Handling interrupts

```
--  This chapter contains routines for implementing a stand-alone
--  terminal handler.
--

PROTOCOL SIGNAL
  CASE
    signal
:

--
--  hardware constants
--

VAL INT keyboard.interface IS ... : -- address of keyboard channel
VAL INT vdu.interface      IS ... : -- address of screen channel

--
--  screen output handler
--

PROC screen.handler(CHAN OF BYTE outgoing, CHAN OF SIGNAL error)

  CHAN OF BYTE screen.out :
  PLACE screen.out AT vdu.interface :

  VAL BYTE bell.character IS '*#07' :
  WHILE TRUE
    PRI ALT
      error ? CASE signal
        screen.out ! bell.character
      BYTE char :
      outgoing ? char
        screen.out ! char
:
```

```
--
--  keyboard input handler
--

VAL INT type.ahead IS 100 :     -- number of characters that can be
                                -- typed ahead of being wanted

PROC keyboard.handler(CHAN OF SIGNAL request,
                      CHAN OF BYTE reply,
                      CHAN OF SIGNAL error   )

  -- Characters typed at the keyboard can be read from reply. A
  -- signal is required on request before each item is read. If
  -- more than type.ahead are typed ahead, an error is signalled.

  CHAN OF BYTE keystroke.in :
  PLACE keystroke.in AT keyboard.interface :

  INT reader, writer, count :
  SEQ
    reader, writer, count := 0, 0, type.ahead

    [type.ahead]BYTE datum :
    WHILE TRUE
      ALT

        BYTE lost :
        count = 0 & keystroke.in ? lost
          error ! signal

        count > 0 & keystroke.in ? datum[writer]
          count, writer := count - 1, (writer + 1) \ type.ahead

        count < type.ahead & request ? CASE signal
          SEQ
            reply ! datum[reader]
            count, reader := count + 1, (reader + 1) \ type.ahead
:
```

```
--
--  handler for echoing of input to output
--

VAL BYTE release IS 0(BYTE) :

PROC echo.handler(CHAN OF SIGNAL request,
                  CHAN OF BYTE reply, echo, inward )
  VAL BYTE enter IS '*c' :
  WHILE TRUE
    BYTE char :
    SEQ
      request ! signal
      reply ? char
      inward ! char    -- Transmit character to user
      IF
        ('*s' <= char) AND (char <= '~')
          echo ! char       -- Echo visible input to screen
        char = enter
          SEQ
            echo ! '*c'     -- Echo carriage return ...
            echo ! '*n'     -- ... and newline
            echo ! release  -- Release screen at end of line
        TRUE
          SKIP              -- No action on other characters
:

--
--  multiplexer for screen output
--

PROC output.multiplexer([]CHAN OF BYTE from, CHAN OF BYTE outgoing )
  WHILE TRUE
    ALT selected.process = 0 FOR SIZE from
      BYTE char :
      from[selected.process] ? char
        WHILE char <> release
          SEQ
            outgoing ! char
            from[selected.process] ? char
:

--
--  template for user process
--

PROC user(CHAN OF BYTE terminal.keyboard, terminal.screen)
  ... omitted code
:
```

```
--
--  structure of the program
--

CHAN OF BYTE reply, outgoing, from.keyboard :

CHAN OF SIGNAL request, error :

VAL INT from.echo.handler IS 0 :
VAL INT from.user         IS 1 :
VAL INT number.of.outputs IS 2 :

[number.of.outputs]CHAN OF BYTE to.screen :

PRI PAR
  --
  --  high priority processes
  --
  PAR
    keyboard.handler(request, reply, error)
    echo.handler(request, reply,
                 to.screen[from.echo.handler],
                 from.keyboard                )
    screen.handler(outgoing, error)
  --
  --  low priority processes
  --
  PAR
    output.multiplexer(to.screen, outgoing)
    user(from.keyboard, to.screen[from.user])
```