Unix & Shell-Programmierung SS21

Vorlesungswoche 8

Helga Karafiat

FH Wedel

Prozesse - Überblick / Definitionen

- Prozess ist eine laufende Instanz eines Programms
- Prozesse werden über Prozessnummern sog. PIDs (Process-IDs) referenziert
- Prozesse werden durch Systemaufrufe `fork()` und `execve()` gestartet
- Der erste gestartete Prozess ist immer `init` (PID 1)
- Jeder Prozess besitzt einen Elternprozess (Parent) und eine PPID (Parent-Process-ID)
- Verwaiste Prozesse werden dem `init`-Prozess untergeordnet
- Beim Herunterfahren werden absteigend (anhand der PID) alle Prozesse beendet, `init` ist der letzte Prozess, der beendet wird
- Vorteile Unix:
  - sehr einfache Prozesserzeugung
  - begünstigt Kombination von vielen kleinen (weniger komplexen Programmen) um größere Aufgaben zu bewältigen

Multitasking

- Unter Unix schon immer Unterstützung mehrerer gleichzeitiger Prozesse (Multitasking)
- So lange es nur eine CPU gibt, ist Multitasking immer nur "Illusion"
  - Prozesse dürfen immer abwechselnd eine kurze Zeitspanne arbeiten (wenige Millsekunden, sog. Zeitscheibe)
  - Nicht laufende Prozesse werden suspendiert
  - Umschalten zwischen Prozessen wird Kontextwechsel genannt
- Verhalten wird komplett vom Betriebssystem/Kernel gesteuert, weder Benutzer noch Programme bekommen davon etwas mit
- Scheduler ist für die Verwaltung der Prozesse zuständig
- Wenn mehrere Kerne / CPUs vorhanden sind, versucht der Scheduler alle zu benutzen um die Arbeitslast bestmöglich zu verteilen

Prozessprioritäten

- Prozesse haben Prioritäten
  -> zeitkritische Prozesse werden vor weniger wichtigen ausgeführt
- Priorität wird auch Niceness genannt (“be nice to the system”)
  - -20 (höchste Priorität) bis 19 (niedrigste Priorität)
  - Standardwert: 0
- Setzen von Prioritäten mit:
  - `nice` beim Starten eines Prozesses
    z.B. `nice -n 19 ./backup_skript`
  - `renice` für laufende Prozesse über die PID, GID oder UID
    (bestimmter Prozess, alle einer Gruppe, alle eines Users)
  - Zusätzlich:
    `ionice` um die I/O-Priorität zu setzen (Festplattenzugriff etc.)
  - Normale User dürfen üblicherweise nur Niceness von 0 bis 19 setzen

Garantien für Prozesse (von Kommandozeile aus gestartet)

- Prozess besitzt Kernel-Kontext (Kernel kann Prozess verwalten und kontrollieren)
- Prozess besitzt privaten und geschützten virtuellen Adressraum
  - eingeschränkt durch Ressourceneinschränkungen und Adressierungsbreite
  - privater Adressraum stellt sicher, dass Prozesse und Kernel sich nicht gegenseitig stören können (robust)
- Dateideskriptoren (`stdin`, `stdout`, `stderr`) sind bereits offen und verfügbar (Programm muss sich meistens nicht extra um I/O kümmern)
- Wildcards in Kommandozeilenargumenten wurden erweitert (einheitliche Vorverarbeitung)
- Auf Umgebungsvariablen kann über einen Umgebungsbereich zugegriffen werden (Kommunikationsmöglichkeit)
- Prozess, der von interaktiver Shell aus gestartet wurde, besitzt Kontroll-Terminal (Senden von Signalen an Prozess möglich)
- Garantien sind fair, d.h. alle Prozesse mit der selben Priorität werden gleich behandelt (Programmiersprache spielt keine Rolle)

Signale

- Nachrichten mit denen man Einfluß auf ein laufendes Programm nehmen kann
- Absender kann das System selbst sein oder ein Benutzer, der das System "bittet" ein bestimmtes Signal zu senden
- Wichtige Signale (alle Signale unter `man 7 signal`)

