mandel_cond.c

```c
/*
 * mandel_cond.c
 *
 * A program to draw the Mandelbrot Set on a 256-color xterm, with condition variables.
 *
 */

#include <signal.h>
#include <errno.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <semaphore.h>

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

/***************************
 * Compile-time parameters *
 ***************************/

/*
 * POSIX thread functions do not return error numbers in errno,
 * but in the actual return value of the function call instead.
 * This macro helps with error reporting in this case.
 */
#define perror_pthread(ret, msg) \
        do { errno = ret; perror(msg); } while (0)

/*
 * Output at the terminal is is x_chars wide by y_chars long
*/
int y_chars = 50;
int x_chars = 90;

/*
 * The part of the complex plane to be drawn:
 * upper left corner is (xmin, ymax), lower right corner is (xmax, ymin)
*/
double xmin = -1.8, xmax = 1.0;
double ymin = -1.0, ymax = 1.0;

/*
 * Every character in the final output is
 * xstep x ystep units wide on the complex plane.
 */
double xstep;
double ystep;

void sig_handler(int signum)
{
    reset_xterm_color(1);
```

```c
    exit(1);
}

/*
 * A (distinct) instance of this structure
 * is passed to each thread
 */
struct thread_info_struct {
        pthread_t tid; /* POSIX thread id, as returned by the library */

        int *color_val; /* Pointer to array to manipulate */
        int thrid; /* Application-defined thread id */
        int thrcnt;
};

int safe_atoi(char *s, int *val)
{
        long l;
        char *endp;

        l = strtol(s, &endp, 10);
        if (s != endp && *endp == '\0') {
                *val = l;
                return 0;
        } else
                return -1;
}

void usage(char *argv0)
{
        fprintf(stderr, "Usage: %s thread_count array_size\n\n"
                "Exactly one argument required:\n"
                "    thread_count: The number of threads to create.\n",
                argv0);
        exit(1);
}

void *safe_malloc(size_t size)
{
        void *p;

        if ((p = malloc(size)) == NULL) {
                fprintf(stderr, "Out of memory, failed to allocate %zd bytes\n",
                        size);
                exit(1);
        }

        return p;
}

/*
 * This function computes a line of output
 * as an array of x_char color values.
 */
void compute_mandel_line(int line, int color_val[])
{
    /*
     * x and y traverse the complex plane.
     */
    double x, y;
```

```
118
119     int n;
120     int val;
121
122     /* Find out the y value corresponding to this line */
123     y = ymax - ystep * line;
124
125     /* and iterate for all points on this line */
126     for (x = xmin, n = 0; n < x_chars; x+= xstep, n++) {
127
128         /* Compute the point's color value */
129         val = mandel_iterations_at_point(x, y, MANDEL_MAX_ITERATION);
130         if (val > 255)
131             val = 255;
132
133         /* And store it in the color_val[] array */
134         val = xterm_color(val);
135         color_val[n] = val;
136     }
137 }
138
139 /*
140  * This function outputs an array of x_char color values
141  * to a 256-color xterm.
142  */
143 void output_mandel_line(int fd, int color_val[]) {
144     char point ='@';
145     char newline='\n';
146
147     for (int i = 0; i < x_chars; i++) {
148         /* Set the current color, then output the point */
149         set_xterm_color(fd, color_val[i]);
150         if (write(fd, &point, 1) != 1) {
151             perror("compute_and_output_mandel_line: write point");
152             exit(1);
153         }
154     }
155
156     /* Now that the line is done, output a newline character */
157     if (write(fd, &newline, 1) != 1) {
158         perror("compute_and_output_mandel_line: write newline");
159         exit(1);
160     }
161 }
162
163 /* Start function for each thread */
164 pthread_mutex_t lock;
165 pthread_cond_t *cond;
166 int counter = 0;
167
168 void *compute_and_output_mandel_line(void *arg)
169 {
170     struct thread_info_struct *thr = arg;
171
172     for(int line = thr->thrid; line < y_chars; line += thr->thrcnt) {
173         compute_mandel_line(line, thr->color_val);
174         pthread_mutex_lock(&lock);
175         if(line != counter) pthread_cond_wait(&cond[thr->thrid], &lock);
176         counter++;
177         output_mandel_line(1, thr->color_val);
```

```
178         pthread_cond_broadcast(&cond[(line+1) % thr->thrcnt]);
179         pthread_mutex_unlock(&lock);
180     }
181         return NULL;
182 }
183
184 int main(int argc, char *argv[])
185 {
186     int thrcnt, ret;
187     struct thread_info_struct *thr;
188
189     // Signal Handler
190     struct sigaction sa;
191     sa.sa_flags = SA_RESTART;
192     sa.sa_handler = sig_handler;
193     if (sigaction(SIGINT, &sa, NULL) < 0)
194     {
195         perror("sigaction");
196         exit(1);
197     }
198
199     /*
200          * Parse the command line
201          */
202         if (argc != 2)
203             usage(argv[0]);
204         if (safe_atoi(argv[1], &thrcnt) < 0 || thrcnt <= 0 || argv[1][0] == '1') {
205             fprintf(stderr, "`%s' is not valid for `thread_count'\n", argv[1]);
206             exit(1);
207         }
208
209     xstep = (xmax - xmin) / x_chars;
210     ystep = (ymax - ymin) / y_chars;
211
212     thr = safe_malloc(thrcnt * sizeof(*thr));
213     cond = safe_malloc(thrcnt * sizeof(pthread_cond_t));
214
215     for(int i=0; i<thrcnt; i++)
216     {
217         thr[i].thrcnt = thrcnt;
218         thr[i].thrid = i;
219         thr[i].color_val = safe_malloc(x_chars * sizeof(int));
220
221         /* Spawn new thread */
222         ret = pthread_create(&thr[i].tid, NULL, compute_and_output_mandel_line, &thr[i]);
223         if (ret) {
224             perror_pthread(ret, "pthread_create");
225             exit(1);
226         }
227     }
228
229     /*
230     * Wait for all threads to terminate
231     */
232     for (int i=0; i<thrcnt; i++) {
233         ret = pthread_join(thr[i].tid, NULL);
234         if (ret) {
235             perror_pthread(ret, "pthread_join");
236             exit(1);
237             }
```

```
238     }
239
240     reset_xterm_color(1);
241     return 0;
242 }
243
```