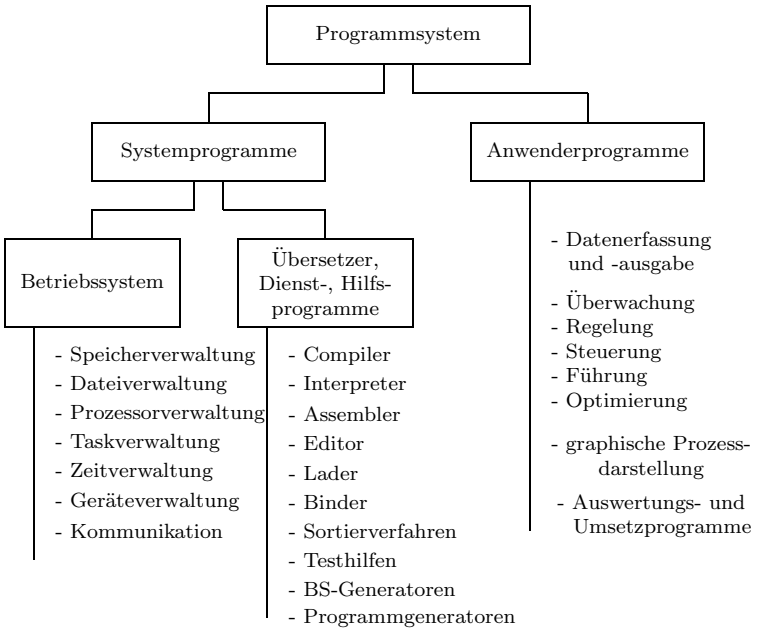
G Software in der Prozessautomatisierung



G.1 Echtzeitbetriebssysteme

- **Eigenschaft** eines Rechners neben der Hardware vor allem durch das **Betriebssystem** festgelegt
- Für die Prozessautomatisierung werden **Echtzeitbetriebssysteme** eingesetzt
- **Besonderheiten** von Echtzeitbetriebssystemen:
  - ◆ Rechner muss **schritthaltend** mit dem techn. Prozess arbeiten
  - ◆ Abarbeitung vieler Aufgaben im **Millisekundenbereich**
    - ➤ Reibungsloser Produktionsablauf
    - ➤ Optimale Nutzung von Maschinen
    - ➤ Beherrschbarkeit von Gefahrensituationen

Besonderheiten von Echtzeitbetriebssystemen(2)

- Für wichtige Aufgaben → **Deadlines**
  - **Hard Deadlines**
    - ➤ Müssen auf jeden Fall eingehalten werden
  - **Soft Deadlines**
    - ➤ Sollten mit grosser Wahrscheinlichkeit eingehalten werden

Besonders **wichtige Komponenten** von Echtzeitbetriebssystemen:

- **Taskverwaltung**
  - ➤ Scheduler, Prioritätensteuerung
- **Zeitverwaltung**
  - ➤ Start von Programmen zu bestimmten Zeiten bzw. nach dem Ablauf einer bestimmten Zeit
  - ➤ Zeitüberwachung, time out
- **Unterbrechungsverwaltung**
  - ➤ wichtige Ereignisse aus techn. Prozess über Interrupt an Rechner gemeldet
  - ➤ Reaktionszeit auf Ereignisse im techn. Prozess minimiert

1 Zeitverwaltung

- Echtzeitbetriebssystem muss in der Lage sein, die einzelnen Abläufe (Tasks) **zeitgesteuert** auszuführen
  - Starten und beenden von Tasks,
  - Werte abfragen und ausgeben
  - Zu vorgegebenen Zeitpunkten
  - Nach einer festen Zeit oder in festen Zeitintervallen
- Dafür wird ein **Absolutzeitgeber** bzw. ein **Differenzzeitgeber** benötigt
  - schickt zu **festen Uhrzeiten** oder nach einem **festen Zeitintervall** der Zentraleinheit Signale
  - woraufhin diese bestimmte Aktionen ausführt.
- Der **Absolutzeitgeber** wird z.B. zur Realisierung **zeitlicher Protokollierung** eingesetzt.
- Eine Anwendung für den **Differenzzeitgeber** ist z.B. das **zyklische** Abarbeiten von Aufgaben in fest definierten Zeitintervallen.

2 Unterbrechungsverwaltung

- Ein Prozessrechner benötigt eine **Unterbrechungsverwaltung** die nach Erhalt eines Unterbrechungsimpulses (Interrupt)
  - das laufende Programm schnell **stoppt**
  - anschließend wichtige Daten **rettet**
  - und höherpriore Programme **aktiviert**.
- **Auslöser** einer solchen Unterbrechung sind z.B.
  - Alarmmeldungen,
  - Kontaktendstellungen,
  - Zeitmeldungen
  - und Bereitmeldungen von E/A-Geräten.

