Directory di lavoro

`bochs` 1905 `compile` 1905 `linker.ld` 1907 `makeit` 1905 `mount` 1905 `umount` 1905

Prima di iniziare gli esperimenti, si predispone una directory di lavoro, da utilizzare in qualità di utente comune. Nella directory si copia il file '`floppy.img`' e si mettono alcuni script molto semplici:

Listato u164.1. '`./mount`'

```
#!/bin/sh
chmod a+rw floppy.img
su root -c "mount -o loop,uid=1001 -t vfat floppy.img /mnt/fd0"
```

Listato u164.2. '`./umount`'

```
#!/bin/sh
su root -c "umount /mnt/fd0"
```

Listato u164.3. '`./bochs`'

```
#!/bin/sh
bochs -q 'boot:a' 'floppya: 1_44=floppy.img, status=inserted' 'megs:32'
```

Il senso di questi script è evidente e il loro scopo è solo quello di ridurre al minimo l'impegno di digitazione. In questa directory viene poi predisposto anche lo script '**`compile`**', ma viene descritto nella sezione successiva.

Directory «05/»

A partire dalla directory di lavoro si crea la sottodirectory '`05/`', nella quale viene poi messo il codice del sistema che si va a creare. Ma per evitare di fare confusione con i file-make, si predispone uno script per la compilazione che li crea al volo, in base ai contenuti effettivi delle sottodirectory.

Listato u164.4. '`./05/makeit`'

```
#!/bin/sh
#
# makeit...
#
OPTION="$1"
#
edition () {
    local EDITION="include/kernel/build.h"
    echo -n                                 > $EDITION
    echo -n "#define BUILD_DATE \""     >> $EDITION
    echo -n 'date "+%Y%m%d%H%M%S"'      >> $EDITION
    echo  "\""  >> $EDITION
}
#
#
#
makefile () {
    #
    local MAKEFILE="Makefile"
    local TAB=" "
    #
    local SOURCE_C=""
    local C=""
    local SOURCE_S=""
    local S=""
    #
    local c
    local s
    #
    # Trova i file in C.
    #
    for c in *.c
    do
        if [ -f $c ]
        then
        C='basename $c .c'
            SOURCE_C="$SOURCE_C $C"
        fi
    done
    #
    # Trova i file in ASM.
    #
    for s in *.s
    do
        if [ -f $s ]
        then
        S='basename $s .s'
            SOURCE_S="$SOURCE_S $S"
        fi
```

```
    done
    #
    # Prepara il file make.
    #
    echo -n                                                 > $MAKEFILE
    echo "# Questo file è stato prodotto automaticamente"  >> $MAKEFILE
    echo "# dallo script \"makeit\", sulla base dei"       >> $MAKEFILE
    echo "# contenuti della directory."                    >> $MAKEFILE
    echo "#"                                               >> $MAKEFILE
    echo "c = $SOURCE_C"                                   >> $MAKEFILE
    echo "#"                                               >> $MAKEFILE
    echo "s = $SOURCE_S"                                   >> $MAKEFILE
    echo "#"                                               >> $MAKEFILE
    echo "all: \$(s) \$(c)"                                >> $MAKEFILE
    echo "#"                                               >> $MAKEFILE
    echo "clean:"                                          >> $MAKEFILE
    echo "${TAB}@rm *.o 2> /dev/null ; pwd"                >> $MAKEFILE
    echo "#"                                               >> $MAKEFILE
    echo "\$(s):"                                          >> $MAKEFILE
    echo "${TAB}@echo \$@.s"                               >> $MAKEFILE
    echo "${TAB}@as -o \$@.o \$@.s"                        >> $MAKEFILE
    echo "#"                                               >> $MAKEFILE
    echo "\$(c):"                                          >> $MAKEFILE
    echo "${TAB}@echo \$@.c"                               >> $MAKEFILE
    echo "${TAB}@gcc -Wall -Werror -o \$@.o -c \$@.c" \
         "-nostdinc -nostdlib -nostartfiles -nodefaultlibs" \
        "-I../include -I../../include -I../../../include"  >> $MAKEFILE
    #
}
#
#
#
main () {
    #
    local CURDIR=‘pwd‘
    local OBJECTS
    local d
    local c
    local s
    local o
    #
    edition
    #
    for d in ‘find .‘
    do
        if [ -d "$d" ]
        then
            #
            # Ci sono sorgenti in C o in ASM?
            #
            c=‘echo $d/*.c | sed "s/ .*//"‘
            s=‘echo $d/*.s | sed "s/ .*//"‘
            #
            if [ -f "$c" ] || [ -f "$s" ]
            then
                CURDIR=‘pwd‘
                cd $d
                makefile
                #
                if [ "$OPTION" = "clean" ]
                then
                    make clean
                else
                    if ! make
                    then
                        cd "$CURDIR"
                        exit
                    fi
                fi
                cd "$CURDIR"
            fi
        fi
    done
    #
    cd "$CURDIR"
    #
    #
    #
    if [ "$OPTION" = "clean" ]
    then
        true
    else
        OBJECTS=""
        #
        for o in ‘find . -name \*.o -print‘
        do
            if    [ "$o" = "./kernel/kernel_boot.o" ] \
               || [ "$o" = "./kernel/kernel_main.o" ] \
               || [ ! -e "$o" ]
            then
                true
            else
                OBJECTS="$OBJECTS $o"
            fi
        done
        #
        echo "Link"
        #
        ld --script=linker.ld -o kernel_image \
```



```
                kernel/kernel_boot.o \
                $OBJECTS \
                kernel/kernel_main.o
        #
        cp -f kernel_image /mnt/fd0/kernel
        sync
    fi
}
#
# Start.
#
if [ -d include ] && [ -d kernel ] && [ -d lib ]
then
    main
else
    echo "Mi trovo in una posizione sbagliata e non posso svolgere" \
         "il mio compito"
fi
```

