Der Posix Prozess

Ein Prozess ist ein laufendes Programm.

Process Creation

Der allererste Prozess heißt `init` und wird vom kernel nach Initialisierung der Hardware (-Treiber) gestartet. Er hat traditionellerweise die `PID = 1` und läuft solange das System läuft.

Ein Prozess wird erzeugt, indem der Parent-Prozess zuerst

- `fork()`'ed (siehe man fork). Es gibt jetzt 2 Instanzen, eine ist das Child, die andere der Parent.
- Dann ruft das child ein File (in dem maschinenlesbarer Code enthalten ist) mittels `exec()` auf.
- Der Parent `wait()`'ed auf das child, wenn es ihn interessiert.

PID und PPID

- pid process ID
- ppid parent process id

Signale

Signale sind eine Möglichkeit, Prozesse über bestimmte Ereignisse zu informieren oder deren Lauf zu beinflussen. Signale werden über den `kill()` Syscall an Prozesse übermittelt. `kill -l` listet alle verfügbaren Signale des Systems auf. Für die meisten (?) Signale sind default actions vorgegeben, welche per Signal-Handler des signalisierten Prozesses geändert implementiert werden können. Manche Signals können aber nicht durch Handler abgefangen werden:

nicht handelbare Signale

- STOP .. unterbricht die Ausführung eines Prozesses bis ein CONT kommt.
- CONT .. setzt die Ausführung eines gestoppten Prozesses fort.
- KILL .. beendet den Prozess *sofort*.
- SEGV .. Segmentation Violation: Der Prozess hat auf Speicher ausserhalb seiner Reservierungen (malloc()) zugegriffen und wird sofort beendet.

filehandles

Jeder Prozess hat zu Beginn 3 offene Filehandles:

- stdin (read-only). Java: System.in, Python: sys.stdin
- stdout (write-only). Java: System.out, JS: console.log(), Python: sys.stdout
- stderr (write-only). Java: System.err, JS: console.error(), Python: sys.stderr

Environment

.. ist ein key-value store, dieser wird bei fork / exec vererbt. deno: process.env, Python: os.environ

argv

.. ist ein string-array, und manchmal (zumeist) ist argv[0] der Name des Programmes selbst. Die folgenden Elemente sind die weiteren (Command-Line) Argumente. Beispiel: `find /etc -type f`

- `argv[0]`: `find`
- `argv[1]`: `/etc`
- `argv[2]`: `-type`
- `argv[3]`: `f`

Process Handling mittels Shell

Management von Prozessen in der Shell

- einfach das Programm aufrufen, Beispiel: `date`
- Starten im Hintergrund mit `&`. Beispiel: `firefox &`
- Holen in den Foreground mit `fg`.
- Crtl-Z stoppt einen Vordergrund Prozess. Dieser kann mit `fg` im Vordergrund oder mit `bg` im Hintergrund weiter ausgeführt werden.
- `jobs` listet laufende Prozesse.
- `%` Symbol, um Hintergrund Prozesse gezielt zu bearbeiten. Zb. `fg %3`

Umleitungen

- `ls > filename.txt`: stdout von 'ls' wird ins file geschrieben
- `ls 2 > errors.txt`: stderr von 'ls' wird geschrieben
- `suchprogramm.exe < logfile`: logfile steht als stdin im suchprogramm.exe zur Verfügung
- `find /etc > file.txt 2>&1`: stdout und stderr landen im file
- `ps wwwaux | grep myprocess`: sucht 'myprocess' in der Prozessliste