- Nach **Start** einer Unterbrechungs-/Interrupt-Routine:
  - ➤ alle **wichtigen Daten** in speziell dafür vorgesehenen Registern **gerettet**
  - ➤ danach **höherpriore Programme gestartet.**
  - ➤ Den verschiedenen **Unterbrechungen** sind unterschiedliche **Prioritäten** zugeordnet.

- Wenn mehrere Unterbrechungen **gleichzeitig**, bzw. in kurzen Abständen auftreten, werden sie
  - ➤ **geschachtelt** und zwischengespeichert
  - ➤ und gemäß ihrer **Priorität** seriell **abgearbeitet**.

- Um bestimmte, wichtige Programmabläufe nicht zu unterbrechen, gibt es die Möglichkeit, sogenannte **Unterbrechungs-/Interruptmasken** zu setzen, die vorübergehend eine Unterbrechung verhindern.

- **Einzelne Aktionen der Unterbrechungsverwaltung:**

  - ◆ **Zwischenspeicherung** des Unterbrechungswunsches in entsprechenden Registern

  - ◆ **Ermittlung der Priorität** des Unterbrechungssignals und dessen Weiterleitung an das Leitwerk

  - ◆ **Vergleich der Unterbrechungspriorität** mit der des laufenden Tasks und gegebenfalls diesen blockieren

  - ◆ **Sperren weiterer Unterbrechungen** durch das Setzen von bestimmten Maskierungsbits

Einzelne Aktionen der Unterbrechungsverwaltung (2):

- **Retten der Informationen** über den bisher laufenden Task (Befehlszähler, Dateninhalte, Zustand etc)
- **Starten der Unterbrechungsantwort** mittels entsprechender Programmroutinen
- **Erlauben** weiterer, **höherpriorer Unterbrechungen** durch das Rücksetzen der Maskierungsbits.
- Nach Abarbeitung der Unterbrechung mit **normalem Programmablauf fortfahren**

Einzelzeiten der Unterbrechungsbearbeitung:

T1: Durchlasszeit, die Zeit, die benötigt wird, um vom Zeitpunkt des Auftretens der Unterbrechung zum Leitwerk durchzuschalten

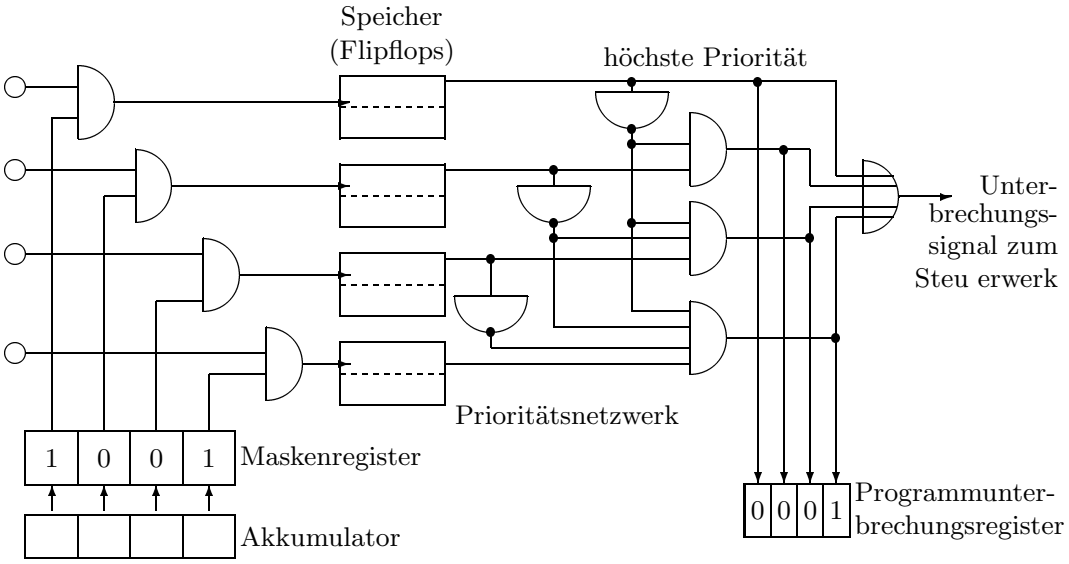
T2: Latenzzeit, die Zeit, die benötigt wird um den laufendenTask abzubrechen (da das nicht an jeder beliebigen Programmstelle möglich ist)

