

## X-Windows in Assembly Language: Part I - THE.UNIX.WORLD

Pubblicato da mammon\_ il 01/1999

Livello **intermedio**

### Introduzione

/\* Questo articolo e' stato tratto dall'Assembly Programming Journal numero 2 e tradotto da Little-John \*/

```

::/ \:.....
:/___\:.....
/|   \:.....
:|   _/\:.....
:|   | \ \:.....
::\___\:.....
.....THE.UNIX.WORLD

```

### Iniziamo

La maniera assennata per scrivere programmi per X-Windows e' usare un toolkit come Xt o Gtk; la via facile sarebbe usare uno scripting package come Python o Tcl/Tk. I coders assembly moderni, comunque, sono rinomati per sacrificare tranquillita' e sensibilito' in nome della curiosita' e della velocita' di esecuzione; e' in questo spirito che la programmazione potenziale di X-Windows in assembly language saro' ora analizzata.

#### X-Windows Programming

Come le altre GUIs, X-Windows usa uno stile di programmazione event-driven (dipendente dagli eventi) in cui una applicazione si registra con il sistema, mostra la sua interfaccia main user, e aspetta eventi di sistema che segnalino che l'utente abbia interagito con il programma. Ci sono quattro 'livelli' principali della X-Windows Programming: XProtocol, XLib, Xt o programmazione 'toolkit', e scripting.

#### XProtocol

X-Windows consiste di un X Server che gestisce l'output grafico, l'input della tastiera e del mouse, la segnalazione degli eventi, e i comandi mandati dal programma client (Window Managers, applicazioni). I client comunicano con l'X Server usando l'XProtocol, che consiste di flussi di byte scambiati tra il client e il server --- da un certo punto di vista come i pacchetti che una rete client scambia con una rete server. XProtocol e' virtualmente di nessun valore per la programmazione delle applicazioni, [for the coding overhead for each server request makes development impractical]. I dettagli delle richieste dell'XProtocol possono essere trovati in '/usr/include/X11/Xproto.h'.

#### XLib

L'equivalente delle Win32 API in X-Windows e' XLib. Anche se qualcuno usa i toolkits per la programmazione delle applicazioni, non c'e' modo di evitare l'XLib coding. XLib e' una interfaccia tra i programmi client e l'X Server; essenzialmente, e' una libreria di funzioni XProtocol esportate per essere usate dalle applicazioni.

#### Xt

La programmazione con un Toolkit e' come usare le class libraries (come MFC, OWL, o VCL) sulla piattaforma Win32. Ci sono diversi toolkit disponibili, come Qt, Gtk, Xt Intrinsics, Athena, e il toolkit Motif. Ogni toolkit consiste di widgets estendibili (come le risorse in Win32) che definiscono tipi di oggetti di base: bottoni, scrollbars, dialogs, edit windows, ecc.

#### Scripting

Una vasta varieto' di linguaggi di scripting sono disponibili per la piattaforma Unix, e molti di questi hanno anche windowing toolkits che li abilitano a creare applicazioni X-Windows. I piu' popolari sono Tcl/Tk, Python, e Java; inutile dire che questi metodi di programmazione potrebbero non essere implementati in assembly language.

#### The XLib Programming Model

Un'applicazione scritta per l'interfaccia XLib dimostra i principi principali della programmazione per X-Windows. Questi principi si sintetizzano in un metodo in 5-passi:

##### Passo I : Connessione al Display

Il primo passo di una applicazione X-Windows e' anche il piu' semplice: una call a XOpenDisplay; il risultato ---restituito in eax naturalmente--- e' un pointer ad una struttura Display. Questo dovrebbe essere salvato, dato che ce ne saro' necessito' per quasi ogni altra call seguente:

```
p_disp = XOpenDisplay( NULL );
```

Nota: Sto fornendo il codice d'esempio in C per questa sezione; la ricostruzione in assembler saro' presentata piu' avanti.

##### Passo II : Inizializzazione delle risorse (Colori e Fonts)

Prima che una finestra sia visualizzata, essa richiede un Graphic Context (simile al DC di Win32); prima che il GC possa essere creato, esso richiede che i colori e i fonts da essere usati dalla finestra siano inizializzati.

