

Übungen zur Vorlesung Einführung in die Informatik III

Aufgabe 39 **Prozesswechsel auf der MI**

- (a) Schreiben Sie einen Prozesskontrollblock für einen Benutzerprozess (siehe MI-Handbuch, S. 17).

```

-- Konstante fuer Prozessliste
EQU      PID      = 92
EQU      PRIO     = 100
EQU      ZUST     = 104
EQU      RKNR     = 108
EQU      WURS     = 112
EQU      NEXT     = 124

RES 3000
prozsyskeller: RES 0
RES 3000
prozkeller:   RES 0
prozpcb:      DD W prozsyskeller  -- Systemstack
               DD W prozkeller    -- Prozessesstack
               DD W (0)*14        -- Register R0 bis R13
               DD W prozkeller    -- R14 / SP
               DD W startadrproz  -- R15 /PC
               DD W H'03C00000'   -- PSW (siehe MI-handbuch, S. 17)
               DD W (0)*4        -- ohne Bedeutung
                                   -- (direkte Adressierung verwendet)

-- Zusatzinformation
DD W 100      -- Prozess-ID (PID)
DD W 100      -- Benutzterkennzeichen
DD W 100      -- Priorität
DD W 1        -- Arbeitszustand:
               -- 0: beendet, 1: rechenbereit,
               -- 2: rechnend, 3: wartend
DD W -1       -- zugeordneter Rechnerkern
DD W 0        -- Ursache für den Wartezustand
DD W 0        -- Zeiger für Verkettung in der
               -- Warteschlange
DD W 0        -- Zeiger auf Vorgänger in der
               -- Prozessliste
```

```
DD W 0                                -- Zeiger auf Nachfolger in der
                                      -- Prozessliste
```

- (b)** Damit Systemdienst-Aufrufe des Benutzerprozesses erledigt werden können, benötigen Sie einen Systemkontrollblock und einen CHMK-Anfangsbehandlung. Diese soll wie der gesamte Systemkern (mit Unterbrechungsbehandlung und Rechnerkernvergabe) von den Rechnerkernen nur exklusiv benutzt werden.

Schreiben Sie Systemkontrollblock und CHMK-Anfangsbehandlung.

Systemkontrollblock (siehe MI-Handbuch, S. 15/16):

```
anfscha:      DD W (simulationsende)*6
              DD W sysdienste          -- Systemdienste
              DD W (simulationsende)*12

-- Variablen im System
plist:        DD      W proz0pcb          -- Zeiger auf 1. Element in der
                                      -- Prozessliste
SysFrei:      DD      B 1                -- fuer exklusiven Systemkern
NrRkFrei:     DD      B 1                -- fuer Zugriff auf NrRk/ImSystem
NrRkImSystem: DD      W -1               -- zunaechst kein RK im System
NrRk:         DD      W -1               -- zunaechst kein RK im System

-- nicht naeher definierte UBH
simulationsende: HALT
```

CHMK-Anfangsbehandlung:

```
sysdienste:
uab:          LIPL I 30                  -- Unterbrechungssperre setzen
                                      -- (siehe MI-Handbuch, S. 31):
                                      -- nur katastrophale Fehler mit
                                      -- Priorität IPL= 31 kommen durch)

-- exklusives Betreten des Systemkerns:
JBCCI I 7, SysFrei, systembelegt

systemfrei:   -- aktuelle RK-Nummer unter gegens. Ausschl. eintragen
JBCCI I 7, NrRkFrei, systemfrei
SPRKMAP NrRkImSystem
SH I -16, NrRkImSystem, NrRkImSystem
MOVE B I 1, NrRkFrei

CALL verlangteAktion
ADD W I 4, SP                            -- Keller zurücksetzen
JUMP rechnerkernvergabe

systembelegt: JBCCI I 7, NrRkFrei, uab
              -- aktuelle RK-Nummer vergleichen mit RkNrImSystem
SPRKMAP NrRk
SH I -16, NrRk, NrRk
```

```

CMP W NrRk, NrRkImSystem
JEQ wiederbetreten
MOVE B I 1, NrRkFrei
JUMP uab

```

wiederbetreten: -- Unterbrechung in Unterbrechungsbehandlung

```

MOVE B I 1, NrRkFrei
CALL verlangteAktion
ADD W I 4, SP          -- Keller zurücksetzen
REI                    -- Rückkehr zu unterbrochener
                        -- Systemaktivität

