Systemprogrammierung

2. Hausaufgabe

Gruppe 1

Marc Radau

Ramin Khorsandi

Mustafa Karaca

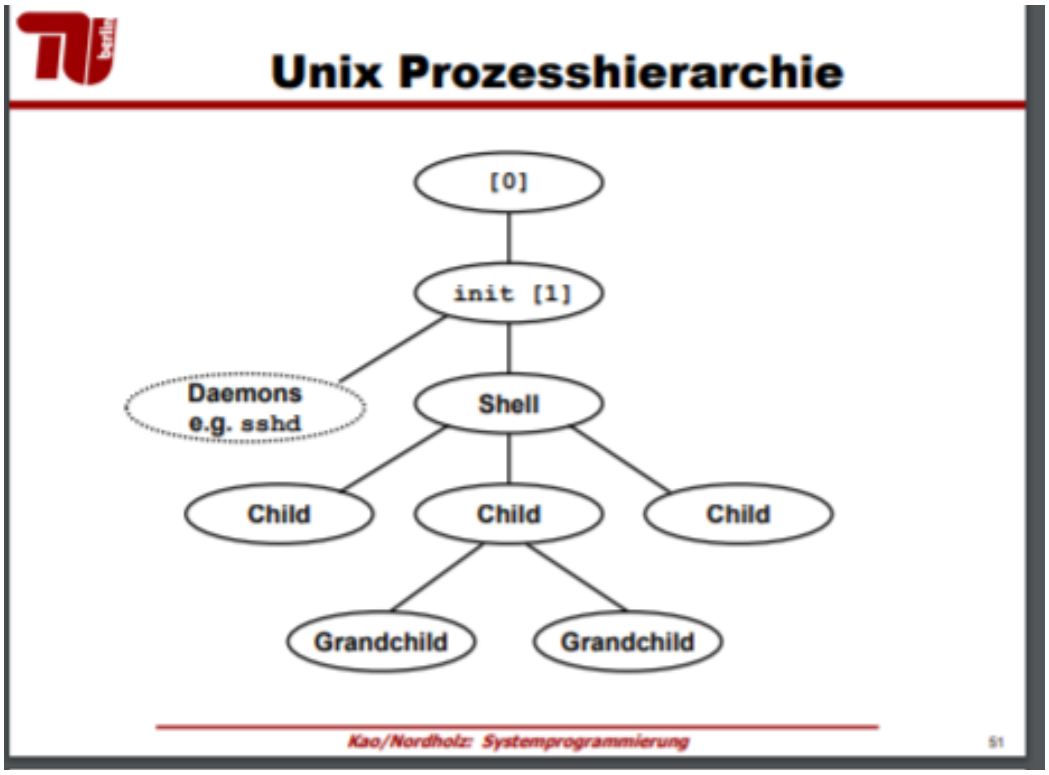
Diego Nils Romero Koehler

Prozesse und Threads

2.3)

a) Das Betriebssystem erschafft ein Prozess um unabhängig voneinander laufende Programme zu realisieren.

b)



Zwischen den Prozessen besehen Vater-Kind-Beziehungen.
Der erste Prozess wird direkt vom Systemkern gestartet mit init [1], wobei 1 die Prozessnummer ist. Dabei läuft im Hintergrund ein Programm, wie zum Beispiel Daemon, die bestimmte Dienste wie Terminaldienste, Netzwerkdienste usw. realisiert. Dann haben wir unseren Unix-Shell, die unsere Benutzerschnittstelle ist, in der Benutzer Kommandos in einer Eingabezeile eintippen können, die der Computer dann ausführt. Jetzt werden Prozesse andere Prozesse erzeugt. Der Prozess wir aufgeteilt in mehrere nebenläufige oder parallele Aktivitäten, die alle als eigene Prozesse initialisiert werden. Dies geschieht durch fork() welches einen neuen Prozess erzeugt, dabei eine Kopie des Vaters erstellt, mit der einzigen Unterscheidung der Prozessnummer.

Quelle: https://de.wikipedia.org/wiki/Unix-Shell
https://de.wikipedia.org/wiki/Daemon
Vorlesungsfolie: Folie 48-51

c)

PCB = Prozesskontrollblock -> verwaltungstechnischer Repräsentant des Prozesses enthält Informationen über Prozessnummer, Prozesszustand, Betriebsmittel usw.

Prozessumschaltung: Ein aktiver Prozess wird vom Prozesszustand Laufend in den Zustand Blockiert aktualisiert, dabei werden die Registerinhalte im PCB gespeichert. Der bereite Prozess wird in den Zustand Laufend aktualisiert, hierbei wird der virtuelle Adressraum auf PCB des neuen Prozesses umgeschaltet, die Registerinhalte aus dem PCB geladen und der Prozess wird dort fortgesetzt, wo als letztes der Befehlszähler war.

1.Ereignis: E/A Geräte führt zu einem Interrupt (anfänglicher Prozess wird blockiert), folge Prozessumschaltung, E/A Operationen werden ausgeführt und wechsle zum anfänglichen Prozess.

2.Ereignis: Aktuelle Prozess wird zu Ende ausgeführt (Terminated), und es wird zum nächsten bereites Prozess umgeschaltet.

Quelle:Folie 9,18

d)

Der Kontextwechsel in einem User-Level-Thread zwischen Userthreads geht deutlich schneller, da man keine Syscalls wie bei Kernel-Level-Threads oder Prozessen benötigt.

Der Threadwechsel ist Teil einer Applikation und somit funktioniert der Threadwechsel ohne Betriebssysstem.

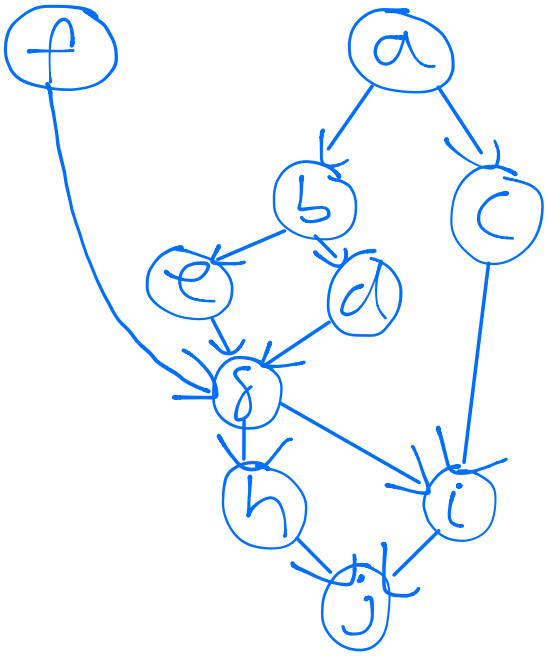
User-Level-Thread ist nicht für Multiprozessoren gegeignet, da BS nur einem KL-Thread einer CPU zugeordnet werden kann.

Quelle: Folie 33

2.4

a) Wenn "a = jobA()" die Zeile 5 des C-Programms ist, sind die unabhängige Zeilen:

- Zeilen 5 und 10
- Zeilen 6 und 7 und 10
- Zeilen 7 und 8 und 9 und 10
- Zeilen 12 und 13



b)

```
fork f
a
fork c
b
fork e
d
join f
join e
g
fork h
join c
i
join h
j
```