Tutorial 4

Semaphores in Linux

1 How to stop cars falling off a bridge

This exercise uses POSIX semaphores to co-ordinate “cars” on a “bridge” in a multi-threaded application. Take a look at the C program specified by the attached files colourdraw.c, specs.h and bridge.c, which are all available via the course homepage. This program may be compiled with a command line such as

```
gcc -Wall -ggdb -o bridge colourdraw.c bridge.c \
   -L/usr/X11R6/lib -lX11 -lpthread -lm
```
(Leave out the back-slash if you type this onto one line.) Make sure you can build the program, and observe what happens when you run it.

This program represents cars crossing a bridge. The bridge has two directions, right and left.

The program has been written using POSIX threads, with separate threads for

- Cars entering from the right and left
- Cars leaving the bridge at the right and left
- Cars moving over the bridge

Threads here are just a type of process — specifically, a type of process that makes it easy to share data between threads. Hence, I implemented this example with threads rather than (full) processes.

The program also has a graphical interface to display the bridge; this is in colourdraw.c.

2 What to do

2.1 Deadlock

Explain how the current program can result in deadlock. Try to understand how that can happen.

2.2 Using Semaphores to Stop Cars “Dropping Off”

Use semaphores to improve the current program in the following way: Ensure mutual exclusion when global data representing the state of the bridge is modified, i.e. make sure that only one thread at a time can be accessing the global data structures representing the state of the bridge.

The change you implement here should eliminate the possibility of the program deadlocking.

**Important:** Use POSIX semaphores to implement this. You can find out about the API for POSIX semaphores with the command `man 3thr sem_init`. Also, note that I have already used semaphores to control access to the display, so you can “copy” from that implementation.
3 specs.h

// Department of Computing, Imperial College London
// Operating Systems II
// Olav Beckmann
// Assessed Tutorial 3

/* Number of positions the bridge has in each direction */
#define POSN_EACH_DIRECTION 10

#define MAX_COLOURS 5
enum {
  red = 0,
  brown = 1,
  blue = 2,
  yellow = 3,
  green = 4,
  white = 5
};
extern char *colournames[];

extern void initialise_display( );
extern void close_display();
extern void update_display( int right[POSN_EACH_DIRECTION],
                           int left[POSN_EACH_DIRECTION] );

4 bridge.c

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>

#include <pthread.h>
#include "specs.h"

#include <semaphore.h>

#define LOCAL_TO_CURRENT_THREAD 0

sem_t display_mutex;

/* Global variables that hold the "positions" on the bridge in each
direction. */
int right[POSN_EACH_DIRECTION];
int left[POSN_EACH_DIRECTION];

#define TRUE 1
#define FALSE (! TRUE)

int done = FALSE;
int num_cars;
int cars_out_right = 0;
int cars_out_left = 0;

void my_sleep( ) {
  usleep( 100000 );
}

void *enter_right( void *ptr ) {
  int i;
  for( i = 0; i < num_cars; ++i ) {
    int car = floor( drand48() * MAX_COLOURS );
    right[0] = car;
    sleep( 1 );
  }
  pthread_exit( NULL );
void *leave_right ( void *ptr ) {  
    while(! done) {  
        if( right[POSN_EACH_DIRECTION-1] != white ) {  
            right[POSN_EACH_DIRECTION-1] = white;  
            cars_out_right = cars_out_right + 1;  
        }  
        sem_wait( &display_mutex );  
        update_display( right, left );  
        sem_post( &display_mutex );  
        my_sleep();  
    }  
    pthread_exit( NULL );  
}  

void *leave_left ( void *ptr ) {  
    while(! done) {  
        if( left[0] != white ) {  
            left[0] = white;  
            cars_out_left = cars_out_left + 1;  
        }  
        sem_wait( &display_mutex );  
        update_display( right, left );  
        sem_post( &display_mutex );  
        my_sleep();  
    }  
    pthread_exit( NULL );  
}  

