Integrazione di Sistemi Embedded
Laboratorio 03

Matteo Perotti 251453
Giuseppe Puletto
Luca Romani 255244
Giuseppe Sarda 255648

November 18, 2018

1 Esercizio 1

Il programma si compone di tre files .c e tre header files. Un ciclo infinito si occupa di leggere il prossimo carattere in arrivo salvandolo nella posizione più a destra di un array di caratteri lungo quanto il numero di caratteri del comando previsto più lungo. Prima della memorizzazione del carattere i dati in ogni cella dell'array vengono spostati di una cella a sinistra.

Ogni volta che un nuovo carattere è stato memorizzato, la funzione readCommand analizza il buffer per vedere se può essere arrivato un comando valido. Questa operazione è svolta andando a controllare la cella dell'array nella quale, se fosse stato ricevuto il comando più corto (draw a point), sarebbe salvato il carattere identificativo del comando stesso. In caso di mancato riscontro viene analizzata la prima cella dell'array, ovvero quella in cui potrebbe essere presente una delle altre due lettere identificative di un comando valido.

In caso di riscontro, i parametri relativi al comando vengono salvati nella struttura apposita, convertendo ogni carattere in intero. In seguito viene fatto un controllo sui dati salvati, per vedere se sono parametri validi, ossia se ogni coordinata è un numero compreso tra 0 e 127, e se il "modo" è un numero compreso tra 0 e 2.

Se il comando non viene riconosciuto o se i parametri non sono corretti, la funzione restituisce 0 e viene letto il prossimo carattere.

Nel caso in cui il comando venga riconosciuto e possegga parametri validi, allora viene chiamata la funzione corrispondente.

1.1 main.c

Il file main.c contiene la definizione della funzione main, la funzione principale del programma. Essa si occupa di definire l'array di char cmdBuffer in cui viene memorizzato un carattere alla volta e la struttura in cui i parametri del comando vengono memorizzati. Prima del ciclo, l'array di caratteri viene inizializzato con null characters per evitare che valori casuali possano portare a riconoscimenti errati di comandi mai ricevuti.

Nel ciclo infinito viene traslato il buffer di arrivo e viene salvato il nuovo carattere nell'ultima posizione. Il buffer è quindi passato alla funzione readCommand insieme alla struttura; a seconda dell'intero ritornato da quest'ultima funzione viene controllato il comando valido ed invocata la funzione corrispondente, oppure il ciclo riprende da capo.

Nel caso in cui la funzione readCommand dovesse ritornare una lettura corretta ma non fosse stato salvato un comando valido nella struttura, la funzione main restituisce un 1 per segnalare un errore.

1.2 read.c and read.h

Nel file header read.h sono presenti i comandi #define per aumentare la leggibilità e la manutenibilità del codice, insieme con la dichiarazione del tipo "basicCmd", ossia un tipo-struttura utile per la variabile in cui saranno salvati i vari parametri del comando, e la dichiarazione delle funzioni relative alla lettura del prossimo carattere dal periferico di input, al riconoscimento del comando e al salvataggio dei relativi parametri e alla conversione di un carattere ad intero.

readChar(void) La funzione ritorna il carattere che viene letto dal periferico di input, in questo caso lo standard input. Viene effettuata una chiamata a funzione "getc()" con arcomento "stdin" (il tutto corrisponde a "getchar()").

char2int(char charIn) La funzione riceve un carattere in ingresso e restituisce uno short int che corrisponde alla cifra rappresentata in ASCII. Se il carattere non è una cifra da 0 a 9, viene restituito 10, perché è un valore che aiuta il parser a capire se i parametri dell'ipotetico comando sono validi oppure no.

readCommand(char* cmdBuffer, basicCmd* cmdStruc_pt) La funzione riceve l'array buffer in cui è salvato il possibile comando, insieme con la struttura dati in cui i parametri del comando vengono salvati.

La funzione si occupa di controllare nelle posizioni chiave se è presente un identificativo del comando. Se è presente, la struttura viene aggiornata grazie all'utilizzo di char2int(char charIn).

In seguito viene eseguito un controllo sui parametri appena aggiornati: se sono validi, allora viene ritornato un 1.

In caso il comando non sia riconosciuto come completamente valido, viene ritornato un 1.

