Avevamo visto l’ultima volta il gestore delle SWI, proseguiamo parlando del lato utente.

Vogliamo che la nostra applicazione scriva qualche cosa. Dovremo quindi aver preparato anticipatamente un buffer in memoria che contenga la nostra stringa da stampare (es. “ciao”). Dopo aver salvato la stringa nel buffer applicativo dobbiamo chiedere al OS di scriverla. Bisogna quindi scrivere innanzitutto l’indirizzo della cella in un registro e passarlo come parametro nella system call. Prendiamo l’indirizzo della cella iniziale e lo mettiamo nel registro 0. Possiamo poi avere due versioni: una che passa le dimensioni della stringa e un'altra che stampa tutto fino a che non trova il terminatore di stringa ‘\0’. Ipotizziamo di usare questo secondo metodo (ma in realtà useremo il primo). Per chiedere al sistema operativo di effettuare la stampa, dopo aver messo l’indirizzo in R0, bisogna chiamare la SWI con un codice particolare (per esempio write = 2): dunque SWI + 2 come bit di motivazione provoca la stampa della stringa di caratteri presenti nel buffer indirizzato da R0. Questa SWI dunque funzionerà in maniera simile a una chiamata di funzione.

La differenza rispetto a una funzione normale è che in questo caso non è ammesso fare pasticci, infatti l’handler è avviato in modalità supervisore.

Proviamo a scrivere questo handler della SWI:

Swi\_h: 336: STM SP!, R4, R5, R6, R7, R8, R9, R10, LP //la maggior parte delle volte non ci serviranno tutti ma dobbiamo considerare il caso peggiore

BIC LP, LP, #2113929216 //il BIC corrisponde al BIT CLEAR = AND NOT, ci serve per essere sicuri di mantenere soltanto l’indirizzo dell’istruzione (puliamo la parte di “registro di stato”): sappiamo infatti che gli ultimi due bit saranno a 00, ma non abbiamo nulla di garantito per i primi 6. La costante equivale a 111111000…000.

LDR R8, [LP, #-8] //questo ci salva in R8 il codice operativo della nostra SWI

BIC R8, R8, #2139095040 //questo ci lascia in R8 soltanto i 24 bit magici di motivazione

/\*A questo punto si definisce una convenzione sul significato dei 24 bit\*/

CMP R8, #0

BEQ exit //se in R8 c’è 0 si fa il branch a una cella contenente il codice exit

CMP R8, #1

MOVEQ R10, #buffers //a 1 associamo il significato di lettura da standard input

BEQ read\_stdin //si definisce convenzionalmente che read\_stdin vuole in R0 l’indirizzo del buffer applicativo, in R1 la lunghezza di tale buffer (= n° effettivo di caratteri letti)

CMP R8, #2

BEQ write\_stdout

…

B error //in caso il numero di motivazione non sia presente tra le opzioni viene restituito un errore

Scriviamo dunque la funzione write\_stdout.

write\_stdout: MOV R6, #0 //carico 0 in R6 (si ricorda che i registri dal 4 al 10 sono già stati salvati)

LDR R9, [R10, #+16] //R10 è l’indirizzo del vettore di caratteristiche dei buffers, R10+16 è l’indirizzo del buffer di uscita

LDR R8 [R10, #+20] //mettiamo in R8 le dimensioni del buffer di uscita

LDR R7 [R10, #+24] //R7 = indice di inserzione dell’output buffer

write\_again: CMP R6, R1 //facciamo finta che abbiamo passato, tramite R1, il numero di caratteri da dover stampare

LDMDBGE SP!, R4, R5, R6, R7, R8, R9, R10, PC^ //se superiamo il numero di caratteri da stampare, allora facciamo il ritorno

ADD R4, R7, #1 //prendiamo l’indice di inserzione aumentato di 1

CMP R4, R8

MOVGE R4, #0 //se superiamo la dimensione del buffer di uscita riportiamo l’indice di inserzione a 0

LDRB R5, [R9, +R4] //prendiamo il valore che dovremmo rimpiazzare nel buffer di uscita

CMP R5, #0 //se questo valore è ancora 0 allora c’è ancora spazio nel buffer

MOVNE R1, R6 //copio il numero di caratteri che sono stati scritti e rimpiazzo il numero di caratteri che mi era stato richiesto di scrivere

LDMDBNE SP!, R4, R5, R6, R7, R8, R9, R10, PC^

LDRB R5, [R0, +R6] //carico in R5, visto che ora non mi serve più lo 0, il carattere (B = byte) da inserire nel buffer di uscita

CMP R5, #0 //se R5 è 0 allora è un terminatore di stringa e non ha senso copiarlo e continuare

MOVEQ R1, R6

LDMDBEQ SP!, R4, R5, R6, R7, R8, R9, R10, PC^

STRB R5, [R9, +R7] //carico il carattere nel buffer di output

CMP R6, #0

BLEQ print\_char //avvio la stampa dei caratteri

ADD R6, R6, #1

MOV R7, R4 //incremento R7 di 1

STR R7, [R10, #+24] //aggiorno l’indice di inserzione

B write\_again

Vediamo adesso la read\_stdin (che è chiamata quando la SWI è passata con motivazione 1). Usiamo il registro 6 per indicare il numero di caratteri letti e passiamo R0 e R1 come parametri del buffer applicativo

read\_stdin: MOV R6, #0

LDR R9, [R10] //inseriamo in R9 l’indirizzo del buffer di ingresso

LDR R8, [R10, #+4] //in R8 teniamo la dimensione del buffer

LDR R7, [R10, #+12] //in R7 c’è l’indice di estrazione

read\_again: CMP R6, R1 //se i caratteri letti sono più dei caratteri da leggere bisogna fermarsi

LDMBGE SP!, R4, R5, R6, R7, R8, R9, R10, PC^

LDRB R5, [R9, +R7] //prendo il carattere dal buffer di sistema/ di ingresso

CMP R5, #0 //se è il terminatore di stringa posso fermarmi

MOVEQ R2, R6

LDMBEQ SP!, R4, R5, R6, R7, R8, R9, R10, PC^

STRB R5, [R0, +R6] //altrimenti si copia il carattere nel buffer applicativo

MOV R5, #0

STRB R5, [R9, +R7] //sostituisco il valore appena copiato con 0 nel buffer di ingresso, per indicare che è stato letto

ADD R7, R7, #1 //aggiorno l’indice di estrazione

CMP R7, R8

MOVGE R7, #0

STR R7, [R10, #+12] //salvo l’indice di estrazione nel descrittore del buffer

ADD R6, R6, #1

B read\_again