Take Command of your SWIs

Jonathan Hunt describes a special module that lets you issue SWI calls from the command line

Most programmers at one time or another will have wished there was a way to call SWI routines more easily, perhaps to test out an effect or to get information from the operating system. It is very easy to call SWIs from within programs, and even from within Basic (using the Basic keyword SYS), but unfortunately RISC OS doesn't provide a method of calling a SWI routine from the command line. The program presented here rectifies this.

WHAT THE PROGRAM DOES

The Basic program given in listing 1 creates and saves a Relocatable Module called CLI_SWI in the currently selected directory. When loaded, this Module adds two new star commands to the OS: *SWI and *SYS. Both commands do the same job (and call the same piece of code) so it doesn't matter which one you use, but Basic programmers will find it more natural to use *SYS while ARM code programmers will find *SWI preferable.

USING CLISWI

Type in the listing carefully and save it as a Basic program before running it (you could call the program MakeCLISWI if you want to give it a meaningful name). Once you have run the program to create the module, double-clicking on the module itself will install it (or you can include a line in your boot file to do this).

The commands work in a similar way to the Basic SYS keyword as far as parameter passing is concerned. The *SYS or *SWI command is followed by either the name of the SWI, enclosed in quotes, or the SWI number itself. This in turn is followed by the parameters to be passed to the SWI, separated by commas. Numbers may be passed in hexadecimal if they are preceded by a & character. If you want to pass a string as a parameter, you must enclose it in quotes. As with the Basic SYS keyword, if you pass a string as one of the parameters, it is stored in a buffer and the pointer to this buffer is passed instead. Values returned by the SWI are stored in the system variables R$0 to R$7.

As an example of how to use *SWI and *SYS, here's how to call the same SWI routine using all three methods:

Basic

```
SYS "OS_File",17,"MyFile"
```

Assembly Language

```
MOV R0,#17
ADR R1,filename
SWI "OS_File"
-
.filename
EQUS "MyFile"+CHR$(0)
```

Using CLI SWI

```
*SWI "OS_File",17,"MyFile"
```

or

```
*SYS "OS_File",17,"MyFile"
```

HOW THE MODULE WORKS

When either of the commands is issued, the first thing the module has to do is make a copy of the parameter string so it can be processed. As it is copied, commas are replaced by ASCII 1 and quotes by ASCII 2. This allows them to be used as end of string terminators while still allowing them to be distinguished from each other. The first parameter, the SWI identifier, is then processed. If this is a number it is converted from the parameter string using OS_ReadUnsigned. This SWI has the added bonus of translating hexadecimal numbers if the & character is present. If the identifier is a name, it is converted into a number by OS_SWINumberFromString.

Once the SWI number is known, it is turned into the actual SWI machine code instruction by adding &FF000000. Then, to make sure it doesn't produce any errors when called, it is turned into the X form by adding &20000. The instruction is then

stored just before the end of the code, where it will be executed later on.

The next job to be done is to load registers R0-R7 with the passed parameters. They are not loaded directly because some of the registers need to be used as counters in the processing loop, so they are held in a block of memory for later transfer. The parameters are processed in order. If a parameter is a number it is stored as it stands. If it is a string then the pointer to that string is stored. After all the parameters have been processed and stored in memory, they are loaded into R0-R7.

The last job to be done is to scan the copied parameter string, and replace any instances of character 1 or 2 with character 0. This is done in case any fussy SWI routines need zero as an end of string terminator.

