```
//
//  main.c
//  catgrepmore
//
//  Created by GUOWEIZHE on 11/6/18.
//  Copyright © 2018 GUOWEIZHE. All rights reserved.
//

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <signal.h>
#include <sys/wait.h>
#include <errno.h>

#define READ 0
#define WRITE 1

int closePipe(int pipe1[], int pipe2[]);
void exitHandler();

void *buffer;
unsigned long long byteCounter = 0;
unsigned int fileCounter = 0;

int BUFSIZE = 4096;
char* pattern;
int pipe1[2];
int pipe2[2];

const struct sigaction ignoreSig = {
    .sa_handler = SIG_IGN,
    .sa_mask = 0,
    .sa_flags = 0
};

const struct sigaction interruptSig = {
    .sa_handler = exitHandler,
    .sa_mask = 0,
    .sa_flags = 0
};

int main(int argc, char **argv) {
    if(argc < 3){
        fprintf(stderr, "Error: Incorrect input format. The correct usage is: ./
         catgrepmore pattern infline1 [...infile2...]\n");
        return -1;
    }

    pattern = argv[1];
    sigaction(SIGPIPE, &ignoreSig, NULL);
    sigaction(SIGINT, &interruptSig, NULL);
```

```c
for(int i = 2; i < argc; ++i){
    if (pipe(pipe1) == -1) {
        fprintf(stderr, "Error while creating pipe for grep: %s\n",
         strerror(errno));
        return -1;
    }
    if (pipe(pipe2) == -1) {
        fprintf(stderr, "Error while creating pipe for more: %s\n",
         strerror(errno));
        return -1;
    }
    int grep_stat, more_stat;
    int grep_pid = fork();
    if(grep_pid < 0){
        fprintf(stderr, "Error while forking the process for grep: %s\n",
         strerror(errno));
        return -1;
    }
    else if(grep_pid == 0){
        // in children process
        if((dup2(pipe1[READ], 0)) < 0){
            fprintf(stderr, "Error while redirecting input using dup2: %s\n",
             strerror(errno));
            return -1;
        }
        if((dup2(pipe2[WRITE], 1)) < 0){
            fprintf(stderr, "Error while redirecting output using dup2: %s\n",
             strerror(errno));
            return -1;
        }
        if(closePipe(pipe1, pipe2) == -1){
            return -1;
        }

        execlp("grep", "grep", pattern, NULL); // should never get here
        fprintf(stderr, "Error while doing exec: %s\n", strerror(errno));
        return -1;
    }

    int more_pid = fork();
    if(more_pid < 0){
        fprintf(stderr, "Error while forking the process for more: %s\n",
         strerror(errno));
        return -1;
    }

    else if(more_pid == 0){
        if((dup2(pipe2[READ], 0)) < 0){
            fprintf(stderr, "Error while redirecting input using dup2: %s\n",
             strerror(errno));
            return -1;
        }
        if(closePipe(pipe1, pipe2) == -1){
            return -1;
        }

        execlp("more", "more", NULL);
        fprintf(stderr, "Error while doing exec: %s\n", strerror(errno));
        return -1;
    }

    if(more_pid > 0 && grep_pid > 0){
        // in parent process
        int readsize, infd;

        if((infd = open(argv[i], O_RDONLY)) < 0){
            fprintf(stderr, "Error while openning the input file named %s:
             %s\n", argv[i], strerror(errno));
            return -1;
        }
        fileCounter++;

        if(!(buffer = malloc(BUFSIZE))){
            fprintf(stderr, "Error while allocating memory for malloc\n");
            return -1;
        }

        do{
            readsize = read(infd, buffer, BUFSIZE);
            if(readsize < 0){
                fprintf (stderr, "Error when reading from the input file named
                 %s: %s\n", argv[i], strerror(errno));
                return -1;
            }
            else {
                int writesize, size;
                char * wbuffer;
                wbuffer = buffer;
                size = readsize;

                byteCounter += readsize;
                while (((writesize = write(pipe1[1], wbuffer, size)) != size) &&
                 size > 0) {
                    if(writesize < 0) {
                        fprintf(stderr, "Error when writing to the output:
                         %s\n", strerror(errno));
                        return -1;
                    }
                    size -= writesize;
                    wbuffer += writesize;
                }
            }
        } while(readsize != 0);

        if((close(infd)) < 0){
            fprintf(stderr, "Error while closing input file named %s: %s\n",
             argv[i], strerror(errno));
```

```
                return -1;
            }

            if(closePipe(pipe1, pipe2) == -1){
                return -1;
            }
            if (waitpid(grep_pid, &grep_stat, 0) < 0 || waitpid(more_pid,
             &more_stat, 0) < 0){
                fprintf(stderr, "Error while waiting for children to end: %s\n",
                 strerror(errno));
                return -1;
            }
            else if (!WIFEXITED(grep_stat) || !WIFEXITED(more_stat)) {
                perror("Error while exiting grep or more \n");
                return -1;
            }
        }
    }
    free(buffer);
    return 0;
}

int closePipe(int pipe1[], int pipe2[]) {
    if((close(pipe1[READ])) < 0){
        fprintf(stderr, "Error while closing pipe's read terminal for grep: %s\n",
         strerror(errno));
        return -1;
    }
    if((close(pipe1[WRITE])) < 0){
        fprintf(stderr, "Error while closing pipe's write terminal for grep: %s\n",
         strerror(errno));
        return -1;
    }
    if((close(pipe2[READ])) < 0){
        fprintf(stderr, "Error while closing pipe's read terminal for more: %s\n",
         strerror(errno));
        return -1;
    }
    if((close(pipe2[WRITE])) < 0){
        fprintf(stderr, "Error while closing pipe's write terminal for more: %s\n",
         strerror(errno));
        return -1;
    }
    return 0;
}

void exitHandler() {
    fprintf(stderr, "%d files, %llu bytes read\n", fileCounter, byteCounter);
    exit(0);
}
```