Va osservato che la variabile ‘**TAB**’ deve contenere esattamente una tabulazione orizzontale (di norma il codice 09_{16}. Pertanto, se si riproduce il file o se lo si scarica, occorre verificare che il contenuto sia effettivamente una tabulazione, altrimenti va corretto. Se la variabile ‘**TAB**’ contiene solo spazi, i file-make che si ottengono non sono validi.

```
local TAB=" "
```

In pratica, attraverso questo script, i file-make che si generano hanno un aspetto simile a quello del listato seguente:

```
c =  elenco_file_c_senza_estensione
#
s =  elenco_file_asm_senza_estensione
#
all: $(s) $(c)
#
clean:
        @rm *.o 2> /dev/null ; pwd
#
$(s):
        @echo $@.s
        @as -o $@.o $@.s
#
$(c):
        @echo $@.c
        @gcc -Wall -Werror -o $@.o -c $@.c ↩
↪               -nostdinc -nostdlib -nostartfiles ↩
↪               -nodefaultlibs -I../include ↩
↪               -I../../include -I../../../include
```

Il «collegamento» (*link*) dei file avviene attraverso un comando contenuto nello script ‘**makeit**’, dove si fa in modo di mettere all’inizio il file-oggetto che è responsabile dell’avvio, dal momento che contiene l’impronta di riconoscimento per il sistema di avvio aderente alle specifiche *multiboot*.

Nella directory di lavoro descritta nella sezione precedente, conviene mettere uno script che richiami a sua volta ‘**makeit**’ e che provveda a copiare il file del kernel nel file-immagine del dischetto:

Listato u164.6. ‘`./compile`’

```
#!/bin/sh
cd 05
./makeit clean
./makeit
cd ..
```

Script di collegamento

Sempre all’interno della directory ‘`05/`’ va predisposto lo script usato da GNU LD per eseguire correttamente il collegamento dei file oggetto in un file eseguibile unico. Dal momento che nel progetto che si intraprende si intende usare la memoria linearmente, si intende che il blocco minimo sia della dimensione di un registro, ovvero pari a 4 byte:

Listato u164.7. ‘`./05/linker.ld`’

```
/*****************************************************************************
 * La memoria viene usata in modo lineare, senza controlli dei
 * privilegi, così non si usano nemmeno gli allineamenti tradizionali
 * di 4096 byte, ma solo di 4 byte, ovvero di un registro.
 *****************************************************************************/

ENTRY (kernel_boot)
SECTIONS {
    . = 0x00100000;
    k_mem_total_s = .;
    .text : {
        k_mem_text_s = .;
        *(.text)
        . = ALIGN (0x4);
        k_mem_text_e = .;
    }
    .rodata : {
        k_mem_rodata_s = .;
        *(.rodata)
        . = ALIGN (0x4);
        k_mem_rodata_e = .;
    }
    .data : {
        k_mem_data_s = .;
        *(.data)
        . = ALIGN (0x4);
        k_mem_data_e = .;
    }
    .bss : {
        k_mem_bss_s = .;
        *(.bss)
        *(COMMON)
        . = ALIGN (0x4);
        k_mem_bss_e = .;
    }
    k_mem_total_e = .;
}
```

Il codice contenuto nel file del kernel che si va a produrre, deve iniziare a partire da 00100000_{16}, ovvero da 1 Mibyte, come prescrive il sistema di avvio *multiboot*, il quale va a collocarlo in memoria, a partire da quella posizione. Inoltre, per consentire di individuare i blocchi di memoria utilizzati, vengono inseriti dei simboli; per esempio, ‘**`k_mem_total_s`**’ individua l’inizio del kernel, mentre ‘**`k_mem_total_e`**’ ne individua la fine.

Si dà per scontato che GNU AS predisponga un file eseguibile in formato ELF.

Altre directory

All’interno di ‘`05/`’ si creano ancora: ‘`lib/`’, per la libreria standard e altre librerie specifiche del sistema; ‘`include/`’, per i file di intestazione della libreria; ‘`kernel/`’ con i file iniziali usati dal kernel; ‘`app/`’ per le applicazioni (ovvero le funzioni avviate dal kernel quando tutto è pronto).