SOFTWARE GUIDELINES -- Assorted and sketchy

crs/24.10.90 -

This note lists out some broad guidelines for developing real-time software on systems for which we have not planned to buy any operating system. In particular, this refers to the station computer (80186-based) and monitor/control module (8031-based) for which we have to build a miniature operating system using the Intel Development set available with us.

Interrupt Handling:

In GMRT software, we will assume that the maximum number of interrupts recognised by the system is 8. There will be a status word **int_flag** -- one byte long, being the low order byte if natural wordlength is 16 for the processor -- which will be nonzero whenever an interrupt occurs in the system. In this word, each interrupt type will be associated with a unique bit which is set by the interrupt-service routine and cleared by a higher level associated with processing the interrupt. In addition, there will be an 8-byte array **int_count** with a byte corresponding to each interrupt. Whenever an interrupt occurs, the corresponding byte in **int_count** is incremented by the interrupt-service routine and decremented by the higher-level routine. The counter **int_count** thus signifies the number of unprocessed interrupts of this kind existing at any given time.

The interrupt handling routines consist of 3 layers. The lowest layer is an interrupt-service routine **intSer** triggered by the hardware interrupt. This routine sets the associated bit in **int_flag**, and increments the corresponding counter in the array **int_count**. In addition, it will perform critical operations associated with the interrupt. For instance, for the HDLC, this will be checking the status and, if there is an error, marking a repeat-request. In the case of interrupt from a serial line, the received character should be transferred to the appropriate location of a buffer. If an absolute clock is being maintained, the internal timer is incremented by the appropriate interval between the clock-ticks. A skeleton of Fortran-equivalent interrupt service routine is given below (Usually, this routine will be in Assembler or perhaps C, never in Fortran !):

```
subroutine intSer
implicit integer * (a-h)
parameter (int_num = 6)     /*  0,1,...8
parameter (int_val = 64)  /* should be 2^int_num
parameter (mid_enable = .true.)
common/intCMN/int_flag,int_fmid,int_count(0:7)
int_flag = OR(int_val,int_flag)
int_count(int_num) = int_count(int_num) + 1
if(mid_enable)int_fmid = OR(int_val,int_fmid)
return
end
```

In general, **intSer** should have as minimal functions as possible, typically less than 20 instructions long. Any function beyond this should be entrusted to a second-level routine **intMidl** which should perform the less important but system-level functions associated with the interrupt. This level is not necessarily present for all interrupts. When it is present, it will be part of the kernel rather than the application task running under the supervisor kernel. For instance, for a packet coming through HDLC, this will check if it is meant to be processed by the kernel or the application task. The application task often may not get control until kernel functions are completed.

The kernel is expected to perform intMid corresponding to all pending interrupts as part of normal house-keeping activities. A typical kernel (if written in Fortran!) could look like the following:

```
      program kernel
      implicit integer * (a-z)
      parameter (int_lev=5)   /* max no. of interrupts handled
      dimension int_order(int_lev)
      common/intCMN/int_flag,int_fmid,int_count(0:7)
      common/childCMN/proc_id,proc_key, addr_st,addr_end
      data int_val/1, 2, 3, 4, 5/
      data int_order/2, 4, 1, 8, 0 /    /* int. priority !
      call iniKer
      proc_key = 0   /* for use by application process
                     /* appln process sets key_proc > 0 to
                     /* indicate it isn't finished;
                     /* < 0 in case of abnormal error
      proc_id = 0    /* no child process yet
      proc_load = 0   /* no process to be loaded yet

      while (.true.)
      call intWait     /* sleep until some interrupt occurs

      call sigProc(intval, proc_id,proc_load)
           /* proc_load = 1 if new process has to be loaded

      if(proc_load .eq. 1)call loadProc(proc_id,proc_key,err)
           /* proc_key =  0   ==> no process loaded
           /*            -1   ==> loaded for execution later
           /*            >0   ==> process-execution is required

      if(err .ne. 0)call errHand(err)
      if(proc_key .gt. 0)then
           call childProc(proc_key)
           if(proc_key .eq. 0)call clearProc
           if(proc_key .lt. 0)call errInform(proc_id,proc_key)
           /* e.g. inform master of abnormal termination
           end if
      end While
      stop
      end


      subroutine iniKer
      implicit integer * (a-z)
      common/intCMN/int_flag,int_fmid,int_count(0:7)
      int_flag = 0
      int_fmid = 0
      do k = 0,7
           int_count(k) = 0
           end do
      call iniKer2
      return
      end

      subroutine iniKer2

      ...
           reset timers, start and end address of application
           routine to be invoked;
           read Table of application process names, start-
           and end-addresses of their executable code as
           stored in EPROM;
           check the health of various subsystems
      ...
      return
      end
```

The third level of interrupt handling is done by a routine **intAppl** which is part of the application task running under the supervisor kernel. This can be as elaborate as necessary, and the application task can be typically assumed to choose its own order of priority of interrupts.