

2. Prozeßverwaltung

- 2.1 Prozeßerzeugung und -verwaltung
- 2.2 Scheduling
- 2.3 Thread-Systeme
- 2.4 Kommunikation

2.1 Einführung

Ein **Prozeß** (*process, task*) ist ein in Ausführung befindliches Programm bestehend aus

- einer Folge von Maschinenbefehlen, die durch das ausgeführte Programm festgelegt sind (**program code, text section**) ,
- dem Inhalt des Stapel-Speichers (**stack**), auf dem temporäre Variablen und Parameter für Funktionsaufrufe verwaltet werden,
- dem Inhalt des Speichers, in dem die globale Daten des Prozesses gehalten werden (**data section**) und
- Verwaltungsinformationen, die den aktuellen **Bearbeitungszustand (Kontext)** beschreiben.

Bemerkung: Der aktuelle Bearbeitungszustand wird u.a. durch den Programmzähler und die Registerinhalte des Prozessors beschrieben (engl. *internal state*).

Prozeßverwaltung:

- Die Verwaltungsinformationen speichert das Betriebssystem in der **Prozeßtabelle**, die pro Prozeß einen Eintrag enthält.
- Ein Eintrag in der Prozeßtabelle heißt **Prozeßkontrollblock** oder auch Prozeßleitblock (engl. process control block).

Ein Prozeßkontrollblock beinhaltet:

- Prozeßidentifikationsnummer (pid)
- Platz für Registerinhalte des Prozessors (CPU)
- Platz für Programmzähler
- Zeiger auf Text-, Daten-, Stacksegment
- Daten über Ein-/ Ausgabegeräte, die dem Prozeß zugeordnet sind
- Daten über geöffnete Dateien und Netzwerkverbindungen
- **Prozeßzustand**
- Daten, die die Bearbeitungsfolge regeln (**Prioritäten**)
- Accounting-Daten über benutzte Betriebsmittel/Ressourcen (z.B. die bisher verbrauchte CPU-Zeit)
- Parameter, die die Zugriffsrechte des Prozesses beschreiben

Man unterscheidet:

Hardwarekontext: Registerinhalte, Befehlszähler, . . .

Softwarekontext: Prozeß-ID, Prozeßzustand, . . .

Datenstruktur für die Realisierung der Prozeßtabelle: meist als doppelt verkettete Liste

Definition 12:

Ein Prozeß befindet sich in genau einem der folgenden Zustände:

rechnend: Der Prozessor ist dem Prozeß zugeteilt.

rechenbereit: Der Prozeß ist ausführbar und wartet auf Prozessorzuteilung.

blockiert: Der Prozeß wartet auf ein Ereignis, z.B. auf Eingabe, Ausgabe oder ein Semaphor.

Abnormales Prozeßverhalten

Definition 13:

Ein Prozeß befindet sich in einem:

Deadlock := Der Prozeß befindet sich im Zustand blockiert und es tritt kein Zustandswechsel mehr ein.

Livelock := Der Prozeß wechselt zwar zwischen den Zuständen, kommt aber in seiner Programmausführung nicht weiter.

Starvation := Man sagt ein Prozeß **verhungert**, falls er sich im Zustand rechenbereit befindet und es tritt kein Zustandswechsel ein.

Unterbrechungen

Unterbrechungen sind das Schlüsselkonzept zur effizienten Ablaufsteuerung.

Definition 14:

- ① Hardware Interrupts (auch **asynchrone Unterbrechung**) werden von externen Ereignissen außerhalb des laufenden/rechnenden Prozesses verursacht.
- ② Software Interrupts (auch: Traps, Program Interrupts) (**synchrone Unterbrechung**) werden von dem laufenden/rechnenden Prozeß selber verursacht.

Beispiele:

① Hardware Interrupts (asynchrone Unterbrechung)

- I/O-Controller (Platte, Tastatur, Network Interface Card (NIC))
- Timer: ermöglicht dem Betriebssystem periodisch bestimmte Arbeiten durchzuführen

② Software Interrupts (auch: Traps, Program Interrupts) (synchrone Unterbrechung)

- Systemdienstaufruf (System Call): `read`, `write`, `open`, `close`, ...
- Division durch Null
- ungültige Speicheradresse

E. W. Dijkstra: *My recollections of operating system design*, Operating Systems review, Vol. 39, Number 2, 2005.

Definition 15:

Der **Unterbrechungsvektor** enthält für jeden Unterbrechungstyp (Disk-Interrupt, Terminal-Interrupt, ...) die Adresse einer zugehörigen Unterbrechungsbehandlungsroutine.