La via piu' facile per far cio' e' usare XLoadQueryFont e le macro WhitePixel e BlackPixel:

```
mfontstruct = XLoadQueryFont( p_disp, "fixed");
WhitePix = WhitePixel( p_disp, DefaultScreen(p_disp));
BlackPix = BlackPixel( p_disp, DefaultScreen(p_disp));
```

Ancora una volta, i valori sono salvati per una utilizzazione successiva. Nota che un metodo piu' complesso per l'allocazione dei colori sarò utilizzato in seguito nell'assembly code; li', un handle alla colormap default di X Windowse' ottenuto con una call a XDefaultColormap, e XAllocNamedColor e' usato per allocare i valori del pixel bianco e nero pixel: cio' realizza lo stesso codice come sopra, ma senza usare le macro.

Passo III : Crea la Finestra

Ci sono 4 cose che devono esser fatte per creare una finestra: la stessa finestra e' registrata con l'X Server ed e' ad essa dato un Resource ID, il GC e' registrato con l'X Server ed a lui dato il suo Resource ID, la finestra deve specificare a quali eventi deve rispondere, e infine la finestra deve essere mappata nell'X display.

Per creare la finestra ci vuole una call a XCreateWindow o XCreateSimpleWindow. XCreateSimpleWindow, usato sotto, richiede il display, la parent window, le coordinate dello schermo x e y, la larghezza e l'altezza della finestra, lo spessore del bordo, il valore pixel del bordo, e il valore pixel del background. A XCreateWindow, usata nella versione assembly, e' passato il display, la parent window, x & y, larghezza & altezza, spessore del bordo, la profondito' di colore, la window class, gli attributi visuali, il valore della mask, e una struttura XSetWindowAttributes. E' restituito un handle alla finestra creata.

```
Main = XCreateSimpleWindow( \
    p_disp, DefaultRootWindow( p_disp ), 100, 100, 100, 50, 1, BlackPix, WhitePix);
```

Creare un GC non e' strettamente necessario; comunque farne a meno rende imprevedibile l'aspetto dell'applicazione (ho visto che il background della mia finestra diventa trasparente). Un GC viene creato chiamando XCreateGC, a cui e' passato il display, l'handle della finestra, il valore della mask, e una struttura GraphicsContextValues:

```
theGC = XCreateGC(p_disp, Main,(GCFont | GCForeground | GCBackground), &gcv);
```

Gli eventi input sono selezionati usando la funzione XSelectInput, a cui e' passato il display, l'handle della finestra, i valori ORati delle mask dell'evento:

```
XSelectInput( p_disp, Main, ExposureMask );
```

Infine, la finestra e' mappata sul display (e quindi visualizzata) con la call XMapWindow, che e' relativamente autoesplicativa:

```
XMapWindow( p_disp, Main );
```

A questo punto la procedura deve essere creata per ogni child window (bottoni, scrollbars, ecc); cio' che segue espone la creazione di un bottone con il suo GC, e la selezione delle mask degli eventi Exposure e ButtonPress:

```
Exit = XCreateSimpleWindow(p_disp, Main, 15, 1, 60, 15, 1, WhitePix, BlackPix);
XSelectInput(p_disp, Exit, ExposureMask | ButtonPressMask );
XMapWindow(p_disp, Exit);
exitGC = XCreateGC(p_disp, Exit,(GCFont | GCForeground | GCBackground),&gcv);
```

Nota che non sarò necessario un altro GC per ogni finestra se condivide lo stesso background, foreground, e i colori dei font.

