

Betriebssysteme

Übungen 07

Prof. Dr. Rainer Werthebach


Studiengang Informatik

Hochschule Aalen - Technik und Wirtschaft

Makefiles

- Aufruf: `make [-f makefiles] [options] [targets]`
- Dateien zum Erstellen und Verwalten von Programmen
- Bietet die Möglichkeit, Programme in Abhängigkeit von Bedingungen zu erstellen
- Tool: autotools
 - Automatisiert den Erstellungsprozess von Makefiles
 - Siehe <http://markuskimius.wikidot.com/programming:tut:autotools>
- Alternative: Apache Ant

Makefiles – Aufbau

- Dependency lines:
 - Erstellung eines „Ziels“ unter Voraussetzen von Bedingungen
- Command lines:
 - Befehl der ausgeführt werden soll
 - Mit TAB in 1. Spalte oder durch „;“ getrennt
- Abstraktes Beispiel:
 - TARGET_A: FILE_1 FILE_2 FILE_3
 cmd_A
- Konkretes Beispiel:
 - `programm: main.c libmathe.a`
`gcc main.c -o programm -L. -lmathe -static`

Unix Prozess im Speicher

Inhalt	Erklärung
Argumente Stack	Stack des Prozesses (Argumente liegen zu Ausführungsbeginn oben)
Heap	Dynamisch allokierbarer Speicher
BSS Segment	Uninitialisierte Daten
Data Segment	Initialisierte Daten, wie Konstanten und Variablen
Text Segment	Eigentlicher Programmcode

Prozess-IDs, PIDs

- `getpid()` → Gibt die Prozessidentifikation (PID) zurück
- `getppid()` → Gibt die PID des Elternprozesses zurück
- `getuid()` → Liefert die reale Benutzeridentität/User-ID (UID) des aktuellen Prozesses
- `geteuid()` → Liefert die effektive Benutzeridentität des aktuellen Prozesses
- `getgid()` → Gibt die reale Gruppen-ID/Group-ID (GID) des laufenden Prozesses zurück
- `getegid()` → Gibt die effektive Gruppen-ID des laufenden Prozesses zurück
- Die reale ID (UID, GID) entspricht der ID des aufrufenden Prozesses. Die effektive ID (EUID, EGID) entspricht dem gesetzten ID-Bit der Datei, die ausgeführt wird.

fork()

- Erzeugen eines Kindprozesses
- Benötigt:
 - `#include <sys/types.h>` → Enthält benötigte Typendeklarationen
 - `#include <unistd.h>` → Enthält `fork()`-Deklaration
- Aufruf:
 - Allgemein (Prototype):
`pid_t fork(void);`
- Rückgabewerte:
 - 0 - im Kindprozess
 - Prozess-ID des Kindprozesses im Elternprozess
 - -1 bei Fehler

fork() - Beispiel

Luke.c:

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main() {
    int pid;
    pid = fork();
    if (pid == -1) {
        perror("fork call");
        exit(1);
    }
    if (pid == 0) {
        sleep(2);
        printf("Luke: Neiiiiiiin\n");
    } else {
        printf("Vader: Ich bin dein Vater!\n");
        sleep(4);
    }
    return 0;
}
```

wait()

- Warten auf Statusänderung eines Kindprozesses
- Der Prozess, welcher `wait()` aufruft, unterbricht seine Ausführung bis zu einer Statusänderung im Kindprozess
- Benötigt:
 - `#include <sys/types.h>`
 - `#include <sys/wait.h>` → Enthält die `wait()`-Deklaration
- Aufruf:
 - Allgemein (Prototype): `pid_t wait(int *status);`
 - Oder: `pid_t waitpid(pid_t pid, int *status, int options);`
- Rückgabewerte:
 - Prozess-ID bei Erfolg
 - 0 - kein Kindprozess aktiv
 - -1 Fehler

Beenden von Prozessen

- normale Terminierung mit `exit()`
 - Schließen aller Dateideskriptoren
 - Übertragen des exit-Wertes in die Prozesstabelle
 - MSByte: exit-Wert, LSByte: immer 0
- Abnormale Terminierung durch Signale:
 - Wird z.B. bei Fehlern vom BS gesendet
 - Kann explizit durch einen anderen Prozess gesendet werden
 - MSByte: 0, LSByte: Signalart
 - Falls Bit8 des LSByte gesetzt ist, wurde ein „core dump“ erzeugt

	MSByte								LSByte							
Wert	2 ¹⁵	2 ¹⁴	2 ¹³	2 ¹²	2 ¹¹	2 ¹⁰	2 ⁹	2 ⁸	2 ⁷	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰
Daten	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Auswertung der Prozessterminierung

- In `sys/wait.h` sind Makros definiert, die die Auswertung der Prozessterminierung vereinfachen
- `#define WIFEXITED(stat) ((int)((stat)&0xFF) == 0)`
 - `WIFEXITED` ist wahr, falls Prozess normal mit `exit()` beendet wurde
- `#define WEXITSTATUS(stat) ((int)(((stat)>>8) &0xFF))`
 - `WEXITSTATUS` liefert den `exit`-Wert, falls Prozess normal mit `exit()` beendet wurde

- Beispiel:

```
int status;
...
pid = fork();
...
wait(&status);
if (WIFEXITED(status)) {
    printf("Kind wurde normal beendet\n");
}
```

Deamons

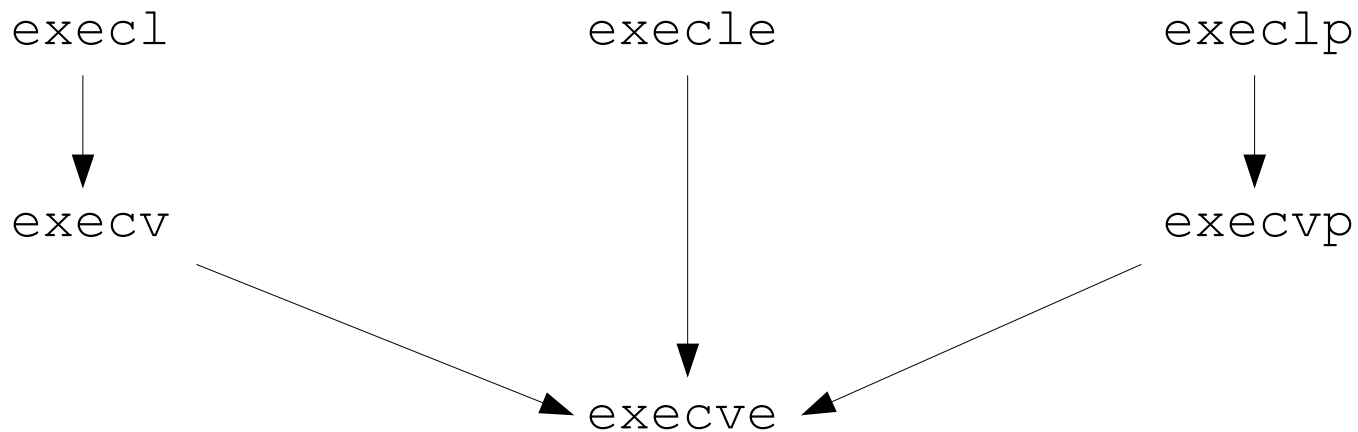
- `fork()`, dann Beenden des Elternprozesses, Kind wird Daemon
- `setsid()` → Neue Session erstellen
 - Prozess wird Sessionführer der neuen Session
 - Prozess wird Prozessgruppenführer der neuen Prozessgruppe
 - Prozess hat kein Kontrollterminal
- `chdir()` → Setzen des Directorys
 - Deamons haben typischerweise „/“ als Verzeichnis
- `umask()` → Setzen der Dateikreierungsmaske
 - Deamons haben typischerweise 0 als umask
- `close()` → Schließt Filedeskriptoren
 - Nicht zwingend notwendig, aber da ein Deamon kein Kontrollterminal für Aus- und Eingaben hat, sind diese Deskriptoren überflüssig

Zombies

- Ein Zombie Prozess ist ein fertig abgearbeiteter Kindprozess, dessen Rückgabewert (exit-Status) noch nicht ausgelesen wurde
- Zombies belegen einen Eintrag in der Prozesstabelle und sind durch den Status `Z` und den Namenszusatz `<defunct>` zu erkennen
- Wenn ein Kindprozess im Zombie-Zustand ist während sich sein Elternprozess beendet, übernimmt der init-Prozess die Vaterschaft und entfernt den Zombie Prozess aus der Prozesstabelle

exec

- Die `exec`-Familie besteht auf Funktionen, die zum Aufrufen eines anderen Programm dienen
- Hierbei wird der Speicher des aufrufenden Prozesses überschrieben und zwar mit dem Programmcode des auszuführenden Programms
- Alle Argumentenlisten der `exec`-Funktionen müssen Null-terminierte Strings sein (`'\0'`)



execl(), fork() - Beispiel

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>

int main() {
    int pid;
    int state;
    pid = fork();
    if (pid == 0) {
        execl("/bin/ls", "ls", "-la", (char *)NULL);
        exit(1);
    }
    wait(&state);
    exit(WIFEXITED(state));
}
```