Beispiel für eine asynchrone Unterbrechung:

Prozeß A rechnet, Prozeß B wartet auf Eingabe von der Platte.

Es erfolgt ein Disk-Interrupt, wenn die Daten für Prozeß B eingelesen sind.

Ablauf:

- ① Programmzähler etc. von Prozeß A werden auf den Stack geschrieben (Hardware).
- ② Unterbrechungsbehandlungsroutine wird geladen.
- ③ Unterbrechungsbehandlungsroutine schreibt die aktuellen Registerinhalte in den Prozeßleitblock, der Stack wird rekonstruiert (Prozeßzustand von A := rechenbereit) (Assembler).
- ④ C-Routine setzt Prozeß B auf rechenbereit.
- ⑤ Scheduler wird aktiv.

Erzeugung von Prozessen

- Der erste Prozeß wird beim Systemstart vom Betriebssystem erzeugt.
- Neue Prozesse können von einem existierenden Prozeß durch spezielle Systemaufrufe erzeugt werden.
- POSIX-Systeme benutzen hierzu den `fork()` Systemaufruf.
Der neu erzeugte Prozeß (**child**) ist eine **Kopie** des erzeugenden Prozesses (**parent**), der **nebenläufig** zum erzeugenden Prozeß läuft.
- Da beliebige Prozesse neue Prozesse erzeugen können, entsteht eine Hierarchie von Prozessen.

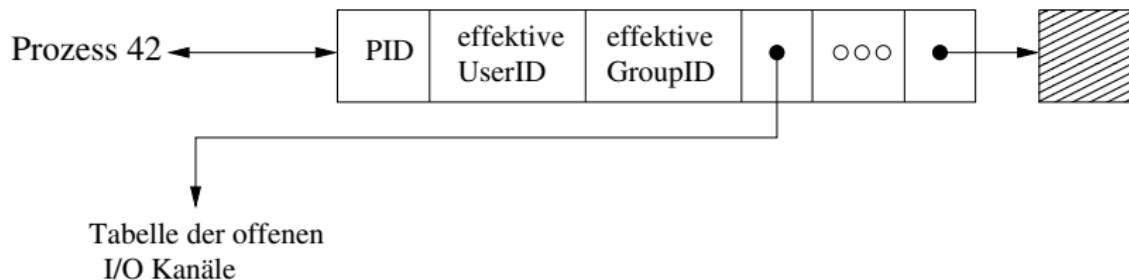
Anekdote: Das B-S-D Maskottchen Beastie



Hier eine Version des BSD Maskottchen von John Lasseter (Chief Creator Officer bei Pixar). Es tauchte erstmals 1988 auf dem Buch-Cover auf.

Prozesskontrollblock

virtueller Adressraum



Descriptor	Beschreibung
0	stdin
1	stdout
2	stderr
○	weitere Einträge durch open(),
○	socket(), pipe(), ...

