```c
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <stdint.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>

const int32_t EXIT = 0;
const int32_t ONE = 1;
const int32_t DONE = -1;
const int32_t READ_ALL = -2;


int main(int argc, char *argv[]) {
int fd = open(argv[1], O_RDONLY ); //get input filename from argv
unsigned long n = strtoul(argv[2], NULL, 10);

int i = 0;
char c;

// Array to hold file descriptors (up to 16 pipes)
int pipefd[16][2];
// Create n pipes
for (int i = 0; i<n; i++){
assert(pipe(pipefd[i])==0);
}

pid_t pid[16]; // Store process ID (up to 16 child processes)
// Create n child processes and its PID
for (size_t i = 0; i<n; i++) {
pid[i] = fork();
if (pid[i] != 0) // If parent process, skip to next iteration
continue;
//child
int32_t line_num;
while (read(pipefd[i][0], &line_num, 4) == 4) {
switch (line_num) {
case EXIT:
printf("%d I'm exiting... \n", getpid());
if (i>0)
assert(write(pipefd[i-1][1], &EXIT, 4)==4); // Signal to exit process from the last one
for (int i = 0; i<n; i++){ // Close all pipes ends
close(pipefd[i][0]);
close(pipefd[i][1]);
}
return 0;
case DONE:
if (i+1 == n)
assert(write(pipefd[i][1], &EXIT, 4)==4); // If it's last process, signal to exit
else
assert(write(pipefd[i+1][1], &DONE, 4)==4); // Else, pass Done information to the next process
break;
case READ_ALL:
printf("%d Read all data\n", getpid()); // Only process that see "READ_ALL" print this (last process)
assert(write(pipefd[i][1], &DONE, 4)==4); // Pass Done to next process
break;
default:
if (read(fd, &c, 1) == 0) {
assert(write(pipefd[(i + n - 1)%n][1], &READ_ALL, 4)==4); // If EOF, initiate READ_ALL
} else { // Prints a line from text file with PID, line number
printf("%d #%d ", getpid(), line_num);
putchar(c);
while (c != '\n') { // Keep reading from input file until EOF
assert(read(fd,&c,1)==1);
putchar(c);
}
line_num++; // Increment line number
assert(write(pipefd[(i+1)%n][1],&line_num, 4)==4); // Send line number to next process
}
}
}
}
//parent
assert(write(pipefd[0][1], &ONE, 4)==4);
for (int i = 0; i<n; i++) { // Wait for child processes to stop
waitpid(pid[i], NULL, 0);
}
// Close all pipe ends
for (int i = 0; i<n; i++){
close(pipefd[i][0]);
close(pipefd[i][1]);
}
return 0;
}
```