K1 Prozessdatenverarbeitung

1.1 Einführung

1.1.1 Ziel der Prozessdatentechnik

= Lenkung eines technischen Prozesses
- Wichtig: Einhaltung von strikten Bearbeitungszeiten in zeitlichen Abfolgen (= Realzeitbedingungen)
- Steuerung des Vorgangs mit Prozessrechner mit Realzeitbetriebsystem

1.1.2 Automatisierung von Prozessen

Ebenen eines automatisierten Prozesses:
- Vorgang
- Vorrichtung um Vorgang zu steuern (automatische Steuerung = Automat)

Automat
- EVA Struktur: kausale Folge von Eingabe, Verarbeitung und Ausgabe
- Entscheidungen zwischen Möglichkeiten hängen von der Eingabe und/oder dem aktuellen Zustands des Systems ab
- Fehlende Eigenschaften:
  - freie Entscheidungsfindung
  - Kreativität zur Verbesserung
  - Abstraktionsfähigkeiten ...

Automatisierung
= Einführung von Automaten in beliebige Abläufe

Automatisierung
= Umfang in dem Vorgang/Prozess automatisiert ist

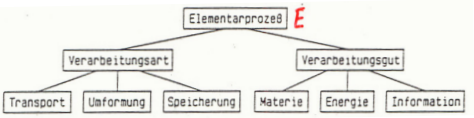
1.1.3 Ziele der Automatisierung in der PDV

- Kosteneinsparung
- Automatisierungsbereiche
  - bei monotonen, regelmässigen Arbeiten
  - Arbeiten mit hoher Konzentration
  - Arbeiten mit hoher Prozessgeschwindigkeit
  - Arbeiten mit großer Datenflut
- Grenzen
  - Problem der Vollständigkeit der Automatisierung
    = System darf nur auf einprogrammierte Ereignisse reagieren
  - Problem der Korrektheit eines Programms
    = Programmfehler in seltenen Anlagenzuständen können lange bestehen bleiben
  - Problem der mangelhaften Erkennungsfähigkeit

1.2 Begriffe und deren Bedeutung in der PDV

1.2.1 Prozess allgemein, Unterscheidung nach Verarbeitungsart und Verarbeitungsgut

Prozess
= Ein Prozess ist der Vorgang zur Umformung, zum Transport oder zur Speicherung von Materie, Energie oder Information



Elementarprozess
= genau eine Verarbeitungsart und genau ein Verarbeitungsgut

Einzelprozess
= kleinste geschlossene Prozesseinheit, in der ein kompletter Bearbeitungsablauf durchgeführt wird
- besteht aus mehreren Elementarprozessen

Verbundprozess
= mehrere Einzelprozesse zusammen, die eine gemeinsame Aufabge erledigen

Betriebsprozess
= Umfasst verschiedene Verarbeitungsbereiche im Betrieb

Computer Integrated Manufacturing (CIM)
= Einbeziehung aller Betriebsbereiche in einen Prozess
- Ziel: Betriebsverbesserung mittels
  - globaler zentraler Strategie (=Optimierung von oben)
  - Strategie der kleinen Iterationen (= stetige lokale Verbesserung von unten)

1.2.2 Der technische Prozess

= Zustandsgrößen werden mit technischen Mitteln erfasst und beeinflusst (gemessen, gesteuert, geregelt)
- Schnittstelle Prozessrechner <=> Prozess = Sensoren und Aktoren

Struktur:

Führungsebene	Führungsrechner
Fernbus	
Prozeßebene	Prozessrechner / SPS
Nahbus	
Signalumsetzer und Stromversorgung	
Technischer Prozess	Sensoren / Aktoren

1.2.3 Klassifikationsmerkmale von techn. Prozessen

Unterscheidung nach Verarbeitungsstruktur
- Kontinuierliche / stetige Verarbeitung
  = Einzelteile können nicht voneinander identifiziert werden
- Diskontinuierliche / unstetige / diskrete Prozesse
  = Stückprozesse mit einzelidentifiziertbaren Produkten
- Hybride / Chargenprozesse
  = Mischform, d.h. Herstellung von zusammengehörenden Teilmengen als Konti Prozess

Unterscheidung nach innerer Ablaufstruktur
- Deterministische Ablaufstruktur
- Stochastische Ablaufstruktur

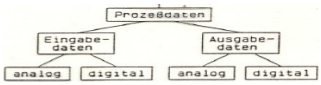
Unterscheidung nach Einsatzgebiet
- Fertigung = Stück-&Erzeugnisprozesse von Material
- Verfahrenstechnik = Konti- und Chargenprozesse von Gütern und Energie
- Verteilungsprozesse
- Verwaltung = EDV
- Mess- & Prüfprozesse = Forschung, Dienstleistung

1.2.4 Prozessrechner

= Rechner der mittels Prozessperipherie direkt an den technischen Prozess gekoppelt ist
1.) Prozessperipherie (Aktoren / Sensoren)
2.) Pegelumformer
3.) Interface (Analog I/O, Digital I/O)
4.) Prozeßbus
5.) Prozssrechner = Realzeitrechnersystem

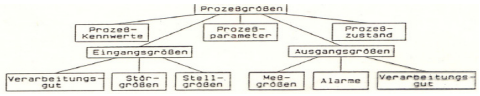
1.2.5 Prozessdaten und Prozessgrößen

Prozessdaten
= Daten zwischen Prozessrechner und Prozess

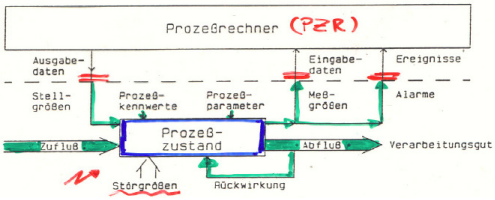


- zusätzlich Alarme = Sonderform von Eingabedaten

Prozessgrößen
= techn., physikalische, chem. Größen die mit techn. Mitteln erfasst und beeinflusst werden können
- Beschreibung unabhängig von Lenkung vom Prozessrechner
- Charakteristische Größen
  - Speicherprozess :[Menge] zB kg, J
  - Transport: :[Durchsatz] zB kg/s, W
  - Umformung :[Art und Umfang der Änderung]
    zB Lochbohren in Metall: Späne + Werkstück



- Prozesskennwerte = technische Festwerte, Grenzwerte
- Eingangsgrößen = Fließen in Prozess
  - Zufluss von Verarbeitungsgut = konti oder Charge
  - Störgrößen
  - Stellgrößen
- Prozessparameter = Festwert für einen Durchlauf
- Ausgangsgrößen = Rückwirkung auf Prozess möglich
  - Meßgrößen = Sollen keine Rückwirkung haben
  - Alarme = Besonders wichtige Ereignisse
  - Abfluss von Verabeitungsgut



1.2.6 Rechenprozess, Task und Multitasking

Rechenprozess
= Umformung, Verarbeitung und Transport von Informationen
- Aufgabe: Verarbeitung der erfassten Messgrößen aus dem techn. Prozess und dessen Beeinflussung

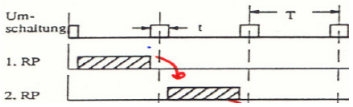
Programm
= statische Aufschreibung von Befehlen (Code)

Rechenprozess = Task
= dynm. Ablauf eines Programms in einem Rechner
- im Single-Prozessor-System kann immer nur ein Rechenprozess aktiv sein
=> Gleichzeitige Verarbeitung nur durch Aufteilung
- Mehrere Rechenprozesse können auf das selbe Programm zugreifen brauchen aber alle einen eigenen Datenbereich
- Berechnung auf verteiltem Prozessrechnersystem oder einem zentralen Rechner

Multitasking
= Quasiparallele Ausführung mehrerer Rechenprozesse, falls genug Rechnerleistung vorhanden

Kontextswitch
= Wechsel eines laufnden Prozesses auf einen anderen

Overhead
= Verlust von Rechenzeit durch Kontextswitch
- Berechnung als % der Dauer der Zeitscheibe
Overhead = Kontextswitchzeit t / Zeitscheibendauer T



