

# Prozesse klonen mit fork()

Inhalt:

- \* Houston, wir haben ein Problem!
- \* Was ist ein Prozess?
- \* Prozesse klonen mit fork()
  - \* So setze ich fork() ein:
- \* Interprozesskommunikation
- \* Ergänzungen, Kommentare
  - \* fork() in anderen Sprachen

## Houston, wir haben ein Problem!

Wir sollen 100 mal das gleiche machen - aber bitte gleichzeitig! Ein Beispiel, wo so etwas vorkommen könnte ist ein Server, der 100 Verbindungen gleichzeitig eingehen kann. Oder es sollen in der Bioinformatik 100 Sequenzen gleichzeitig berechnet werden.

Um das Prinzip zu erklären, wie man dieses Problem lösen kann, werde ich mich auf die Erzeugung von 2 parallelen Prozessen beschränken.

## Was ist ein Prozess?

Ein Prozess ist ein laufendes Programm. Er besteht aus

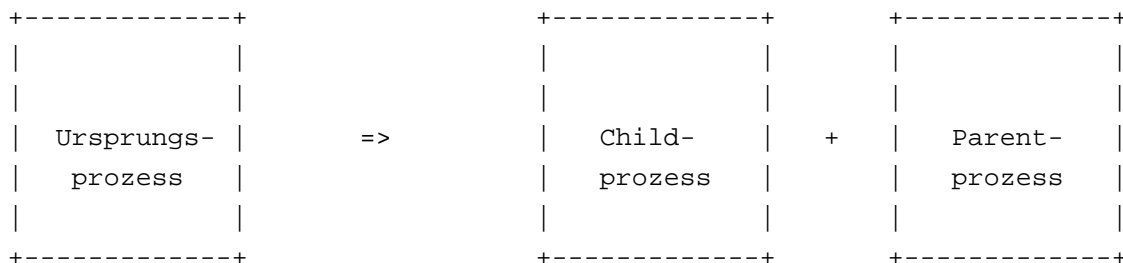
- \* statischen Daten wie TEXT, BSS, DATA
- \* dynamischen Daten wie STACK, HEAP, Register-Belegung,...

## Prozesse klonen mit fork()

Ein Prozess kann mit dem Systemaufruf fork() geklont werden. Das heißt, es wird eine identische Kopie des ursprünglichen Prozesses erzeugt. fork() erwartet keine Parameter und gibt einen numerischen Wert (Integer) zurück. Das Duplikat, das als Child-Prozess bezeichnet wird, übernimmt die aktuellen Werte aller Variablen und weiterer Datenstrukturen.

Man kann sich das ganze bildlich so vorstellen: Ein Mann lässt sich klonen. Der Klon erwacht im Zimmer nebenan und hat die gleichen Erinnerungen wie das Original; einschließlich dem Betreten des anderen Zimmers, in dem noch das Original ist. Die Kopie denkt dann "Ups, bin ich vorhin nicht in das andere Zimmer gegangen? Und wer ist der nette Herr in dem anderen Zimmer?".

Original und Kopie arbeiten aber ab dem fork() unabhängig voneinander. Das heißt, die Kopie kann in die Stadt einkaufen gehen, ohne dass das Original etwas davon mitbekommt - und umgekehrt. Ab dieser Stelle hat also jeder der beiden Prozesse alle bisher benutzten Variablen mit ihren aktuellen Belegungen zur Verfügung; Änderungen dieser Belegungen wirken sich aber nur noch innerhalb des eigenen Prozesses aus, der andere Prozess bekommt davon nichts mit. Nur Filehandles werden nicht kopiert - beide Prozesse schreiben und lesen aus den bzw. in die selben Filehandles.



Um Probleme zu vermeiden, müssen Parent- und Child-Prozess wissen, wer von ihnen wer ist. Dieses Problem wird mit den sogenannten Prozess-IDs (kurz: PID) gelöst. Dies sind eindeutige positive Ganzzahlen.

## So setze ich fork() ein:

```
#!/usr/bin/perl
```

```

use strict;
use warnings;

my $x = 4;
$x++;

my $pid = fork();

if($pid == 0){
    # Hier bin ich im Child-Prozess
    $x++;
    print "Ich bin das Kind:  ", $$, "\tx:$x\n";
    exit(0);
}
else{
    # Hier bin ich im Parent-Prozess
    print "Ich bin der Vater:  ", $$, "\tx:$x\n";
    wait();
}

```

Ausgabe:

```

(reneeb) Mo Mär 21 14:12:54 [/bin/bash]
~/entwicklung 18> perl fork.pl
Ich bin das Kind:  23160          x:6
Ich bin der Vater: 23159          x:5

```

## Interprozesskommunikation

Diese Unabhängigkeit kann aber auch zu einem Problem werden. Was ist, wenn Original und Kopie miteinander kommunizieren müssen?