Prozesskontrollblock

virtueller Adressraum

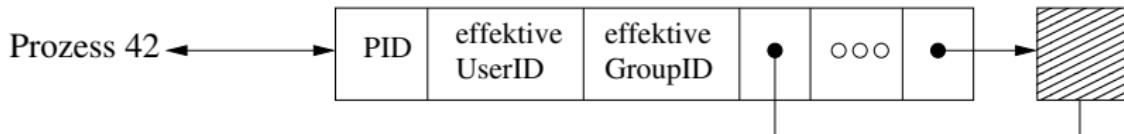


Tabelle der offenen
I/O Kanäle

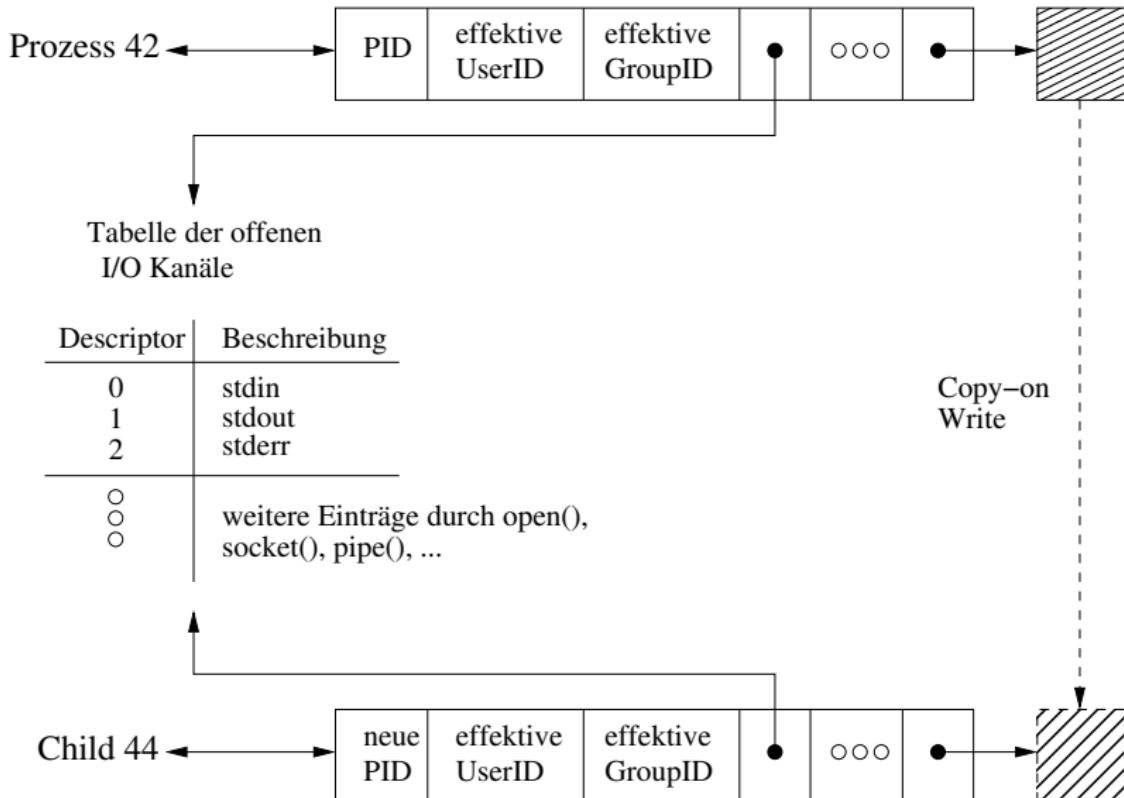
Descriptor	Beschreibung
0	stdin
1	stdout
2	stderr
○ ○ ○	weitere Einträge durch open(), socket(), pipe(), ...

