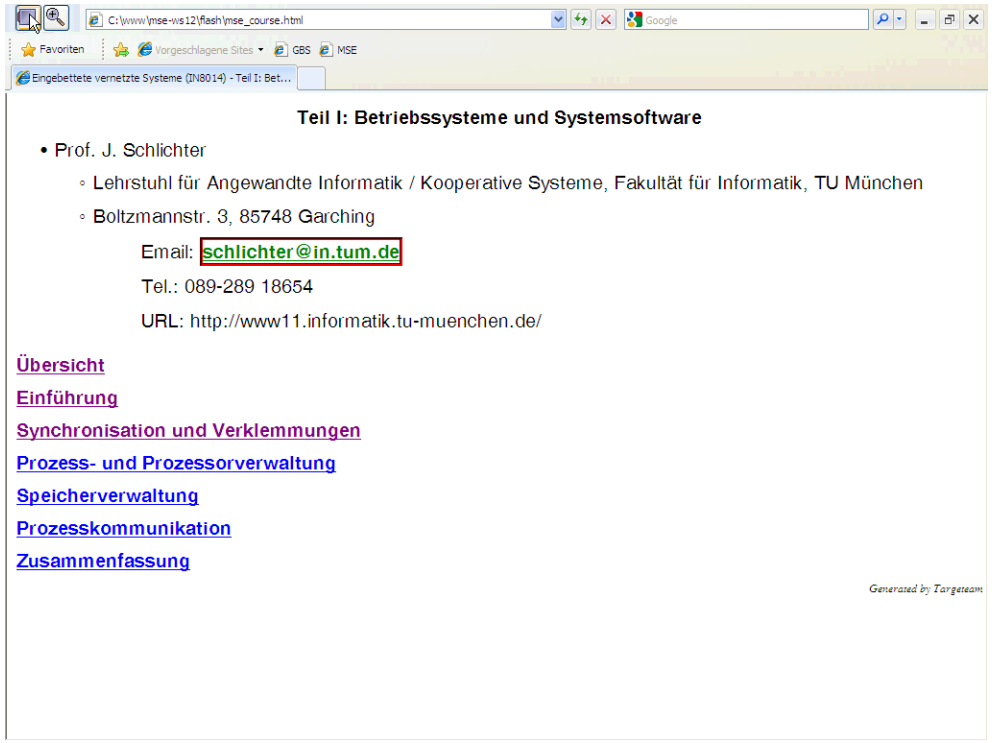
Script **generated by TTT**

Title: Eingebettete_Systeme (30.10.2012)

Date: Tue Oct 30 14:47:28 CET 2012

Duration: 79:51 min

Pages: 19



Ein Prozess ist der Ablauf eines Programms in einem Rechensystem. Dieser Ablauf ist eine Verwaltungseinheit im jeweiligen Betriebssystem.

Fragestellungen

Dieser Abschnitt gibt eine kurze Einführung in eine der wichtigen Verwaltungsaufgaben eines Betriebssystems:

Verwaltung von Prozessen.

Verwaltung des Prozessors, d.h. Zuteilung der CPU an rechenbereite Prozesse (Scheduling).

Unterbrechungskonzept.

Prozessverwaltung

Prozessorverwaltung

Unterbrechungskonzept

Generated by Targeteam

Dieser Abschnitt behandelt das Prozesskonzept, Datenstrukturen zur Beschreibung des aktuellen Prozesszustandes sowie Dienste zum Aufruf von Systemfunktionen.

Prozesse repräsentieren eine Aktivität; sie haben ein Programm, Eingaben, Ausgaben und einen Zustand.