Passo IV : Event Loop

L'event loop e' il 'corpo' del programma, dove l'applicazione risponde agli eventi dell'utente. Questo loop chiama XNextEvent per ricevere l'evento di sistema successivo, e risponde a quelli mandati alla sue finestre. Il loop seguente cattura l'evento Expose e scrive del testo in ogni finestra usando XDrawString sulla exposure iniziale di ogni finestra (xexpose.count ==0). In piu' quando e' premuto il bottone Exit, il loop while esce e l'applicazione si conclude.

```
while( !Done ){
    XNextEvent(p_disp, &theEvent);
    if( theEvent.xany.window == Main){
        if( theEvent.type == Expose && theEvent.xexpose.count == 0){
            XDrawString(p_disp, Main, theGC, 1, 40, msgtext, strlen(msgtext));
        }
    }
    if( theEvent.xany.window == Exit){
        switch(theEvent.type){
            case Expose:
                if( theEvent.xexpose.count == 0){
                    XDrawString(p_disp, Exit, exitGC, 2, 11, extext, strlen(extext) );
                }
                break;
            case ButtonPress:
                Done = 1;
        }
    }
}
```

```

    }
  }
}

```

### Passo V : Pulisci e chiudi il Display

A questo punto l'applicazione e' terminata; i vari handles devono essere liberati, le finestre distrutte, e il display chiuso. Le funzioni tipiche per far cio' sono queste sotto:

```

XFreeGC(p_disp, theGC);
XFreeGC(p_disp, exitGC);
XUnloadFont(p_disp, mfontstruct->fid);
XDestroyWindow(p_disp, Main);
XCloseDisplay(p_disp);
exit(0);

```

Nota che tutte le funzioni, le strutture, e i messaggi usati sopra sono definiti in '/usr/include/X11/Xlib.h', './X11/Xutil.h' e './X11/X.h'.

### Inline Assembler Con GCC

Per via della presenza dell'assembler GAS insieme al GCC, l'inline assembler e' alquanto chiaro. Col GCC, la keyword 'asm' e' usata per anticipare un blocco di istruzioni asm; la sintassi di 'asm' e' cosi':

```
asm( statements : output vars : input vars : modified registers);
```

Nota che gli ultimi tre parametri sono usualmente utilizzati solo se stai scrivendo una intera funzione in assembly language, o se stai modificando i registri che non salvi (e' meglio salvare tutti i registri che modificherai, se contengono valori che saranno necessari dopo).

Le linee di asm sono passate direttamente a GAS, e quindi devono essere in formato riconoscibile dal GAS. Per questa ragione, espressioni asm multilinea richiederanno una nuova linea (e, opzionalmente, un tab) dopo ogni espressione, come qui:

```
asm( "
statement1 \n
statement2 \n
statement3 \n
statement4"
: "g" (outvar)
: "g" (invar)
: eax, ebx, ecx
);
```

o, come faccio qui:

```
asm( "statement1 \n\t"
"statement2 \n\t"
"statement3 \n\t"
"statement4 \n\t");
```

Piu' che questo non ci sono altre restrizioni. Le strutture non passano bane tra C e GAS; se hai bisogno di far riferimento a variabili di strutture specifiche dall'inline assembly, e' consigliabile mettere queste variabili in variabili C temporanee, che possono essere normalmente utilizzate dal blocco assembler.

L'esempio seguente dimostra cio':

```
fid = mfontstruct->fid;
asm( "
push fid\n
push mainGC\n
push p_disp\n
call XSetFont\n
add $12, %esp");
```

Piu' informazioni sull'assembler inline GCC possono essere reperite qui:

- Avly's Programming Page (<http://www.castle.net/~avly/djasm.html>)
- CodeX Software ([http://www.gameprog.com/codex/tut/att\\_asm.html](http://www.gameprog.com/codex/tut/att_asm.html))
- Brennan's DGPP Resources (<http://brennan.home.ml.org/djgpp/>) [Down al momento]