- Berechnung bei kleinen Zeitscheiben

$L = L_{max} - t / T * L_{max}$

L_{max} = Max. Rechenleistung

$t/T * L_{max}$ = Verlust

=> Schnellere Hardware => t sinkt, T steigt

1.2.7 Grundprinzipien der Rechner-Prozesskoppelung

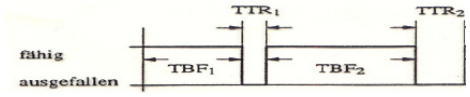
- offline Betrieb
  = Mensch transportiert Daten zwischen Prozess und Rechner
- online Betrieb
  = Rechner und Prozess über Peripherie gekoppelt
  - offener Betrieb
    = Datenerfassung (Prozess => PR)
    = Datenausgabe, steuerung (Prozess <= PR)
  - closed-loop Betrieb
    = Regelung (Prozess <=> PR)

1.2.8 Zuverlässigkeit und Sicherheit

Zuverlässigkeit
= Fähigkeit eines Systems unter vorgegebener Zeitdauer und zulässigen Betriebsbedingungen die spezifizierte Funktion zu erbringen.
- Beeinträchtigt durch Fehler und ggfs. Funktionsausfall
- Ausfall der Anlage führt immer zu hohen Kosten

Sicherheit
= Sicherheit ist das Nichtvorhandensein einer Gefahr für den Menschen, die Umwelt und den Sachwert

zu 1.2.8 Verfügbarkeit (1)



TBF = Time Between Failure
MTBF = Mean TBF (Mittelwert aller Fälle)
TTR = Time To Repair inkl. Fehlerkennung
MTTR = Mean TTR (Mittelwert alle Fälle)

Stationärer Fall p,q != f(t)

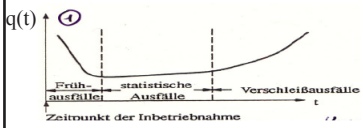
$$\text{Verfügbarkeit } p = \frac{MTBF}{MTTR + MTBF}$$

$$\text{Unverfügbarkeit } q = \frac{MTTR}{MTTR + MTBF}$$

$$\text{mit } p + q = 1$$

Ausfallrate: $\lambda = 1 / MTBF$
Reparaturrate: $r = 1 / MTTR$

Nicht stationärer Fall



Bei (1) : MTTR hoch : Lange Repzeit, Neue Anlage
MTBF niedrig : Frühausfälle

zu 1.2.8 Fehlertolerante Prozessrechnersysteme zur Erhöhung der Verfügbarkeit (1)

Fehler
= Abweichung des Gesamtsystems oder Teilen des Systems vom gewünschten / geplanten Verhalten
- Fehlertypen
  - Permanent = Bei auftreten dauerhaft anliegend
  - Transient = Nach auftreten schnell wieder weg
- Fehlerkategorien
  - Entwurfsfehler
  => Vermeidung: diversitäter Lösung (d.h. mehrere Entwicklungsteams)
  - Herstellungs- oder physikalische Fehler
  => Vermeidung: Qualitätssicherung
- Betriebs- und Bedienfehler
  - Störungen, Bedienfehler, Wartungsfehler
  => Vermeidung: Schulung des Personals

=> 1.) Fehlervermeidung wenn möglich
=> 2.) Fehlertoleranz (=Redundanz) falls 1.) nicht möglich
= Funktion wird weiterhin erfüllt obwohl ein Fehler aufgetreten ist
- Fehler muss sicher erkannt werden
- Redundanz / Ersatz vorhanden zB Parallelschaltung
- Nach Fehlererkennung kompensiert Redundanz den Fehler um Funktion zu erbringen

zu 1.2.8 Verfügbarkeit (2)

Annahme: Einzelkomponenten voneinander statistisch unabhängig
Gesucht: Gesamtverfügbarkeit der Systems

a) Serienschaltung der Einzelkomponenten

K_i = Einzelkomponente
p_i = Verfügbarkeit der Einzelkomponente K_i
q_i = Unverfügbarkeit K_i

Gesamtverfügbarkeit: $P_{gesamt} = \prod_{i=1}^{n} P_i$
(Ziel: $P_i = 1$)

$$P_{gesamt} = 1 - q_{gesamt} = \prod_{i=1}^{n}(1 - q_i) \approx 1 - \sum_{i=1}^{n} q_i$$

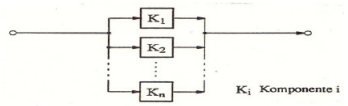
Gesamtausfallrate: $\lambda_{gesamt} = \sum_{i=1}^{n} \lambda_i$

=> $MTBF_{gesamt} = \frac{1}{\sum_{i=1}^{n} \frac{1}{MTBF_i}}$

Gesamtunverfügbarkeit:
(Ziel: $q_i << 10^{-2}$) $q_{gesamt} \approx \sum_{i=1}^{n} q_i = n \cdot q_i$

zu 1.2.8 Verfügbarkeit (2)

b) Parallelschaltung der Einzelkomponenten



$q_{gesamt} = \prod_{i=1}^{n} q_i$! "parallel"

$P_{gesamt} = 1 - q_{gesamt} = 1 - \prod_{i=1}^{n} q_i$

zu 1.2.8 Fehlertolerante Prozessrechnersysteme zur Erhöhung der Verfügbarkeit (2)

- Fehlererkennung durch Diagnosemodell gemäß FMEA (Failure Mode + Effect Analysis)
- Fehlererkennung auch durch
  - zykl. Hard-/Softwaretest
  - Hardware-Voter (2 aus 3 ...)
  - Software-Voter
  - Assertions (Zusicherungen) = vergleich mit vorab bekannten Zwischenergebnissen

zu 1.2.8 Arten der Redundanz

- dynamische Redundanz
  = Ersatz übernimmt erst wenn Fehler aufgetreten ist
  - Probleme:
    - Sichere Fehlererkennung
    - Einphasen des Ersatzsystems
    => Lösung: Backward-Recovery-Systeme
    = Erstellen von Checkpoints an die zurückgekehrt werden kann

- statische Redundanz
  = Kein Unterschied zwischen Ersatz- und Hauptsystem
  - räumlich parallel = mehrfache HW
    => SRU Konzept (smallest replacable unit), bei Fehler wird betroffene HW manuell durch SRU ersetzt
  - zeitlich sequetiell = selbe Ablauf wird zeitlich hintereinander wiederholt

1.2.9 Sicherheitsaspekte

- Sicherheit wichtiger als Verfügbarkeit
- Sicherheitskritischer Fehler => Anlage in fail-safe
- Annahme:
  3-fach redundante HW $q_{hw} << 10^{-4}$
  idealer Voter (q=0)

Mögliche Betriebsarten:

1.) Sicherheitsbetrieb
= d.h. Bei Fehler fährt Anlage in fail-safe Zustand
- Unverfügbarkeit: $q_{ges} = 1-p_{ges} = 1-(1-q_{hw})^3 = ca.\ 3q_{hw}$
- Wahrscheinlichkeit eines unsicheren Zustands W_a
  = Unerkannter Fehler
  hier: Alle 3 HW fallen zeitgleich aus und produzieren den gleichen Fehler => Voter findet keinen Fehler

$$W_{as} \le q^3 \quad q_{hw} \approx 10^{-4} \quad W_{as} \approx 10^{-12}$$

2.) TMR-Betrieb (Triple Modular Redundancy)
= 2 aus 3 Mehrheitsentscheidung
- Unverfügbarkeit: $q_{ges} = q^3 + 3 * (q^2 * (1-q))$
  q^3 = Wahrscheinlichkeit alle HW fallen aus
  $q^2(1-q)$ = Wahrscheinlichkeit zwei HW ausgefallen
- W_a
  hier: 3 oder 2 HW ausgefallen und liefern gleiches falsches Ergebnis

$$W_{atmr} \le q^3 + 3 * (q^2 * (1-q)) \approx 10^{-7}$$

3.) Höchste Verfügbarkeit
= Betrieb solang eine HW noch funktionsfähig
- Unverfügbarkeit: $q_{ges} = q_{hw}^3$
  = Ausfall erst wenn alle HW ausgefallen
- W_a : Voter entscheidet sich für fehlerhaften Rechner als richten Fehler wenn ein Fehler auftritt

$$W_{av} \le 3*q$$

1.3 Prozessbeschreibung (-modelle) und Systembetrachtung

- Prozess = Prozessbeschreibung + Prozessmodell
- Mögliche Modellformen
  - Gegenständliches Modell
  - Abstraktes oder mathematisches Modell
  - Rechnermodell mit Computer

