Rechnerkommunikation

- Steuerung der Eisenbahn mit 2 Rechnern
- Fehlertoleranz durch hot-standby
  (Heiße Redundanz)
- Aktiver/passiver Rechner
- Ausfall simulieren durch Ctrl-C (Booten geht nicht)
- Nachrichtenaustausch über serielle Schnittstelle 4
  zur gegenseitigen Überwachung
- Nachrichtenaustausch zur Übermittlung des
  Systemzustandes (aktiv/passiv)
  - bei Rechnerinitialisierung erforderlich
  - sonst rechnet passiver Rechner mit und schreibt
    nicht auf das Interface
- Paket `communication` entwickeln
- Zeitüberwachtes Senden und Empfangen von
  Nachrichten
- mögliche Fehler
  - Verbindung kann nicht aufgenommen werden
    (`COMMUNICATION_ERROR`)
  - erwartete Nachricht kann nicht empfangen
    werden (`RECEIVE_TIMEOUT`)
  - gesendete Nachricht wird nicht bestätigt
    (`SEND_ERROR`)

Beispielspezifikation

```
  COMMUNICATION_ERROR: exception;
  RECEIVE_TIMEOUT:     exception;
  SEND_ERROR:          exception;

OPEN_COMMUNICATION (TIME: in DURATION);
  -- Eröffnen eines bidirektionalen
  -- Kommunikationsweges durch einmaligen
  -- Nachrichtenaustausch
  -- nach TIME ⇒ COMMUNICATION_ERROR

RECEIVE_MESSAGE (
  MESSAGE: out STRING;
  LENGTH:  out NATURAL;
  TIME:    in  DURATION);
  -- blockierendes Empfangen einer
  -- Zeichenkette, das nach TIME
  -- abgebrochen wird
  -- ⇒ RECEIVE_TIMEOUT
  -- andernfalls Bestaetigung senden

SEND_MESSAGE (MESSAGE: STRING);
  -- nicht blockierendes Senden einer
  -- Zeichenkette
  -- erfolgt keine Bestaetigung nach
  -- ACKNOWLEDGE_TIMEOUT: ⇒ SEND_ERROR
```

- vorgegeben: Paket **`serial_io`**

```
with MACHINE_TYPES; use MACHINE_TYPES;
with TEXT_IO;

package SERIAL_IO is
-- Es sind 4 Terminalports zum lesen
-- (get) und schreiben (put) nutzbar
subtype Term_T is BYTE range 1..4;

-- Konstanten fuer die Ausgabe von
-- Gleitkommazahlen
DEFAULT_FORE: constant Text_Io.Field := 2;
DEFAULT_AFT : constant Text_Io.Field := 2;
DEFAULT_EXP : constant Text_Io.Field := 3;

-- Alle Prozeduren verhalten sich wie die
-- entsprechenden Standard-ADA-Prozeduren
-- aus Text_IO. Der Name endet stets auf
-- S (fuer Serial).
-- Wichtig ist die Angabe des anzuspre-
-- chenden Terminalports (1 bis 4) als
-- ersten Parameter. Werte ausserhalb des
-- zulaessigen Bereiches fuehren zu einer
-- Exception (CONSTRAINT_ERROR).
procedure PutS(T:Term_T; X:Byte);
procedure PutS(T:Term_T; X:Character);
procedure PutS(T:Term_T; X:String);
procedure PutS(T:Term_T; N:Integer;
   Width:in Text_Io.Field:=Integer'Width);
```

```
procedure PutS(T:Term_T; F:Float;
   Fore:in Text_Io.Field:=DEFAULT_FORE;
   Aft :in Text_Io.Field:=DEFAULT_AFT;
   Exp :in Text_Io.Field:=DEFAULT_EXP);

procedure GetS(T:Term_T; X:out Byte);
procedure GetS(T:Term_T; X:out Character);
procedure GetS(T:Term_T; X:out Integer);
procedure GetS(T:Term_T; X : out Float);

procedure Get_ByteS(T:Term_T; X:out Byte;
   Got_Byte : out BOOLEAN);
-- gibt FALSE zurueck, falls kein Byte zum
-- Lesen bereit stand

procedure Put_LineS(T:Term_T; X:String);
procedure Get_LineS(T:Term_T;
   X:out String; Last:out NATURAL);
-- Last bezeichnet die Position des
-- letzten Zeichens in X
procedure New_LineS(T:Term_T;
   X : POSITIVE := 1);

end SERIAL_IO;
```