## The XHell Sample Program

---

Per poter usare gli header files C per X-Windows, il programma seguente e' stato scritto in C per GCC, usando codice C per le dichiarazioni e l'assembler come 'corpo' del programma. Nella seconda parte di questo articolo (sulla prossima issue) convertirò questo programma nel modello Xt e lo implementerò in NASM.

```
// xhell.c =====
#include
#include
/* ===== Global Variable Declarations ===== */
char *msgtext = "You are in XHell",
*extext = "Exit XHell",
*m_font = "fixed",
*app_name = "xhello",
>window_title = "XHell",
*szWhite = "white",
*szBlack = "black";
XFontStruct *mfontstruct;
Display *p_disp;
Window Main, Exit;
GC mainGC, exitGC;
XEvent theEvent;
Font fid;
Colormap cmap;
int Done = 0;
unsigned long pxBlack, pxWhite;
XSetWindowAttributes xswa;
XColor pixBlack, pixWhite;
XGCValues gcv;
/* ===== Start della Main Function ===== */
main()
{
/* ===== Connessione al Display ===== */
asm( "push $0\n\t"
"call XOpenDisplay\n\t"
"movl %eax, p_disp\n\t"
"add $4, %esp\n\t");

/* ===== Setup Colors e Fonts ===== */
asm( "push m_font\n\t"
"push p_disp\n\t"
"call XLoadQueryFont\n\t"
"add $8, %esp\n\t"
"movl %eax, mfontstruct");
/* ===== Prepara la Main Window ===== */
fid = mfontstruct->fid;
/* ===== Crea il Main Graphics Context ===== */
// Obtain Colormap Handle
asm( "push p_disp\n\t"
"call XDefaultScreen\n\t"
"add $4, %esp\n\t"
"push %eax\n\t"
"push p_disp\n\t"
"call XDefaultColormap\n\t"
"add $8, %esp\n\t"
"movl %eax, cmap");
// Alloca i colori White and Black
asm( "push $pixWhite\n\t"
"push $pixWhite\n\t"
"push szWhite\n\t"
"push cmap\n\t"
"push p_disp\n\t"
"call XAllocNamedColor\n\t"
"add $20, %esp");
asm( "push $pixBlack\n\t"
"push $pixBlack\n\t"
"push szBlack\n\t"
"push cmap\n\t"
"push p_disp\n\t"
"call XAllocNamedColor\n\t"
"add $20, %esp");
xswa.background_pixel = pixWhite.pixel;
asm( "push $xswa\n\t"
```

```

"movl $1, %ebx\n\t"
"shl $1, %ebx\n\t" //CWBackPixel = 1 << 1
"push %ebx\n\t"
"push $0\n\t" //CopyFromParent = 0 (X.h)
"push $1\n\t" //InputOutput = 1 (X.h)
"push $0\n\t" //CopyFromParent = 0 (X.h)
"push $1\n\t"
"push $50\n\t"
"push $100\n\t"
"push $100\n\t"
"push $100\n\t"
"push p_disp\n\t"
"call XDefaultRootWindow\n\t"
"add $4, %esp\n\t"
"push %eax\n\t"
"push p_disp\n\t"
"call XCreateWindow\n\t"
"add $48, %esp\n\t"
"movl %eax, Main");
gcv.font = fid;
asm( "push $gcv\n\t"
"movl $1, %ebx\n\t"
"shl $14, %ebx\n\t" //GCFont = 1 << 14
"push %ebx\n\t"
"push Main\n\t"
"push p_disp\n\t"
"call XCreateGC\n\t"
"add $16, %esp\n\t"
"movl %eax, mainGC");
pxBlack = pixBlack.pixel;
pxWhite = pixWhite.pixel;
asm( "push fid\n\t"
"push mainGC\n\t"
"push p_disp\n\t"
"call XSetFont\n\t"
"push pxBlack\n\t"
"push mainGC\n\t"
"push p_disp\n\t"
"call XSetForeground\n\t"
"push pxWhite\n\t"
"push mainGC\n\t"
"push p_disp\n\t"
"call XSetBackground\n\t"
"add $36, %esp");
asm( "movl $1, %ebx\n\t"
"shl $15, %ebx\n\t" //ExposureMask = 1 << 15
"push %ebx\n\t"
"push Main\n\t"
"push p_disp\n\t"
"call XSelectInput\n\t"
"add $12, %esp");
asm( "push Main\n\t"
"push p_disp\n\t"
"call XMapWindow\n\t"
"add $8, %esp");
/* ===== Crea le Child Windows ===== */
asm( "push pxWhite\n\t"
"push pxBlack\n\t"
"push $1\n\t"
"push $15\n\t"
"push $60\n\t"
"push $1\n\t"
"push $15\n\t"
"push Main\n\t"
"push p_disp\n\t"
"call XCreateSimpleWindow\n\t"
"movl %eax, Exit\n\t"
"add $36, %esp");
asm( "movl $1, %ebx\n\t"
"shl $15, %ebx\n\t" //ExposureMask = 1 << 15
"movl $1, %ecx\n\t"
"shl $2, %ecx\n\t" //ButtonPressMask = 1 << 2
"or %ecx, %ebx\n\t"
"push %ebx\n\t"