1.3.1 Prozessbeschreibung

Vorgang:
1.) Strukturierung des Prozesses
2.) Zerlegung in Teilprozesse
3.) Verfeinerung bis zu Elementarprozessen
=> Elementarprozesse meist als mathematische Modelle

Prozessstrukturen meist graphisch:
- statische Strukturen mit Blockschaltbildern
- stationäre Abläufe mit Flussplan, Netzplan
- dynamische Abläufe mit Zustandsdiagramm, PETRI

1.3.2 Prozesserkennung (Modell-Bildung)

Methoden, Wahl je nach Prozess
- Empirische Methode
  => Bei nachträglicher Automatisierung eines Prozess, Beschreibung anhand von Erfahrungsdaten (=Prozessidentifikation)
- Analystische Methode
  => Beschreibung des Ablaufs durch Gleichungen (Automatentheorie)

Adaptive Prozessmodelle
- Unbekannte Prozessgrößen (zB Störgrößen) werden im laufenden Betrieb erfasst und als Paramter in die Berechung eingezogen
- Problem: Schwer vorhersagbar

Lernende Prozessmodelle
- Programme die aktiv suchend die optimalen Werte ermitteln
- Bisher sehr selten

Vorgehen bei Prozesserkennung:
1.) Elementarprozeß besimmen
2.) Ablauf bestimmen
3.) Merkmale (Prozessart, Verarbeitungsstruktur, innere Ablaufstruktur siehe 1.2.1 und 1.2.3)
4.) Methode wählen (idR analytisch d.h. Formel)

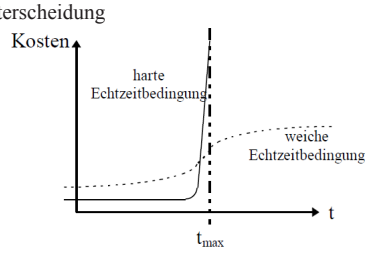
Allgemeines:
- Modellbildung meisten unzureichend da Störgrößen nur approximiert
  => iterative Verbesserung des Modells

K2 Grundlagen Echtzeitbetriebssystem

E1 Allgemeines

Echtzeitverhalten
= Der Rechner (PZR) muss mit den Vorgängen des technischen Prozesses, die er erfassen und steuern soll, trägheitslos Schritt halten können.

Unterscheidung



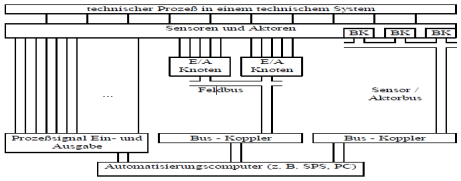
Problem der zeitlichen Synchronisation
= Eingabe / Ausgabe benötigen ebenfalls Zeit
=> Priorisierung notwendig, da Rechenzeit nur bei einem Task liegen kann

Determiniertheit = EVA Verhalten
= Ein Echtzeitsystem heißt determiniert, wenn sich für jeden möglichen Zustand und für jede Menge an Eingangsinformationen eine eindeutige Menge von Ausgabeinformationen und ein eindeutiger nächster Zustand bestimmen läßt.

E2 Prozesssignalanbindung und PZR-interne Verarbeitung

- Prozesssignale müssen gemäß ihrer Zeitanforderung rechtzeitig verarbeitet werden
=> Prozesssignale müssen durchgereicht werden
=> Signalverarbeitung muss rechtzeitig erfolgen

Möglichkeiten zur Signalanbindung



links E/A Verdrahtung:
- Vorteil: je nach Komplexität ggfs. günstiger
- Probleme: evlt auch sehr teur, EMV anfällig

mitte Feldbus:
- Problem: Defintive Aussagen zu Zeitverhalten nötig

rechts Sensor/Aktorbus:
- Vorteil: striktes, zyklisches Verfahren (kein Polling)
- Problem: Leitungsbruch führt zu Gesamtausfall (außer bei redudanten Kabeln)

Detektion eines Signalswechsels:
- Polling
= periodisches Abfragen aller Teilnehmer
- sehr rechenintensiv und zeitaufwändig
- Hardwareinterrupts / ISR
- mehrere Interrupteingänge für unterschiedliche externe Ereignisse nötig, da diese gleichzeitig auftreten können

Prozesssignale
= externe Ereignisse
- Können gleichzeitig auftreten
- Verarbeitung: Annahme in ISR durch Devicetreiber, danach Umwandling in Softwareereignis

Softwareinterrupts /-signale
= interne Ereignisse
- Dienen zur Tasksynchronisation
- Können NICHT zeitgleich auftreten

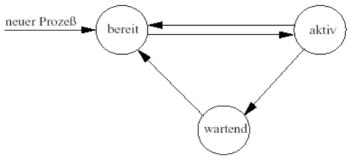
Ereignisse (events, signals)
= Prozesssignale + Softwaresignale
- Dienen zur Aktivierung von Tasks

E3 Tasks (Rechenprozess)

Programm
= statische Aufschreibung von Befehlen (Code)

Task / Rechenprozess
= geladenes Programm im ausgeführten Zustand

Taskeigenschaften
- TaskID
- Priorität
- Privater Adressraum, Zugriff auf Betriebsmittel
- Muss Prozesssignale rechtzeitig bedienen
- Bearbeitung
  - unterbrechbar und fortsetzbar
  - oder teilweise ununterbrechbar
  - oder ununterbrechbar
- Min/max Taskreaktionszeit (Ende - Start +Wartezeiten)
- Min/max Verarbeitungszeit
- Prozesszeit = Wiederkehr desselben Prozesssignals
- Taskzustand
  - Bereit / Rechenwillig = Warten auf Prozessor
  - Aktiv / Rechnend = Aktiv im Prozessor
  - Warten auf ... = Kein Anfrage auf Prozessorzeit

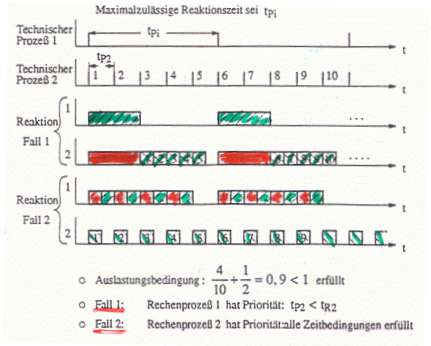


=> Da meist mehrere Tasks gleichzeit rechenwillig sind, ist eine Verteilung (Scheduling) notwendig

E4 Tasksscheduling = Zuteilung von Rechenzeit

E4.1 Ausgangssituation

Realisierung der Software im Prozessrechner als
- Embedded System ohne Betriebssystem
- Mit Echtzeitbetriebssystem

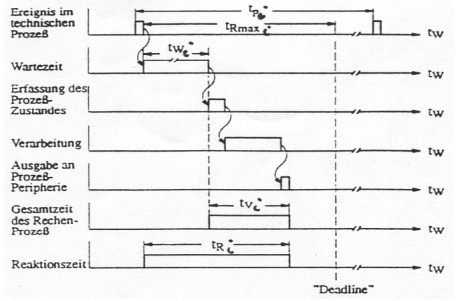


=> je nach Priorität wird das Zeitverhalten beeinflusst

Wunsch und Lösung
- Zerlegung des Programms in Teilprogramme mit einzelnen Prozessereignissen
=> Mehrere Tasks die Daten austauschen
- Zuteilung der Rechenzeit der CPU an die einzelnen Tasks derart das die Realzeitbedingungen erfüllt werden

