School of Computer and Information Science

《**Operating-System-Concepts(10th)**》

Project Report

Student information：

Name：张乐之 ID：222022321102120

Major/Grade：2022 计科中外 04 Time：2024-10-31

Experiment information：

Topic:

Programming Project 4

Requirements:

1. Master the basic concepts, design ideas and status of the process. Understanding the basic principles and ideas of process synchronization and mutual exclusion. Use the basic means of realizing process synchronization and mutual exclusion to analyze and solve the basic problems of process synchronization and mutual exclusion.
2. Master the basic concepts and design ideas of the computer file system. Have the ability to write simple codes to read or write a file.

Procedure：

1. Choose the algorithm and code.
2. Compile the codes and link with the Linux library so that some functions like creating threads can work properly.
3. Run the program and check whether it meet the requirements.

Results：

（Code and Figures）

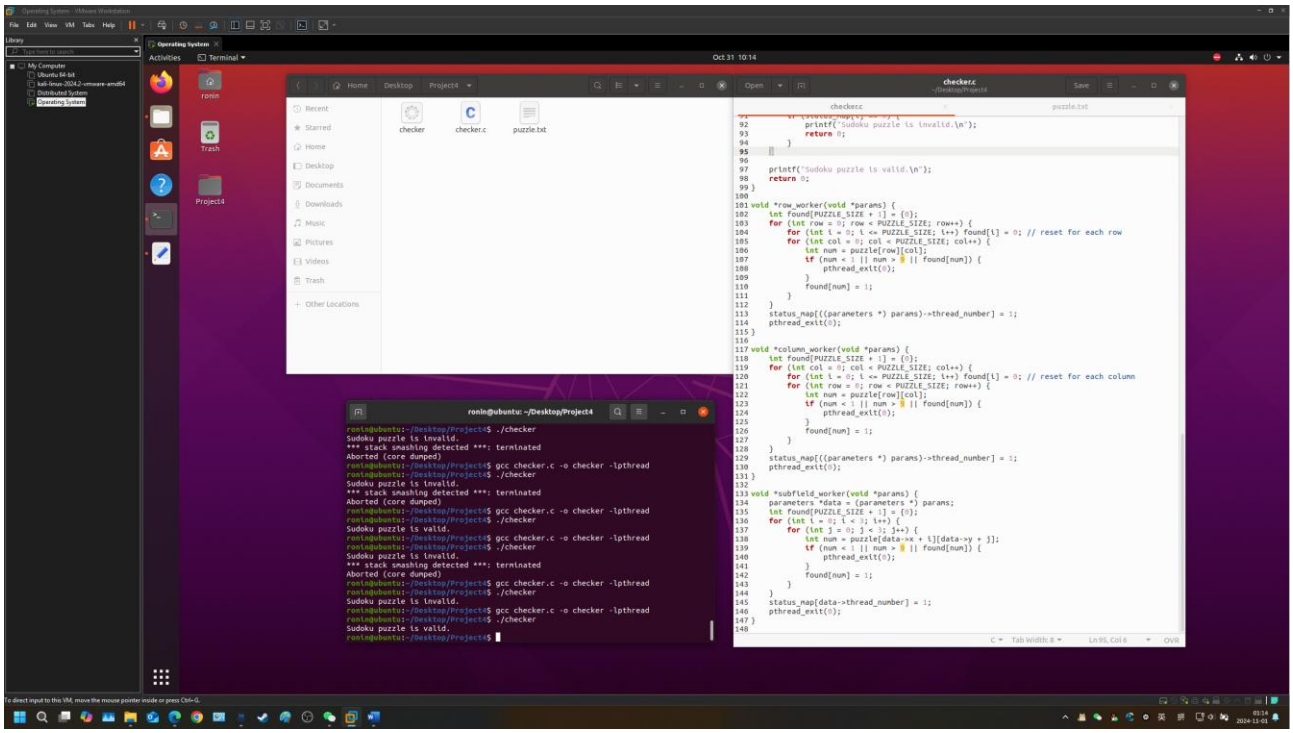
Codes are as follows:

```
1.  /**
2.   * Multithread solution for checking a Sudoku puzzle
3.   * Solution to project 1
4.   *
5.   * Operating System Concepts - Ninth Edition
6.   * Copyright John Wiley & Sons - 2013.
7.   */
8.
9.  #include <pthread.h>
10. #include <stdio.h>
11. #include <stdlib.h>
```

```
12.
13. #define NUMBER_OF_THREADS        11
14. #define PUZZLE_SIZE                  9
15.
16. // Function declarations
17. void *column_worker(void *param);       /* thread that checks columns */
18. void *row_worker(void *param);            /* thread that checks rows */
19. void *subfield_worker(void *param);    /* thread that checks subfields */
20.
21. // Shared result array
22. int status_map[NUMBER_OF_THREADS] = {0};
23.
24. // Data structure for passing data to threads
25. typedef struct {
26.         int thread_number;
27.         int x;
28.         int y;
29. } parameters;
30.
31. // Sudoku puzzle array
32. int puzzle[PUZZLE_SIZE][PUZZLE_SIZE];
33.
34. int load_puzzle_from_file(const char *filename) {
35.         FILE *file = fopen(filename, "r");
36.         if (file == NULL) {
37.                 perror("Error opening puzzle file");
38.                 return -1;
39.         }
40.
41.         for (int i = 0; i < PUZZLE_SIZE; i++) {
42.                 for (int j = 0; j < PUZZLE_SIZE; j++) {
43.                         if (fscanf(file, "%d,", &puzzle[i][j]) != 1) {
44.                                 perror("Error reading puzzle data");
45.                                 fclose(file);
46.                                 return -1;
47.                         }
48.                 }
49.         }
50.
51.         fclose(file);
52.         return 0;
53. }
54.
55. int main(int argc, char *argv[]) {
56.         pthread_t workers[NUMBER_OF_THREADS];
57.         int i;
```

```
58.
59.         // Load the Sudoku puzzle from file
60.         if (load_puzzle_from_file("puzzle.txt") != 0) {
61.                 fprintf(stderr, "Failed to load puzzle\n");
62.                 return 1;
63.         }
64.
65.         // Create thread for row check
66.         parameters *row_data = (parameters *) malloc(sizeof(parameters));
67.         row_data->thread_number = 0;
68.         pthread_create(&workers[0], NULL, row_worker, row_data);