T3: **Erkennungszeit**, die Zeit, die bis zum Aufruf der richtigen Unterbrechungs-routine (Unterbrechungserkennung und Zuordnung) vergeht

T4: **Ausführungszeit**, die Zeit, die die Unterbrechungsroutine benötigt

T5: Rückkehrzeit, die Zeit, die vom Ende der Antwortroutine bis zum ersten Befehlsstart des nächsten regulären Tasks vergeht

Unterbrechungslogik:



- **FlipFlops**, zum Speichern der Unterbrechungsmeldung
- **Maskenregister** zum Maskieren der Unterbrechungsmeldung
- **Prioritätsnetz**, das die Speicherung und Prioritätsabarbeitung der Unterbrechungsmeldungen sicherstellt
- **Programmunterbrechungsregister** zum Abspeichern der aktuellen Unterbrechung
- **Register mit der Anfangsadresse** des startenden Antwortprogramms

Unterbrechungsarten

◆ **Traps**:

- **Unterbrechungen**, die aus dem **laufenden Programm** heraus bzw. vom Rechner selbst kommen.
- man spricht auch von einer **synchronen** Unterbrechung.
- Hierzu gehören:
  - ➤ Datenüberlauf
  - ➤ Speicherplatzüberschreitungen
  - ➤ Datenfehler (etwa durch Division durch Null),
  - ➤ Übertragungsfehler (fehlende Quittungen, Paritätsfehler, Adressierungsfehler),
  - ➤ Warten auf bestimmte Ereignisse,
  - ➤ Programmstart- bzw. Programmendemeldungen
  - ➤ Schreib-/Lesezugriffe auf vorher festgelegte Adressen

Unterbrechungsarten (2):

◆ **Interrupts**:

- Unterbrechungen, die von **nebenläufigen Geschehnissen**, externen Geräten oder Maschinenfehlern ausgelöst werden,
- sogenannte **asynchrone** Unterbrechungen.
  - ➤ Unterbrechungen, die vom **Bedienungspersonal** über Eingabemechanismen durchgeführt werden
    - – Stop-, Starttastenbetätigung,
    - – Tastatureingaben
  - ➤ Alle Arten von **Notfällen**:
    - – Prozessalarme durch Grenzwert- oder Bereichsmelder,
    - – Meldungen über gefährliche Umweltbedingungen wie Luftfeuchtigkeit, Temperaturüberschreitung etc.
    - – Gerätedefektmeldungen und Stromausfälle

Reaktionen auf Unterbrechungen (Antwortprogramme):

- **Programmabruch**
  - ➤ Falls aufgrund des aufgetretenen Fehlers eine **Weiterführung** des laufenden Programms **nicht möglich** ist (bei Datenfehlern, unzulässigen Operationen, etc.), wird dieses vom Betriebssystem aus **abgebrochen**.
  - ➤ Ein spezielles Organisationsprogramm versorgt entsprechende Standardprogramme mit den **notwendigen Daten** und **startet** sie (Bedienmeldungen, Fehlersuchprogramme, Protokollausdrucke usw.)

Reaktionen auf Unterbrechungen (2):

- Programmunterbrechung **mit Programmwechsel:**
  - ➤ Durch ein **Unterbrechungssignal** an das Betriebssystem wird das bisher **laufende Programm unterbrochen**, sein **Zustand** und seine Daten in entsprechende Register **gerettet** und ein **höherpriores Programm** zur Unterbrechungsbearbeitung **gestartet**.
  - ➤ Nach dessen **Beendigung** läuft das **zuvor unterbrochene Programm** ordnungsgemäß weiter (Standardfall in der Prozessautomatisierung)
- Programmunterbrechung **ohne Programmwechsel**
  - ➤ Wenn das laufende Programm nur **Informationen übernehmen** soll, z.B. Fertigmeldungen von E/A-Geräten, Geräteanforderungen oder Zeitimpulse, so muss es **kurzzeitig unterbrochen** und angehalten werden, um die entsprechenden Bits oder Bytes zu übernehmen.
  - ➤ Anschließend läuft es weiter, **ohne** dass ein **Programmwechsel** stattfinden muss.

3 Fehlerbehandlung

Error Isolation:

- Während des Ablaufs der einzelnen Tasks können unterschiedliche **Fehler** auftreten:
  - Bereichsüberschreitungen,
  - Datenfehler (Division durch Null),
  - Ausfall von Betriebsmitteln usw.