E4.2 Überprüfung des Echzeitverhaltens

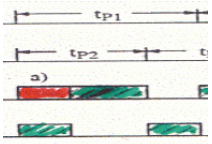
Notwendige Zeitgrößen



t_{pi} = minimale Prozesszeit des Prozesssignals
t_{wi} = Wartezeit der Task i (zB durch Unterbrechung)
t_{vi} = Netto-maximale-Verarbeitungszeit der Task i
WCET = Worst Case Execution Time (Schätzung)
BCET = Best Case Execution Time (Schätzung)
$t_{Ri} = t_{vi} + t_{wi}$ = erreichte / tatsächliche Reaktionszeit
$t_{Ri,max}$ = vorgegebene maximal erlaubte Reaktionszeit
$t_{d,max}$ = max Deadline zur Bearbeitung Prozesssignals i
$t_{Ri,min}$ = vorgegebene minimal erlaubte Reaktionszeit
$t_{di,min}$ = min. Deadline zur Bearbeitung Prozesssignal i

zu E4.2 Notwendige Bedingungen für Echtzeit

1.) Gleichzeitigkeit (notwendig)



$t_{pi,1} = 7$
$t_{pi,2} = 5$
$t_{vi,1} = 3$ (grün)
$t_{vi,2} = 2$ (grün)

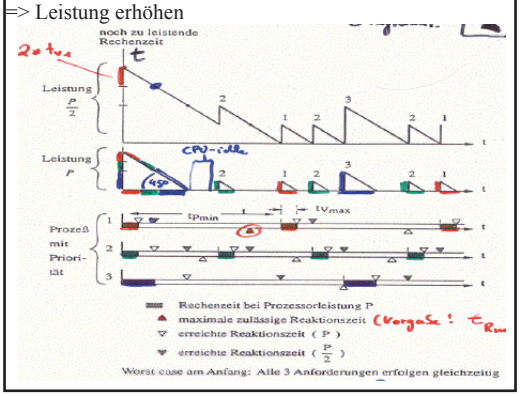
Die Auslastung A der Rechnerkapazitäten muss < 1 sein

$$A = \sum_{alle} \frac{tvi\,max}{tpi\,min} < 1$$ hier: A = 3/7 + 2/5 = 0,83 < 1

2.) Rechtzeitigkeit (notwendig)
- Für jedes einzelne auftretende Prozessignal i muss gelten: $t_{Ri,min} < t_{Ri} < t_{Ri,max}$
=> Überprüfung Graphisch mit A-t-Diagramm oder Algebraisch siehe Beispiel E4.3

zu E4.2 Verstoss gegen Echtzeitbedingungen

=> Leistung erhöhen



E4.3 Rechenzeit Zuteilungsstrategie (Taskscheduling)

Grundprinzipien
- Zyklische Zuteilung
- Ereignisgesteuerte, priorisierte Zuteilung

Zyklische CPU-Zuteilung an die Tasks:
= Jede Task erhält periodisch ausreichend Rechenzeit um seine individuelle Zeitanforderung einzuhalten
- round-robin Verfahren
= periodische Verteilung von gleichgroßen oder individuell unterschiedlichen Zeitscheiben an die Tasks
- Rückgabe der Zeitscheibe möglich falls Zeit nicht gebraucht
- mind. 1 interruptgebender Timer nötig
- Prozesssignalwechselerkennung durch Polling
- Kontextswitchverluste
= Rechenzeitverlust beim Umladen der Register bei Taskwechsel auf dem Prozessor
- Maximale Taskreaktionszeit:

$$T_{RMAXi} = T * (N * \lceil T_{Vi}/T \rceil - N + 1)$$

T = Timelsice
N = Anzahl Tasks
T_{vi} = Verarbeitungszeiten
$\lceil T_{Vi}/T \rceil$ = Gauss Klammern, Ergebnis auf nächste ganze Zahl runden zB $\lceil 5/9 \rceil = 1$

Grundprinzip ereignisgesteuerte priorisierte Zuteilung:
- Priorisierung der Tasks mit dazugehörigem Prozesssignal
=> Unterschiedlich wichtige Prozessereignisse müssen wichtiger sein
- Taskwechsel erfolgen mittels ISR durch Umformung des externen Ereignisses in ein internes Ereignis
- Vorteil: Belastung des Systems nur bei Auftreten von Prozesssignalen
- Prüfung durch Scheduler
Kriterien:
  - Ende einer Systemfunktion
  - RBS-Systemfunktion hat aktiven Prozess blockeirt
  - Prioriätenänderung
  - Aktiver Prozess wurde termininiert
  - Interrups
  - Timer-Interrupts
  - Höher Priorer Prozess wartet

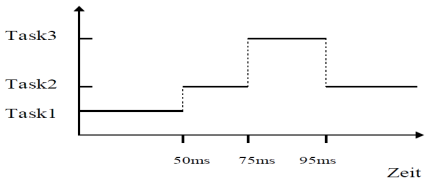
Alternative Bewertung nach ...
- Gerechtigkeit
- Effizienz
- Durchlaufzeit
- Durchsatz
- Antwortzeit

zu E4.3 Ratenmonotones Prioritätsscheduling (RMS)

Vereinbarungen:
- Prioritäten statisch und niemals gleich
- Tasks haben Prozesssignale
- A < 100%
- Alle Tasks sind unterbrechbar und fortsetzbar
- Verarbeitungszeiten bleiben konstant
- **Taskprioritätsverteilung:**
  - höchste Priorität an Task mit Prozesssignal mit höchster Wiederholrate
  - zweithöchste an PS mit zweithöchster Widrate usw.
- Unterbrechung durch höherpriore Tasks sofort

Algorithmus:
- Externe oder interne Ereignisse löst den Prioritäts-scheduler aus der dann die laufwillige Task mit höchster Priorität aktiviert

Darstellung von Rechenprozessen (A-t-Diagramm)



zu E4.3 Algebraische Überprüfung

Alternative zur Graphischen Variante:

1.) $U_{RMS} = n^* (2^{1/n} - 1) * 100\%$
Falls $A \le U_{RMS}$ Echtzeitfähigkeit garantiert
=> Falls $U_{RMS} < A < 100\%$ zusätzliche Prüfung nötig

2.) Annahme: Task j höherprior als Task i
HPRIO = Tasks die Höherprior sind

3.) Reaktionszeitzeit $T_{Ri} = T_{vi} + T_{wi}$

$$T_{Wi} = \sum_{\forall j \in HPRIO} \left(\lceil T_{Ri} / T_{Pj} \rceil * T_{Vj} \right)$$

$$T_{Ri} = T_{Vi} + \sum_{\forall j \in HPRIO} \left(\lceil T_{Ri} / T_{Pj} \rceil * T_{Vj} \right)$$

4.) Lösung durch Rekursion für n = 0, ..., n
1. Schritt: $T^0_{Ri} = T_{vi}$

... n $$T^{n+1}_{Ri} = T_{Vi} + \sum_{\forall j \in HPRIO} \left(\lceil T^n_{Ri} / T_{Pj} \rceil * T_{Vj} \right)$$

5.) Lösung existiert falls
T^0_{Ri} .... T^n_{Ri} konvergiert und $T^{n+1}_{Ri} = T^n_{Ri}$ existent für alle Tasks

zu E4.3 OS-9 Timesharing Algorithmus

- Mischform: Zyklisch + Ereignisgesteuerte prioritätsgesteuertes Scheduling
=> Echtzeit und Nicht-Echtzeit-Tasks im System
- Teilung nach Priorität
  Echtzeittasks: Priorität 256 bis 65535
  Nicht-Echtzeit-Tasks: Priorität 0 bis 255
  - Aging: pro Timeslice indem rechenwilliger Prozess nicht bedient wurde Priorität (Age) + 1 bis 255
  - Nach Rechenscheibe => Rückstufung auf Startprio und +1 für alle anderen Prozesse

zu E4.3 Auswahl bekannter scheduling Verfahren

1.) Statisches Scheduling vor Laufzeit des Systems
= für sicherheitsrelevante Systeme
- Streng zyklische Zuteilung gemäß Planung
- Nicht gebrauchte Rechenzeit verfällt