```

- (c) Die Rechnerkernvergabe bestimme in R0 mit Hilfe der Prozessliste die Anfangsadresse des Prozesskontrollblocks jenes Benutzerprozesses, der als nächster einen Rechenkern erhält. Schreiben Sie eine MI-Befehlsfolge für den Start bzw. die Fortsetzung des ausgewählten Benutzerprozesses.

rechnerkernvergabe:

```

SPPCB                    -- aktuellen Prozess sichern
                        -- (siehe MI-Handbuch, S. 30)

SPCBADR R0
-- setze aktuellen Prozess rechenbereit, falls er rechnend war
-- sonst wähle den ersten rechenbereiten Prozess mit höchster
-- Priorität aus
-- (gefundene Prozessadresse steht anschliessend in R0)

MOVE W I 2, ZUST+!R0     -- setze gefundenen Prozess rechnend
SPRKMAP R1
SH I -16 R1, R1          -- aktuelle RK-Nummer in R1
MOVE W R1, RKNR+!R0      -- RK-Nummer bei Prozess in PCB eintragen
LPCBADR R0               -- ausgewählten Prozess starten/fortsetzen
LPCB                     -- (siehe MI-Handbuch, S. 30/31)

```

```

warte:  JBCCI I 7, NrRkFrei, warte
        MOVE W I -1, NrRkImSystem
        MOVE B I 1, NrRkFrei
        MOVE B I 1, SysFrei
        REI

```

- (d) Implementieren Sie den Systemdienst 0, aufgerufen mit CHMK I 0, der die Beendigung einer Prozessen auslöst.

verlangteAktion:

```

PUSHR                    -- Register sichern
MOVEA 64+!SP, R12        -- Systemdienstnummer in R12
MOVE W SP, R13

-- Systemdienst 0 (Prozessende)

```

```

        CMP W I 0, !R12
        JEQ prozessende

        -- andere Systemdienste (ohne Wirkung)
        JUMP endesysdienste

prozessende: -- aktuellen Prozess beenden
        SPPCBADR R0
        MOVE W I -1 RKNR+!R0      -- aktuellem Prozess kein RK zugeordnet
        MOVE W I 0 PRIO+!R0      -- Priorität auf 0 setzen
        MOVE W I 0 ZUST+!R0      -- Prozess auf beendet setzen.

endesysdienste:
        MOVE W R13, SP
        POPR                      -- Register wiederherstellen
        RET                      -- (zu CALL verlangteAktion)

```

Aufgabe 40 Natürlichzahlige Semaphore auf der MI

(a) **sema** nat $n := N$ wird simuliert durch:

```

var nat  $n := N$ ;
sema bool mutex := true;
sema bool access := ( $N > 0$ );

```

P(n) wird simuliert durch

```

P(access);
P(mutex);
 $n := n - 1$ ;
if  $n > 0$  then V(access) fi ;
V(mutex);

```

V(n) wird simuliert durch

```

P(mutex);
 $n := n + 1$ ;
V(access);
V(mutex);

```

(b) **sema** nat $n := N$ wird simuliert durch:

```

n:          DD W N                -- var nat  $n := N$ ;

mutex_acc:  DD B 3                -- sema bool mutex := true;
                                   -- sema bool access := ( $N > 0$ );

```

P(n) wird simuliert durch

```

P_access:   JBCCI I 7, mutex_acc, P_access  -- P(access);
P_mutex:    JBCCI I 6, mutex_acc, P_mutex   -- P(mutex);
            SUB W I 1, n                    --  $n := n - 1$ ;

```

```

        JEQ continue                -- if n>0 then V(access) fi;
        OR B I 1, mutex_acc
continue: OR B I 2, mutex_acc        -- V(mutex);

```

V(n) wird simuliert durch

```

P_mutex2: JBCCI I 6, mutex_acc, P_mutex2 -- P(mutex);
          ADD W I 1, n                    -- n := n+1;
          OR B I 1, mutex_acc            -- V(access);
continue: OR B I 2, mutex_acc            -- V(mutex);

