Project 4
CS 4471

This programming project has two parts, listed below.

Create a Gitlab repository labeled Project4 inside the group I invited you to. Work on your project in your Gitlab repository. When finished, tag your latest commit with "Finished" and I will know to grade it. The timestamp of the "Finished" commit will be used to determine if you completed the project on time.

Notes:

1. For this homework you will implement the sender-initiated load balancing algorithm described in Sec. 8.2. Your output should exactly match the output given below.
   - Use the given starter code in `balance/`.
   - In the example output below for test file *test3.dat*, the first table (not the descriptive text at the beginning) prints which processes are assigned to run when on each CPU. For example, CPU A will run process 0 from 0-7. It would like to run process 1 from 6-13, but that isn't possible since they overlap. The final table shows how the processes were actually run. CPU B ended up running process 1.
   - The middle table describes states of CPUs and actions. If a CPU is underutilized, then "under" is printed. If it is overutilized, then "over" is printed. In the case of overutilization, the system attempts to find an underutilized CPU and transfer a process to it. For example, in the *test3.dat* example, process 3 is transferred from CPU B to CPU C in timestep 3.
   - You should make sure you understand the example output before starting to code.
   - You can assume that we will have at most 20 time steps.
   - It is recommended that you store CPU ids as integers. So CPU 'A' would be 0, 'B' would be 1, etc. You can compute this using `cpu_idx = cpu_id - 'A'`.

```
> ./balancing test1.dat
CPU A creates process 0 at time 0 requiring 8 time units.
CPU A creates process 1 at time 6 requiring 8 time units.

   0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19
A: 0  0  0  0  0  0  0  0
A:                    1  1  1  1  1  1  1  1

Time Proc State Action
6    A    over
7    A    over

   0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19
A: 0  0  0  0  0  0  0  0
A:                          1  1  1  1  1  1  1  1
```

```
> ./balancing test2.dat
CPU A creates process 0 at time 0 requiring 8 time units.
CPU A creates process 1 at time 6 requiring 8 time units.
CPU B creates process 2 at time 0 requiring 4 time units.
CPU B creates process 3 at time 3 requiring 5 time units.
CPU B creates process 4 at time 10 requiring 6 time units.

   0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19
A: 0  0  0  0  0  0  0  0
A:                    1  1  1  1  1  1  1  1
B: 2  2  2  2
B:          3  3  3  3  3
B:                               4  4  4  4  4  4

Time Proc State Action
3    B    over
6    A    over
7    A    over
9    B    under

   0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19
```

```
A: 0  0  0  0  0  0  0  0
A:                        1  1  1  1  1  1  1  1
B: 2  2  2  2
B:             3  3  3  3  3
B:                              4  4  4  4  4  4
```

```
> ./balancing test3.dat
CPU A creates process 0 at time 0 requiring 8 time units.
CPU A creates process 1 at time 6 requiring 8 time units.
CPU B creates process 2 at time 0 requiring 4 time units.
CPU B creates process 3 at time 3 requiring 5 time units.
CPU B creates process 4 at time 10 requiring 6 time units.
CPU C creates process 5 at time 0 requiring 1 time units.

   0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19
A: 0  0  0  0  0  0  0  0
A:                   1  1  1  1  1  1  1  1
B: 2  2  2  2
B:          3  3  3  3  3
B:                              4  4  4  4  4  4
C: 5

Time Proc State Action
1    C    under
2    C    under
3    B    over  process 3 -> CPU C
4    B    under
5    B    under
6    A    over  process 1 -> CPU B
8    A    under
8    C    under
9    A    under
9    C    under
10   A    under
10   B    over  process 4 -> CPU A
10   C    under
11   C    under
12   C    under
13   C    under
14   B    under
14   C    under
15   B    under
15   C    under

   0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19
A: 0  0  0  0  0  0  0  0
A:                              4  4  4  4  4  4
B: 2  2  2  2
B:                   1  1  1  1  1  1  1  1
C: 5
C:          3  3  3  3  3
```

2. You will implement a reader for steganographic images. Start from the given source code in `stega/`.

   - The skeleton code reads a .bmp file for you and places the file in `buffer`. See comments in *stega.cpp* for instructions on how to interpret the buffer.
   - The bitmap files are encoded as follows: each byte of image data (NOT header data, so be sure to skip the header as discussed in *stega.cpp*) has one bit of encoded data – the least significant bit. The first byte of data's least significant bit is the first character's least significant bit. The second byte of data contributes to the first character's second-least significant bit. Here are two examples:

```
bytes of data                       bits         ASCII     character
-------------------------------     --------     -----     ---------
255 254 254 118 118 118 121 124 --> 01000001 --> 65    --> 'A'
200 198 202 199 153 211 223 188 --> 01111000 --> 120   --> 'x'
```

Note: The bytes of data contribute to the ASCII bits in *reverse* order. For example, in the first example above, the byte 255 contributes the last one in the bits; 124 contributes the very first zero in the bits.

- How do you know if the last bit in a byte is a zero or one? There are two ways: you can either use a bitwise AND, or you can take the number modulo 2.
- How can you contruct an ASCII byte of data from a bunch of bits? Initialize the byte `b` to zero, then if, say, the third-to-last bit is a one, `b = b | (1 << 3)`, which left-shifts a one three places and ORs it with byte `b`.
- Test using the three included data files.
- **IMPORTANT!** To receive full credit you must decode a certain image found on the web. Go to `www2.cose.isu.edu/~mcgrmic2` and look for an image that has two gears. Download the image and run it through your steganography decoder. Follow the secret instructions for the final 10% of the project.

Following is a sample output.

```
> ./stega hello.bmp
Hello world!
```