Now everything is set up, the SWI instruction can be executed. If any errors occur while executing the SWI, the V flag will be set, and when the module returns control to the OS, the error will be handled automatically. If no errors occur, the returned registers are stored back in memory and are then copied one by one into the system variables R$0 to R$7 before the module returns control to the OS.

```
 10 DIM module% 1024
 20 PROCassem
 30 SYS "OS_File",10,"CLI_SWI",&FFA,,m
odule%,module%+P%
 40 END
 50:
 60 DEFPROCassem
 70  FOR pass%=4 TO 6 STEP 2
 80   P%=0
 90   O%=module%
100   [OPT pass%
110   EQUD 0
120   EQUD 0
130   EQUD 0
140   EQUD 0
150   EQUD title
160   EQUD help
170   EQUD commands
180 .title
190   EQUS "CLI_SWI"+CHR$(0)
200   ALIGN
210 .help
220   EQUS "CLI SWI"+CHR$(9)+CHR$(9)+"
1.00 ("+MID$(TIME$,5,11)+")"+CHR$(0)
230   ALIGN
240 .commands
250   EQUS "SWI"+CHR$(0)
260   ALIGN
270   EQUD swi
280   EQUB 1
290   EQUB 0
300   EQUB 1
310   EQUB 0
320   EQUD swisyntax
330   EQUD swihelp
340   EQUS "SYS"+CHR$(0)
350   ALIGN
360   EQUD swi
370   EQUB 1
380   EQUB 0
390   EQUB 1
400   EQUB 0
410   EQUD syssyntax
420   EQUD syshelp
430   EQUD 0
440 .swi
450   STMFD R13!,{R7-R11,R14}
460   MOV R2,#0
470   ADR R1,buffer
480 .transferloop
490   LDRB R3,[R0,R2]
500   MOV R4,R3
510   CMP R3,#44
520   MOVEQ R4,#1
530   CMP R3,#34
540   MOVEQ R4,#2
550   STRB R4,[R1,R2]
560   ADD R2,R2,#1
570   CMP R2,#256
580   BGE endtransfer
590   CMP R3,#32
600   BGE transferloop
610 .endtransfer
620   MOV R4,#0
630   SUB R2,R2,#1
640   STRB R4,[R1,R2]
650   LDRB R3,[R1,#0]
660   CMP R3,#2
670   BNE number
680 .name
690   ADD R1,R1,#1
700   MOV R9,R1
710   SWI "XOS_SWINumberFromString"
720   LDMVSFD R13!,{R7-R11,PC}
730   MOV R10,R0
740   B loadregs
```

```
750 .number
760   MOV R9,R1
770   MOV R0,#0
780   SWI "XOS_ReadUnsigned"
790   LDMVSFD R13!,{R7-R11,PC}
800   MOV R10,R2
810 .loadregs
820   LDR R11,swiaddr
830   AND R11,R11,#&FF000000
840   ORR R11,R11,R10
850   ORR R11,R11,#&20000
860   STR R11,swiaddr
870   MOV R8,#0
880   MOV R11,#0
890   ADR R12,regs
900 .loadloop
910   LDRB R3,[R9,R8]
920   ADD R8,R8,#1
930   CMP R8,#256
940   BGE endload
950   CMP R3,#2
960   BGE loadloop
970   CMP R3,#0
980   BEQ endload
990   LDRB R3,[R9,R8]
1000  CMP R3,#2
1010  BNE regnumber
1020 .regstring
1030  ADD R7,R9,R8
1040  ADD R7,R7,#1
1050  STR R7,[R12,R11]
1060  ADD R11,R11,#4
1070  CMP R11,#32
1080  BLT loadloop
1090  B endload
1100 .regnumber
1110  MOV R0,#0
1120  ADD R1,R9,R8
1130  SWI "XOS_ReadUnsigned"
1140  LDMVSFD R13!,{R7-R11,PC}
1150  STR R2,[R12,R11]
1160  ADD R11,R11,#4
1170  CMP R11,#32
1180  BLT loadloop
1190 .endload
1200  MOV R2,#0
1210 .charloop
1220  LDRB R3,[R9,R2]
1230  MOV R4,R3
1240  CMP R3,#2
1250  MOVLE R4,#0
1260  STRB R4,[R9,R2]
1270  ADD R2,R2,#1
1280  CMP R2,#256
1290  BGE endchar
1300  CMP R3,#0
1310  BGT charloop
1320 .endchar
1330  LDMIA R12,{R0-R7}
1340 .swiaddr
1350  SWI 0
1360  LDMVSFD R13!,{R7-R11,PC}
1370  ADR R12,regs
1380  STMIA R12,{R0-R7}
1390  ADR R0,varname
1400  ADR R1,buffer
1410  MOV R2,#4
1420  MOV R3,#0
1430  MOV R4,#1
1440  ADD R5,R0,#2
1450  MOV R6,#48
1460  MOV R7,#0
1470 .returnloop
1480  STRB R6,[R5]
1490  ADD R6,R6,#1
1500  LDR R8,[R12,R7]
1510  STR R8,[R1]
1520  SWI "XOS_SetVarVal"
1530  ADD R7,R7,#4
1540  CMP R7,#32
1550  BNE returnloop
1560  LDMFD R13!,{R7-R11,PC}
1570 .swihelp
1580  EQUS "*SWI calls a SWI routine."
+CHR$(13)+"Syntax: *SWI "+CHR$(34)+"<SWI
name>"+CHR$(34)+"|<SWI number>,r0,r1,r2
..."+CHR$(0)
1590 .swisyntax
```

Important Announcement

You may have noticed that in this issue the name of RISC Developments has been dropped from the magazine. Unfortunately RISC Developments Ltd was forced to cease trading on 8th August 1994 as a result of losses associated with the production of Beebug magazine during the last year of its life.

RISC User magazine and the software and hardware products previously produced by RISC Developments will now be part of BEEBUG Ltd. This will not affect the magazine in any way, and