```

(c)

```

sema:     DD W I 0
semafrei: DD B I 1                    -- für den exklusiven Zugriff auf sema

p:
p_loop:   JBCCI I 7, semafrei, p_loop   -- P(semafrei);
          CMP W sema, I 0                -- if sema != 0
          JNE p_decr                    -- goto p_decr
          MOVE B I 1, semafrei           -- V(semafrei);
          MOVEA sema, -!SP                -- Parameter sema im Keller ablegen
          CHMK I 10                      -- in Warteschlange einreihen
          ADD W I 4, SP                  -- Keller bereinigen nach dem Aufwachen
          JUMP p_loop                   -- neuer Versuch nach dem Aufwachen
p_decr:   SUB W I 1, sema                -- sema := sema - 1;
          MOVE B I 1, semafrei           -- V(semafrei);
p_ende:

v:         JBCCI I 7, semafrei, v_loop   -- P(semafrei);
          ADD W I 1, sema                -- sema := sema + 1;
          CMP W sema, I 1                -- if sema = 1;
          JEQ v_aufwecken                -- then goto v_aufw
          MOVE B I 1, semafrei           -- else V(semafrei);
          JUMP v_ende
v_aufw:    MOVE B I 1, semafrei           -- V(semafrei):
          MOVEA sema, -!SP                -- Parameter sema im Keller ablegen
          CHMK I 11                      -- Prozesse aufwecken
          ADD W I 4, SP                  -- Keller bereinigen;
v_ende:

```

Aufgabe 41 MI Gerätetreiber — Terminal E/A von Zeichenketten

```

SEG
IMP  inout
EQUAL estr = H'203FFFA18'
EQUAL edr  = H'203FFFA1A'
EQUAL sstr = H'203FFFA1C'
EQUAL sdr  = H'203FFFA1E'
MOVE  W I H'10000', SP
CALL  inout

```

```

        HALT

getchar: SEG                                -- liest ein Zeichen von der Tastatur
                                           -- und "ubergibt es dem Aufrufer

        PUSHR
        MOVEA 64+!SP,R12
        MOVE  W SP, R13
        MOVE  H I H'010D',estr           -- Steuerregister bereit setzen (9600 baud)
busywait:
        ANDNOT H I H'FF7F',estr,R0       --Auf Eingabe warten
        JEQ   busywait
        MOVE  H edr,!R12                  --Eingabe an den Aufrufer weitergeben
        CLEAR H estr                      --Steuerregister loeschen
        MOVE  W R13,SP
        POPR
        RET

putchar: SEG                                --gibt das vom Aufrufer erhaltene Zeichen
                                           -- auf dem Bildschirm aus

        PUSHR
        MOVEA 64+!SP,R12
        MOVE  W SP, R13
        MOVE  H I H'010D', sstr          -- Steuerregister bereit setzen (9600 baud)
        MOVE  H !R12,sdr                  -- Zeichen ins Datenregister legen

busywait:
        ANDNOT H I H'FF7F',sstr,R0       -- auf Abholung warten
        JEQ   busywait
        CLEAR H sstr                      --Steuerregister loeschen
        MOVE  W R13,SP
        POPR
        RET

inout:  SEG                                -- leitet einen durch linefeed beendeten,
                                           -- aber hoechstens 76 Zeichen langen String
                                           -- von der Tastatur zum Bilderscreen weiter

        IMP   getchar,putchar
        PUSHR
        MOVEA 64+!SP,R12
        MOVE  W SP,R13
        MOVE  W I 76,R0                  -- R0 dient als Zeichenzaehler
        CLEAR H -!SP                     -- Platz fuer Eingabezeichen

loop:
        CALL  getchar
        CMP   H I H'A',!SP               -- linefeed beendet die Ein/Ausgabe
        JEQ   fertig
        CALL  putchar
        SUB   W I 1,R0                    -- Zeichenzaehler weiterschalten
        JLE   fertig                      -- nach 76 Zeichen ist Schluss
        JUMP  loop

```

```
fertig:
    MOVE    H I H'A', !SP      -- Ausgabe eines linefeed
    CALL    putchar
    MOVE    W R13, SP
    POPR
    RET

    END
```

```
----- < START SIMULATOR > -----
```

```
=> go
go### Kanal 3 Empfaenger (Tastatur) : zu uebertragender String ?
hallo
```

```
### Kanal 3 Sender (Bildschirm)
hallo
```

```
+++ instruction trap #1 HALT ( 00 )
=> go
go### Kanal 3 Empfaenger (Tastatur) : zu uebertragender String ?
768690789
```

```
### Kanal 3 Sender (Bildschirm)
768690789
```

```
+++ instruction trap #1 HALT ( 00 )
+++ user break
=> quit
quit
Number of simulated machine instructions : 970
```

```
----- < END SIMULATOR > -----
```