```

```

"push Exit\n\t"
"push p_disp\n\t"
"call XSelectInput\n\t"
"add $12, %esp");
gcv.foreground = pxBlack;
gcv.background = pxWhite;
asm( "push $gcv\n\t"
"movl $1, %ebx\n\t"
"shl $14, %ebx\n\t" //GCFont = 1 << 14
"movl $1, %ecx\n\t"
"shl $2, %ecx\n\t" //GCForeground = 1 << 2
"or %ecx, %ebx\n\t"
"movl $1, %ecx\n\t"
"shl $3, %ecx\n\t" //GCBackground = 1 << 3
"or %ecx, %ebx\n\t"
"push %ebx\n\t"
"push Exit\n\t"
"push p_disp\n\t"
"call XCreateGC\n\t"
"add $16, %esp\n\t"
"movl %eax, exitGC");
asm( "push Exit\n\t"
"push p_disp\n\t"
"call XMapWindow\n\t"
"add $8, %esp");
/* ===== Event Loop ===== */ while( !Done ){ //Implementeto in C per salvare spazio ;)
XNextEvent(p_disp, &theEvent);
if( theEvent.xany.window == Main){
if( theEvent.type == Expose && theEvent.xexpose.count == 0){
asm( "push $16\n\t"
"push msgtext\n\t"
"push $40\n\t"
"push $1\n\t"
"push mainGC\n\t"
"push Main\n\t"
"push p_disp\n\t"
"call XDrawString\n\t"
"add $28, %esp");
}
}
if( theEvent.xany.window == Exit){
switch(theEvent.type){
case Expose:
if( theEvent.xexpose.count == 0){
XDrawString(p_disp, Exit, exitGC, 2, 11, extext, strlen(extext) );
}
break;
case ButtonPress:
Done = 1;
}
}
}
/* ===== Chiude il Display ===== */
asm( "push mainGC\n\t"
"push p_disp\n\t"
"call XFreeGC\n\t"
"add $8, %esp\n\t"
"push exitGC\n\t"
"push p_disp\n\t"
"call XFreeGC\n\t"
"add $8, %esp\n\t"
"push fid\n\t"
"push p_disp\n\t"
"call XUnloadFont\n\t"
"add $8, %esp\n\t"
"push Main\n\t"
"push p_disp\n\t"
"call XDestroyWindow\n\t"
"call XCloseDisplay\n\t"
"add $8, %esp");
}
; EOF =====

```

Come puoi vedere, creare un programma XLib in linguaggio assembly e' abbastanza 'unwieldly'. Il codice prodotto e' soprattutto manipolazione dei dati e chiamate C; non c'e' molto che l'assembly puo' offrire, neanche nell'event loop. Infatti l'unica vera ottimizzazione —apparte l'overhead aggiunto dal compiler, che nell'esempio sopra non bypassiamo— e' nell'uso di chiamate dirette invece delle macro su cui l'originario "hello world" in C si basava.

Anche se questo in se' e' alquanto un trionfo —programmando applicazioni C in assembler impari a vedere esattamente di quanto codice superfluo puoi fare a meno— non e' abbastanza. Nella prossima issue, parlero' della programmazione Xt in assembler, che usero' i widgets/risorse piu' che creare finestre dal nulla, quindi riponendo la maggior parte del codice nelle librerie di sistema esistenti e rendendo l'applicazione finale molto piu' piccola.