- Schnittstelle 4 benutzen
- GetS und PutS verwenden (nicht GetLineS)

Ablauf Kommunikation

- Vorschlag
  - handshaking-Protokoll implementieren
    - zwei Prozesse – senden und empfangen
    - jeweils zeitüberwacht
  - zyklisch „Lebenszeichen“ austauschen (s. unten)

- Initialisierung
  - Eröffnen der Kommunikation (zeitüberwacht)
  - kommt Verbindung zustande, dann Einigung aktiver/passiver Rechner
  - kommt keine Verbindung zustande, dann „ich bin allein“

- Einigung aktiver/passiver Rechner (über timeouts)
  - Eingabe eines Parameters im Dialog beim Starten des Programms
  - Fehlereingaben (beide aktiv, beide passiv) sollen erkannt werden. Einigung über Zeitschranken.

- „ich bin allein“ (und damit aktiver Rechner)
  - zyklisch „Eröffnen der Kommunikation“ versuchen (alle 5 Sekunden)
  - der ggf. hinzukommende Rechner ist dann passiv

- „ich bin aktiver Rechner“
  - der passive Rechner fällt aus
    - „ich bin allein“
    - laut Aufgabenstellung keine Konsequenzen
  - der andere Rechner glaubt ebenfalls, er sei der aktive
    - Einigung aktiv/passiv s. oben
    - wegen der Kommunikationsstruktur wird einer der Rechner die Situation zuerst bemerken

- „ich bin passiver Rechner“
  - der aktive Rechner fällt aus
    - „ich bin allein“
    - Übernahme der Steuerungsaufgaben
  - der andere Rechner glaubt ebenfalls, er sei der passive
    - Einigung aktiv/passiv s. oben

Kommunikationsstruktur

Variante1

- synchrone Kommunikation
- der passive Rechner testet die Reaktionsfähigkeit des aktiven

```
ACTIVE_LOOP:
  loop
    begin
      RECEIVE_CHECK (CHECK_TIMEOUT);
      -- kein delay !!!
    exception
      when RECEIVE_TIMEOUT
        => SINGLE := TRUE;
        -- Ausfall passiver Rechner
        exit ACTIVE_LOOP;
    end;
  end loop ACTIVE_LOOP;
```

```
PASSIVE_LOOP:
 loop
  begin
   SEND_CHECK;
   delay TEST_AGAIN;
  exception
   when SEND_ERROR
    => SINGLE := TRUE;
    -- Ausfall aktiver Rechner
    exit PASSIVE_LOOP;
  end;
 end loop PASSIVE_LOOP;
```

- Abschätzung der Zeitbedingungen:

  CHECK_TIMEOUT(1.0) > TEST_AGAIN(0.5) >> ACKNOWLEDGE_TIMEOUT(0.1)

  – Werte hängen von Systemlast ab, wesentlich kürzere Zeiten sind möglich

  – Übertragungszeit einer Nachricht (10 Zeichen) max. 0.02 s

  – Task muß ggf. priorisiert werden

  – wichtigste Information (aktiver Rechner ausgefallen) muß am schnellsten erkannt werden

  – Optimierung von TEST_AGAIN hinsichtlich

    - Systemauslastung
    - Unfallwahrscheinlichkeit

Variante2

- asynchrone Kommunikation
- passiver Rechner überwacht die Funktion des aktiven

```
ACTIVE_LOOP:
  loop
   begin
     SEND_ALIVE;
     delay BEAT_TIME;
   exception
     when SEND_ERROR
       => SINGLE := TRUE;
       -- Ausfall passiver Rechner wird
       -- durch ausbleibende Bestaetigung
       -- bemerkt
       exit ACTIVE_LOOP;
   end;
  end loop ACTIVE_LOOP;
```

```
PASSIVE_LOOP:
 loop
  begin
   RECEIVE_ALIVE (ALIVE_TIMEOUT)
   -- kein delay
  exception
   when RECEIVE_TIMEOUT
    => SINGLE := TRUE;
    -- Ausfall aktiver Rechner
    exit PASSIVE_LOOP;
  end;
 end loop PASSIVE_LOOP;
```

- Abschätzung der Zeitbedingungen:

  ALIVE_TIMEOUT(0.6) > BEAT_TIME(0.5)

- Fazit
  - synchron (Variante 1)
    - harte Zeitbedingungen für aktiven Rechner
  - asynchron (Variante 2)
    - einfache Zeitbedingungen
    - Möglichkeit des prozeduralen Aufrufs bei „Leerlauf“ im Programm