2.) Dynamisches Scheduling zur Laufzeit des Systems
= Zuteilung gemäß aktueller Bedarfsituation
=> Prozesse müssen unterbrechbar (preemptiv) sein

2a) First Come First Serve (FCFS)
= für Batch Systeme um gleiche, mittlere Wartezeiten zu erzeugen
- Bereite Prozesse werden in Warteschlage nach Erzeugungszeitpunkt eingeordnet
- Jeder Prozess läuft bis zum Ende, außer er geht in den Blockiert Zustand
- Bei Wiedereintritt wird der laufende Prozess nicht unterbrochen

2b) Round-Robin (Zeitscheibenverfahren)
- Jeder Prozess in einer Warteschlange erhält eine Zeitscheibe (time slice)
- Prozesswechsel (freiwillig oder Zeitscheibenende)
  - Aktueller Prozess wird verdrängt
  - Aktueller Prozess ans Ende der Warteschlange
  - Erster Prozess in Warteschlange erhält CPU
- Geeignetes Verhältnis Zeitscheibe / Kontextwechsel
- Große Zeitscheibe: Effizient aber lange Wartezeiten
- Kleine Zeitscheibe: Kurze Antwortzeiten aber großer Overhead durch häufige Prozesswechsel

2c) Time-Division-Multiplexing
- Zyklische Zuweisung der Zeitscheiben
- bei n Tasks: Je Task bekommt 1/n CPU Zeit

2d) Deadline-Scheduling
- Scheduling gemäß Priorität der Tasks
- Dynamische Prorität je nach Deadline
- Prorität der Task wächst, je näher Taskdeadline
- Verfahren
  - EDF = Earliest Deadline First, Prozess dessen Frist als nächstes endet bekommt CPU
  - LST = Least Slack Time, Prozess mit geringstem Spielraum erhält CPU
    Spielraum = Deadline - Bereitzeit der Task- Ausführungszeit der Task
  - Vorteil: Frühere Fristverletzungserkennung

2e) Scheduling nach POSIX 1003.1b
- Prioritätsgesteuertes Scheduling
- Prioritätsebenen mit mehreren Prozessen
- In jeder Ebene FCFS oder Round Robing
- Prio 0 = niedrigste Priorität

zu E4.3 Beispiel Überprüfung der Echtzeitfähigkeit

RechenProzess P_i	Prozesszeit T_{pimin}	Max. Reaktionszeit erlaubt T_{Rmaxi}	Verarbeitungszeit T_{vimax}
P1	150 ms	150 ms	30 ms
P2	100 ms	100 ms	10 ms
P3	200 ms	200 ms	100 ms

Bei Zeiten Worst Case annehmen
Kontextswitch wird vernachlässigt

Vorgehen
0.) Weitere Bedinungen Prüfen:
Gleichzeitigkeit
A = 30/150 + 10 /100 + 100/200 = 0.80 < 1
U_{RMS} = 3 ($2^{1/3}$-1) =0.78 => also 78% < A*

1.) Prioritäten gemäß Wiederholungsrate ($T_{pi,min}$)
=> P2(rot) > P1(blau) > P3(grün)

2.) Ereignisse mit Verarbeitungszeit ($T_{vi,max}$) für jeden Prozess einzeichnen

3.) Maximal zulässige Reaktionszeit (T_{Rmaxi}) für jeden Prozess einzeichnen. Zeichen: Δ

4.) Prozessbelegung eintragen, wobei höherpriore Tasks niederpriore bei Auftreten verdrängen. Verdrängte Tasks bearbeiten den Rest anschließend.

5.) Erzielte Reaktionszeit eintragen:
Ende in Prozessorbelegung => ∇ für jeden Prozess

6.) Echtzeit gegeben falls ∇ niemals Δ überschreitet

zu E.4 Beispiel: Algebraische Überprüfung

$$T^{n+1}_{Ri} = T_{Vi} + \sum_{\forall j \in HPRIO} \left(\lceil T^n_{Ri} / T_{Pj} \rceil * T_{Vj} \right)$$

Vorgehen: Formel für alle Tasks prüfen
1.) Überprüfung Task P3 (niederpriorste)
HPRIO = Task P2 und Task P1

T^{n+1}_{R3}	=	T_{V3}	+	$T^n_{R3}/T_{P1} * T_{V1}$ +	+	$T^n_{R3}/T_{P2} * T_{V2}$ +
140	=	100		⌈100/150⌉ * 30 +		⌈100/100⌉ * 10+
150	=	100		⌈140/150⌉ * 30+		⌈140/100⌉ * 10+
150	=	100		⌈150/150⌉ * 30+		⌈150/100⌉ * 10+

► Lösung existiert — Task P1 — Task P2
= weil 150 Endwert => konvergiert

2.) Überprüfung Task P1
HPRIO = Task P2

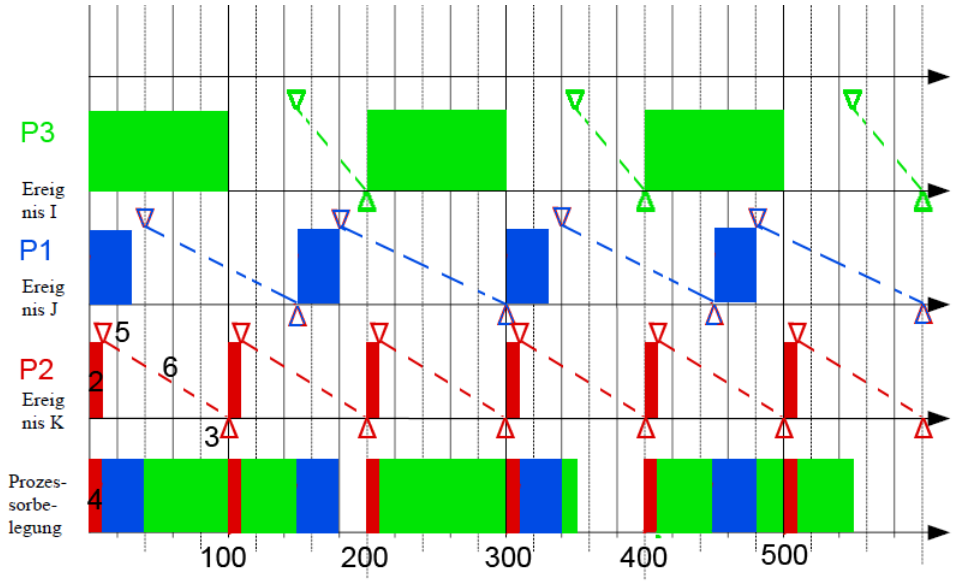
$T^{n+1}_{R1} = T_{V1} + T^n_{R1}/T_{P2} * T_{V2}$

40	30	⌈30/100⌉ * 10+
40	30	⌈40/100⌉ * 10+

► Lösung existiert — Task P2
= weil 40 Endwert => konvergiert

3.) Überprüfung Task P2
HPRIO = Keine
=> $T^{n+1}_{R2} = T^n_{R2} = T^0_{R2} = T_{V2} = 10{,}0$ ms , Lösung existiert

=> Echtzeitfähig laut Algebraischem Verfahren



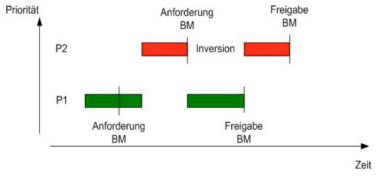
E 4.4 Prioritätsinversion

= Falls niederpriorer Prozess einen höherprioren blockiert
- Situation entsteht bei gemeinsam exklusiv nutzbaren Betriebsmitteln
- insb. gemeinsam genutzer Speicher(Shared Memory)
=> Es darf immer nur eine Task in den Speicher schreiben / lesen
=> Höhere Tasks muss ggfs. warten bis niedere fertig

