mandel-fork1.c

```c
/*
 * mandel.c
 *
 * A program to draw the Mandelbrot Set on a 256-color xterm.
 *
 */

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

#include <semaphore.h>
#include <sys/mman.h>
#include <sys/wait.h>

/*TODO header file for m(un)map*/

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

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

/*
 * 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;

sem_t *sem;

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

        l = strtol(s, &endp, 10);
        if (s != endp && *endp == '\0') {
```

```c
                *val = l;
                return 0;
        } else
                return -1;
}

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

/*
 * 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;

    int n;
    int val;

    /* Find out the y value corresponding to this line */
    y = ymax - ystep * line;

    /* and iterate for all points on this line */
    for (x = xmin, n = 0; n < x_chars; x+= xstep, n++) {

        /* Compute the point's color value */
        val = mandel_iterations_at_point(x, y, MANDEL_MAX_ITERATION);
        if (val > 255)
            val = 255;

        /* And store it in the color_val[] array */
        val = xterm_color(val);
        color_val[n] = val;
    }
}

/*
 * This function outputs an array of x_char color values
 * to a 256-color xterm.
 */
void output_mandel_line(int fd, int color_val[])
{
    int i;

    char point ='@';
    char newline='\n';

    for (i = 0; i < x_chars; i++) {
        /* Set the current color, then output the point */
        set_xterm_color(fd, color_val[i]);
```

```
118         if (write(fd, &point, 1) != 1) {
119             perror("compute_and_output_mandel_line: write point");
120             exit(1);
121         }
122     }
123
124     /* Now that the line is done, output a newline character */
125     if (write(fd, &newline, 1) != 1) {
126         perror("compute_and_output_mandel_line: write newline");
127         exit(1);
128     }
129 }
130
131 void compute_and_output_mandel_line(int fd, int init_line, int procs)
132 {
133     /*
134      * A temporary array, used to hold color values for the line being drawn
135      */
136     int color_val[x_chars];
137
138     for(int line = init_line; line < y_chars; line += procs) {
139         compute_mandel_line(line, color_val);
140         if(sem_wait(&sem[line % procs]) < 0) {perror("sem_wait"); exit(1);}
141             output_mandel_line(fd, color_val);
142         if(sem_post(&sem[(line+1) % procs]) < 0) {perror("sem_post"); exit(1);}
143     }
144 }
145
146 /*
147  * Create a shared memory area, usable by all descendants of the calling
148  * process.
149  */
150 void *create_shared_memory_area(unsigned int numbytes)
151 {
152     int pages;
153     void *addr;
154
155     if (numbytes == 0) {
156         fprintf(stderr, "%s: internal error: called for numbytes == 0\n", __func__);
157         exit(1);
158     }
159
160     /*
161      * Determine the number of pages needed, round up the requested number of
162      * pages
163      */
164     pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;
165
166     /* Create a shared, anonymous mapping for this number of pages */
167     /* TODO:*/
168     addr = mmap(NULL, pages * sysconf(_SC_PAGE_SIZE), PROT_READ | PROT_WRITE, MAP_SHARED |
MAP_ANONYMOUS, -1, 0);
169     if(addr == MAP_FAILED) {perror("mmap"); exit(1);}
170
171     return addr;
172 }
173
174 void destroy_shared_memory_area(void *addr, unsigned int numbytes) {
175     int pages;
176
```

```
177     if (numbytes == 0) {
178         fprintf(stderr, "%s: internal error: called for numbytes == 0\n", __func__);
179         exit(1);
180     }
181 
182     /*
183      * Determine the number of pages needed, round up the requested number of
184      * pages
185      */
186     pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;
187 
188     if (munmap(addr, pages * sysconf(_SC_PAGE_SIZE)) == -1) {
189         perror("destroy_shared_memory_area: munmap failed");
190         exit(1);
191     }
192 }
193 
194 int main(int argc, char *argv[])
195 {
196     pid_t p;
197     int procs;
198 
199     /*
200          * Parse the command line
201          */
202         if (argc != 2)
203                 usage(argv[0]);
204         if (safe_atoi(argv[1], &procs) < 0 || procs <= 0) {
205                 fprintf(stderr, "`%s' is not valid for `procedures_count'\n", argv[1]);
206                 exit(1);
207         }
208 
209     xstep = (xmax - xmin) / x_chars;
210     ystep = (ymax - ymin) / y_chars;
211 
212     sem = create_shared_memory_area(procs * sizeof(sem_t));
213     if(sem_init(&sem[0], 1, 1) < 0) {perror("sem_init"); exit(1);}
214     for(int i = 1; i < procs; i++)
215         if(sem_init(&sem[i], 1, 0) < 0) {perror("sem_init"); exit(1);}
216 
217 
218     /*
219      * draw the Mandelbrot Set, one line at a time.
220      * Output is sent to file descriptor '1', i.e., standard output.
221      */
222     for (int init_line = 0; init_line < procs; init_line++) {
223         p = fork();
224         if(p < 0) {perror("fork"); exit(1);}
225         if(p == 0) {
226                     compute_and_output_mandel_line(1, init_line, procs);
227                     exit(0);
228                 }
229     }
230 
231     //Finish program
232     for(int i=0; i<procs; i++)
233         wait(NULL);
234     destroy_shared_memory_area(sem, procs*sizeof(sem_t));
235 
236     reset_xterm_color(1);
```

```
237     return 0;
238 }
239
```