CS69001 Computing Laboratory – I

Assignment No: C2

Date: 10–September–2018

An alien is spotted by a spacecraft on the object A94D75 in the Kuiper Belt. The spacecraft immediately takes a photo of the alien, and transmits the image to earth. Ten observatories receive the transmission. Since the spacecraft is quite far away and has feeble transmission power, the images received are quite corrupted. The observatories individually reconstruct some noisy lines of the image. The astronomers now want to stitch together the partial images and remove the noises as much as possible.

Assume that the spacecraft sends a 64×128 two-tone (black-and-white) image. Each pixel of the image is represented either by 0 (white) or 1 (black). The lines of the image are numbered $0 \ldots 63$, whereas the pixels in each line are numbered $0 \ldots 127$. Each line of the image is a bit-string of length 128, and can be compressed to 32 hexadecimal digits (lower-case in this assignment). For example, the bit string 0000 1001 1100 1111 0011 ... can be represented as $09cf3\ldots$.

Write a program that uses twelve collaborating processes to reconstruct the image.

- **Master process** This is the initial process U that you launch by running your executable.
- **Observatory processes** Simulate the work of ten observatories by the processes $V_1, V_2, \ldots, V_{10}$.
- **Stitching process** This process W stitches the raw lines received from the observatory processes.

The program involves the following tasks.

Part 1: Initial set up

You launch Process U by running your code. This process first creates two pipes P_r and P_i for transmitting the raw lines and the reconstructed image, respectively. U then forks the ten observatory processes $V_1, V_2, \ldots, V_{10}$ and the stitching process W. Subsequently, U waits until all the processes V_i are done.

Part 2: Sending raw data

Each observatory process V_i reads a data file **`Add.txt`**, where **`dd`** is a two-digit representation of i (with a leading zero for $i < 10$). The files **`A01.txt`**, **`A02.txt`**, ..., **`A10.txt`** will be supplied to you. Each line of these files stores a noisy information about a line of the image, and consists of a line number (in decimal) followed by 32 lower-case hex digits. The lines are not sorted with respect to the image line numbers. Moreover, not all image lines may be present in each input file. Here is a sample format for these lines.

```
29 0009fe7fffffffffe6fefff07fff9100
 1 000000080e0edff7bff8000000800004
46 000020000020e0007ff07fca02400000
51 00000000000000000081fffdfc0000000
 8 004807fffbdfffffffffbffffa20400
41 000000000007f0000003800183000000
```

Each V_i reads its input file line by line, inserts its tag i at the beginning of each line, converts the string of 32 hex digits to a string of 128 bits (characters 0 and 1), and writes the processed line to the pipe P_r. When all the lines are read from the respective input file, V_i writes to P_r a line containing its tag followed by the line number -1 indicating end of transmission. Write a function **`senddata()`** for this part.

Part 3: Receiving raw data

The stitching process W calls a function **`recvdata()`** to collect the raw data sent by *all* the observatory processes $V_1, V_2, \ldots, V_{10}$. It keeps on reading processed lines from the pipe P_r until it reads ten end-of-transmission notifications. W stores the image lines in a suitable data structure T after stripping off the process tags (and possibly also the image line numbers). Notice that T contains at most 640 128-bit strings. The function **`recvdata()`** returns the raw table T.

Reading and writing in P_r must be accomplished by the low-level **`read()`** and **`write()`** primitives.

Part 4: Reconstructing the image

After `recvdata()` returns T, the stitching process W invokes another function `stitchdata()` with T as input. The output of this function is a 64×128 image I of characters. Let i, j be a pixel of the image with $i \in [0,63]$ and $j \in [0,127]$. Suppose that W received the i-th line of the image from k of the observatory processes (we have $0 \leqslant k \leqslant 10$). W prepares the count n_0 of zeros and the count n_1 of ones in the received k lines at the j-th position. W then sets $I(i,j) = 0$ if $n_0 > n_1$, or $I(i,j) = 1$ if $n_1 > n_0$, or $I(i,j) = ?$ if $n_0 = n_1$ (*majority decision*). If $k = 0$ (that is, line i of the image is not transmitted by any of the observatory processes), we have $n_0 = n_1 = 0$, so the third case applies.

Part 5: Sending the reconstructed image

The stitching process W now sends the reconstructed image I to the master process U. This transmission happens via the pipe P_i in a stylistic manner. W redirects its `stdout` to the write end $P_r[1]$ of the pipe. It then uses `printf()` to send the image line by line. Write a function `sendimage()` for this part.

Part 6: Receiving and final processing of the reconstructed image

The master process U redirects its `stdin` to the read end $P_i[0]$ of the pipe. It uses `scanf()` to read and store the image I in a two-dimensional character array J. U then makes a final attempt to clean the image by removing its isolated ambiguous pixels marked by ?. Let $J(i,j)$ be such a pixel not on the boundary. If all the eight neighbors of $J(i,j)$ store the same bit (all zeros or all ones), U replaces the ? at $J(i,j)$ by that bit. Even after this postprocessing on the whole image, some pixels may still remain as ?.

Finally, U prints (to its default `stdout`) the cleaned up image for the astronomers. All of this part is to be written in a function `recvimage()`.

Notes

1. The *Kuiper Belt* is a region extending beyond the orbit of Neptune (at a distance of 30–50 AU from the sun). It consists of thousands of objects including the dwarf planet Pluto. Beyond the Kuiper belt, a sphere (0.8–3.5 LY) of small objects is loosely bound by the gravity of the sun, and is called the *Oort Cloud*. Except for these facts, the story given in the assignment is only a fiction.
2. The input file format is specified in Part 2. The sample output is not shown here. You need to work out how the alien looks like.
3. If you use a dark terminal with light fonts, replacing ones by spaces in the final printing of the image gives you a good visual impact. Conversely, if your terminal background is light and the font color is dark, print the zeros as spaces. In either case, print the other bits and the remaining question marks as they are.

Submit a single C source file. Do not use global/static variables.