=> Prioritäsinversion nötig

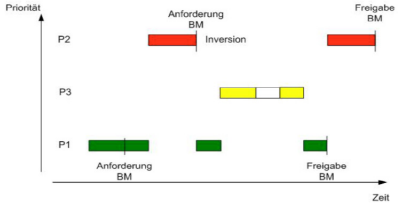
Arten von Prioritätsinversion:

a) begrenzte (bounded) Prioritätsinversion
= P1 blockiert P2 für einen begrenzten Zeitraum



- kann nicht vermieden werden

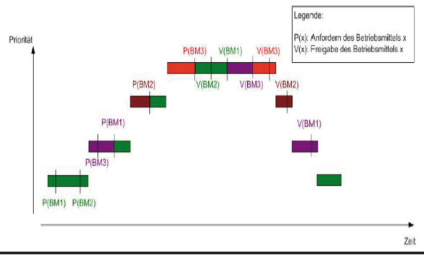
b) unbegrenzte (unbounded) Prioritätsinversion
= P3 blockiert P1 für beliebig langen Zeitraum
=> P1 blockiert P2 belibig lange => Lösung nötig



=> Lösung durch Priovererberbung oder Priogrenzen

zu E 4.4 Prioritätsvererbung

= Lösung für Problem der unbegrenzten Prioinversion
= Niederpriorer Prozess erbt bei Inversion die Priorität des höheren Prozesses bis BM wieder freigegeben
=> Mittlere Prozesse können nicht mehr blockieren (P3)
=> unbegrenzte Inversion wird begrenzt auf Dauer des kritischten Abschnitts
=> Es entstehen Blockierungsketten
ABER Deadlocks weiterhin möglich



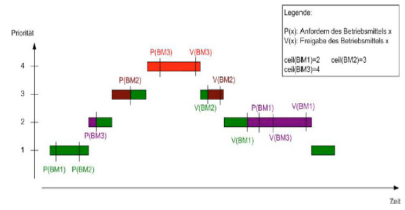
zu E 4.4 Prioritätsgrenzen

= Lösung für Problem der unbegrenzten Prioinversion
= Lösung zum Verhindern von Deadlocks
= Jedes Betriebsmittel s (BMs) erhält eine Prioritätsgrenze ceil(s), dieses entspricht dem Maximum der Prioritäten aller Prozesse j, die je auf s Zugreifen könnten
- Anfang: Alle BMs unbelegt aktceil=0

Zuteilungsregel:
- **BMs belegt** durch anderen Prozess j
=> Prozess p blockiert sofort (1)
- **BMs frei**
=> Prüfung bestimme aktceil=Maximum aller ceil(BMx) im Augenblick zugeteilter BMs (2)
- aktprio(p) > actceil, BMs wird zugeteilt(2.1)
- aktprio(p) <= actceil, BMs wird nur zugeteilt wenn
- Prozess p bereits BMx passend zu aktuellem actceil besitzt(2.2)
- sonst blockiert Prozess p an freiem! BMs (2.3)

- Zusätzlich Vererbung bis Freigabe alle Ressourcen BMx inkl BMs

Spezialfall: Immediate Priority Ceiling
= ceiling: Prozesse, die ein Betriebsmittel s belegen, bekommen sofort die Priorität ceil(s) zugewiesen



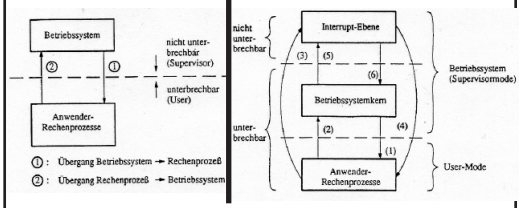
K3 Echtzeitbetriebssystem OS9

E5.1 Einführung

Realzeitbetreibssystem
= Bereitstellung Infrastruktur zur Verwaltung des PZR
= Aufgabe eines Realzeit-Betriebssystems (insbesondere des Schedulers,), den einzelnen Rechenprozessen zu jedem Zeitpunkt die prozessbedingte erforderliche Rechenzeit just-in-time Verfügung zu stellen.
- Verwaltet Betriebsmittel (CPU, Speicher, E/A, Kommunikations und Synchronisationsressourcen)
- Anforderung: Garantierte max. Reaktionszeit auf Ereignisse

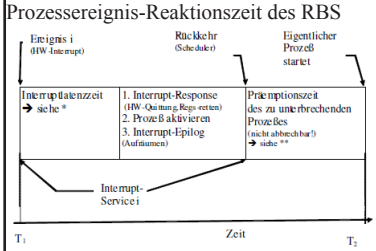
Unterscheidung normales BS <=> RBS
- Forderung an RBS: Prozesse unterbrechbar

normales OS RBS



Abtrennung OS<=>Anwender
- Verschiedene HW Betriebsmodi (Supervisor <=> User) auf CPU mit unterschiedlichen Befehlssätzen
- Adressabschirmung (OS Calls)
- HW Zugriff je nach Betriebsmodus
- Speicherschutz durch Memory Protection oder Memory Management Unit

Prozessereignis-Reaktionszeit des RBS



Ereigniss i
=> Interrupt i
=> Task i an
=> Task n aus

|----------Interruptlatenzzeit ----------|

* worst case: Maximum aus längster Befehl des CPU-Befehlsvorrats und längste programmgesteuerte Interruptsperrzeit
** RBS Aufruf der blockiert und verzögert (zB Schedular)

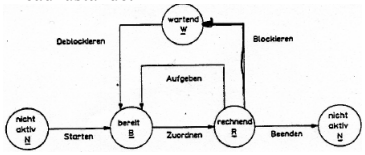
zu E5.1 weitere Eigenschaften typischer RBS

- dynamische Prioritätensteuerung der Tasks
- Tasksynchronisation und -kommunikation
- Minimaler RAM Footprint
- Anwenderschnittstellen:
- Command Shell = Bedienoberfläche
- System Calls = Aufrufschnittstelle, C Befehle

E5.2 Task- und Threadkonzept der RBS

E5.2.1 Task-/Threadmodell und Zustände

- Scheduler wird bei jedem System Call aktiv
- Task hat Haupthread und ggfs. weitere Threads
- Thread = Ablauffaden (Programmcoutner) der eigenständig Programmcode abarbeitet
- Threadeigenschaften:
- Teilt sich Datenbereich einer Task
- Eigener Stack und Registersatz
- ThreadID bzw. HaupthreadID, gültig nur während der Laufzeit (in OS9: max 65535 Threads)
- Threadmultiplexing = Schnelles Umschalten zwischen Tasks / Threads durch Scheduler um quasiparallele Abarbeitung zu erzeügen
- Threadzustände:



- bereit = Warten auf CPU
- rechnend = Thread hat CPU
- warten = Warten auf Ereignis

E5.2.2 Threadzustandswechsel

Vertagung (blockieren, suspend, OS9: sleeping)
= Zustandswechsel Rechnend => Warten
Auslösung durch
- Thread gibt CPU ab und wartet auf Ereignis
- BM Anforderung aber BM nicht verfügbar
- OS9: Device -I/O-/Thread-Condition-Variable – Queue (Serielle Warteschschlange)
- Parent-Task wartet auf Beendigung seiner Kinder

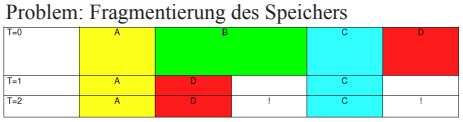
Fortsetzung (deblocking)
= Zustandswechsel Wartend => Bereit
Auslösung durch
- Ereignis, auf das gewartet wird, tritt ein
- OS9: Thread wird geweckt, sobald Signal eintritt unabhängig vom eigentlichen Eventwert
- BM wird frei, auf das gewartet wurde

zu E.5.2 Threadzustandswechsel

Verdrängung
= Zustandswechsel Rechnend => Bereit
Auslösung durch
- höherpriorer Task wird bereit gesetzt
- OS9: Interrupt => Treiber löst Event aus
- zugeteilte Zeitscheibe abgelaufen
- OS9: alle 1ms Uhreninterrupt

