

```
/*
```

Process.c was originally all my work, but a couple of group members added to it. I cut out their code in this pdf. - Blake Barton

Group: B

Name: Blake Barton

Email: blake.barton@okstate.edu

Date: 9/25/22

Description:

This file currently has these primary functions:

- 1 - take user input on which input file and column (temporary)
- 2 - read the input file and save them into an array [readFile()]
- 3 - traverse array and find unique values in column and count occurrence of each [processSetup()]

- 4 - create parallel processes - one for each unique value in column [processCreation()]

- 5 - send row data to each process based on its unique value [processCreation()]

- 6 - closes parallel processes [processCreation()] (will likely need to be changed for the menu options)

compile: gcc process.c -lrt

execute: ./a.out

tested all options succesfully in csx2

```
*/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>
```

```
#include <time.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <fcntl.h>
#include <errno.h>
#include <mqueue.h>
```

```
#include "process.h"
```

```
#define QUEUE_NAME "/Queue-3-test"
#define PERMISSIONS 0660
#define MAX_MESSAGES 5
#define MAX_MSG_SIZE 210
#define MSG_BUFFER_SIZE 220
```

```
typedef char* String;
```

```
char bookInfo[703][6][210];
char amazonBestsellers[550][7][210];
```

```
/* readFile()
```

desc: reads the input file and parses data and stores in either bookInfo[][][] or amazonBestsellers[][][] based on file

input: file name, # of columns from file

```
*/
```

```
void readFile(char inputFile[25], int cols) {
```

```

char (*array)[cols][210];
if (cols == 6) array = bookInfo;
else array = amazonBestsellers;

FILE* f;
char ch;

f = fopen(inputFile, "r");

if (NULL == f) {
    printf("file could not be opened.\n");
}

int count = 0;
int row = 0;
int col = 0;

char str[210];
do {
    ch = fgetc(f);
    if (ch == '"') {
        //printf("TEST 1\n");
        count = count + 1;
        strncat(str, &ch, 1);
    } else if (ch == ',') {
        //printf("TEST 2\n");
        if (count % 2 == 1) {
            strncat(str, &ch, 1);
        } else {
            //printf("%s\n", str);
            sprintf(array[row][col], "%s", str);
            //printf("%s\n", bookInfo[row][col]);
            //printf("TEST $\n");
            memset(str, 0, 210);
            col++;
        }
    } else if (ch == '\n') {
        //printf("TEST 3\n");
        //printf("%s\n", str);
        sprintf(array[row][col], "%s", str);
        //printf("%s\n", bookInfo[row][col]);
        memset(str, 0, 210);
        col = 0;
        row++;
    } else {
        //printf("TEST 4\n");
        strncat(str, &ch, 1);
        //printf("TEST 5\n");
    }
} while (ch != EOF);

fclose(f);
}

/* processSetup()

```

```

desc: looks at the users selected column and divides it up by unique
categories
it collects the names of each unique value and the number of times it occurs
then it calls processCreation() to create the new processes and communicate
via message queue
input: # of rows, # of columns, user specified column, # of unique processes
output: 0
*/
int processSetup(int rows, int columns, int col, int processes, int *fout,
int *fin) {

    //selects file array to read from
    char (*array)[columns][210];
    if (rows == 703) array = bookInfo;
    else array = amazonBestsellers;

    //creates arrays to store category/process name (example: each book genre
or year) and the number of items per category
    char* category[processes];
    int amount[processes];

    //initializes the new arrays
    for (int k = 0; k < processes; k++) {
        category[k] = "";
        amount[k] = 0;
    }

    int index = 0; //counts through the new arrays
    int flag = 0; //0: name not in array, 1: name already in array

    //iterates through file array
    for (int i = 1; i < rows; i++) {

        //searches through new array to check if category from file is
already present, if so flags and increases the count
        for (int k = 0; k < index; k++) {
            if (strcmp(category[k], array[i][col]) == 0) {
                amount[k]++;
                flag = 1;
            }
        }

        //adds category to the array if flag is not raised
        if (flag == 0) {
            category[index] = array[i][col];
            amount[index] = 1;
            index++;
        }
        flag = 0; //clears flag for next row
    }

    //prints unique categories and number of items associated (TEMP)
    /*for (int k = 0; k < processes; k++) {
        printf("%s %dTEST\n", category[k], amount[k]);
    }
    */
}