Prozesskonzept

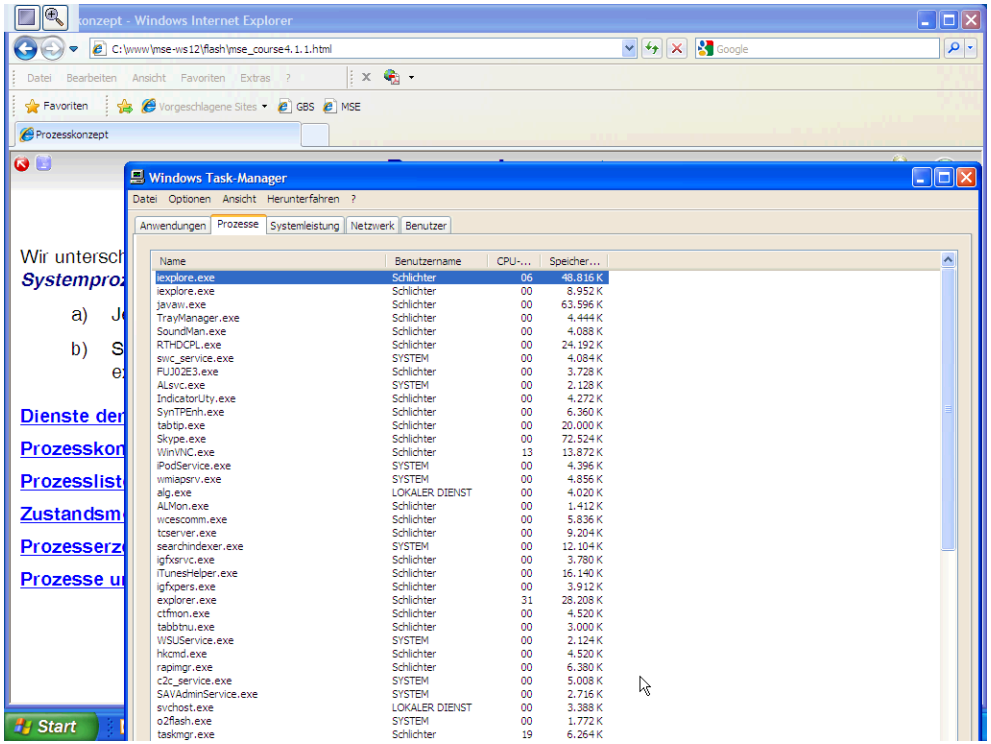
Dispatcher

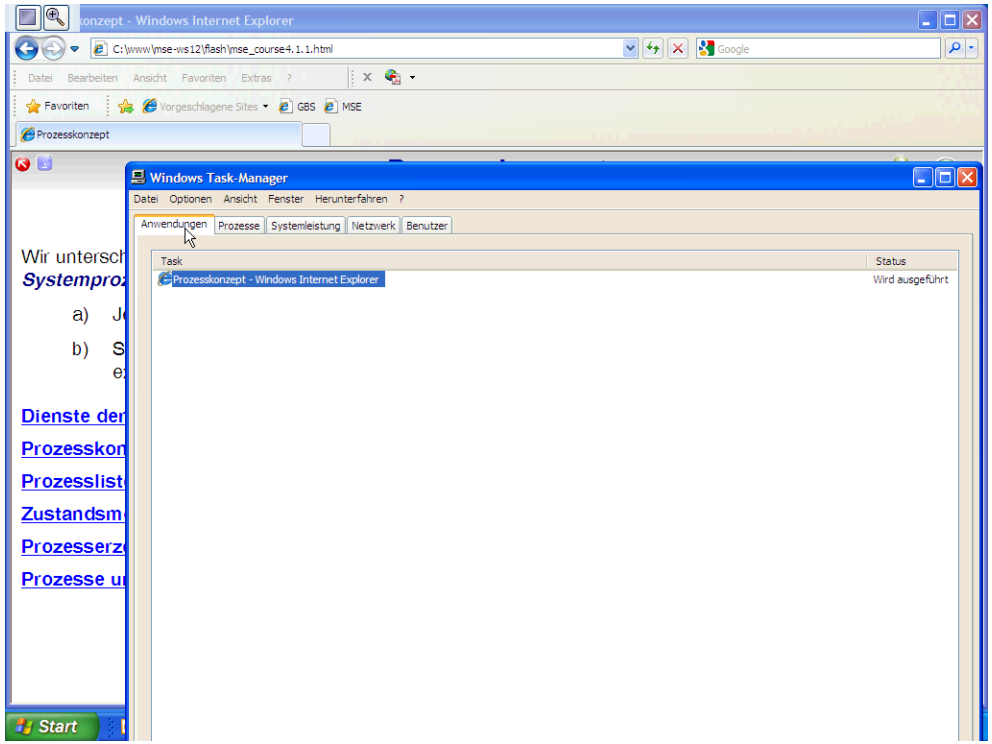
Arbeitsmodi

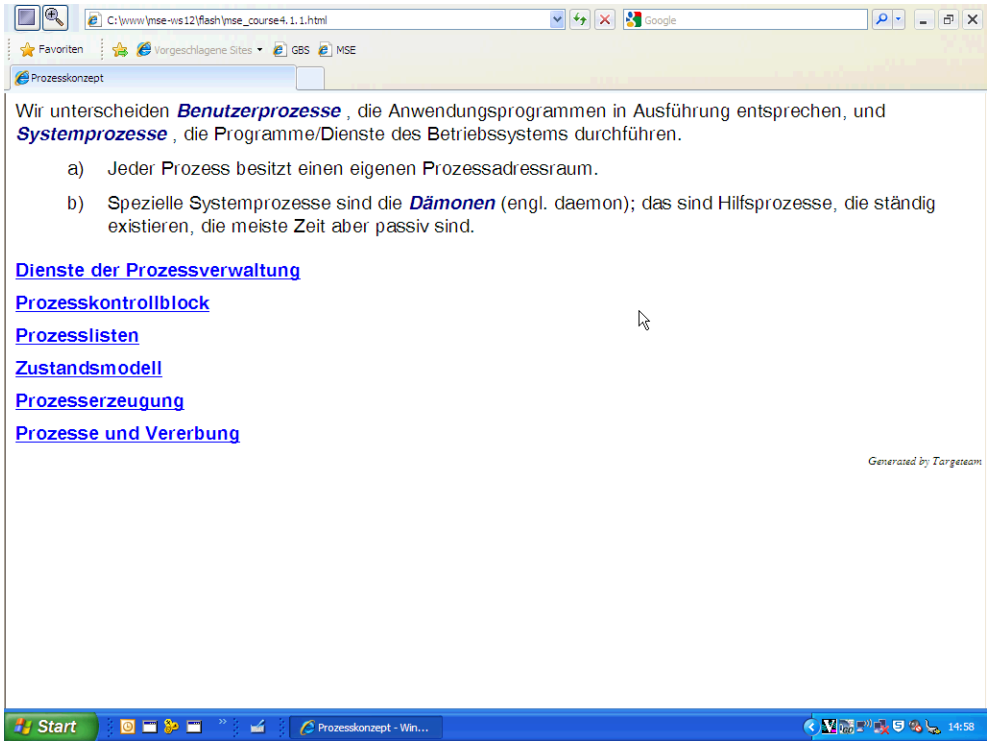
Systemaufrufe

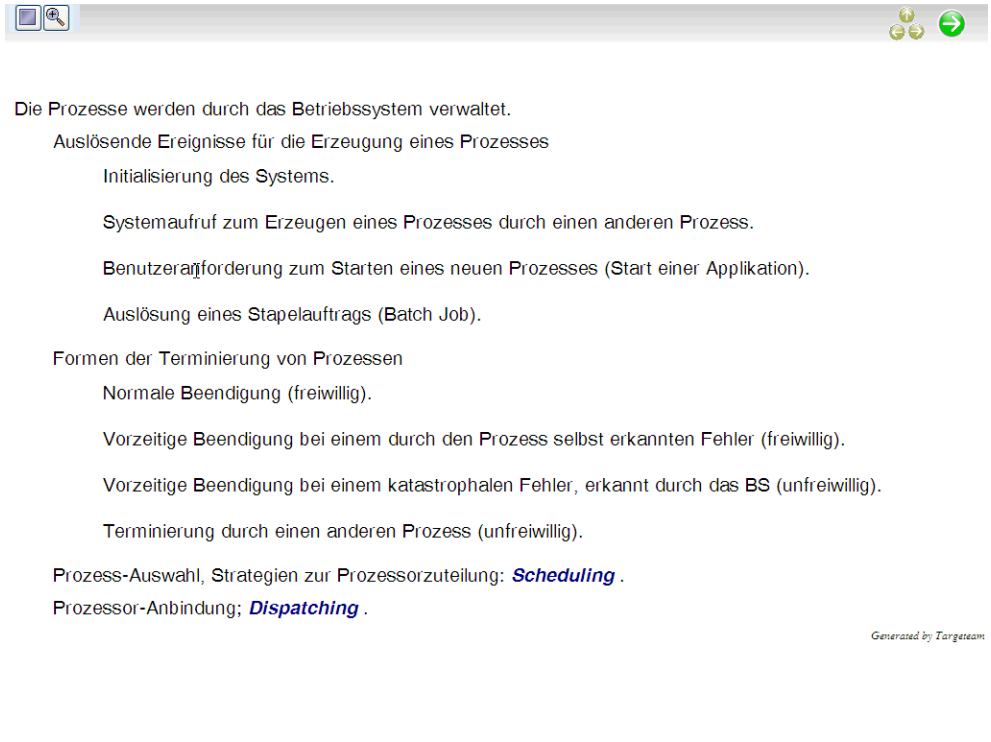
Realisierung von Threads

Generated by Targeteam


Windows Task-Manager
Datei Optionen Ansicht Herunterfahren ?
Anwendungen Prozesse Systemleistung Netzwerk Benutzer
Name Benutzername CPU-... Speicher...
Wir unterscheiden Benutzerprozesse , die Anwendungsprogrammen in Ausführung entsprechen, und Systemprozesse , die Programme/Dienste des Betriebssystems durchführen.
a) Jeder Prozess besitzt einen eigenen Prozessadressraum.
b) Spezielle Systemprozesse sind die Dämonen (engl. daemon); das sind Hilfsprozesse, die ständig existieren, die meiste Zeit aber passiv sind.
Dienste der Prozessverwaltung
Prozesskontrollblock
Prozesslisten
Zustandsmodell
Prozesserzeugung
Prozesse und Vererbung