- Das Betriebssystem enthält **Fehlerroutinen**, die
  - feststellen, **wo** ein Fehler aufgetreten ist
  - und dafür sorgen, dass alle daran **nicht beteiligten Tasks ungestört** weiterlaufen können.

Error Recovery:

- Falls die entsprechenden Fehlerroutinen die **Störung selbst** wieder **beheben** können:
  - durch **Umleiten** der Anforderungen auf andere Geräte,
  - **Speicherrekonfiguration**,

  sollten alle Systemeinheiten wieder normal weiterarbeiten, unter bestimmten Umständen z.B. Ausfall von:
  - Prozessoren
  - Speichereinheiten

  auch mit **verminderter Leistungsfähigkeit** (graceful degradation).

4 Echtzeitbetrieb

- Beim Echtzeitbetrieb sind mit der Verarbeitung eines Auftrags **strenge Zeitbedingungen** verbunden.

  - **Nicht immer** dreht es sich dabei um **Schnelligkeit**, sondern es müssen bestimmte Aufgaben
    - ➤ zu **festen Zeitpunkten** bzw.
    - ➤ innerhalb fest vorgegebener **Zeitschranken** (deadlines)

    erledigt werden.

  - Der **zeitliche Bereich** ist von der jeweiligen **Anwendung** abhängig.
    - ➤ **Temperaturregelungen** i.a. relativ **zeitunkritisch**,
    - ➤ **Antriebsregelungen** fordern eine Reaktionszeit im **Millisekundenbereich**.
    - ➤ Die typische Reaktionszeit von automatisierten Prozessen liegt zwischen **10 und 1000 ms.**

- **Eigenschaften des Echtzeitbetriebssystems eines Prozessrechners:**

  - **Aktive Rechtzeitigkeit:**
    - Aufgrund bekannter, meist zyklischer Abläufe, deren Zeitpunkte er über einen Zeitgeber erhält, muss der Rechner
      - ➤ Programme starten, blockieren oder beenden
      - ➤ Daten ein- und ausgeben
      - ➤ periphere Geräte anstoßen

  - **Passive Rechtzeitigkeit:**
    - Auf unvorhergesehene Ereignisse (Alarmmeldungen) muss der Prozessrechner sofort reagieren können
      - ➤ d.h. laufende Programme unterbrechen und
      - ➤ Unterbrechungsroutinen starten.

Eigenschaften des Echtzeitbetriebssystems eines Prozessrechners (2):

◆ **Simultanverarbeitung**:

- Simultanverarbeitung **erster Art**:
  - ➤ Der Prozessrechner muss in der Lage sein, eine echte **Parallelarbeit** von **Peripherie** und **Zentraleinheit** zu leisten, d.h. bei gleichzeitigem Auftreten mehrerer Bedienungsanforderungen eine **Bearbeitungsreihenfolge** festlegen und für diese die Bedienung vornehmen (Koordination und Synchronisation)

- Simultanverarbeitung **zweiter Art:**
  - ➤ Auf einem Rechner laufen im allgemeinen **mehrere Programme** (Tasks) mit unterschiedlichen Aufgabenbereichen (Multi-Programming).
  - ➤ **Strategien** (Prioritäten, Round Robin RR, First Come First Served FCFS etc), für die **Zuteilung** der Zentraleinheit.

Eigenschaften des Echtzeitbetriebssytems eines Prozessrechners (3):

◆ **Vordergrund- und Hintergrundverarbeitung:**

- **Vordergrundprogramme**:
  - ➤ Programme die sich mit dem eigentlichen Prozessgeschehen befassen,
    - – Unterbrechungsbearbeitung,
    - – zyklische Prozessbearbeitung,
    - – Initiierung von Betriebssystemdiensten (Supervisorcall, SVC),

    sind nach Prioritäten gestaffelt.

- **Hintergrundprogramme**:
  - ➤ Weniger wichtige Verwaltungsprogramme
  - ➤ dienen zur Auslastung des Rechners
  - ➤ werden immer dann ausgeführt, wenn der Prozessor gerade Zeit zur Verfügung stellt.
  - ➤ Der Rechner wird dadurch optimal ausgenutzt.

5 Das PEARL Betriebssystem (PBS)

- Wurde zur Unterstützung der Prozessprogrammiersprache **PEARL** entwickelt.
- **Aufgabenen** von PBS:
  - **Steuerung** des parallelen Taskablaufs,
  - **Synchronisation** der einzelnen Tasks untereinander,
  - **Verwaltung von E/A-Aufträgen** und ein mehrprozessorfähiges Dateiverwaltungssystem,
  - **Verwaltung** und **Abarbeitung** von **Programmstarts** zu bestimmten Zeitpunkten oder aufgrund bestimmter Ereignisse (hier als Einplanungen bezeichnet)
  - Verwaltung von **Interrupts** und **Traps**.