```

```

    printf("processes running...\n");
    processCreation(processes, col, category, amount, rows, columns, fout,
fin);

    return 0;

}

/* processCreation()
desc: creates child processes for each unique value within user's column
also sends row data from the parent to the corresponding child processes
using POSIX message queue
input: # of processes, user specified column, array of unique value names,
array of occurrence of each unique value, # of rows, # of columns
output: 0
*/
int processCreation(int processes, int location, char* values[], int sizes[],
int rows, int cols, int *fout, int *fin)
{

    char (*array)[cols][210];
    if (cols == 6) array = bookInfo;
    else array = amazonBestsellers;

    //creates 11 child processes

    pid_t pids[processes]; //stores all the child pids

    mqd_t server_qd, client_qd; //server queue descriptor

    struct mq_attr attr = {
        .mq_flags = 0,
        .mq_maxmsg = MAX_MESSAGES,
        .mq_msgsize = MAX_MSG_SIZE,
        .mq_curmsgs = 0, //num messages on queue
    };

    //forks() for total number of processes - all are the child of the same
parent
    for (int i = 0; i < processes; i++) {

        if ((pids[i] = fork()) < 0) {
            perror("Fork failed.\n");
            exit(1);
        }

        else if (pids[i] == 0) { //child
            close(fd[0]);

            //array for storing contents of rows
            char list[sizes[i]][cols][210];

            //creates client name in format '/process'
            char client_name[64];
            sprintf(client_name, "%s", values[i]);

```

```

        //open client
        if ((client_qd = mq_open(client_name, O_RDWR | O_CREAT,
PERMISSIONS, &attr)) == -1) {
            perror("Child: mq_open client\n");
            exit(1);
        }

        sleep(1); //allows time for server to open

        //open server
        if ((server_qd = mq_open(QUEUE_NAME, O_RDWR | O_NONBLOCK)) == -1)
{
            perror("Child: mq_open server\n");
            exit(1);
        }

        char in_buffer[MSG_BUFFER_SIZE];

        int row = 0;
        int col = 0;

        //loops until server says to stop
        while (strcmp(in_buffer, "quit") != 0) {

            //receives message from server
            if (mq_receive(client_qd, in_buffer, MSG_BUFFER_SIZE, NULL)
== -1) {
                printf("IN_BUF: %s, MSGBUFSIZE: %i, ERROR %i\n",
in_buffer, MSG_BUFFER_SIZE, i);
                perror("Child: mq_receive\n");
                exit(1);
            }

            //adds to matrix unless closing message
            if (strcmp(in_buffer, "quit") != 0) {
                sprintf(list[row][col], "%s", in_buffer);

                //increments placement in columns and rows
                if (col < cols - 1) {
                    //printf("%s %s %d %d\n", client_name, in_buffer,
row, col);

                    col ++;
                }
                else {
                    col = 0;
                    row ++;
                }
            }
        }

        /*
        //test for printing: REMOVE