1.3 draw.c, draw.h and shared.h

Il file draw.c contiene la definizione e la descrizione dell'interfaccia delle funzioni drawPoint, drawLine e drawEllipse e la direttiva #include al file locale draw.h. IL file draw.h contiene la dichiarazione delle funzioni drawPoint, drawLine e drawEllipse e la definizione di tre macro, tre variabili di nome DRAW_MODE_CLEAR, DRAW_MODE_SET e DRAW_MODE_XOR. draw.h ha anche una direttiva #include al file locale shared.h e una al file di sistema stdio.h. IL file shared.h contiene la definizione delle macro rowsFrame, colsFrame e wordPixels e la definizione della variabile globale frameBuffer, corrispondende allo schermo virtuale su cui disegnare punti, ellissi e linee a richiesta. frameBuffer viene poi definito nel main.c

int drawPoint(int x, int y, int m) La funzione drawPoint è quella più basilare delle tre disponibili per disegnare. Come dichiarato dal prototipo nel file header draw.h, la funzione riceve tre interi, di cui i primi due sono le coordinate x e y del frame buffer. Dato che, dal punto di vista del piano cartesiano le y rappresentano l'altezza in cui si troverà il punto e le x la distanza orizzontale dall'origine, nel momento in cui si passano le due coordinate alla funzione è necessario che queste vengano invertite per puntare alla cella corretta del frameBuffer, che è una matrice di caratteri da 128 righe e 16 colonne da 1 Byte ciascuna. Per cui la variabile y corrisponderà alla i-esima riga della matrice, mentre la variabile x dovrà essere divisa per 8 per capire a quale degli 8 bit del j-esimo carattere si vuole puntare. Bisogna tener presente che le coordinate passate a questa funzione sono state controllate da readCommand(), di conseguenza si assume che i valori rispettivamente di x e y sono compresi tra 0-15 e 0-127.
Facendo sempre riferimento ad un ipotetico piano cartesiano, se si vuole scrivere l'i-esimo pixel della matrice a partire dal basso occore complementare la coordinata y, sottraendola al numero 127. Siccome nel linguaggio C non esiste la possibilità di accedere direttamente ad uno specifico bit di un carattere è necessario effettuare operazioni "bitwise" combinando le funzioni booleane e lo shifting di bit, che in questo caso avviene da destra verso sinistra di una quantità pari al resto della divisione di x per otto (il MSB è l'ultimo a sinistra).
Se l'intero m che viene passato alla funzione, corrisponde ad una delle tre modalità di scrittura del buffer definite nell'header "shared.h" allora la funzione modificherà l'opportuno bit nel modo desiderato e ritornerà zero, altrimenti verrà ritornato un uno per segnalare la presenta di un errore.

int drawLine(unsigned int x1,unsigned int y1,unsigned int x2,unsigned int y2,int m) La funzione drawLine descrive, dati due estremi, un segmento all'interno del frameBuffer.
Riceve, come da prototipo, le coordinate dei due punti (x1,y1) e (x2,y2), ed infine la modalità di disegno. Ritorna il valore intero '0' se la scrittura all'interno del frameBuffer o '1' se le coordinate degli estremi sono superiori all dimensioni massime dei "pixel" scrivibili. L'algoritmo utilizzato per la scrittura del codice è stato ricavato da un documento reperibile al seguente indirizzo web a pagina 6: Bresenham's Algorithm.
Lo pseudo-codice si basa sull'algoritmo di Bresenham implementato con numeri interi. Tuttavia nel pdf viene considerato il punto $P_1(x_1,y_1)$ sempre l'estremo di partenza, dando per scontato che le ordinate e coordinate siano di modulo inferiore a quelle del punto $P_2(x_2,y_2)$. È stata dunque necessaria una rivisitazione per considerare tutti gli altri casi e permettere un corretto funzionamento del codice. La funzione richiede, infine, l'utilizzo di 6 *int* e 1 *char*.

int drawEllipse(int xc, int xy, int dx, int dy, int m) La funzione drawEllipse riceve cinque variabili intere come parametri: le cordinate cartesiane del centro dell'ellisse, i diametri orizontale e verticale e il modo attraverso cui scrivere su un array di char.

drawEllipse è definita nel file draw.c, dichiarata nel file draw.h. Fa uso di tre macro xmax, ymax e wordPixels, definite le prime due in read.h, l'ultima in shared.h, tre variabili intere di cui la funzione fa ampio uso.

Essa si serve dell'algoritmo di Bresenham per disegnare sull'array di char. Il codice dell'algoritmo è stato ricavato correggendo e modificando la porzione di righe di codice trovata alla pagina web

Bresenham's circle or ellipse drawing algorithm. L'algoritmo trovato viene corretto usando in totale due variabili locali di tipo int in meno, non usando la funzione DrawPixel() ed ottimizzando l'algoritmo in modo che non scriva un pixel più di una volta. Quest'ultima correzione è la più importante: senza questa, drawEllipse non sarebbe stata in grado di disegnare secondo il draw mode xor. Altra importante correzione riguarda il controllo della posizione del pixel da scrivere. Se questa cade al di fuori dello schermo virtuale a disposizione, i bit dell'array di char, il pixel non viene disegnato.

La funzione ritorna uno zero.