Nummer	Signalname	Grund / Bedeutung
1	SIGHUP	Beenden der Verbindung ("Auflegen"), Schließen des Terminals
2	SIGINT	Unterbrechen des im Vordergrund laufenden Prozesses (Strg-C)
3	SIGQUIT	Verlassen des Programms, Absturz
4	SIGILL	Illegale Instruktion
8	SIGFPE	Illegale Fließkommainstruktion
9	SIGKILL	Sofortiges Beenden, kann nicht abgefangen werden, "last resort"
11	SIGSEGV	Schutzverletzung (illegaler Zugriff auf Speicher)
13	SIGPIPE	Schreiben in eine geschlossene Pipe
15	SIGTERM	Hinweis: Programm soll sich sofort beenden (netter als 9)
18,20,24	SIGTSTP	Stoppen des Prozesses (Strg-Z)

- Senden von Signalen an Prozesse mittels `kill`
  - Syntax `kill [-SIGNALNO] PID...`,
    z.B. `kill -9 1234` um Prozess 1234 sofort (!!!) zu beenden
  - ohne explizite Angabe eines Signals sendet `kill` 15 (`SIGTERM`)
  - `killall` arbeitet wie `kill`,
    nur dass statt PIDs Prozessnamen angegeben werden
    z.B. `killall chrome` um alle Prozesse von `chrome` zu beenden
- Abfangen von Prozesssignalen mittels `trap`
  - Syntax: `trap [ACTION] [SIGNAL]`
  - Nützlich um z.B. beim Beenden eines laufenden Programms entsprechende Ausgaben machen zu können oder aufzuräumen. Beispiel:

```
trap "echo 'Terminal hung up'; readCfg" HUP
trap "echo 'Ctrl-C pressed'; exit" INT
trap "echo 'Program crashed'; exit" QUIT
trap "echo 'Program was told to stop'; exit" TERM
trap "echo 'Program about to exit'; rm -f $TMPFILE" EXIT
```

  - `EXIT` Signal wird immer ganz am Schluss verarbeitet

Ausführung im Hintergrund

- Laufende Prozesse blockieren üblicherweise die Shell, von der aus sie gestartet wurden; startende Shell ist Elternprozess
- Starten im Hintergrund möglich mit `&`
  z.B. klassischer Tee-Timer (3 Minuten): `sleep 180 && echo "Tee ist fertig!" &`
- Shell vergibt für jeden Hintergrundprozess aufsteigende Nummer (in eckigen Klammern) und zeigt PID an
- Prozesse nachträglich in den Hintergrund verschieben:
  - Prozess mit `Strg-Z` schlafen legen / stoppen
  - `bg` startet den zuletzt gestoppten Prozess im Hintergrund
  - optionale Angabe der Nummer mit `%`, z.B. `bg %1` um den ersten gestoppten Prozess im Hintergrund wieder zu starten
- Prozesse aus dem Hintergrund zurückholen
  - `fg` holt den zuletzt in den Hintergrund verschobenen Prozess wieder in den Vordergrund, optionale Angabe der Nummer analog zu `bg`
- Aktuelle Kind-Prozesse und deren Status anzeigen mit: `jobs`

Tools

- Einige Tools um Prozesse anzuzeigen
  - `ps`
    - ähnlich zu `ls`, aber für Prozesse, Ausgabe ist Momentaufnahme
    - Vielzahl von Kommandozeilenoptionen (siehe `man ps`)
    - Vorsicht: Eingaben im classic style (Optionen mit -) oder BSD Style triggern ggfs. unterschiedliches Verhalten
    - `ps aux` listet alle laufenden Prozesse auf
  - `top`
    - dynamische Anzeige der aktuell laufenden Prozesse
    - Standardansicht: Prozesse nach CPU-Zeit absteigend sortiert
    - Für jeden erdenklichen Anwendungfall anpassbar (siehe `man top`)
    - In etwas kleinerem Umfang auch dynamisch anpassbar, Kurzübersicht der Optionen mit `?` zur Laufzeit
    - Kann Prozesse auch bearbeiten (`kill` & `renice`)

- Einige Tools um Prozesse anzuzeigen (Fortsetzung):
  - `pstree`
    - Übersicht aller Prozesse als Baumstruktur
    - Diverse Varianten, Sortierungen und Zusammenfassungen möglich
    - Konfigurierbar über Kommandozeilenargumente (siehe `man pstree`)
    - `pstree -a` zeigt die Kommandozeilenargumente der Prozesse mit an
- Anzeige der durchschnittlichen Systemlast mit `uptime`
  - load average: Anzahl an Prozessen, die aktuell Ausführung erwarten
  - Durchschnittswerte: letzte Minute, letzte fünf und letzte 15 Minuten
- Weitere nützliche Tools: `iostat`, `netstat`, `procinfo`, ...