Generated by Targeteam


Windows Task-Manager
Datei Optionen Ansicht Fenster Herunterfahren ?
Anwendungen Prozesse Systemleistung Netzwerk Benutzer
Task Status
Prozesskonzept - Windows Internet Explorer Wird ausgeführt


Wir unterscheiden Benutzerprozesse , die Anwendungsprogrammen in Ausführung entsprechen, und Systemprozesse , die Programme/Dienste des Betriebssystems durchführen.
a) Jeder Prozess besitzt einen eigenen Prozessadressraum.
b) Spezielle Systemprozesse sind die Dämonen (engl. daemon); das sind Hilfsprozesse, die ständig existieren, die meiste Zeit aber passiv sind.
Dienste der Prozessverwaltung
Prozesskontrollblock
Prozesslisten
Zustandsmodell
Prozesserzeugung
Prozesse und Vererbung
Generated by Targeteam

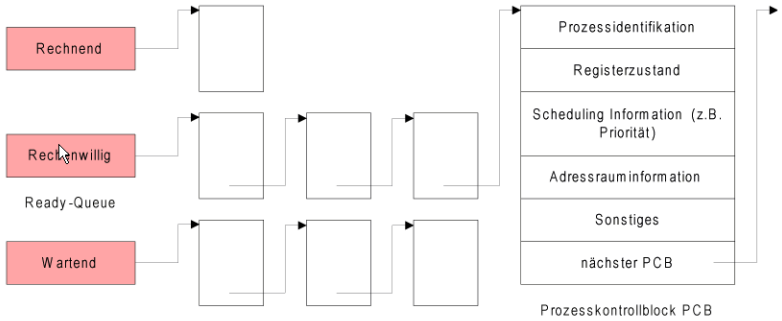

Die Prozesse werden durch das Betriebssystem verwaltet.
Auslösende Ereignisse für die Erzeugung eines Prozesses
Initialisierung des Systems.
Systemaufruf zum Erzeugen eines Prozesses durch einen anderen Prozess.
Benutzeranforderung zum Starten eines neuen Prozesses (Start einer Applikation).
Auslösung eines Stapelauftrags (Batch Job).
Formen der Terminierung von Prozessen
Normale Beendigung (freiwillig).
Vorzeitige Beendigung bei einem durch den Prozess selbst erkannten Fehler (freiwillig).
Vorzeitige Beendigung bei einem katastrophalen Fehler, erkannt durch das BS (unfreiwillig).
Terminierung durch einen anderen Prozess (unfreiwillig).
Prozess-Auswahl, Strategien zur Prozessorzuteilung: Scheduling .
Prozessor-Anbindung; Dispatching .
Generated by Targeteam

Jeder Prozess muss als eine ***Verwaltungseinheit*** beschrieben sein. Ein Prozess wird durch seinen Prozess-Kontext und dieser durch den Prozesskontrollblock (PCB) beschrieben. Ein ***PCB*** (process control block) enthält i.d.R. folgende Informationen:

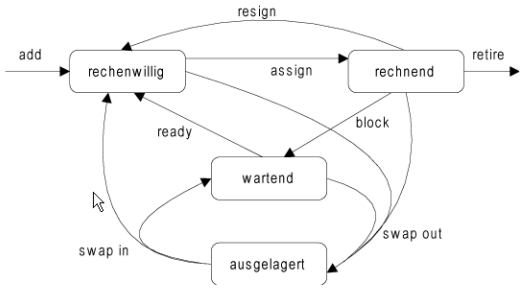
- eindeutiger Name, z.B. fortlaufende Nummerierung des Prozesses (z.B. pid in Unix)
- Name des Benutzers, dem der Prozess zugeordnet ist
- der momentane Prozesszustand (wartend, rechnend, rechenwillig, ...)
- falls der Prozess rechnend ist, Angabe der zugeordneten CPU
- falls der Prozess wartend ist, eine Spezifikation des Ereignisses, auf das der Prozess wartet (z.B. Adresse eines Semaphors).
- die Ablaufpriorität des Prozesses
- die Inhalte der programmierbaren Register (die Anzahl ist abhängig von der jeweiligen CPU-Architektur), z.B. Kellerpointer.
- die Inhalte der Register, in denen die Anfangsadresse und Länge der prozessspezifischen Speicherabbildungstabellen enthalten sind (virtuelle Adressierung).
- das Programmstatuswort (PSW). Das PSW enthält weitere Informationen, die die CPU über den Prozess kennt, z.B. Ablaufcodes, Condition Codes
- PCB unter Linux ist durch die Struktur `task_struct` spezifiert; definiert unter `include/linux/sched.h` .

Die Prozesse werden in Zustandslisten verwaltet, die als verkettete Liste der PCBs realisiert sind.

für E/A-Geräte (z.B. Platte, Terminal) existiert i.d.R. jeweils eine eigene Warteschlange, die die PCBs der wartenden Prozesse enthält.



Das Prozess-Zustandsmodell unterscheidet neben den bereits vorgestellten Zuständen ***rechenwillig*** , ***rechnend*** , ***wartend*** auch den Zustand ***ausgelagert*** . Letzterer Zustand tritt ein, wenn der Adressraum aufgrund Speichermangels aus dem Arbeitsspeicher auf den Hintergrundspeicher verlagert wird ("swapping").



Zustandsübergänge sind:

`add` : ein neu erzeugter Prozess wird zu der Menge der rechenwilligen Prozesse hinzugefügt;

`assign` : als Folge eines Kontextwechsels wird dem Prozess die CPU zugeordnet;

`block` : aufgrund eines EA-Aufrufs oder einer Synchronisationsoperation wird der Prozess wartend gesetzt;

`ready` : nach Beendigung der angestoßenen Operation wechselt der Prozess in den Zustand rechenwillig; er bewirbt sich erneut um die CPU;

`resign` : dem rechnenden Prozess wird die CPU entzogen; er bewirbt sich anschließend erneut um die CPU;

`retire` : der aktuell rechnende Prozess terminiert;

`swap out` : der Prozess wird auf die Festplatte ausgelagert;

Prozesse erzeugen andere Prozesse: parent und child. Vater

kann auf Beendigung von Kind warten, oder

parallel weiterlaufen.

Prozesserzeugung: 2 Vorgehensweisen

Windows NT: Vaterprozess veranlasst eine Reihe von Systemaufrufen, die das Kind entstehen lassen.

Unix: Vater klont sich mit Systemaufruf fork(); Kind ändert selbst sein Aufgabe.

Unix Prozesserzeugung

Linux unterstützt den Systemaufruf `clone()` zur Erzeugung neuer Thread-Kopien.

Unix Prozesserzeugung

Aufruf von fork() führt zu einem fast exakten Duplikat des Vaterprozesses. Unterschiede sind

- unterschiedlicher process identifier (PID)
- der Ergebniswert von fork()
  - Vater: PID des Kindprozesses
  - Kind: 0

Beispielprogramm

- Kind hat vieles mit dem Vater gemeinsam:
  - liest dieselben Dateien, gleicher Benutzername, benutzt dieselben Daten
- Unix Kind fängt mit dem Code des Vaters an und ändert sich dann
  - Systemaufruf exec(): ersetzt das Programmbild des Vaters mit einem anderen.