Aufgabe 42 MI Gerätetreiber — Terminal E/A von Zahlen

```
SEG
EQUAL estr = H'203FFA18'
EQUAL edr  = H'203FFA1A'
EQUAL sstr = H'203FFA1C'
EQUAL sdr  = H'203FFA1E'
EQUAL linefeed = I H'100'
EQUAL notadigit = I H'200'
MOVE  W I H'10000', SP
IMP   main
JUMP  main
```

```

getdigit: SEG                                -- liest eine Ziffer oder linefeed von
        PUSHR                                -- der Tastatur und gibt die Ziffer (als
        MOVEA 64+!SP,R12                     -- Wert,nicht als code) an den Aufrufer.
        MOVE W SP,R13                       -- Bei linefeed wird H'100' uebergeben
        MOVE H I H'010D',estr               -- Steuerregister bereit setzen

busywait:
        ANDNOT H I H'FF7F',estr,R0          -- auf Eingabe warten
        JEQ busywait
        MOVE H edr,R1                       -- Eingabe in R1 zwischenspeichern
        CLEAR H estr                        -- Steuerregister loeschen
        CMP H I H'A',R1                    -- linefeed ?
        JNE test
        MOVE H linefeed,!R12
        JUMP done

test:
        CMP H I H'30',R1                   -- Vergleich mit Code für '0'
        JGT nonval
        CMP H I H'39',R1                   -- Vergleich mit Code für '9'
        JLT nonval
        SUB H I H'30',R1,!R12              -- Eingabecode umrechnen und Ergebnis
        JUMP done                          -- an Aufrufer uebergeben

nonval:
        MOVE H notadigit, !R12

done:
        SH I -16,!R12,!R12                -- Halbwort verschieben
        MOVE W R13,SP
        POPR
        RET

putchar: SEG                                -- wie in Aufgabe 1
        PUSHR
        MOVEA 64+!SP,R12
        MOVE W SP,R13
        MOVE H I H'010D',sstr
        MOVE H !R12,sdr

busywait:
        ANDNOT H I H'FF7F',sstr,R0
        JEQ busywait
        CLEAR H sstr
        MOVE W R13,SP
        POPR
        RET

readnum: SEG                                -- liest einen String von der Tastatur
        IMP error                           -- und wandelt ihn in eine natuerliche Zahl

```



```

IMP    getdigit                -- um. Zeichen, die keine Ziffern oder
                                -- linefeed sind, werden dabei ignoriert.

PUSHR
MOVEA  64+!SP,R12
MOVE   W SP, R13
SBPSW  I H'20'                -- setzt b26 des PSW
                                -- Bei Ueberlauf wird das Ergebnis auf -1
                                -- gesetzt und die Proz. "error" aufgerufen

CLEAR  W -!SP
CLEAR  W R0

loop:
CALL   getdigit
CMP    W linefeed,!SP
JEQ    fertig
CMP    W notadigit,!SP
JEQ    loop
MULT   W I 10,R0              -- naechste Stelle => *10
JV     fehler
ADD    W !SP,R0
JV     fehler
JUMP   loop

fehler:                                -- leere Zeichenpuffer
CALL   getdigit
CMP    W linefeed,!SP
JNE    fehler
MOVE   W I -1,R0
CALL   error

fertig:
MOVE   W R0,!R12
MOVE   W R13,SP
POPR
RET

error: SEG                        -- gibt eine Fehlermeldung am Bildschirm aus
IMP    putchar
JUMP   start

text:  DD 'OVERFLOW-ERROR!', H'A',0
length: DD W 15

start:
PUSHR
MOVEA  64+!SP,R12
MOVE   W SP,R13
CLEAR  H R2
MOVEA  text, R1
CLEAR  H -!SP

```

```

loop:
    MOVE    B !R1+,R2
    JEQ     end
    MOVE    H R2,!SP
    CALL    putchar
    JUMP    loop

```

```

end:
    -- CLEAR H !SP+
    MOVE    W R13,SP
    POPR
    RET

```

```

main: SEG
    IMP     readnum
    CLEAR   W -!SP
    CALL    readnum
    MOVE    W !SP+,R0
    HALT
    END

```

----- < START SIMULATOR > -----

```

=> go
go### Kanal 3 Empfaenger (Tastatur) : zu uebertragender String ?
350

+++ instruction trap #1 HALT ( 00 )
=> R0/wd
R0/wd<wd>
R0      :      350
      OK
=> go 0
go 0### Kanal 3 Empfaenger (Tastatur) : zu uebertragender String ?
9999999999999999

### Kanal 3 Sender (Bildschirm)
OVERFLOW-ERROR!

```

```

+++ instruction trap #1 HALT ( 00 )
=> R0/wd
R0/wd<wd>
R0      :      -1
      OK
+++ user break
=> quit
quit
Number of simulated machine instructions : 1169

```

----- < END SIMULATOR > -----