2 Esercizio 2

2.1 Script

Per automatizzare il processo di compilazione e generazione del file eseguibile è stato elaborato lo script "universalDrawerScript.sh", che quando viene lanciato riceve come argomento il tipo di compilatore che si vuole usare: gcc per piattaforme x86-64, arm-none-eabi-gcc per piattaforme ARM prive di sistema operativo. Nel caso in cui il primo argomento non sia tra quelli attesi o sia assente, lo script termina immediatamente l'esecuzione.
Una volta scelta la piattaforma target lo script compila i file draw.c, read.c e main.c (con opzione "-Wall" attiva per mostrare tutti i warning), viene eseguito il linking con il compilatore scelto e tutti i file oggetto vengono spostati in una cartella generata dallo script che può avere due nomi differenti:

- x86_64_platform
- arm_platform

in entrambi i casi all'interno della cartella corrispondente si troveranno i file oggetto e un file log (universalDrawerArmLog per ARM e universalx86_64Log per x86_64) al cui interno sono contenuti i risultati dei comandi "nm" e "size" rispettivamente.
Dato che per la piattaforma ARM non c'è un sistema operativo che si interfaccia con il programma scritto (direttiva data con "arm-none-eabi-gcc"), è necessario aggiungere l'opzione "specs=nosys.specs" al compilatore per dirgli di non preoccuparsi delle funzioni di sistema dato che è compito di chi programmerà la piattaforma gestire i vari ritorni e funzionalità del programma.

Di seguito viene riportato il contenuto dei file di log per ciascuna piattaforma:

x86_64

```
text       data     bss     dec     hex filename
400           0       0     400     190 main.o
1825          0       0    1825     721 read.o
6017          0       0    6017    1781 draw.o
# size output


main.o:
U drawEllipse
U drawLine
U drawPoint
U _GLOBAL_OFFSET_TABLE_
0000000000000000 T main
U readChar
U readCommand
U __stack_chk_fail

draw.o:
00000000000016d0 T abs
0000000000000437 T drawEllipse
000000000000019a T drawLine
```

```
0000000000000000 T drawPoint
0000000000000800 C frameBuffer

read.o:
0000000000000015 T char2int
U _GLOBAL_OFFSET_TABLE_
U _IO_getc
0000000000000000 T readChar
0000000000000044 T readCommand
U stdin
#nm output
```

ARM

```
text       data     bss     dec     hex filename
396           0       0     396     18c main.o
2656          0       0    2656     a60 read.o
8096          0       0    8096    1fa0 draw.o
# size output


main.o:
00000000 t $a
U drawEllipse
U drawLine
U drawPoint
00000000 T main
U readChar
U readCommand

draw.o:
00000000 t $a
00000220 t $a
000005e0 t $a
000016b4 t $a
00001f64 T abs
0000021c t $d
000005dc t $d
000016b0 t $d
000005e0 T drawEllipse
00000220 T drawLine
00000000 T drawPoint
00000800 C frameBuffer

read.o:
00000000 t $a
00000088 t $a
00000088 T char2int
00000084 t $d
U _impure_ptr
00000000 T readChar
000000e0 T readCommand
U __srget_r
#nm output
```

Script

```
#!/bin/bash

#  compiling commands
#  This script generates a directory and a log file
#  according to the first argument provided:
#  if it is gcc, the script uses gcc compiler for x86_64 platforms
#  and creates object files in the x86_64_platform directory. The log
#  file cointains informations regarding memory allocation of the code
#  Same behaviour if first argument is arm-none-eabi-gcc
#  Exit 1 is asserted in the wrong or missing argument case


if [[ $1 = gcc ]]
then

$1 -Wall -c read.c
$1 -Wall -c draw.c
$1 -Wall -c main.c
$1 -o universalDrawer main.o read.o draw.o
chmod +x universalDrawer

mkdir x86_64_platform
mv *.o x86_64_platform
cd x86_64_platform

size main.o read.o draw.o | cat > sizeFile
printf "# size output\n\n" >> sizeFile
nm main.o draw.o read.o | cat > nmFile
cat sizeFile nmFile > universalDrawerx86_64Log
printf "#nm output" >> universalDrawerx86_64Log

rm sizeFile nmFile
cd ..

elif [[ $1 = arm-none-eabi-gcc ]]
then
$1 -Wall --specs=nosys.specs -c read.c
$1 -Wall --specs=nosys.specs -c draw.c
$1 -Wall --specs=nosys.specs -c main.c
$1 --specs=nosys.specs -o universalDrawer main.o read.o draw.o
chmod +x universalDrawer

mkdir arm_platform
mv *.o arm_platform
cd arm_platform

size main.o read.o draw.o | cat > sizeFile
printf "# size output\n\n" >> sizeFile
nm main.o draw.o read.o | cat > nmFile
cat sizeFile nmFile > universalDrawerArmLog
printf "#nm output" >> universalDrawerArmLog

rm sizeFile nmFile
cd ..

else

echo Invalid argument 1, you must insert gcc or arm-none-eabi-gcc
exit 1

fi
```