**Requirement (C based on Linux using multi processing):**

Implement a multi-process program for character frequency analysis in files.

This program must use the ring communication structure based on the pipe system call to accumulate a count of the occupancy of each character(a-z) in a set of text files within a sub-directory. Case can be ignored treating uppercase letters also in the count of lower case letters by lowercasing those.

Parameters accept:

Number of processes to create

Name of the sub-directory containing the text files as command-line arguments

Algorithm

1. The main process gets a list of the files in the target directory and store those file names in an array.
2. Main process then creates each process and add it to the ring communication structure.
3. Each process (including main process) then begins working through a sub-set of the files, summing up the frequency of each character
4. Once each process has finished processing its allocated files, it will need to communicate the results.
   1. Main process will simply send its frequency array to the next process in the ring and wait to receive the results from last process in the ring
   2. All other processes will need to receive the array of frequencies from the previous process in the ring, add their counts to the totals for each character and pass this result to the next process in the ring
   3. Then main process report to the terminal with final frequencies.

Potential approach:

1. File Retrieval: The main process will retrieve a list of files in the specified sub-directory and store their names in an array for further processing.  
  
2. Ring Communication Structure: I will employ the ring communication structure using the pipe system call to establish communication between processes. Each process, including the main process, will be created and added to the ring structure.  
  
3. Character Frequency Analysis: Each process will work on a subset of the files, accumulating the frequency count of each character (ignoring case by lowercasing uppercase letters).  
  
4. Communication and Result Aggregation:  
a. The main process will send its frequency array to the next process in the ring and await the results from the last process.  
b. Each intermediate process will receive the frequency array from the previous process, add its character counts to the totals, and pass the updated array to the next process.  
c. Finally, the main process will report the final frequencies to the terminal, including a simple star visualization based on the given formula.  
  
To facilitate the implementation, I will utilize the provided helper functions, `procFileCounts`, `main`, `add\_new\_node`, and `make\_trivial\_ring`. These functions will be modified and expanded as necessary to meet the project's requirements.

Example output:

A group of people in rows

Description automatically generated

Output must contain a simple star visualization to depict the frequencies. To calculate \* counts use formula (char\_count/max\_count)\*BARLENGTH, char\_count : count for a character, max\_count is the count for the most frequent character and BARLENGTH is the maximum size of the chart.

Need to use these helper functions.

long procFileCounts(char inFile[], long char\_stats[]){

char cur\_fname[MAX\_FILENAME] = TEXT\_DIR;

strcat(cur\_fname, inFile);

FILE\* file = fopen(cur\_fname, "r");

if (file == NULL) {

perror("Failed to open file");

exit(1);

}

char c;

long char\_count = 0;

/\* for as long as we can get characters... \*/

while((c=fgetc(file))) {

/\* break if end of file \*/

if(c == EOF){

break;

}else if((tolower(c)-'a')>=0 && (tolower(c)-'a') < 26){

char\_stats[tolower(c)-'a'] += 1;

}else{

continue;

}

/\* otherwise add one to the count of that particular character \*/

char\_count+=1;

}

return char\_count;

}

int main(int argc, char \*argv[ ]){

int i; /\* number of this process (starting with 1) \*/

int childpid; /\* indicates process should spawn another \*/

int nprocs; /\* total number of processes in ring \*/

if(parse\_args(argc,argv,&nprocs) < 0) exit(EXIT\_FAILURE);

if(make\_trivial\_ring() < 0){

perror("Could not make trivial ring");

exit(EXIT\_FAILURE); };

for (i = 1; i < nprocs; i++) {

if(add\_new\_node(&childpid) < 0){

perror("Could not add new node to ring");

exit(EXIT\_FAILURE); };

if (childpid) break; };

/\* ring process code \*/

fprintf(stderr, "node %d of %d\n", i, nprocs);

exit(EXIT\_SUCCESS);

}

int add\_new\_node(int \*pid){

int fd[2];

if (pipe(fd) == -1)

return(-1);

if ((\*pid = fork()) == -1)

return(-2);

if(\*pid > 0 && dup2(fd[1], STDOUT\_FILENO) < 0)

return(-3);

if (\*pid == 0 && dup2(fd[0], STDIN\_FILENO) < 0)

return(-4);

if ((close(fd[0]) == -1) || (close(fd[1]) == -1))

return(-5);

return(0);

}

int make\_trivial\_ring(){

int fd[2];

if (pipe (fd) == -1)

return(-1);

if ((dup2(fd[0], STDIN\_FILENO) == -1) ||

(dup2(fd[1], STDOUT\_FILENO) == -1))

return(-2);

if ((close(fd[0]) == -1) || (close(fd[1]) == -1))

return(-3);

return(0); }