COL331/COL633: Operating Systems
Assignment 2

Total: 12.5 Marks
Due Date: 15th March 23:59

Key Points:

1. The assignment has to be done individually.
2. -15% penalty per day, after the deadline.
3. We shall use Ubuntu Linux 18.04 to test the assignment. If the assignment does not work, it is the student's fault.
4. The whole assignment need to be done as a **user program**. You **DO NOT** need to change the kernel code.
5. You can remove the user programs created as a part of assignment 1.

1 *Jacobi* algorithm for steady state heat distribution [5 marks]

In this part of the assignment, you have to implement the steady state heat distribution algorithm using the *Jacobi* algorithm. Please find a write up of the algorithm on Moodle. Please ensure the following while implementation:

1. You have to implement a parallel version of the Jacobi algorithm, where the number of parallel processes is configurable during runtime.
2. You need to use the unicast and multicast system calls implemented in assignment 1.
3. There are five configurable parameters in the *Jacobi* algorithm.
   - Matrix size $N \times N$, with $1 \leq N \leq 20$.
   - Epsilon value, e, with $1.0 \times 10^{-5} \leq e \leq 1.0$.
   - Boundary temperature, T, with $100.0 \leq T \leq 5000.0$. In the algorithm (Figure 13.10) provided in the supplementary material it is currently set to 100.0.
   - Number of processes, P, with $1 \leq P \leq 8$.
   - Maximum number of iterations of the main loop, L, with $20,000 \leq L \leq 100,000$. This is a hard limit on the total number of iterations. The code will exit after these number of iterations, even if the *diff* is not less than *epsilon* (maximum value of the *count* variable in the code provided. See *jacob.c* for variable definitions and their usage).

4. These parameters will be provided in the following format in a file (named: **assig2a.inp**):

```
N
e
T
P
L
```

5. The final output of the algorithm is a $N \times N$ matrix, which contains the steady state temperature values.

6. The values are of datatype *float*. However, XV6 does not support printing floating point values. Hence, you need to type cast it to *int* (floor value) before printing. Ensure that the calculations are done using *float* and only while printing they are type casted to *int*.

7. The output format is an $N \times N$ matrix. The number of rows and columns is equal to N. The cells are separated by spaces. An example is shown below.

```
1 2 3 4 5
6 7 8 9 10
.
.
1 4 5 7 8
```

8. This part of the assignment will be graded with different **assig2a.inp** files as per the ranges mentioned.

2 Maekawa algorithm for mutual exclusion [5 Marks]

In this part of the assignment, you need to implement the Maekawa algorithm for mutual exclusion. This is used by parallel processes to get an exclusive lock to access a shared resource.

1. The input to this algorithm is the number of processes P, $P \in \{4, 9, 16, 25\}$.

2. There are three kinds of processes: P1, P2, and P3.

   - $P1$: Do not ask for the lock.
   - $P2$: Asks for the lock, and after acquiring the lock prints the following.

     `<pid> acquired the lock at time <T>.`

     waits for 2 seconds

     `<pid> released the lock at time <T>.`

     where, T is the global clock time (uint type: see *ticks* in *trap.c*). After releasing the lock, it never asks for the lock again.

- $P3$: Asks for the lock, after acquiring the lock it prints:

  `<pid> acquired the lock at time <T>.`

  `<pid> released the lock at time <T>.`

  where, T is the global clock time, and releases the lock immediately. After releasing the lock, it never asks for the lock again.

- The input will be given in a file named **assig2b.inp** with the following format:

```
P
P1
P2
P3
```

  where, P is the total number of processes and P1, P2 and P3 are the number of processes of their respective types. Please note that: $P1 + P2 + P3 = P$ and $0 \leq P1, P2, P3 \leq P$. The choice of processes in the P1, P2 and P3 types is up to you.

3. Run the program till all the processes asking for the lock (i.e. $P2 + P3$) get the lock (and release the lock).

3 Linux Implementation and Report [2.5 Marks]

XV6 runs on the Qemu emulator. The number of CPUs can be configured in the *Makefile*. However, the mapping is virtual and it uses a single physical CPU for emulation. Thus, the performance of a parallel program does not scale with an increase in the number of CPUs. It actually worsens because of the extra scheduling overheads.

Hence, for this part of the assignment, you need to implement the part 1 and part 2 of this assignment in Linux (Ubuntu). You are not required to write any kernel code. You can use already existing methods for IPC. Name the file as **jacob_linux.c** and **maekawa_linux.c** for part 1 and part 2 respectively. Please follow the same input-output format as defined above.

3.1 Report

1. For part 1 implemented in Linux, you need to plot the performance vs the number of processes.

2. For part 2 implemented in Linux, plot the total time taken by the program by varying P, P1, P2 and P3.

3. Comment on your observations and explain the reasons behind the trends.

4. Show how you verified the correctness of your programs in both XV6 and Linux.

4 Submission Instructions

- We will run MOSS on the submissions. We will also include last year's submissions. Any cheating will result in a zero in the assignment, a penalty as per the course policy and possibly much stricter penalties (including a fail grade and/or a DISCO).
- There will be NO demo for assignment 2.

How to submit:

1. Copy your report in the xv6 root directory.
2. Copy jacob_linux.c and maekawa_linux.c in the xv6 root directory.
3. Then, in the root directory run

```
make clean
tar czvf assignment2_<entry_number>.tar.gz *
```

   This will create a tar ball with name, assignment2_ ⟨ entry_number⟩.tar.gz in the same directory. Submit this tar ball on moodle.

4. Please note that we will use your *Makefile*. Ensure that you include the file **assig2a.inp** and **assig2b.inp** while building XV6.