void *move ( void *ptr ) {  
    int i;  
    while(! done) {  
        for( i = POSN_EACH_DIRECTION - 2; i >= 0; i-- ) {  
            if( right[i + 1] == white && right[i] != white ) {  
                right[i + 1] = right[i];  
                right[i] = white;  
            }  
            sem_wait( &display_mutex );  
            update_display( right, left );  
            sem_post( &display_mutex );  
            my_sleep();  
        }  
        for( i = 1; i < POSN_EACH_DIRECTION; ++i ) {  
            if( left[i - 1] == white && left[i] != white ) {  
                left[i - 1] = left[i];  
                left[i] = white;  
            }  
            sem_wait( &display_mutex );  
            update_display( right, left );  
            sem_post( &display_mutex );  
            my_sleep();  
        }  
        pthread_exit( NULL );  
    }  
}
int main( int argc, char* argv[] ) {
    int i;
    pthread_t move_thread, leaver_thread, leavel_thread;
    pthread_t enterr_thread, enterl_thread;
    sem_init( &display_mutex, LOCAL_TO_CURRENT_THREAD, 1 );

    if( argc != 2 ) {
        fprintf( stderr, " Usage : ./bridge < number of cars>\n" );
        exit( 1 );
    }
    num_cars = atoi( argv[1] );
    srand48( 145789 );

    for( i = 0; i < POSN_EACH_DIRECTION; ++i ) {
        right[i] = white;
        left[i] = white;
    }

    initialise_display();
    update_display( right, left );
    my_sleep();

    pthread_create( &move_thread, NULL, move, NULL );
    pthread_create( &leaver_thread, NULL, leave_right, NULL );
    pthread_create( &leavel_thread, NULL, leave_left, NULL );
    pthread_create( &enterr_thread, NULL, enter_right, NULL );
    pthread_create( &enterl_thread, NULL, enter_left, NULL );

    pthread_join( enterr_thread, NULL );
    pthread_join( enterl_thread, NULL );

    while( num_cars != cars_out_right ) {
        printf( "Right: %d cars entered, %d left\n", num_cars, cars_out_right );
        sleep(1);
    }

    printf( "Right: %d cars entered, %d left\n", num_cars, cars_out_right );
    while( num_cars != cars_out_left ) {
        printf( "Left: %d cars entered, %d left\n", num_cars, cars_out_left );
        sleep(1);
    }

    printf( "Left: %d cars entered, %d left\n", num_cars, cars_out_left );
    done = TRUE;

    close_display();
    return 0;
}

 colourdraw.c

//
/* color-drawing.c — demonstrate drawing of pixels, lines, arcs, etc, using
 * different foreground colors, in a window.
 /* Acknowledgement: Little Unix Programmers Group */

#include <X11/Xlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include "specs.h"

XColor colours[MAX_COLOURS+1];
char *colournames[] = { "red",
    "brown",
    "blue",}
"yellow",
"green",
"white ");

const unsigned int line_width = 2; // line width when drawing on
unsigned int width, height;
// height and width for the new window.
Display* display; // pointer to X Display

GC gc; // GC (graphics context) used for
displaying in our window. */
Window win; // pointer to the newly created

/*
 * function: create_simple_window. Creates a window with a white background
 * in the given size.
 * input: display size of the window (in pixels), and location of
 * the window (in pixels).
 * output: window's ID.
 * notes: window is created with a black border, 2 pixels wide.
 * the window is automatically mapped after its creation.
 */

Window create_simple_window( Display* display,
    int width, int height, int x, int y ) {
int screen_num = DefaultScreen(display);
int win_border_width = 2;
Window win;

/* create a simple window, as a direct child of the screen's */
/* root window. Use the screen's black and white colors as */
/* the foreground and background colors of the window, */
/* respectively. Place the new window's top-left corner at */
/* the given 'x', 'y' coordinates. */
win = XCreateSimpleWindow(display, RootWindow(display, screen_num),
    x, y, width, height, win_border_width,
    BlackPixel(display, screen_num),
    WhitePixel(display, screen_num));

/* make the window actually appear on the screen. */
XMapWindow(display, win);