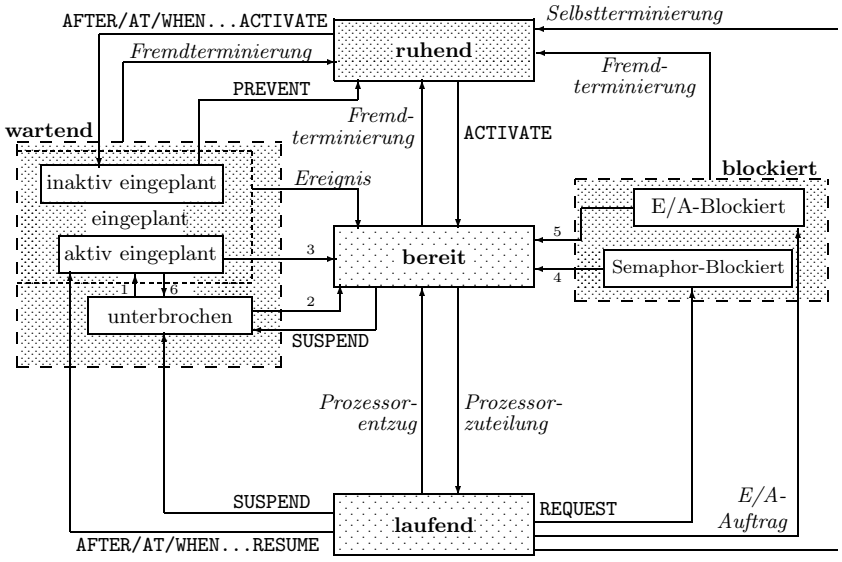
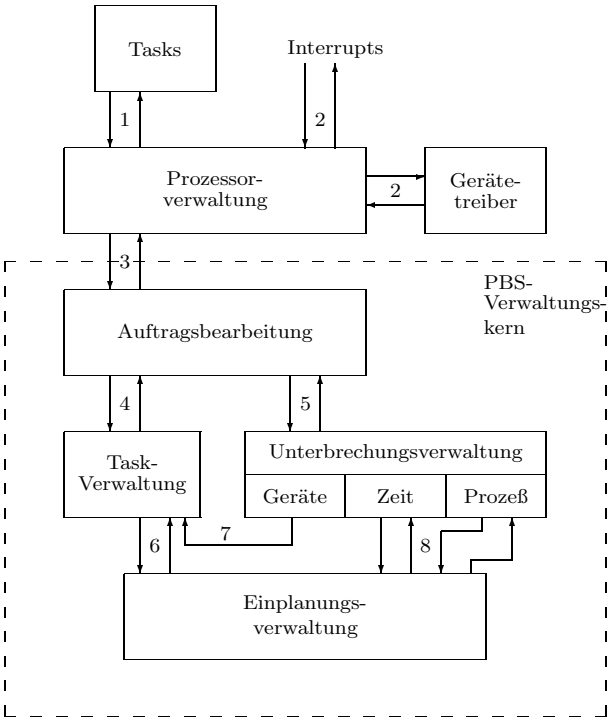


Fig. 1
Taskzustände und ihre Übergänge im PEARL-Betriebssystem
1: `AFTER/AT/WHEN...CONTINUE`; 2: `CONTINUE`; 3: *Ereignis*; 4: `RELEASE`; 5: *E/A-Ende*; 6: `PREVENT`

- **Fremdterminierung**: Beenden eines Tasks durch einen anderen
- **Selbstterminierung**: Beenden des Tasks durch das Erreichen seines Befehlsendes
- AFTER/AT/WHEN...RESUME bzw. ACTIVATE: **PEARL- Befehle** die die entsprechenden Zustandübergänge im PBS auslösen
- PBS besitzt ein **statisches Auftragskonzept:**
  - Es wird mit allen Tasks gestartet
  - Nachladen weiterer Tasks, also eine dynamische Erweiterung des Systems, ist nicht möglich
  - Es gibt keine Taskzustände vor dem Zustand "ruhend"
  - Im Zustand "ruhend" befindet sich ein Task vor seinem Start und nach seinem Ende.
  - Kein Task kann das System verlassen.