Kopie

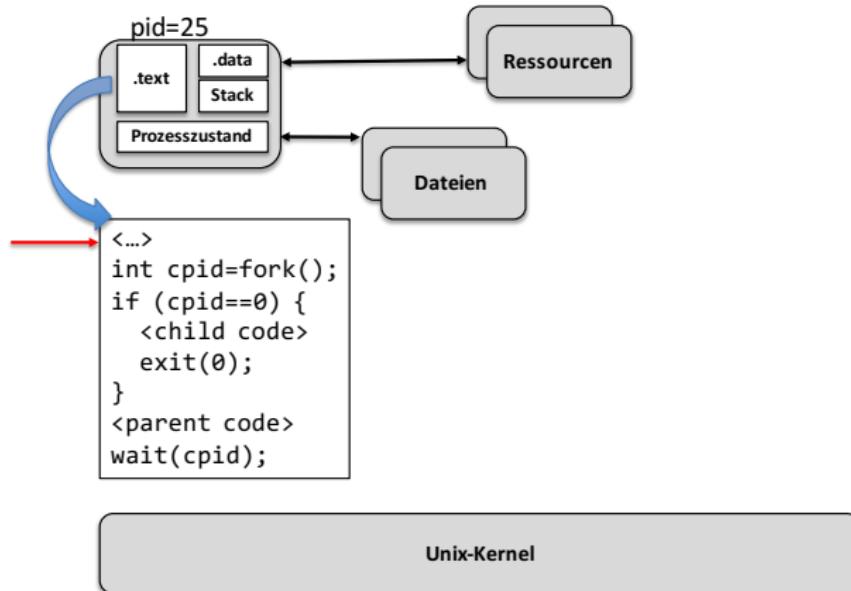


Prozesskontrollblock

virtueller Adressraum

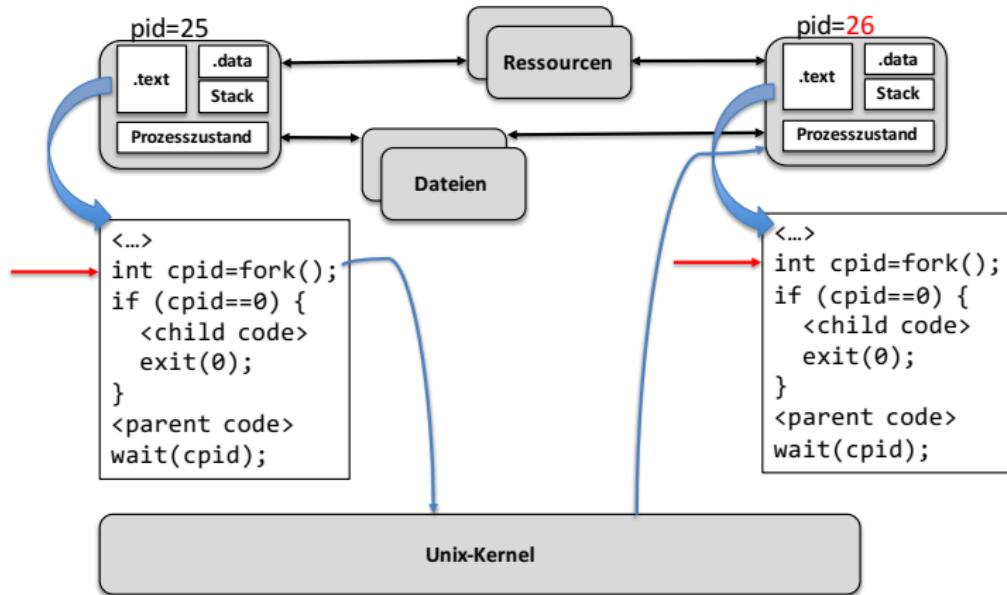


Ablauf von fork⁹:



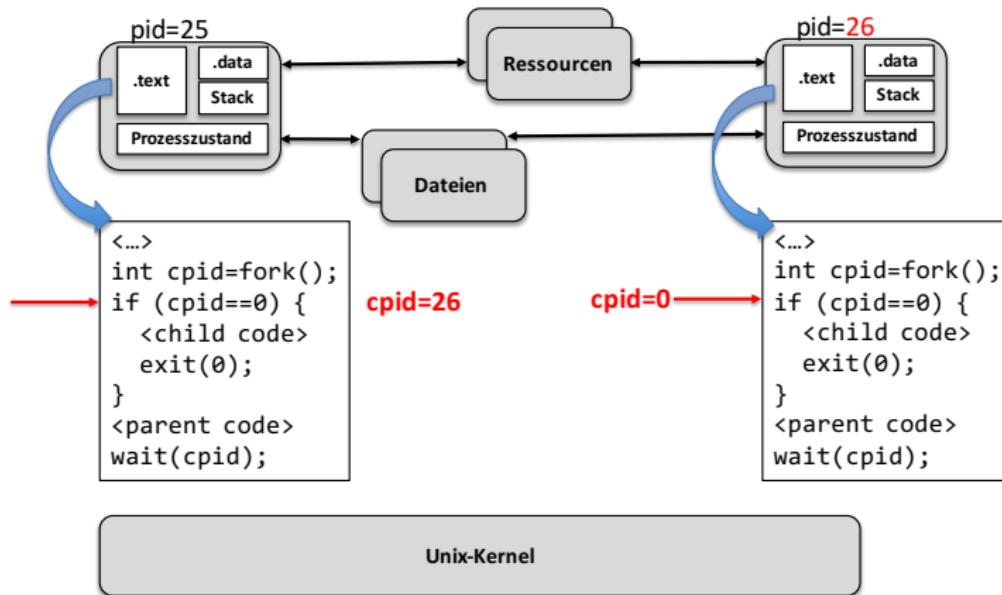
⁹Einfaches Beispiel von Michael Engel ohne execve() im <child code>

