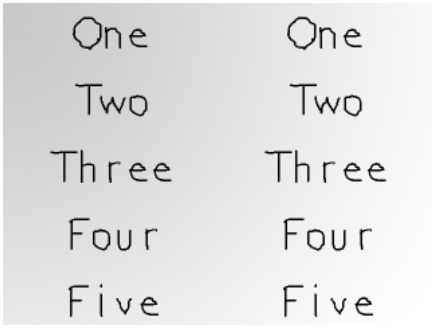
Semaphores



Last Updated 3/28/10

When you have multiple threads running you have to make sure that they don't try to manipulate the same piece of data at the same time. In this tutorial we'll use semaphores to lock threads to prevent them from crashing into each other.

Before we start there's something that has to be said. In this tutorial we have video functions running in separate threads. You should never do this in a real application. It's just bad software design and in some cases can cause your OS to become unstable. The only reason we're doing it here is because it's a small program and nothing's going to go wrong. We're doing it here just as a simple demonstration of semaphores in action.

Now with that out of the way let's start with the tutorial. In this program we're going to have two threads trying to blit surfaces to screen and update the screen at the same time. Having two threads trying to manipulate the same data at the same time is a recipe for disaster. This is where the semaphores come into play. What the semaphore will do is allow only one thread to manipulate to the screen at a time.

```
//The threads that will be used
SDL_Thread *threadA = NULL;
SDL_Thread *threadB = NULL;

//The protective semaphore
SDL_sem *videoLock = NULL;
```

At the top of our program we have our two threads we're going to have running along with our semaphore.

```
bool init()
{
    //Initialize all SDL subsystems
    if( SDL_Init( SDL_INIT_EVERYTHING ) == -1 )
    {
        return false;
    }

    //Set up the screen
    screen = SDL_SetVideoMode( SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_BPP, SDL_SWSURFACE );

    //If there was an error in setting up the screen
    if( screen == NULL )
    {
        return false;
    }

    //Initialize SDL_ttf
```

```
    if( TTF_Init() == -1 )
    {
        return false;
    }

    //Create the semaphore
    videoLock = SDL_CreateSemaphore( 1 );

    //Set the window caption
    SDL_WM_SetCaption( "Testing Threads", NULL );

    //If everything initialized fine
    return true;
}
```

Before we can use a semaphore you have to create it using SDL_CreateSemaphore().

You may be wondering what that 1 argument does. I'll explain later.

```
int blitter_a( void *data )
{
    //Y offset
    int y = 10;

    //Go through the surface
    for( int b = 0; b < 5; b++ )
    {
        //Wait
        SDL_Delay( 200 );

        //Show surface
        show_surface( ( ( SCREEN_WIDTH / 2 ) - text[ b ]->w ) / 2, y, text[ b ] );

        //Move down
        y += 100;
    }

    return 0;
}

int blitter_b( void *data )
{
    //Y offset
    int y = 10;

    //Go through the surface
    for( int b = 0; b < 5; b++ )
    {
        //Wait
        SDL_Delay( 200 );

        //Show surface
        show_surface( ( SCREEN_WIDTH / 2 ) + ( ( ( SCREEN_WIDTH / 2 ) - text[ b ]->w ) / 2 ), y, te

        //Move down
        y += 100;
    }

    return 0;
}
```

Here are our threads functions. All they do is go through an array of surfaces and show them on the screen. blitter_a() blits surfaces on the left of the screen and blitter_b() blits surfaces on the right.

As you can see we're not using our usual apply_surface() function. We're using show_surface(), a modified version of apply_surface() which applies the surface using SDL_BlitSurface() and updates the screen using SDL_Flip() all in one function. You'll see it in a little bit.

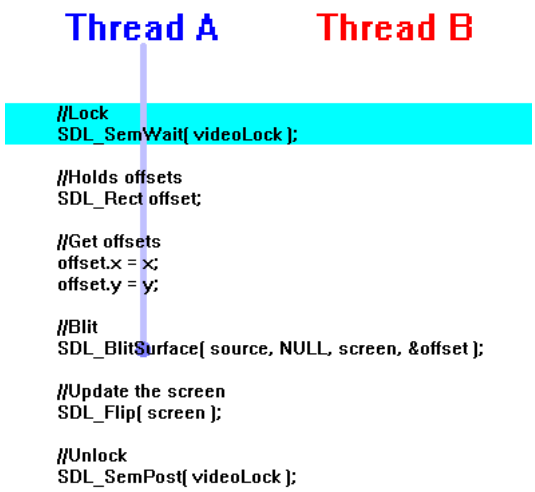
Since we're going to have the threads run at the same time we have prevent them from trying to manipulate the screen at the same time. To do this we're going to have to protect the show_surface() function using our semaphore.

```
void show_surface( int x, int y, SDL_Surface* source )
{
    //Lock
    SDL_SemWait( videoLock );
```