69.
70.         // Create thread for column check
71.         parameters *col_data = (parameters *) malloc(sizeof(parameters));
72.         col_data->thread_number = 1;
73.         pthread_create(&workers[1], NULL, column_worker, col_data);
74.
75.         // Create threads for each 3x3 subgrid
76.         for (i = 0; i < PUZZLE_SIZE; i++) {
77.                 parameters *subfield_data = (parameters *) malloc(sizeof(parameters));
78.                 subfield_data->thread_number = i + 2;
79.                 subfield_data->x = (i / 3) * 3;
80.                 subfield_data->y = (i % 3) * 3;
81.                 pthread_create(&workers[i + 2], NULL, subfield_worker, subfield_data);
82.         }
83.
84.         // Wait for all threads to complete
85.         for (i = 0; i < NUMBER_OF_THREADS; i++) {
86.                 pthread_join(workers[i], NULL);
87.         }
88.
89.         // Check final result
90.         for (i = 0; i < NUMBER_OF_THREADS; i++) {
91.                 if (status_map[i] == 0) {
92.                         printf("Sudoku puzzle is invalid.\n");
93.                         return 0;
94.                 }
95.         }
96.
97.         printf("Sudoku puzzle is valid.\n");
98.         return 0;
99.  }
100.
101. void *row_worker(void *params) {
102.          int found[PUZZLE_SIZE + 1] = {0};
103.          for (int row = 0; row < PUZZLE_SIZE; row++) {
```

```
104.            for (int i = 0; i <= PUZZLE_SIZE; i++) found[i] = 0; // reset for each row
105.            for (int col = 0; col < PUZZLE_SIZE; col++) {
106.                int num = puzzle[row][col];
107.                if (num < 1 || num > 9 || found[num]) {
108.                    pthread_exit(0);
109.                }
110.                found[num] = 1;
111.            }
112.        }
113.        status_map[((parameters *) params)->thread_number] = 1;
114.        pthread_exit(0);
115. }
116.
117. void *column_worker(void *params) {
118.        int found[PUZZLE_SIZE + 1] = {0};
119.        for (int col = 0; col < PUZZLE_SIZE; col++) {
120.            for (int i = 0; i <= PUZZLE_SIZE; i++) found[i] = 0; // reset for each column
121.            for (int row = 0; row < PUZZLE_SIZE; row++) {
122.                int num = puzzle[row][col];
123.                if (num < 1 || num > 9 || found[num]) {
124.                    pthread_exit(0);
125.                }
126.                found[num] = 1;
127.            }
128.        }
129.        status_map[((parameters *) params)->thread_number] = 1;
130.        pthread_exit(0);
131. }
132.
133. void *subfield_worker(void *params) {
134.        parameters *data = (parameters *) params;
135.        int found[PUZZLE_SIZE + 1] = {0};
136.        for (int i = 0; i < 3; i++) {
137.            for (int j = 0; j < 3; j++) {
138.                int num = puzzle[data->x + i][data->y + j];
139.                if (num < 1 || num > 9 || found[num]) {
140.                    pthread_exit(0);
141.                }
142.                found[num] = 1;
143.            }
144.        }
145.        status_map[data->thread_number] = 1;
146.        pthread_exit(0);
147. }
```



As is shown in the picture, we can see that the checker can identify whether the matrix meet the requirements of a Sudoku puzzle or not and print the result.