Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

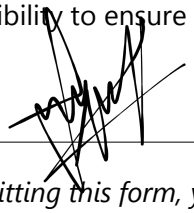
Last name:	Pathirage	Student ID:	21198889
Other name(s):	Rivin Genaka Kariyawasam		
Unit name:	Operating Systems	Unit ID:	COMP2006
Lecturer / unit coordinator:	Mr. Samantha Rajapaksha	Tutor:	
Date of submission:	29/04/2024	Which assignment?	(Leave blank if the unit has only one assignment.)

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature:  Date of signature: 29/04/2024

(By submitting this form, you indicate that you agree with all the above text.)

Report

Operating System Assignment (COMP2006)

(Rivin Pathirage, Curtin ID: - 21198889)

How to compile

Since I have made a makefile, you can just simply type “make” in the ubuntu terminal to compile all the c files.

How to Run

To Run simply type **“./mssv <file name> <delay**>” where the file name is the name of the file that contains the solution of sudoku that needs to be validated, and delay is the number of seconds you can make all children threads sleep. I have included 2 text files with the same solution named “solution.txt” and “testtext.txt” (“testtext.txt” was used to test the program though now it contains the same solution as the “solution.txt”). For an example, the program could run with the prompts **“./mssv solution.txt 1**”.

Synchronization

For the synchronization, I have made two main choices.

1. **Use Mutex variables (Mutal Exclusion) to protect shared data,**
2. **Use Conditional Variable instead of spinlock.**

Why Mutex?

Using the Mutex lock and unlock method was a very effective and a simple way of protecting shared data, especially thanks to pthread library that made using Mutual Exclusion simple and very effective.

What is Mutal Exclusion?

Mutual Exclusion in my program is when one thread is accessing the shared data no other thread can access it. Simply when one of my threads tries to access shared data It

checks whether the Mutex is available if they cannot access shared data until the shared data is released by whatever the thread that is currently using the thread. If the Mutex is available, lock the Mutex (So other threads can access Shared data) and after completing the execution of the Critical Section (Where in the code the shared data is accessed) the current thread will unlock the Mutex (Now other threads can access the shared data). This is done in all 5 threads including the parent thread(Since all threads access the shared data in my program . The Parent prints the solutions, hence will access shared data).

Why Conditional Variable?

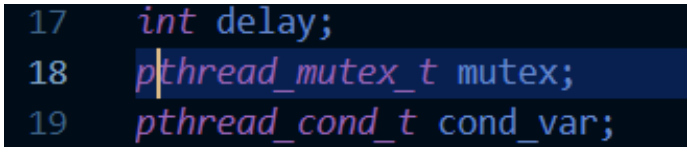
Using conditional variable using the pthread library was rather easy to understand and implement. Also, my other option was to use Spinlock. Though I have a multiprocessor in my laptop I felt the more efficient a program is the better (Spinlock is known to waste CPU resources), and for me using conditional variables got the work done in the most simple manner.

What is a Conditional Variable?

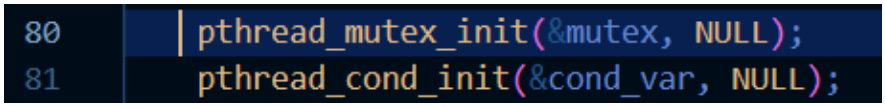
In my program, this is where one thread waits for a signal from another thread to resume execution. This signal is implemented using the Conditional Variable. I this for the parent thread to wait until the 4 other child threads to finish working so the Parent thread can print the correct results.

Mutex Implementation

First I have declared the mutex variable using the pthread library at the beginning of the main class (main.c) and the in the shared data header file (shared.h)



Then in the same main class, I initialized the mutex variable before creating the children threads.



Then after the creation of the 4 threads, I lock the Mutex before making the parent threads wait for the children thread to finish up and unlock it after the signal is received. This way I can ensure that the parent does not miss a signal or wake up at the wrong time. Since the conditional wait function of pthreads library releases the mutex until a signal comes ensures children can access shared data.

```
89      pthread_mutex_lock(&mutex);
90      pthread_cond_wait(&cond_var, &mutex);
91      pthread_mutex_unlock(&mutex);
```

Mutex in Child Threads

I created the 4 threads one after another and used Mutex in these threads in order to make sure the 4 threads execute in the 1,2,3,4 order so things are easy to understand, easy to debug if needed, and make sure the 4th thread (The One that checks the columns) finishes last.

```
pthread_t threads[4];
pthread_create(&threads[0], NULL, thread_func_1, NULL);
pthread_create(&threads[1], NULL, thread_func_2, NULL);
pthread_create(&threads[2], NULL, thread_func_3, NULL);
pthread_create(&threads[3], NULL, thread_func_4, NULL);
```

After creating the 4 threads in order, each thread's entry point function (The function the thread executes) has Mutex implemented so the shared data is accessed one after another, in the right order.