Ablauf von fork:



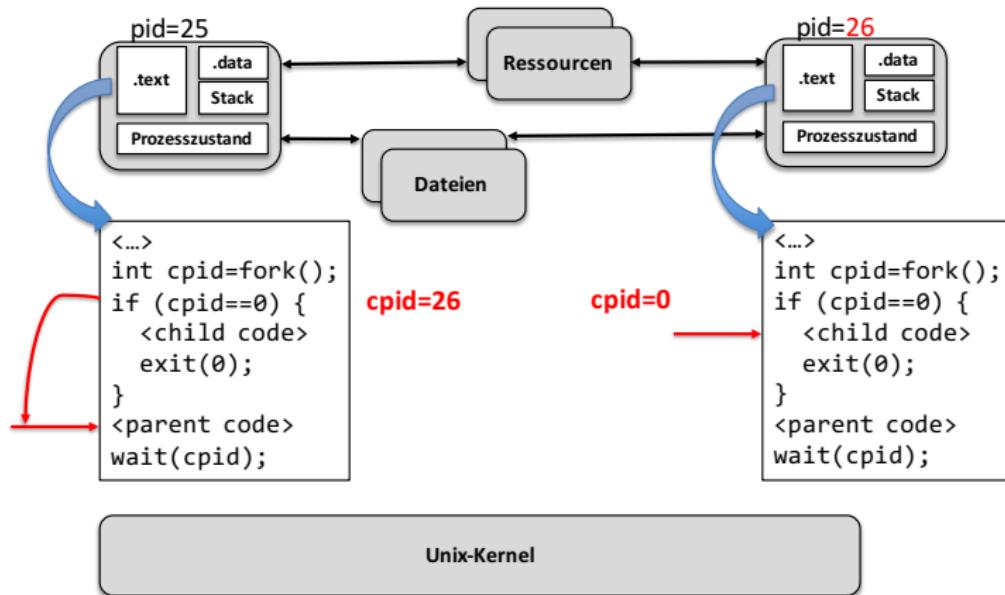
Der Prozeß mit ID = 26 ist eine Kopie vom Parent-Prozeß. Die Programmzähler stehen an derselben rot-markierten Stelle (im fork).

Ablauf von fork:

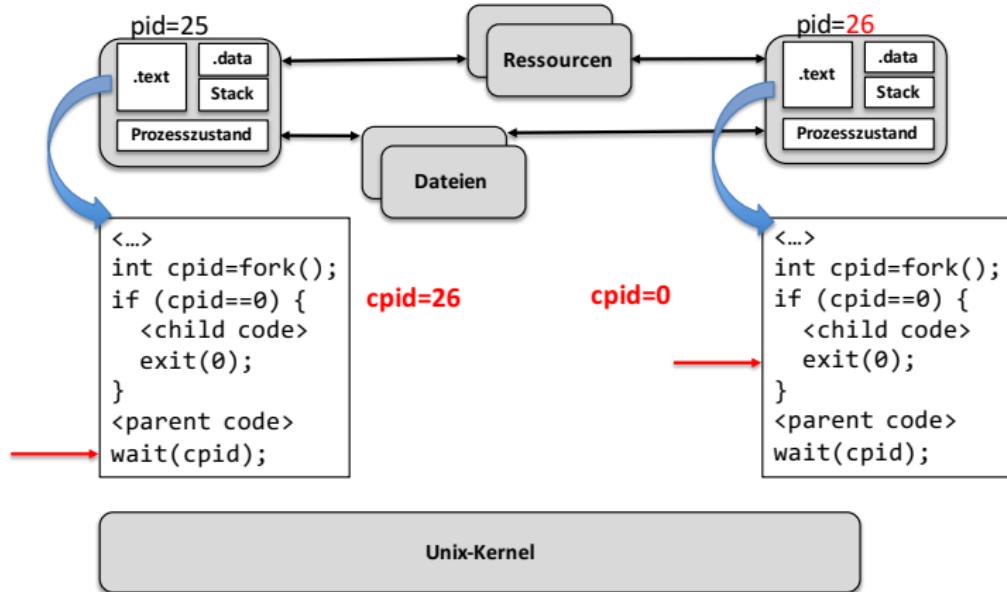


Beide Prozesse kehren aus dem `fork`-Aufruf zurück.
Unterschied: verschiedene Rückgabewerte!

