Assign-05: *The Histogram*

Background

Think of this assignment as a puzzle that you need to solve – and in terms of designing your software there is no single right answer – there are many potential correct solutions. Where requirements have not been stated – you need to figure out a way to do what you need to do in order to (1) satisfy the requirements and (2) get the overall job done. This means that a lot of the design choices in this application are left up to you …

Create an application suite (called HISTO-SYSTEM) that consists of three distinct processing components:

- a "data consumer" (called DC) application
- and **two** different "data producer" (called DP-1 and DP-2) applications

You will need to write the data consumer, and each of the data producer programs in ANSI C under Linux.

The purpose of this assignment is to get you to see how / why and when certain IPC mechanisms are used within UNIX / Linux System Application programming. This program will use semaphores, shared memory and signals. The nice thing about this assignment is that for the most part, it can be built upon the sample code that has been demonstrated and provided in class.

Overall Considerations

Here are a couple of other things to consider:

- all applications should have modular design
- all applications should have module and function comment blocks
- as always – don't forget to comment your code
- also remember to hand in your code in the required directory structure with makefiles for your components
- your submitted code should have no debugging "print" statements (or any other print statements in any of the applications) – the only output from this system of three applications is the histogram data being output every 10 seconds from the DC process.

Data Consumer

Purpose : The data consumer (DC) program's job is to read data out of the shared memory's *circular buffer*. The data will be populated in this shared memory space by the 2 data producers.

Details :

- As noted in the *Data Producer* section below, the circular buffer will be large enough to hold 256 random letters (chosen by the producers). These letters will take on the values 'A' to 'T'
    - Make sure that your DC program follows best practices and checks for the existence of the shared memory before attempting to read data out of the buffer
    - If the shared memory doesn't exist, then the DC application will sleep for 10 seconds and try again to see if the shared memory has been established and created
    - Only then can the DC application enter its main processing loop
- The DC application will be launched from the DP-2 application.
    - DP-2 will pass the sharedMemoryID, the DP-1 processID and the DP-2 processID into the DC application on the command line at the time of launching
- The DC application will read the random letters out of the circular buffer using the *reading* index of the buffer
- It is the job of the DC application to keep track of (keep a count of) the number of each letter that it reads out of the buffer. The DC process will display a histogram of its current counts.
    - It will wake-up every 2 seconds (use a wake-up call (SIGALRM) to remind yourself) and read data out of the buffer
        - The DC application will guard itself from reading the buffer beyond the current write index (Note: this means that the DC application's access to the buffer must also use the semaphore)
        - Once the end of the buffer has been reached (i.e. read index 255), the DC application will wrap the read index back to zero and continue reading
    - Once every 10 seconds, the DC process will display the histogram of its counts
        - The DC process will clear the screen before displaying the histogram
        - Use special symbols in your histogram to signify letter count units of "ones" (-), "tens" (+) and "hundreds" (*)
        - Here are some examples of sample histograms
            - Assume for the purposes of these examples, that I am tracking the count of only the letters 'A' through 'E'
            - If there are 3 A's, 4 B's, 4 C's 5 D's and 3 E's – the histogram will look like this

```
A-003 ---
B-004 ----
C-004 ----
D-005 -----
E-003 ---
```

            - If there are 8 A's, 12 B's, 16 C's 13 D's and 9 E's – the histogram will look like this

```
A-008 --------
B-012 +--
C-016 +------
D-013 +---
E-009 ---------
```

- If there are 58 A's, 62 B's, 66 C's 53 D's and 49 E's – the histogram will look like this
  ```
  A-058 +++++--------
  B-062 ++++++--
  C-066 ++++++------
  D-053 +++++---
  E-049 ++++---------
  ```
- If there are 108 A's, 112 B's, 121 C's 109 D's and 99 E's – the histogram will look like this
  ```
  A-108 *--------
  B-112 *+--
  C-121 *++-
  D-109 *---------
  E-099 +++++++++---------
  ```

- If the DC process receives a SIGINT signal – it will send a SIGINT signal to each of the producers
  - *Question: Think of the launching order of the DC/DP-1/DP-2 applications and ask yourself – which process is in the foreground? How will you send a SIGINT signal to the DC application?*
  - Once the SIGINT signal is received by the DC process, it will set a state within its processing loop such that it will continue to read the data out of the buffer until it catches up (i.e. until the *reading* index catches up to *writing* index)
  - When the DC process has read all of the data out of the buffer, it will
    - Clear the screen and display the histogram one final time
    - Clean up its IPC usage and
    - Exit with the statement "**Shazam !!**"

- Please note that in these requirements, I have not told you exactly how many values to read out of the circular buffer when the Data Consumer is signaled to do so
  - This is left up to you to experiment with and determine how many the DC should read every 2 seconds knowing that
    - one of the DPs is blasting 20 values into the buffer every 2 seconds
    - the other DP is writing 1 value into the buffer every 1/20 of a second
  - things to consider about how many to read might be
    - there are only 256 elements in the buffer
    - remember that the DPs when writing their data are not allowed to overrun (or pass) the point in the buffer where the DC is reading from – so if a DP has to write some data and can't because writing it would surpass the DC – then that data doesn't get written
    - in 2 seconds of running there will be 60 elements written to the circular buffer from the 2 DCs (20 from DP-1 and 40 from DP-2)
    - so your experimentation range is between 1 and 60 values …
- Also don't forget that this is a circular buffer and the read and write indices wrap around back to the beginning once they've reached the end – you will need to do some funny accounting to determine if a DP has enough room to write its data when the indices wrap around

Data Producer(s)

Purpose : The data producer (DP) program's purpose is to setup some shared memory for use as a circular buffer (with a *read* and *write* index), to randomly generate some letters (between 'A' and 'T') and populate the buffer with their choices. For the purposes of the detail write-up, I have identified the 2 producers as DP-1 and DP-2.

Details :

- DP-1 when launched will allocate enough shared memory to hold a circular buffer of 256 characters, plus enough space for the 2 read/write indices
    - It will follow best practices and check for the existence of the shared memory, and if not found, will create it
    - After the shared memory has been created, DP-1 will launch DP-2 and will only pass the sharedMemoryID (shmID) value into DP-2 as a command line argument
        - DP-2 will need to get its processID (PID), and the processID of DP-1 (it's parent) and will need to launch the DC application and pass the required information on the command line (as noted in the DC section above)
        - After launching the DC application, DP-2 will immediately attach to the block of shared memory
        - At this point, both producers are ready to being populating the buffer with their data
- Both DP-1 and DP-2 will guard their writing into the circular buffer through the use of a semaphore
- DP-1 will generate 20 random letters one at a time and write all 20 into the buffer in one shot, then it will sleep for 2 seconds
    - Consider this letter generation and burst-mode write to be an atomic operation
- DP-2 will generate one random letter, write it into the buffer and sleep for 1/20 of a second
    - Consider this single letter generation and single write to be an atomic operation
- Both DP-1 and DP-2 will be capable of
    - Listening for and handling a SIGINT signal
        - When caught, the DP process will programmatically release the shared memory and exit with no statement to the world.
    - Once either the DP-1 or DP-2 process writes the 256 element of the buffer, it will wrap the index of the buffer back to zero and continue writing
        - Both the DP-1 and DP-2 processes will ensure that they never write past the read index
        - If the case of the DP processes ever catching up / overtaking to the read-index of the buffer happens, the DP process will stop writing and enter its sleep mode – hoping that the DC process catches up and reads more