Example: - Thread 1 entry point (thread_func_1(void *arg), class threads (threads.c))

```
10  void *thread_func_1(void *arg)
11  {
12      for (int i = 0; i < 3; i++)
13      {
14          pthread_mutex_lock(&mutex);
15          row[i] = validate_row(i, sol);
16          sub[i] = validate_subgrid(i, sol);
17          counter += row[i] + sub[i];
18          pthread_mutex_unlock(&mutex);
19
20      }
21      sleep(delay);
22
23      return NULL;
24  }
```

All the thread entry points have similar Mutex utilized.

Conditional Variable Implementation

The conditional variable is used so the parent thread can print the results of the solution validation after the children threads (These threads validate the solution) have finished validating. If not, the parent thread will start to print results even before they are validated.

The Mutex implementation ensures the children threads are executing in order so the 4th thread will finish last. Hence the conditional variable only needs to be signaled at the end of the execution of the 4th thread.

```
59  void *thread_func_4(void *arg)
60  {
61      for (int i = 0; i < GRID_SIZE; i++)
62      {
63          pthread_mutex_lock(&mutex);
64          col[i] = validate_col(i, sol);
65          counter += col[i];
66          pthread_mutex_unlock(&mutex);
67
68      }
69      sleep(delay);
70
71      printf("Thread ID-4 is the last thread\n");
72      pthread_cond_signal(&cond_var);
73
74      return NULL;
75  }
```

After the signal is made by the 4th thread, the parent thread should catch the signal and start printing the results.

```
89      pthread_mutex_lock(&mutex);
90      pthread_cond_wait(&cond_var, &mutex);
91      pthread_mutex_unlock(&mutex);
92
93      print_validation_results(row, col, sub, counter);
```

Testing

There are 3 parts that needs to be tested to ensure the program is working correctly.

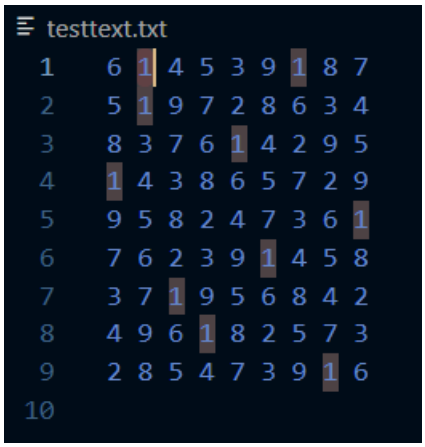
1. **The Sub Grid Validation Function (validate_sub_grid())**
2. **The Row Validation Function (validate_row())**
3. **The Column Validation Function (validate_col())**

If the above functions are correctly functioning, then the program is working as it should. For testing, I used correct inputs and incorrect inputs to see whether it identified the solution was correct or not.

As an example let's test the first sub grid and the 1nd row, and the 2nd Column (Which is in the same subgrid).

- Incorrect Input,

  The value is the 1st row, 2nd Column should be "2"(According to Sudoku Rules) but for testing I've made it "1".



Expected Output,

"Thread ID-1: row 1, sub-grid 1, are invalid

Thread ID-2: valid

Thread ID-3: valid

Thread ID-4: column 2,

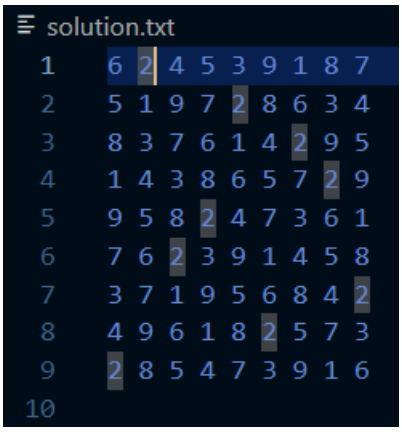
There are in total 24 valid rows, columns, and sub-grids, and the solution is invalid."

Output Displayed,

```
Delay - 1
Thread ID-4 is the last thread
Thread ID-1: row 1, sub-grid 1, are invalid
Thread ID-2: valid
Thread ID-3: valid
Thread ID-4: column 2,
There are in total 24 valid rows, columns, and sub-grids, and the solution is invalid.
rivin@Stallion:/mnt/c/Users/rivin/OneDrive - Curtin/OS Assignment$
```

Since the expected output and the real output matched the program has successfully identified the incorrect solution and the which row, column, and subgrid that it is incorrect in.

❖ Correct Input,



Expected Output,

"Thread ID-1: valid

Thread ID-2: valid

Thread ID-3: valid

Thread ID-4:

There are 27 valid sub-grids, and thus the solution is valid."

Output Displayed,

Since the expected output and the real output match, the program validates the given solution successfully.