Ablauf von fork:

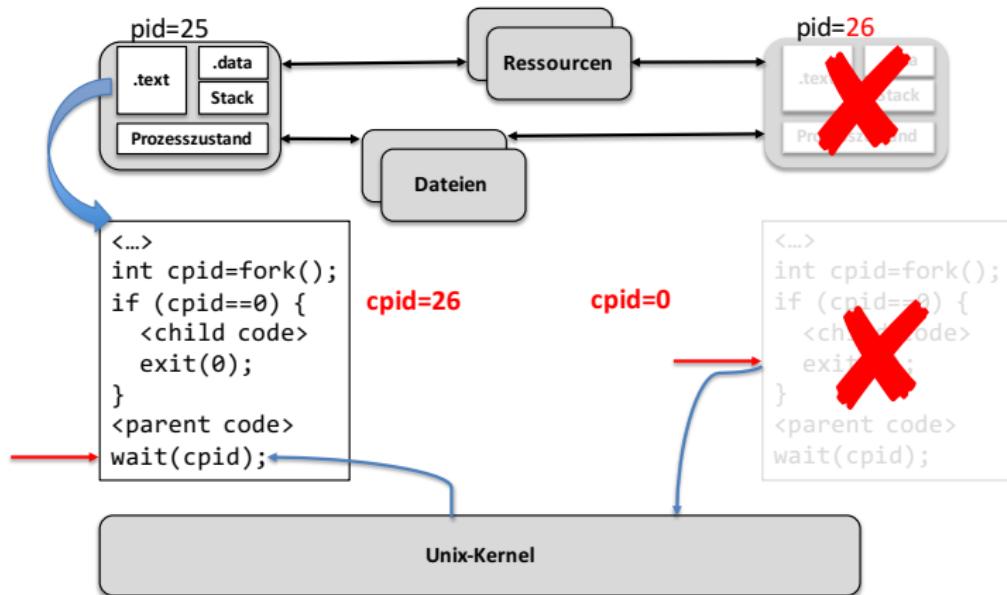


Ablauf von fork:



Der Parent-Prozeß wartet, bis der Kindprozeß sich beendet.

Ablauf von fork:



Beenden von Prozessen

- Prozesse können in POSIX-Systemen mit Hilfe des `execve()` Systemaufrufs das ausgeführte Programm durch ein neues ersetzen.
- Prozesse können sich selbst durch den `exit()` Systemaufruf beenden und damit aus dem System entfernen.
- Ein Prozeß kann die von ihm erzeugten Prozesse beenden, falls
 - ein Prozeß nicht länger gebraucht wird oder falls
 - der erzeugende Prozeß sich beendet und es nicht gewünscht/erlaubt ist, daß die erzeugten Prozesse unabhängig weiter existieren (*cascading termination*).
- Auf die Beendigung eines erzeugten Prozesses kann in POSIX-Systemen mit Hilfe des `wait()` Systemaufrufs gewartet werden.
- Der Ergebniswert (*return code*) eines sich beendenden Prozesses wird an den erzeugenden Prozeß weitergeleitet.

Beispiel: Einfacher Kommandointerpreter

```
while (1) { /* repeat forever */
    type_prompt(); /* display prompt on screen */
    read_command(); /* read input from the terminal */
    pid = fork(); /* create a new process */
    if (pid < 0) { /* repeat if system call failed */
        perror ("fork");
        continue;
    }
    if (pid != 0) { /* parent process*/
        waitpid(pid, &status, 0); /* parent process */
    } else { /* child process */
        execve(command, params, 0); /* execute command */
    }
}
```

Bem.: POSIX spezifiziert:

posix_spawn – spawn a process

The `posix_spawn()` function is used to create a new child process that executes a specified file.

The `posix_spawn()` function provides the functionality of a combined `fork(2)` and `exec(3)`, with some optional housekeeping steps in the child process before the `exec(3)`.

These functions are not meant to replace the `fork(2)` and `execve(2)` system calls. In fact, they provide only a subset of the functionality that can be achieved by using the system calls.

Quelle: https://man7.org/linux/man-pages/man3 posix_spawn.3.html