Die Kommunikation kann man über verschiedene Wege sicherstellen. Z.B. über Shared Memory, Signale oder Pipes.

Pipes kann man sich als "Röhre" vorstellen, über die in nur eine Richtung kommuniziert werden kann.

Dies sieht so aus:

```

#! /usr/bin/perl

```

```

use strict;
use warnings;

pipe(READER,WRITER);

my $pid = fork();

if($pid == 0){
    # Hier bin ich im Child-Prozess
    close READER;
    my $ort = 'bei Freunden';
    print "Ich bin ", $ort, "\n";
    print WRITER $ort;
    exit(0);
}

```

```

else{
  # Hier bin ich im Parent-Prozess
  close WRITER;
  while(my $line = <READER>){
    print "Ach, Du bist also ",$line,"\n";
  }
  wait();
}

```

Ausgabe:

```

(reneeb) Mo Mär 21 14:27:32 [/bin/bash]
~/entwicklung 21> perl fork.pl
Ich bin bei Freunden
Ach, Du bist also bei Freunden

```

Der Parent-Prozess lauscht so lange an der Pipes, bis der Child-Prozess etwas in die Pipe schreibt.

## Ergänzungen, Kommentare

### fork() in anderen Sprachen

C++:

1. Programm:

```

#include<iostream> // fuer cout
#include<unistd.h> // fuer getpid fork

using namespace std;

int main(){

  int x = 4;
  x++;

  int pid;
  pid=fork(); // Prozess teilen

  if(pid<0){ // wenn Fehler bei fork
    cout << "Fehler bei fork" << endl;
    exit(1);
  }

  if(pid==0){ // wenn Kind
    x++;
    cout << "Ich bin das Kind: " << getpid() <<"\tx: " << x << endl;
    exit(0); // Kind mit 0 beenden
  }
  else if(pid>0){ // wenn Vater
    cout << "Ich bin der Vater: " << getpid() <<"\tx: " << x << endl;
    exit(0); // Vaterl mit 0 beenden
  }
}

```

Ausgabe:

```

(reneeb) Thu Mar 24 15:22:13 [-bash]

```

```
~/entwicklung 181> fork
```

```
Ich bin der Vater: 14362      x: 5
```

```
Ich bin das Kind:  14363      x: 6
```

## 2. Programm:

```
#include<iostream> // fuer cout
```

```
#include<unistd.h> // fuer getpid fork
```

```
using namespace std;
```

```
int main(){
```

```
    int fds[2];
```

```
    pipe(fds);
```

```
    char ort[] = "bei Freunden";
```

```
    int BUFSIZE = strlen(ort);
```

```
    int pid;
```

```
    pid=fork(); // Prozess teilen
```

```
    if(pid<0){ // wenn Fehler bei fork
```

```
        cout << "Fehler bei fork" << endl;
```

```
        exit(1);
```

```
    }
```

```
    if(pid==0){ // wenn Kind
```

```
        close(fds[0]);
```

```
        cout << "Ich bin " << ort << endl;
```

```
        write(fds[1],ort,BUFSIZE);
```

```
        exit(0); // Kind mit 0 beenden
```

```
    }
```

```
    else if(pid>0){ // wenn Vater
```

```
        // die Standardeingabe des Vaters mit der lesbaren Seite der Pipe
```

```
        // (also fds[0]) verbunden. Dazu dient die Systemfunktion dup2()
```

```
        // Der Wert 0 steht dabei für den Filedeskriptor der Standardeingabe.
```

```
        // Das andere Ende der Pipe wird im Kind nicht benötigt und daher geschlossen
```

```
        dup2(fds[0],0);
```

```
        close(fds[1]);
```

```
        int length;
```

```
        char antwort[BUFSIZE];
```

```
        do{
```

```
            read(fds[0],antwort,BUFSIZE);
```

```
            cout << "Du bist also bei " << antwort << endl;
```

```
        }while(length > 0);
```

```
        wait(0); // Vaterl mit 0 beenden
```

```
    }
```

```
}
```

## Ausgabe:

```
(reeneb) Thu Mar 24 15:22:13 [-bash]
```

~/entwicklung 186> fork

Ich bin bei Freunden

Du bist also bei bei Freunde

Java:

In Java gibt es erst seit Java 5 eine Umsetzung von fork(). Dort heißt die Klasse

ProcessBuilder

?

- ReneeBaecker 24 Mar 2005

Kommentare werden am besten in folgender Form vorgenommen, damit sie im Inhaltsverzeichnis angezeigt werden (natürlich ohne das <verbatim>):

----- User.??? - 14 Jul 2003 - Betreff