe bidirektional (beide Dateien sind zum lesen und schreiben geöffnet), POSIX spezifiziert aber nur `fildes[1]` als Schreibdeskriptor und `fildes[0]` als Lesedeskriptor.

(In vielen Systemen können die geöffneten Dateien und damit auch die Pipes eines Prozesses mit der Prozess Id PID unter `/proc/<PID>/fd` eingesehen werden.)

Rückgabewert:

0 bei Erfolg, sonst -1.

Nach Benutzung der Pipe müssen die mit `pipe()` erzeugten Deskriptoren wieder mit `close()` geschlossen werden!

Duplizieren eines Dateideskriptors:

Wird ein `exec<l|v>[e|p]` Befehl aufgerufen, werden normalerweise alle offenen Dateien des Prozesses geschlossen (close-on-exec).

Um Dateideskriptoren zu erzeugen, die auch einen Aufruf eines `exec` Befehls überleben, dienen die Befehle `dup()` bzw. `dup2()`.

Syntax von `dup()`:

```
#include <unistd.h>

int dup(int fildes);
```

Parameter:

`fildes`: Der Dateideskriptor, der dupliziert werden soll.

Rückgabewert:

Bei Erfolg wird ein Deskriptor zurückgeliefert, der auf die Datei (oder Pipe) referenziert, auf die auch `fildes` zeigt (beide benutzen dieselbe interne Struktur). Der Deskriptor überlebt den `exec` Befehl, d.h. ein mit `exec` gestartetes Programm kann diesen Deskriptor benutzen.

Der zurückgelieferte Deskriptor ist der kleinste freie Deskriptor im System.

Bei Misserfolg wird -1 zurückgegeben.

Anwendungsbeispiel:

Soll z.B. die Ausgabe eines Befehls in eine Datei umgeleitet werden, kann wie folgt verfahren werden:

```
/* Im Beispiel erfolgt keine Fehlerbehandlung! */
/* Machen Sie es besser! */
int fd;
fd = open("TestDatei.dat", O_RDWR | O_CREAT | O_TRUNC);
/* stdout Strom wird geschlossen. */
/* Damit ist die 1 als Deskriptor frei. */
close(STDOUT_FILENO);

/* mit dup() wird der Deskriptor 1 der Testdatei zugewiesen */
dup(fd);

/* ls wird aufgerufen, Ausgabe geht in Testdatei! */
execlp("ls", "ls", "-l", ".", (char *)NULL);
```

Syntax von `dup2()`:

```
#include <unistd.h>

int dup2(int fildes, int fildes2);
```

Parameter:

`fil-des`: Der Dateideskriptor, der dupliziert werden soll.

`fil-des2`: Zeigt nach Erfolg auf dieselbe Datei wie `fil-des`. Ist `fil-des2` eine geöffnete Datei, so wird diese zuvor geschlossen.

Rückgabewert:

Siehe `dup ()`

Anwendungsbeispiel:

Soll z.B. die Ausgabe eines Befehls in eine Datei umgeleitet werden, kann wie folgt verfahren werden:

```
/* Im Beispiel erfolgt keine Fehlerbehandlung! */
/* Machen Sie es besser! */
int fd;
fd = open("TestDatei.dat", O_RDWR | O_CREAT | O_TRUNC);

/* mit dup2() wird stdout der Testdatei zugewiesen */
dup2(fd, STDOUT_FILENO);

/* ls wird aufgerufen, Ausgabe geht in Testdatei! */
execlp("ls", "ls", "-l", ".", (char *)NULL);
```

Wie die Beispiele zeigen, hat der Ausgabestrom `stdout` einen eigenen Dateideskriptor namens `STDOUT_FILENO`.

in der Headerdatei `unistd.h` sind folgende Dateideskriptoren vordefiniert:

```
#define STDIN_FILENO    0 /* standard input file descriptor */
#define STDOUT_FILENO   1 /* standard output file descriptor */
#define STDERR_FILENO   2 /* standard error file descriptor */
```

Diese Deskriptoren sind standardmäßig für jeden Prozess geöffnet und können von Ihren Programmen benutzt werden.