Terminal interrupt management

```
PROC keyboard.handler(CHAN request, sink, error)
  -- Characters typed at the keyboard can be read from sink.
  -- A signal is required on request before each item is read.

PROC echo.handler(CHAN request, reply, echo, inward)
  -- Read characters from a buffer by request/reply, passing
  -- them on to the inward channel with a copy to echo

PROC output.multiplexer(CHAN from[],VALUE width,CHAN outgoing)
  -- Copy characters from each of from[0 FOR width] to outgoing
  -- switching between inputs on receipt of a release value

PROC screen.handler(CHAN outgoing, error)
  -- Forwards characters from channel outgoing to the screen
  -- and rings a bell in case of a signal on the error channel

PROC user(CHAN terminal.keyboard, terminal.screen)
  -- Template for the user's program

PRI PAR

  PAR   -- High priority process
    --

  PAR   -- Low priority process
    --
```

```
DEF type.ahead = ...,  control = NOT ((NOT 0) << 5),  release = -1 :


PROC keyboard.handler(CHAN request, sink, error) =
  -- Characters typed at the keyboard can be read from sink.
  -- A signal is required on request before each item is read.
  -- If more than type.ahead are typed-ahead, an error is signalled.
  CHAN keystroke.in AT ... :
  VAR reader, writer, count :
  SEQ
    reader  := 0           -- index of next item to be read from buffer
    writer  := 0           -- index of next free location in buffer
    count   := type.ahead  -- number of spare locations in buffer
    VAR datum[type.ahead] :
    WHILE TRUE
      ALT
        count = 0 &                      -- if no room
                 keystroke.in ? ANY     -- but something is typed
          error ! ANY                    --     then signal an error
        count > 0 &                      -- if there is room and
                 keystroke.in ? datum[writer]  -- something is typed
          SEQ                                  --   then store it
            writer  := (writer + 1) \ type.ahead
            count   := count - 1
        count < type.ahead &           -- if something in the buffer
                 request ? ANY          -- and something is requested
          SEQ                           --   then read from the buffer
            sink ! datum[reader]
            reader  := (reader + 1) \ type.ahead
            count   := count + 1                   :



PROC echo.handler(CHAN request, reply, echo, inward) =
  -- Read characters from a buffer by request/reply, passing
  -- them on to the inward channel with a copy to echo
  DEF enter = control /\ 'M' :
  WHILE TRUE
    VAR ch :
    SEQ
      request ! ANY
      reply ? ch
      inward ! ch                       -- transmit character to user
      IF
        ('*s' <= ch)  AND  (ch <= ' ') -- visible input
          echo ! ch                     --  sent to terminal screen
        ch = enter                      -- at the end of a line
          echo ! release                --  release the screen
        TRUE
          SKIP                  :
```

```
PROC output.multiplexer(CHAN from[], VALUE width, CHAN outgoing) =
  -- Copy characters from each of from[0 FOR width] to outgoing
  -- switching between inputs on receipt of a release value
  WHILE TRUE
  VAR ch :
  ALT selected.process = [0 FOR width]
    from[selected.process] ? ch  -- take a message from any of from[]
      WHILE ch <> release        -- and copy it to completion
        SEQ
          outgoing ! ch
          from[selected.process] ? ch                   :



PROC screen.handler(CHAN outgoing, error) =
  -- Forwards characters from channel outgoing to the screen
  -- and rings a bell in case of a signal on the error channel
  DEF bell.character = control /\ 'G' :
  CHAN screen.out AT ... :
  WHILE TRUE
    VAR ch :
    PRI ALT
      error ? ANY                -- signal errors by ringing the bell
        screen.out ! bell.character
      outgoing ? ch              -- and send on outgoing characters
        screen.out ! ch                   :




PROC user(CHAN terminal.keyboard, terminal.screen) =
  ... Template for the user's program



DEF  from.echo.handler = 0. from.user = 1, number.of.outputs = 2 :
CHAN outgoing, from.keyboard, to.screen[number.of.outputs] :

PRI PAR

  CHAN request, reply, error :       -- High priority process
  PAR
    keyboard.handler(request, reply, error)
    echo.handler(request, reply,
                  to.screen[from.echo.handler], from.keyboard)
    screen.handler(outgoing, error)

  PAR                               -- Low priority process
    output.multiplexer(to.screen, number.of.outputs, outgoing)
    user(from.keyboard, to.screen[from.user])
```