Beispielprogramm für exec

Prinzipablauf


```
#include <stdio.h>

int main(int argc, char *argv[]) {
    char *myname = argv[1];
    int cpid = fork();
    if cpid == 0 {
       printf("The child of %s is %d\n", myname, getpid());
       ......
       return (0);
    } else {
       printf("My child is %d\n", cpid);
       /* wird vom Vaterprozess durchlaufen */
       .....
       return(0);
    }
}
```


Unix Prozesserzeugung

Aufruf von fork() führt zu einem fast exakten Duplikat des Vaterprozesses. Unterschiede sind

- unterschiedlicher process identifier (PID)
- der Ergebniswert von fork()
  - Vater: PID des Kindprozesses
  - Kind: 0

Beispielprogramm

- Kind hat vieles mit dem Vater gemeinsam:
  - liest dieselben Dateien, gleicher Benutzername, benutzt dieselben Daten
- Unix Kind fängt mit dem Code des Vaters an und ändert sich dann
  - Systemaufruf exec(): ersetzt das Programmbild des Vaters mit einem anderen.

Beispielprogramm für exec

Prinzipablauf

Generated by Targeteam

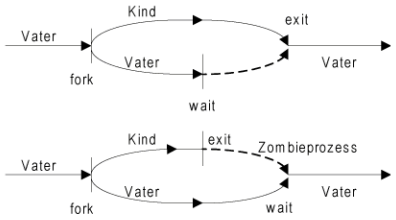
```
#include <stdio.h>

int main(int argc, char *argv[]) {
    char *myname = argv[1];
    int cpid = fork();
    if cpid == 0 {
       int rc;
       rc = execl("/bin/ls", "ls", "-l", (char *) 0);
       printf("Fehler bei execl Aufruf: %d\n", rc);
       exit(1)
    } else {
       /* wird vom Vaterprozess durchlaufen */
    }
}
```

Generated by Targeteam

Mit der Systemfunktion wait wartet der Vaterprozess auf den Kindprozess.

wait vor Beendigung des Kindes: Vater ist blockiert.

wait nach Beendigung des Kindes: Kind wird nach Beendigung zum Zombieprozess.



Unix Prozesserzeugung

Aufruf von fork() führt zu einem fast exakten Duplikat des Vaterprozesses. Unterschiede sind

unterschiedlicher process identifier (PID)

der Ergebniswert von fork()

Vater: PID des Kindprozesses

Kind: 0

Beispielprogramm

Kind hat vieles mit dem Vater gemeinsam:

liest dieselben Dateien, gleicher Benutzername, benutzt dieselben Daten

Unix Kind fängt mit dem Code des Vaters an und ändert sich dann

Systemaufruf exec(): ersetzt das Programmbild des Vaters mit einem anderen.

Beispielprogramm für exec

Prinzipablauf

Bei der Verwaltung von Vater-/Kindprozess sind eine Reihe von Entscheidungen zu treffen:

Ausführung

Vaterprozess und Kind werden gleichzeitig ausgeführt, oder

der Vaterprozess wartet darauf, dass das Kind beendet wird

Ressourcen

Vater und Kind teilen sich alle Ressourcen.

Vater und Kind teilen sich einen Teil der Ressourcen.

Vater und Kind haben keine Ressourcen gemeinsam.

Adressraum

Das Kind ist ein Duplikat des Vaters.

Das Kind führt durch automatisches Laden ein neues Programm aus (exec-Systemaufruf).

Threads

Das Kind dupliziert alle Threads des Vaters.

Das Kind dupliziert nur den Thread des Vaters, der die fork-Operation ausgelöst hat.

Wir unterscheiden ***Benutzerprozesse*** , die Anwendungsprogrammen in Ausführung entsprechen, und ***Systemprozesse*** , die Programme/Dienste des Betriebssystems durchführen.

a) Jeder Prozess besitzt einen eigenen Prozessadressraum.

b) Spezielle Systemprozesse sind die ***Dämonen*** (engl. daemon); das sind Hilfsprozesse, die ständig existieren, die meiste Zeit aber passiv sind.

Dienste der Prozessverwaltung

Prozesskontrollblock

Prozesslisten

Zustandsmodell

Prozesserzeugung

Prozesse und Vererbung