Zuteilung
= Zustandswechsel Bereit => Rechnend
Auslösung durch
- RBS teilt Thread mit höhster Prio die CPU für maximale Dauer seiner TICs zu
- OS9: Anzahl der TIC pro Thread = 1

Thread starten
= Zustandswechsel Nicht aktiv => Bereit (falls möglich)
- Starten des Hauptthreads der Task mit TaskID und BM
- OS9: Arbeitsspeicherzuteilung aus dem vorhandenen freien Pool, dann Laden des Programmcodes
- Problem: Fragmentierung des Speichers



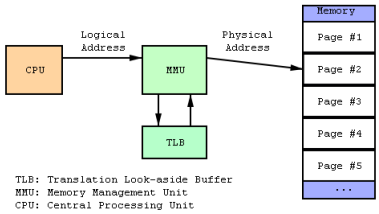
- T=0: B&D rechnend, T=1 B&D fertig, D startet neu
- T=2: B möchte neu starten, aber kein zusammenhängender Speicher verfügbar
=> Lösung: Fertige Tasks gehen wartend oder MMU

Thread beenden
= Zustandswechsel Bereit, Wartend, Rechnend => Nicht aktiv
- Alle Laufdaten werden gelöscht, Arbeitsspeicherzuteilung und BM wieder freigegeben

5.2.3 Thread- und Taskverwaltung

- Anwender-Programmcode, Initialisierungdaten
  OS9: Prozessmodul Teil Objekt-Code
- Anwender-Daten
  OS9: Prozessmodul - Teil Daten-Area
- Laufzeitdaten und Laufzeitparameter zusammengefasst in Task-/Threadkontrollblock
  OS9: ein Systemstack

Schutzmöglichkeiten von Task-Speicherbereichen
- ungeschützt, alle Task gemeinsam (OS9:atomic kernel)
- geschützt durch Memory Management Unit (MMU) oder Memory Protection MPU
  MMU:
  - Aufgabe: Umrechnung virtuelle => physische Adresse, Schutzt durch Trennung der Arbeitsspeicherbereiche (horizontal und vertikal)
  - Trennt CPU und RAM
  - Paged MMU: Adressumsetzung gemäß Seiten bzw. festen Speicherblöcken („Page“)



E5.2.4 Task-/Threaderzeugung und -entfernung

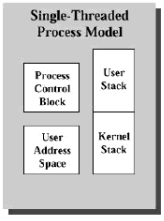
- Per Kommando über die mshell
- Durch andere Task/Threads programgesteuert

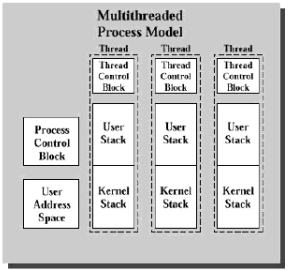
Task und Threads in OS9:
- Bei zwei oder mehr Tasks an einer Aufgabe (Multithreading)
  - Zeitliche Synchronisation mit
    - Events (Condition Variable, OS9 Events)
    - Semaphore (Mutex, OS9 Semaphore)
    - Signalen
  - Datenaustausch über gemeinsamen Speicher (OS9: Datenmodul)

=> Es gibt in einer Task mehrere Threads mit:
- gemeinsamen globalen statischen Speicherbereich für Taskverwaltugnsdaten
- jeder Thread hat weiterhin eigenen Stack, Registersatz, Programmcounter, Priorität und SignalISR

=> Nachteil: wenn ein Thread abstürzt, wird die gesamte Task beendet

Taskmodelle in OS9 gemäß IEEE POSIX 1003.1c

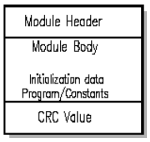




Threadeigenschaften:
- ThreadID (1 - 65535)
- ParentID (=welcher Task hat Thread gestartet)
- Task: Haupthread + Nebenthread
- Eigener Registersatz und Stack
- Priorität, Signalempfang und Eventfähigkeit
- Voreingestellte Signalempfangsroutine
- Child<=> Parent System
  - detached Threads (ParentID=0) / Orphans
  - normal Threads (ParentID!=0), auf diese kann gewartet werden
- Threads nur für user-state Programme

E5.2.5 OS9 Modulearchitektur

= übergeordnete Einheit die im ASP vom Betriebssystem verwaltet wird
- Modulaufbau



- Module müsse reentrant und position independant sein
- Verwaltung im ASP durch Modulverzeichnis
- Zugriff über Berechtigung im Modulkopf
- Modul besitzt Revisionsnummer (für Softwareupdates), es wird immer nur das Modul mit der höhsten Revnummer verwendet

OS9-Geräte und Dateisystem
- hierarchisches Dateisystem
- nicht Case-sensitive
- Zugriffsschutzmechanismen
  - User/Group ID (0,0 = Super User)
  - Änderung der Rechte nur durch Ersteller oder SU

E5.2.6 OS9 Systemstart

1.) Hochfahren OS9 Kernel (Einsprung im Resetvektor)
2.) Init-Modul
3.) Urtask wird gestartet (mshell)

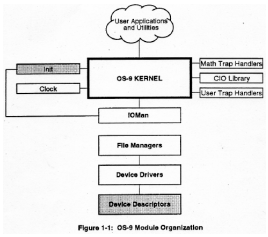


Figure 1-1: OS-9 Module Organization

E5.3 Tasksynchronisation

A) Taskumschaltung durch aktives Steuerung der Prioritäten der Tasks zur Laufzeit
- Prioritäten vergabe durch den Systemverwaltet
- Änderung durch Kommandos oder Aging

B) Freiwillige vorzeitige Rückgabe via Taskscheduler

C) Sperren des Taskwechsel (task-locking)
- ununterbrechbarer Task-Programmabschnitt
- Alternative: tempoäre Erhöhung auf Maxpriorität

D) Vertagung und Fortsetzung mittels Event
- Vertagung bis Even eintrifft:
  - warten auf Nachricht von anderer Task / BM
  - warten auf Eventerzeugung durch Interrupt
  - warten auf Timerablauf
- Befinden sich in wait-for-Event-Queue

zu E5.3 D) OS9 Events

- Eindeutiger Evantname und EventID
- Event = 32 bit Zahlenwert
- Warten auf bestimmten Zahlenwert oder Wertbereich
- Eventaufbau - Listenelemente:

Event ID (32-bit)
Länge des Eventnamens(16 bit)
Zeiger auf Eventname
Linkcount (links)
Zugriffsrechte (MP_OWNER_READ | MP_OWNER_WRITE |MP_GROUP_READ | MP_GROUP_WRITE |MP_WORLD_READ | MP_WORLD_WRITE)
OwnerID
Waitinkrement (Winc 16-bit)
Signalisierungsinkrement (Sinc 16bit)
Zahlenwert (32-bit)
Pointer to next event in event-list (next)
Pointer to previous event in event-list (prev)

- Winc = Waitinkrement, wird bei erhöht falls eine Task die auf Evenzustand wartete geweckt wird
- Sinc = Signalisierungsinkrement, beim Setzen
- Linkcount = Wieviele Tasks ein Event benutzen, nur falls = 0 können Events gelöscht werden
- Next & Prev = Listenverwaltung

zu E5.3 Synronisation mit Signals und Sleeps

Signale
= Signale sind sog. Software-Interrupts die programmgesteuert von einer auslösenden Task an eine oder alle Anwendungstasks gesendet werden können
- bestehts aus Signalnummer (signal code) und EmpfängerTaskID

- 256 bis 4294967295 : User-definable
- Signal 0 = kill an eigene Tasks
- Signal 1 = Wake Up einer Task ohne Empfangsroutine
- Signal 2/3 = Keyboard Interrupts
- Signal 4 = kill
- Broadcast von Signalen möglich
- Empfangen von Signalen mit Signal-Empfangsroutine

zu E5.3 Synronisation mit Signals und Sleeps

Signale senden:
- Sofortiges Ausführen der EmpfangsISR ohne Taskwechsel
  => Setzen einer global-statischen Variablen (Volatile)
  => Rückkehr aus SignalISR

