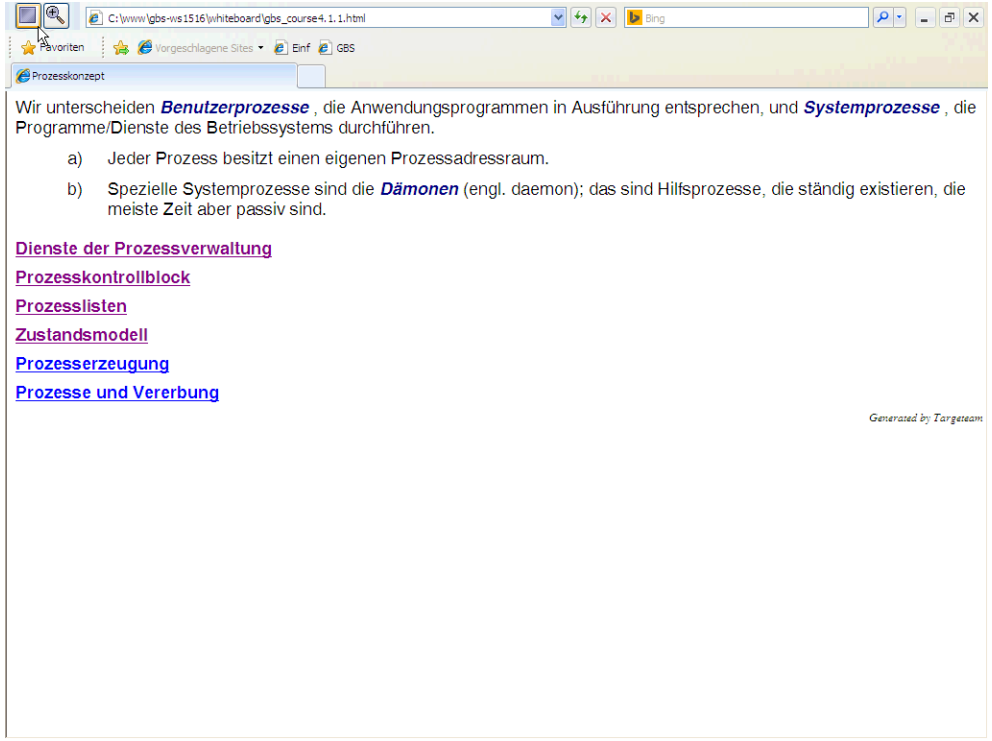
Script **generated by TTT**

Title: Grundlagen_Betriebssysteme (18.11.2015)

Date: Wed Nov 18 13:15:35 CET 2015

Duration: 45:25 min

Pages: 14



Prozesserzeugung

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.

Generated by Targeteam

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

Beispielprogramm

```
#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

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

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

Prozesse und Vererbung

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.

Dispatcher

Aufgabe des Dispatchers: Realisieren der Zustandsübergänge zwischen ***rechnend*** und ***rechenwillig*** : Prozessor binden und entbinden. Dazu ist ein Kontextwechsel erforderlich.

Kontextwechsel

CPU wird entzogen und einer anderen Aktivität zugeteilt; ein Kontextwechsel ist erforderlich, falls der rechnende Prozess P1 in den Zustand ***wartend*** oder z.B. durch Prozessorentzug in den Zustand ***rechenwillig*** übergeführt wird.

Problem

aktueller Ausführungskontext des Prozesses muss gesichert werden und Kontext des nächsten rechenbereiten Prozesses muss geladen werden.

Achtung : je umfangreicher ein PCB ist, desto "teurer" sind Prozesswechsel, d.h. das Umschalten der CPU zwischen den Prozessen.

Threads

Threads haben einen sehr viel kleineren Kontext ⇒ Umschalten zwischen Threads innerhalb eines Prozesses sehr schnell, da Adressraum und andere Ressourcen (z.B. Dateien) gemeinsam genutzt werden.

Beispiel: Kontext-Wechsel in Unix

Beispiel: Kontext-Wechsel in Unix

Kontextwechsel z.B. durch den Aufruf der Systemoperation sleep durch einen Prozess.

1. Maskieren von Interrupts;
2. Lokalisieren der benötigten Warteschlange;
3. Neuberechnung der Priorität des Prozesses;
4. Einfügen des Prozesses in die Warteschlange;
5. Aufruf der Operation zum Kontextwechsel.

Arbeitsmodi

Ziel für den Betrieb von Rechensystemen: kontrollierter Zugriff auf Hardwarekomponenten nur durch BS.

Lösung: alle Zugriffe auf Hardware nur über ***privilegierte*** Befehle zulässig;

Frage: wer darf privilegierte Befehle ausführen ⇒ Antwort: Prozesse in einem privilegierten Modus.

Herkömmlich unterscheidet man zwischen dem Benutzer- (engl. user mode) und dem Systemmodus (engl. kernel mode).

Benutzermodus

nur nicht privilegierten Befehle; Zugriff auf Prozessadressraum und unkritische Register (Befehlszähler, Indexregister); Benutzerprozesse im Benutzermodus.

Systemmodus

Es sind auch die privilegierten Befehle verfügbar (z.B. Anhalten der Maschine, Verändern der Ablaufpriorität). Die Dienste des Betriebssystemkerns werden im Systemmodus ausgeführt.

Nutzung der Hardware-Komponenten nur über Dienste des BS: Aufruf eines BS-Dienstes über spezielle Befehle: den ***Systemaufruf*** .

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.

Lesen von Daten aus einer Datei und Kopieren in eine andere Datei. Dabei treten die folgenden Systemaufrufe auf:

(1) Schreiben des Prompts auf Bildschirm: Angabe der Dateinamen
(2) Lesen der Tastatureingabe (bzw. Analoges bei Mouse-Eingabe)
(3) Öffnen der zu lesenden Datei (open)
(4) Erzeugen der neuen Datei
(5) ggf. Fehlerbehandlung: Nachricht auf Bildschirm
(6) Schleife: Lesen von Eingabedatei (ein Systemaufruf) und schreiben in zweite Datei (auch Systemaufruf)
(7) Schließen beider Dateien
(8) Ausgabe auf Bildschirm

Durch die Verwendung von Laufzeitroutinen ergibt sich eine höhere Abstraktionsebene ⇒ Aufruf einer Routine, die die notwendigen Systemaufrufe durchführt.

Prozessverwaltung

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