1

Zusammenspiel derBausteine des PEARL-Betriebssystems



- Bei **Neustart** des PEARL-Betriebssystems wird eine Initialisierungsroutine aufgerufen, die die PBS-**Verwaltungsstrukturen** aufbaut, z.B.:
  - **Semaphorverwaltungsblöcke**,
  - **Initialisierungs**- und **Interruptliste**,
  - **Taskverwaltungsblöcke**, die Zustand, Priorität, Programmzähler etc. beschreiben
  - **Geräteverwaltungsblöcke** mit Gerätenummer und Statuswort
  - **Auftragsparameterblöcke**, die u.a. Zeiger auf die entsprechenden Verwaltungsblöcke enthalten.
- Anschließend wird der **Starttask** aktiviert der verschiedene Betriebssystemaufrufe absetzen kann.
- Danach arbeiten die **einzelnen Bausteine** wie folgt zusammen:

Prozessorverwaltung

- Nach einem entsprechenden Algorithmus **teilt** sie die **CPU** den Subsystemen, Tasks, Gerätetreibern oder dem Betriebssystemkern PBS **zu**.
- Sie leitet sowohl die Aufträge von Tasks (1) als auch die Unterbrechungen (Gerätefertigmeldungen, Zeit- und Prozessinterrupts) (2) an die Auftragsbearbeitung des PBS weiter (3).
- Hat das PBS alle Aufträge erledigt, teilt die Prozessorverwaltung die CPU entweder dem unterbrochenen Objekt (2) (z.B. Treiber) oder dem höchstprioren Task (1) zu.
- Lag weder Fall (1) noch Fall (2) vor, versetzt sie das PBS in den Zustand "active wait", wo es auf weitere Aufträge wartet.
- Solche Aufträge können nur über Interrupts neu hinzukommen.
- Sind keine Aufträge mehr eingeplant, so gilt das Anwenderprogramm als beendet.

Auftragsbearbeitung

- Sie **verteilt** vorliegende **Aufträge**, je nach ihrer Herkunft, entweder an die **Taskverwaltung** (4) oder an die **Unterbrechungsverwaltung** (5).
- Aufträge, die von **Interrupts** kommen, werden dabei **bevorzugt** behandelt.
- Da **neue Aufträge** hinzukommen können, während das PBS in unterbrechbaren Codestücken läuft, kehrt die Auftragsbearbeitung erst dann wieder über die Prozessorverwaltung zum Subsystem Tasks zurück, wenn **kein weiterer Auftrag** mehr vorliegt.

Taskverwaltung

- Sie erledigt durch **Ein-/Ausketten** der betreffenden Taskkontrollblöcke in/aus prioritätsorientierten **Warteschlangen** alle einfachen Taskaufträge, wie z.B.
  - Aktivieren,
  - Fortsetzen,
  - Suspendieren,
  - Blockieren,
  - Terminieren von Tasks.
- Sie sorgt auch für das **Weiterleiten** von **E/A-Aufträgen** an Gerätetreiber und das Einketten in eine entsprechende **Gerätewarteschlange**.
- Alle Taskaufträge, die **Zeiteinplanungen** (Absolut- oder Relativzeit, einmalig oder zyklisch) betreffen sowie **Einplanungen** von **Interrupts** der Prozesshardware, gibt die Taskverwaltung an die Einplanungsverwaltung weiter (6).

Einplanungsvewaltung

- Einerseits bekommt sie Aufträge von der Taskverwaltung, wobei Tasks in eine **Zeitliste** bzw. in eine **Prozessinterruptliste** eingetragen werden.
- Andererseits prüft sie beim Eintreffen eines Interrupts (der von der Unterbrechungsverwaltung kommt) für welche Tasks **Zeit**- bzw. **Prozessinterrupt-Einplanungen** ganz oder teilweise abgelaufen sind.
- Diese Tasks übergibt sie dann der Taskverwaltung zur Fortsetzung bzw. kettet sie bei kombinierten oder zyklischen Einplanungen erneut in eine der Listen ein.

Unterbrechungsverwaltung

- Sie besteht aus **drei** Teilen, die jeweils einen Interrupttyp bearbeiten:
  - ➤ Die **Geräteverwaltung** stellt fest, für welchen Task eine Gerätefertigmeldung eingetroffen ist und übergibt diesen Task zur Fortsetzung an die Taskverwaltung (7).
  - ➤ Die **Zeitverwaltung** und die **Prozessinterruptverwaltung** prüfen, ob überhaupt entsprechende Einplanungen vorliegen und überlassen gegebenenfalls weitere Maßnahmen der Einplanungsverwaltung (8).

