(1) Test drive a C program that creates **Orphan and Zombie Processes**

(2) Develop a multiprocessing version of **Merge or Quick Sort.** Extra credits would be given for those who implement both in a multiprocessing fashion [increased no of processes to enhance the effect of parallelization]

(3) Develop a C program to count the maximum number of processes that can be created using fork call.

(4) Develop **your own command shell** [say mark it with @] that accepts user commands (System or User Binaries), executes the commands and returns the prompt for further user interaction. Also extend this to **support a history feature** (if the user types !6 at the command prompt; it shud display the most recent execute 6 commands). You may provide validation features such as !10 when there are only 9 files to display the entire history contents and other validations required for the history feature;

(5) Develop a multiprocessing version of Histogram generator to count the occurrence of various characters in a given text.

(6) Develop a multiprocessing version of matrix multiplication. Say for a result 3*3 matrix the most efficient form of parallelization can be 9 processes, each of which computes the net resultant value of a row (matrix1) multiplied by column (matrix2). For programmers convenience you can start with 4 processes, but as I said each result value can be computed parallel independent of the other processes in execution.

Non Mandatory (Extra Credits)..

(7) Develop a parallelized application to check for if a user input square matrix is a magic square or not. No of processes again can be optimal as w.r.t to matrix exercise above.

(8) Extend the above to also support magic square generation (u can take as input the order of the matrix..refer the net for algorithms for odd and even version…)