```

```

        for (int a = 0; a < row; a++) {
            for (int b = 0; b < cols; b++) {
                printf("Client: %s A: %d B: %d List: %s\n", client_name,
a, b, list[a][b]);
            }
        }*/

        //printf("%s closing\n", client_name);
        if (mq_close(client_qd) == -1) {
            perror("Child: mq_close\n");
            exit(1);
        }

        if (mq_unlink(client_name) == -1) {
            perror("Child: mq_unlink\n");
            exit(1);
        }

        return(0);
    }
}

//PARENT PROCESS SECTION

//creates the server (parent)
if ((server_qd = mq_open(QUEUE_NAME, O_RDWR | O_CREAT | O_NONBLOCK,
PERMISSIONS, &attr) == -1)) {
    perror("Server: mq_open server\n");
    exit(1);
}

sleep(1); //required to ensure all forks and clients open before using
them! DO NOT CHANGE

//loops through rows in array
for (int i = 1; i < rows; i++) {

    //loops through unique values in columns to send data to specific
process
    for (int j = 0; j < processes; j++) {

        //locates which process corresponds to the current row
        if (strcmp(array[i][location], values[j]) == 0) {

            //creates child/client name
            char client_name[64];
            sprintf(client_name, "%s", values[j]);

            //open child based on child name
            if ((client_qd = mq_open(client_name, O_RDWR)) == -1) {
                perror("Parent: mq_open client\n");
                exit(1);
            }
        }
    }
}

```

```

        //sends data from all columns to corresponding child process
        for (int k = 0; k < cols; k++) {
            char out_buffer[MSG_BUFFER_SIZE];
            sprintf(out_buffer, "%s", array[i][k]);
            //printf("%s\n", out_buffer);

            //send info to client
            if (mq_send(client_qd, out_buffer, strlen(out_buffer) +
1, 0) == -1) {
                perror("Parent: mq_send\n");
                exit(1);
            }
        }
    }
}

```

```

//sends message to terminate children

for (int i = 0; i < processes; i++) {
    char client_name[64];
    sprintf(client_name, "%s", values[i]);

    //open child based on child name
    if ((client_qd = mq_open(client_name, O_RDWR)) == -1) {
        perror("Parent: mq_open client\n");
        exit(1);
    }

    char out_buffer[MSG_BUFFER_SIZE];
    sprintf(out_buffer, "quit");

    //sends quit message to child process
    if (mq_send(client_qd, out_buffer, strlen(out_buffer) + 1, 0) == -1)
{
        printf("%s\n", out_buffer);
        perror("Parent: mq_send\n");
        exit(1);
    }

}

/*
for(int i = 0; i < rows; i++){
    for(int j = 0; j < cols; j++){
        printf("%-40s ", array[i][j]);
    }
    printf("\n");
}
*/

```

```

//ensures all children fully close

```

```

wait(NULL);

//closes server
if ((mq_close (server_qd)) == -1) {
    perror("Parent: mq_close\n");
    exit(1);
}

if ((mq_unlink(Queue_NAME)) == -1) {
    perror("Parent: mq_unlink\n");
    exit(1);
}

//prints out all processes and their ids
/*for (int i = 0; i < processes; i++) {
    printf("Process %d (pid = %d)\n", i, pids[i]);
}*/
return 0;
}

/*
int main() {
    char input[25];
    printf("Enter a file (bookInfo.txt or amazonBestsellers.txt):\n");
    scanf("%s", input);
    if (strcmp(input, "bookInfo.txt") == 0) {
        printf("Which category: Book category, Star rating, or Stock?\n");
        scanf("%s", input);
        readFile("bookInfo.txt", 6);
        if (strcmp(input, "Book") == 0) {
            processSetup(703, 6, 1, 43);
        } else if (strcmp(input, "Star") == 0) {
            processSetup(703, 6, 2, 5);
        } else if (strcmp(input, "Stock") == 0) {
            processSetup(703, 6, 4, 2);
        } else printf("Incorrect category.\n");
    } else if (strcmp(input, "amazonBestsellers.txt") == 0) {
        printf("Which category: User rating, Year, or Genre?\n");
        scanf("%s", input);
        readFile("amazonBestsellers.txt", 7);
        if (strcmp(input, "User") == 0) {
            processSetup(550, 7, 2, 10);
        } else if (strcmp(input, "Year") == 0) {
            processSetup(550, 7, 5, 11);
        } else if (strcmp(input, "Genre") == 0) {
            processSetup(550, 7, 6, 2);
        } else printf("Incorrect category.\n");
    } else {
        printf("Incorrect file name.\n");
    }
    return 0;
}
*/

```