G.2 Programmiersprachen für die Prozessautomatisierung

- Programmiersprachen für Prozessautomatisierungsaufgaben haben wegen der **Echtzeit**- und **E/A-**Anforderungen andere und weitergehende **Sprachelemente** und dazugehörende Regeln (Syntax), als gewöhnlichen Programmiersprachen.

1 Anforderungen an Prozessprogrammiersprachen

- **Echtzeitbedingungen** müssen programmtechnisch **formulierbar** sein,
- die **Speicher**- und **Dateiverwaltung** muss durch entsprechende **Datentypen** und Objektbeschreibungen unterstützt werden,
- sie sollte sich aus **Modulen** zusammensetzen lassen,

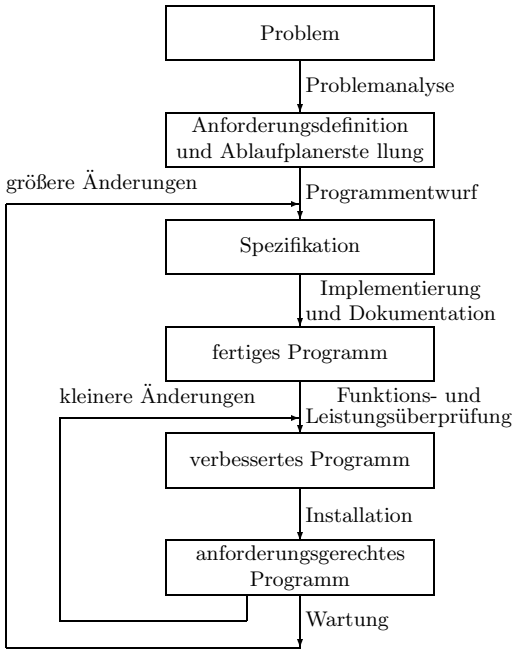
1 Anforderungen an Prozessprogrammiersprachen (2)

- sie sollte möglichst für viele **Prozesstypen** verwendbar sein,
- sie muss **spezielle Anweisungen** zur **Interrupt**- und **Alarm**behandlung aufweisen,
- die **Synchronisation** von verschiedenen Tasks sollte durch Sprachelemente erfolgen, die spezielle Betriebssystemmechanismen unterstützen,
- entsprechende Sprachelemente für die unterschiedlichen **Ein-/Ausgaben** zu den einzelnen Komponenten und den direkten Hardwarezugriff sollten vorhanden sein,
- sie sollte die Spezifikation und Synchronisation von **parallelen Abläufen** ermöglichen,
- die **Fehlererkennung** und -**behandlung** muss gut programmierbar sein.

1 Anforderungen an Prozessprogrammiersprachen (3)

- Aus der **Sicht der Programmierer** sollten noch folgende grundlegende Eigenschaften berücksichtigt sein:
  - **leicht** erlernbar,
  - **schnell** zu formulieren, **standardisiert** und **rechnerunabhängig**,
  - das **Einfügen** von **Assembler**- und Maschinencode in den Quelltext sollte möglich sein,
  - der **byte- bzw. bitweise Zugriff** auf Daten sollte möglich sein,
  - für die jeweiligen **Anwendungen** sollten entsprechende **symbolische** Benennungen enthalten sein,
  - passende **Editoren** für die verwendeten Programmiersprachen sollten zur Verfügung stehen

2 Software - Lebenszyklus



- **Unabhängig** von der **Problemstellung** und der verwendeten **Programmiersprache**.

- Je nach **Bedeutung** des anzufertigenden Programms werden einige Phasen

  - noch **untergliedert** bzw.

  - **mehrmals durchlaufen** wegen durchzuführender Verifikationen und Leistungsüberprüfungen

- **Problemanalyse**:

  - Das zu lösende Problem wird möglichst **vollständig** beschrieben.

  - Berücksichtigt werden muss dabei auch die **Programmumgebung**:
    - Hardware,
    - Betriebssystem,
    - Schnittstellen.

  - Das Ergebnis dieser Phase ist:
    - die **vollständige Anforderungsdefinition** (Lastenheft) des Bestellers und zum anderen
    - das vom Anbieter erstellte **Pflichtenheft**, das seine Aufgaben beschreibt.
    - Diese werden u.U. auch durch **Programmablaufpläne** (o.ä.) strukturiert dargestellt.

◆ **Entwurfsphase**:

- ➤ Das Programm wird in **überschaubare Einheiten** (Module) zerlegt,
- ➤ Die benötigten **Schnittstellen** werden beschrieben.
- ➤ Das Ergebnis dieser Phase ist die **Spezifikation**.
- ➤ Diese sollte unbedingt bezüglich der festgelegten Anforderungsdefinitionen auf ihre **Korrektheit** überprüft werden, da logische Fehler in der **späteren** Programmversion nur noch **schwer zu beheben** sind.