/* flush all pending requests to the X server. */
XFlush(display);

return win;
}

GC create_gc( Display* display, Window win, int reverse_video ) {
    GC gc; // handle of newly created GC. */
    unsigned long valuemask = 0; // which values in 'values' to */
    XGCValues values; // initial values for the GC. */

    int line_style = LineSolid; // style for lines drawing and */
    int cap_style = CapButt; // style of the line's edge and */
    int join_style = JoinBevel; // joined lines. */
    int screen_num = DefaultScreen(display);

    gc = XCreateGC(display, win, valuemask, &values);
    if (gc < 0) {
        fprintf(stderr, "XCreateGC: \n");
    }
    /* allocate foreground and background colors for this GC. */
    if (reverse_video) {
        XSetForeground(display, gc, WhitePixel(display, screen_num));
        XSetBackground(display, gc, BlackPixel(display, screen_num));
    } else {
        XSetForeground(display, gc, BlackPixel(display, screen_num));
        XSetBackground(display, gc, WhitePixel(display, screen_num));
    }

    /* define the style of lines that will be drawn using this GC. */
XSetLineAttributes(display, gc, 
    line_width, line_style, cap_style, join_style);

/* define the fill style for the GC. to be 'solid filling'. */
XSetFillStyle(display, gc, FillSolid);

return gc;

/* open connection with the X server. */
void initialise_display( ) {
    int screen_num; /* number of screen to place the window on. */
    unsigned int display_width, display_height; /* height and width of the X display. */
    char *display_name = getenv("DISPLAY"); /* address of the X display. */
    Colormap screen_colormap; /* color map to use for allocating colors. */
    Status rc; /* return status of various XLib functions. */
    int i,
    int screen_num;
    display = XOpenDisplay(display_name);
    if (display == NULL) {
        fprintf(stderr, "Cannot connect to X server 'n", display_name);
        exit(1);
    }

    /* get the geometry of the default screen for our display. */
    screen_num = DefaultScreen(display);
    display_width = DisplayWidth(display, screen_num);
    display_height = DisplayHeight(display, screen_num);
    width = (display_width / 2);
    height = (display_height / 6);

    /* create a simple window, as a direct child of the screen's */
    /* root window. Use the screen's white color as the background */
    /* color of the window. Place the new window's top-left corner */
    /* at the given 'x,y' coordinates. */
    win = create_simple_window(display, width, height, 0, 0);

    /* allocate a new GC (graphics context) for drawing in the window. */
    gc = create_gc(display, win, 0);
    XSync(display, False);

    /* get access to the screen's color map. */
    screen_colormap = DefaultColorMap(display, DefaultScreen(display));

    /* allocate the set of colors we will want to use for the drawing. */
    for (i = 0; i <= MAX_COLOURS; ++i) {
        rc = XAllocNamedColor( display, screen_colormap,
            colournames[i], &colours[i], &colours[i] );
        if (rc == 0) {
            fprintf(stderr, "XAllocNamedColor - failed to allocate 'n',
            colournames[i]);
            exit(1);
        }
    }

    /* Draw the "grid" of slots on the bridge */
    XSetForeground(display, gc, colours[brown].pixel);
    XDrawLine(display, win, gc, 0, height/2, width, height/2);
    for ( i = 0; i < POSN_EACH_DIRECTION; ++i ) {
        const int step = width / POSN_EACH_DIRECTION;
        XDrawLine(display, win, gc, step * i, 0, step * i, height);
    }
}

void close_display( ) {
    /* close the connection to the X server. */
    XCloseDisplay(display);
}

void update_display( int right [POSN_EACH_DIRECTION],
int left[POSN_EACH_DIRECTION] )
{
    int i;
    for( i = 0; i < POSN_EACH_DIRECTION; ++i )
    {
        const int step = width / POSN_EACH_DIRECTION;
        const int offset = 5;
        const int carheight = ((height - line_width) / 2) - (2 * offset);
        const int carwidth = (((width - (POSN_EACH_DIRECTION * line_width)) / POSN_EACH_DIRECTION) - (2 * offset));

        XSetForeground( display, gc, colours[right[i]].pixel );
        XFillRectangle( display, win, gc, step * i + offset, offset, carwidth, carheight );
        XSetForeground( display, gc, colours[left[i]].pixel );
        XFillRectangle( display, win, gc, step * i + offset, offset + height / 2, carwidth, carheight );
    }
}

/* flush all pending requests to the X server. */
XFlush(display);