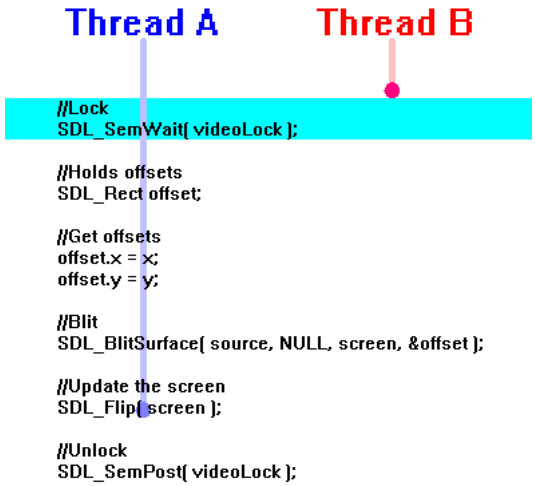
```
    //Holds offsets
    SDL_Rect offset;

    //Get offsets
    offset.x = x;
    offset.y = y;

    //Blit
    SDL_BlitSurface( source, NULL, screen, &offset );

    //Update the screen
    SDL_Flip( screen );

    //Unlock
    SDL_SemPost( videoLock );
}
```

Here's our show_surface() function which blits a surface then updates the screen. We also put SDL_SemWait() at the top of our function and SDL_SemPost() at the bottom.

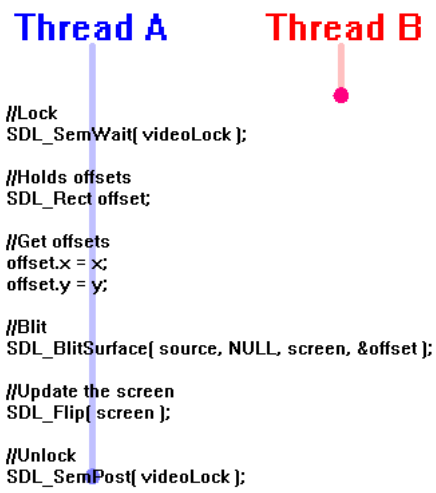
What SDL_SemWait() does is lock the semaphore:



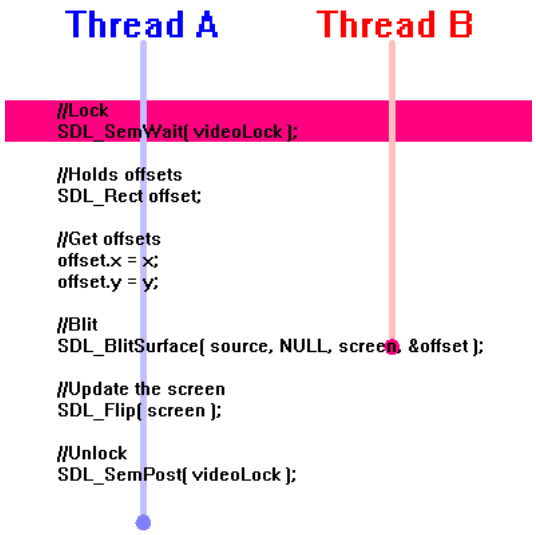
So when another thread tries to get through, it has to wait:



Until SDL_SemPost() is called to unlock the semaphore:



Then the next thread does its thing and locks the semaphore behind it:



Because SDL_BlitSurface() and SDL_Flip() are protected by the semaphore, only one thread can call video functions at a time so there's no conflict.

As I mentioned earlier we passed 1 argument in SDL_CreateSemaphore() in our init() function. That 1 is how many threads can get past the semaphore before it locks. If we wanted to we could have allowed 2 threads to pass, but that would be pointless in this case. Most of the time you will only want one thread to pass through a semaphore before locking.

```
    //Show the background
    show_surface( 0, 0, background );

    //Create and run the threads
    threadA = SDL_CreateThread( blitter_a, NULL );
    threadB = SDL_CreateThread( blitter_b, NULL );

    //Wait for the threads to finish
    SDL_WaitThread( threadA, NULL );
    SDL_WaitThread( threadB, NULL );

    //While the user hasn't quit
    while( quit == false )
    {
```

```
        //If there's an event to handle
        if( SDL_PollEvent( &event ) )
        {
            //If the user has Xed out the window
            if( event.type == SDL_QUIT )
            {
                //Quit the program
                quit = true;
            }
        }
    }
```

In the main() thread after everything is loaded and initialized we show the background on the screen.

Then we run our two threads and wait for them to finish by using SDL_WaitThread(). After the threads are finished we simply wait for the user to quit.

```
void clean_up()
{
    //Destroy semaphore
    SDL_DestroySemaphore( videoLock );

    //Free the surfaces
    SDL_FreeSurface( background );

    //Free text
    for( int t = 0; t < 5; t++ )
    {
        SDL_FreeSurface( text[ t ] );
    }

    //Close the font
    TTF_CloseFont( font );

    //Quit SDL_ttf
    TTF_Quit();

    //Quit SDL
    SDL_Quit();
}
```

Don't forget to clean up your semaphore when you're done with it.

We do that in our clean up function using SDL_DestroySemaphore().

Download the media and source code for this tutorial here.

Previous Tutorial Next Tutorial

SDL Forums SDL Tutorials Articles OpenGL Tutorials OpenGL Forums
News FAQs Games Contact Donations