◆ **Implementierungsphase**:

- ➤ Führt zu einem **fertigen**, dokumentierten **Programm**.
- ➤ Alle **Module** sind eingebunden und beschrieben
- ➤ **Module** sollten mit Hilfe einer geeigneten **Testumgebung** auf ihre richtige Implementierung **überprüft** werden.

◆ **Funktionsüberprüfung:**

- ➤ Durch **Tests** wird das Gesamtsystem auf sein **korrektes** Arbeiten hin überprüft.
- ➤ Mit entsprechenden **Leistungsmessungen** wird das **Zeit**- und **Durchsatzverhalten** getestet.

◆ **Installation:**

- ➤ **Übertragen** des Programmsystems auf seine **reale Umgebung,** d.h. auf die entsprechende Hardware bzw. in die gewünschte Softwareumgebung.

◆ **Wartung:**

- ➤ Da im laufenden Betrieb **unerwartete** Programm- und Datenzustände sowie **Veränderungen** in der **Hardwarekonfiguration** entstehen können, muss nach der Implementierung für die Wartung der Programme gesorgt werden.

- Die hier aufgezeichnete **Top-Down-Methode** für den Programmentwurf wird s**elten so streng sequentiell** durchgezogen. Meist ergeben sich während der einzelnen Phasen noch **Rückgriffe** auf vergangene Abschnitte. Dabei erleichtert das Aufteilen des Gesamtproblems in einzelne Programmteile (Module) spätere Veränderungen in vorangegangen Phasen.

3 PEARL - Kurzbeschreibung

- PEARL verfügt über **Sprachmittel** zur Formulierung von
  - Algorithmen,
  - Ein-/Ausgabe,
  - Echtzeitprogrammierung,
  - Programmstrukturierung.

- Die **Strukturierung** erfolgt durch das Aufteilen eines Programms in einzelne **Module**:
  - Abgeschlossene Programmteile,
  - Können getrennt übersetzt und getestet werden.
  - Jedes dieser Module besteht aus einem Systemteil und einem Problemteil.

◆ **Systemteil und Problemteil:**

- ➤ Der **Systemteil** ist für die **Beschreibung** der **Hardwarestruktur** zuständig.
- ➤ Den benötigten Hardwareeinrichtungen und Signalen werden hier entsprechende **Namen** zugeordnet.
- ➤ Diese so vergebenen **symbolischen Namen** werden **im Problemteil**, dem eigentlichen Programm, **verwendet**.
- ➤ bei **Änderungen** der **Hardwarekonfiguration** wird **nur** der **Systemteil** verändert, der Problemteil hingegen bleibt unverändert.
- ➤ PEARL-Programme lassen sich daher **leicht** auf andere Rechner **übertragen**.

◆ **Ablaufsteuerung**

- ➤ ACTIVATE: Starten eines Tasks
- ➤ TERMINATE: Beenden eines Tasks
- ➤ SUSPEND: Versetzen eines Tasks in den Wartezustand
- ➤ CONTINUE: Fortsetzen eines Tasks

◆ **Alarm- und Zeitbedingungen:**

- ➤ AFTER, AT, WHEN
- ➤ Datentypen DURATION, CLOCK:

```
DECLARE Zahl FLOAT, Nummer FIXED;
DECLARE (Summe, Differenz) FLOAT;
DECLARE Mittagszeit CLOCK, Arbeitsdauer DURATION;
DECLARE Prozesszustaende BIT(16),
Meldungstext CHAR(40);
```

Beispiel: Mögliche Vereinbarungen im Deklarationsteil eines Programms

◆ Anweisungen zur Beschreibung zeitlicher Abläufe:

```
AFTER 10 SEC ALL 5 SEC DURING 70 MIN ACTIVATE Schütz PRIORITY 7;
```

➤ Diese Anweisung sorgt dafür, dass ein Task mit dem Namen "Schütz" nach 10 Sekunden für einen Zeitraum von 70 Minuten alle 5 Sekunden mit der Priorität 7 in den Zustand "bereit" versetzt wird.

```
AT 12:00:00 All 60 MIN UNTIL 24:00:00 ACTIVATE Protokoll;
```

➤ Der Task "Protokoll" wird zwischen 12 Uhr und 24 Uhr jede Stunde gestartet.

G.2 Programmiersprachen für die Prozessautomatisierung

G.2 Programmiersprachen für die Prozessautomatisierung