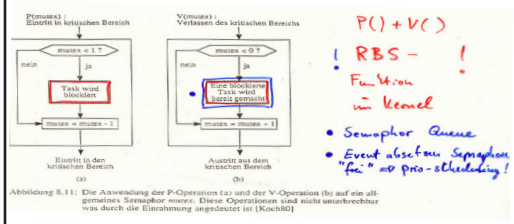
Weitere Reaktion:
- Fall 1: Empfangstask irgendwo im Code und wartet nicht auf Auftreten des Signals
  - Abfrage durch Programmierung, sobald Task Rechenzeit scheduled bekommt
- Fall 2: Empfangstask per os_sleep suspendiert
  - Task wird durch Signalempfang oder Ablauf des Sleep Timers geweckt

Schutz kritischer Abschnitte gegen Signalunterbrechung:
- Signalempfangsmaske einer Task erlaubt nur Signalempfang falls = 0 => Wert ändern

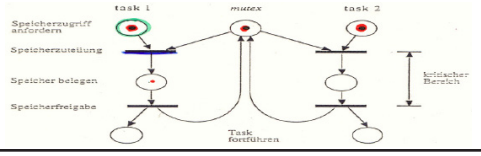
Sonstiges:
- Signale werden sequentiell immer 1x durch Signalinterceptroutine bearbeitet
- Unbearbeite Signale werden gequeued
- Sonderform: Timer-Alarme
  = RBS schickt nach einer einstellbaren Zeit ein Signal
  - Zyklisch oder Einmalig

zu E5.3 Tasksynchronisation durch Semaphore

Semaphore
= spezielles Event welches zur Zugriffssteuerung auf ein begrenzt verfügbares Betriebsmittel dient



- Realisierung:
  - Semaphor Operationen P() und V()
  - Anzahl der exklusiven BM = n
  1.) Anlegen eines Events
  2.) Vorbelegen des Event-Wertes mit n
  3.) Winc -1, Sinc 1
  => Belegen des BM mit P() => Mutex -1
  => Freigeben des BM mit V() => Mutex +1



zu E5.3 Petri Netze

- für nebenläufige Prozesse und Synchronisationsaufgaben

Elemente:

- STELLEN und TRANSITIONEN haben Ein- und Ausgänge
- STELLEN und TRANSITIONEN werden über unidirektionale gerichtete Pfeile miteinender verbunden
-STELLEN und TRANSITIONEN wechseln sich ab; es werden NIE zwei TRANSITIONEN und/oder STELLENaufeinenderfolgend verbunden.

Stellen = Zustände:

- aktive STELLEN enthalten MARKEN, inaktive keine
- jede STELLE besitzt eine MAXIMALE Aufnahmekapazität von MARKEN, default: unendlich viele Marken möglich

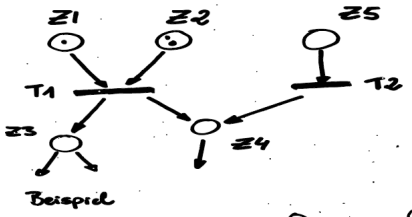
Transitionen = Weiterschaltbedingungen

- können boolsche Gleichungen sein
- verknüpft alle über die gerichteten Pfeile angeschlossenen STELLEN
- nur aktiv wenn Bedingungen erfüllt

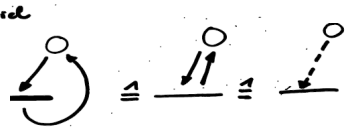
Weiterschalten:

- Geschaltet wird wenn:
  - jede Stelle am Eingang der Transition eine Marke enthält UND
  - die Transistionsbedingung erfüllt ist
- Vorgang:
  - Jede Eingangsstelle verliert eine Marke
  - Jede Ausgangsstelle erhält genau eine Marke
  => Stellen ohne Marke werden inaktiv
  => Stelle mit Marken werden aktiv

Beispiel:



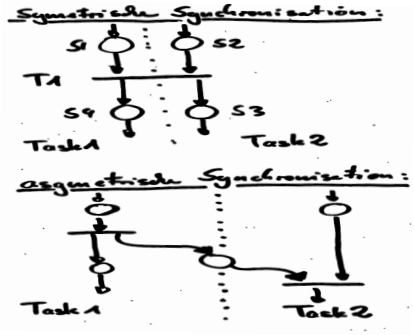
Abkürzungen:



Konsistenzprüfung:

- Verklemmungsdetektion: Gehen Marken verloren oder schalten Transistionen nicht?
- Kapazitätsüberschreitungen: Werden Stellenkapazitäten überschritten?

Beispiele: Synchronisation



zu E5.3 Synchronisation zwischen Threads

- Variablenzugriff über Semaphore um Dateninkonsistenzen zu vermeiden
- Für Semaphore **innerhalb einer Tasks** bzw. den Threads einer Task: Mutex Mechanimus
- Mutex
  - Variable, für alle Threads sicherbar
  - Belegungs / Freigabe Konzept
  - Falls Zugriff auf bereits belegte Mutex
  => Suspend bis Mutex verfügbar
- Alternativ: pthread-Condition-Variable (selten)

zu E5.3 Threadglobale und threadprivate Variable

- Alle Threads einer Task sehen globale und globale statische Variablen
- Funktionslokale Variablen sind threadprivat
- Funktionsstatische Variablen exsistieren bis zum Ende der Task
=> Alle Zugriffe von Thread auf gemeinsam genutzte Variable, die nicht funktionslokal sind, müssen synronisiert werden (zB mit Semaphore) um Dateninkonsistenzen zu vermeiden

zu E5.4 Intertaskkommunikation

= Möglichkeiten zum Austausch von Nachrichten der Prozesse untereinander

A) Messagebuffers (nicht in OS9)

- Ablage von Nachrichten in Poolelemente (buffer)
- Weitergabe über TaskID
- Pool meist als verkettete Liste mit FIFO oder LIFO Prinzip

B) Globaler gemeinsamer Speicher (memory mailbox via shared memory)

- Gemeinsamer Speicherbereich im OS9 Datenmodul
- Zugriffssteuerung über Events oder Semaphore
- Sichtbarkeit durch MMU und RBS je nach Zugriffsrecht
- Bei Multiprozesssorsystemen: jeder hat lokale Kopie, diese wird konstant synchronisiert

Shared Memory (OS9 Datenmodul)

- Speicherschutzmechanismen mittels MMU
- Datenmodul = Modulname + Größe
- Zugriff durch anlinken je nach Zugriffsrecht
=> Datenmodullinkcount wird erhöht
=> Freigabe wieder nötig

Pipes

- Zur unidirektitonalen Daten-Kommunikation zwischen zwei oder mehr Prozessen
- Ein / Mehrere Sender => Ein Empfänger
- Zwischenspeicher: FIFO Queue („Pipe“)
- 128 Byte größ, named pipes ggfs größer
- Arten
  - Unnamed Pipes
    - Entstehen falls mehre Kommandos mittels ! verkettet werden
  - Named Pipes
    - Haben Namen und sind auf Device/pipe abgelegt
    - Lässen sich wie Dateien mit Kommandos ansprechen
    - Zugriffsverwaltung wie bei Dateien
- Leere Pipes die von keinem Prozess geöffnet sind werden automatisch gelöscht
- Besonderheiten beim r/w von named Pipes:
  - Schreiben auf volle Pipe => suspend bis Platz frei
  - Lesen von leerer Pipe => EOF Fehler
  - Lesen wenn zu wenig Daten da und kein EOF im Pipe => Warten auf weitere Zeichen
  - Lesen wenn zu wenig Daten und EOF => Geht
  - Alle Benutzer schließen Zugriff zu named pipe
  => Falls noch Daten drin bleiobt sie bestehen