Chapitre XVIII : Thread avec SDL

par Loka (autres articles)

Date de publication : 08/10/2007

Dernière mise à jour : 08/10/2007

Jusqu'à présent, nous avons construit nos programmes en utilsant qu'un seul processus à la fois. Avec le multithreading, nous pouvons construire des programmes qui peuvent utiliser plusieurs processus en même temps.
Ce tutoriel va introduire les capacités multithreading compatible avec toute plateforme de SDL.

XVIII - Thread

Tout comme à notre habitude, nous allons construire ce tutoriel autour d'un programme d'exemple. Ce programme va utiliser un thread pour animer la barre caption.
La première chose à faire pour utiliser les threads avec SDL est d'utiliser l'include correspondant :

fichiers d'entete
```
//Les fichiers d'entête
#include "SDL/SDL.h"
#include "SDL/SDL_image.h"
#include "SDL/SDL_thread.h"
#include <string>
```

Le fichier d'entête correspondant à l'utilisation des threads est donc... **SDL_thread.h**
Nous avons ensuite deux variables importantes :

```
//Le thread qui sera utiliser
SDL_Thread *thread = NULL;

//Ce qui va nous permettre de quitter
bool quit = false;
```

La première variable sera notre thread que nous allons lancer et qui est donc un SDL_Thread. La seconde est notre variable *quit* qui nous avons pour habitude d'avoir en local dans le *main()* mais cette fois ci on en a besoin en tant que variable globale afin d'être utilisée aussi avec notre thread.
Nous allons maintenant passer à la partie importante de notre programme : la fonction qui va s'occuper d'être notre thread.

```
int my_thread( void *data )
{
    //Tant que le programme n'est pas fini
    while( quit == false )
    {
        //On anime la barre caption
        SDL_WM_SetCaption( "Thread is running", NULL );
        SDL_Delay( 250 );

        SDL_WM_SetCaption( "Thread is running.", NULL );
        SDL_Delay( 250 );

        SDL_WM_SetCaption( "Thread is running..", NULL );
        SDL_Delay( 250 );

        SDL_WM_SetCaption( "Thread is running...", NULL );
        SDL_Delay( 250 );
    }

    return 0;
}
```

Voici donc ce qui sera notre thread. Tout ce que cette fonction fait est de mettre un message évolutif sur la barre de caption et ainsi créer une animation texte. Plus exatement, chaque quart de seconde, le message change tant que l'utilisateur n'a pas quitté.
Maintenant vous comprenez pourquoi notre variable quit est devenue globale.
Afin que la fonction soit utilisée comme un thread par SDL, il faut respecter deux choses :

- Premièrement, elle doit retourner un int
- Deuxièmement, elle doit avoir en argument qui est un pointeur vers un type de données void

Si la fonction ne respecte pas ces deux conditions, elle ne pourra pas être utilisée comme un thread.

Maintenant qu'on a fais notre fonction, il faut l'utiliser dans notre *main()* :

Création et lancement du thread
```
//Creation et lancement du thread
    thread = SDL_CreateThread( my_thread, NULL );
```

Dans notre fonction main, après que tout a été initialisé et chargé, nous appelons **SDL_CreateThread()**.
SDL_CreateThread() prend en premier argument la fonction, la transforme en thread et lance ce nouveau thread construit. La fonction retourne un pointeur vers le thread, ainsi on peut garder une trace de ce dernier.
La suite reste classique, on affiche une image à l'écran et on attend que l'utilisateur quitte le programme.

main
```
//On applique l'image sur l'écran
    apply_surface( 0, 0, image, screen );

    //Mise à jour de l'écran
    if( SDL_Flip( screen ) == -1 )
    {
        return 1;
    }

    //Tant que l'utilisateur n'a pas quitté
    while( quit == false )
    {
        //Tant qu'il y a un événement
        while( SDL_PollEvent( &event ) )
        {
            //Si l'utilisateur a cliqué sur le X de la fenêtre
            if( event.type == SDL_QUIT )
            {
                //On quitte the programme
                quit = true;
            }
        }
    }
```

Tant que c'est en marche, la barre caption change grâce à notre thread qui est lancé en parralléle de ce que nous sommes en train de faire. Grâce aux possibilités multithreading de SDL, nous sommes maintenant capable de faire deux choses en même temps.
Il ne faudra pas oublier de bien stopper le thread quand on quitte le programme

netoyage
```
void clean_up()
{
    //On stop le thread
    SDL_KillThread( thread );

    //On libère la surface
    SDL_FreeSurface( image );

    //On Quitte SDL
    SDL_Quit();
}
```

Pour stopper le thread, on appel la fonction **SDL_KillThread()** qui va stopper instantanement le thread.
Une meilleure façon de faire serait d'attendre que le thread finisse son travail, mais dans ce cas il n'y a aucun incident à simplement le stopper.
Ensuite on libère la surface de notre image et on quitte SDL, comme d'habitude.
Bien qu'il y ai beaucoup de cas où l'utilisation des threads peut se révéler intéressante dans les jeux vidéos, si vous êtes nouveau dans la programmation de jeux, ne les utilisez pas jusqu'à ce que vous soyez plus expérimentés.
Le multithreading peut vite devenir une prise de tête et son apport pas vraiment significatif dans le cadre de petits jeux. Vous avez à connaitre la synchronisation des threads, ainsi que comment gérer la concurrence.
Je recommande leur utilisation seulement si vous êtes devenu à l'aise avec l'architecture logicielle et seulement dans ce cas cet outil deviendra un puissant ami.

Téléchargements

Télécharger les sources du chapitre XVIII (102 ko)

Version pdf (84 ko - 6 pages) 

Remerciements

<= Retour au chapitre XVII : Création d'un Font Bitmap

-=Index Tutos SDL=-