```c
#include <stdio.h>
#include <time.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <string.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <mqueue.h>
#include <ctype.h>
#include "defs.h"

/**
 * This is the program entry point.
 */
int main(int argc, char **argv)
{
    pid_t cpid;
    double price = 0;
    FILE *outFile = NULL;
    struct Item *ptr = NULL;
    int N = 0, serial = 0, itemCount = 0, fd = 0, status;
    bool isValidOrder = FALSE, isValidNumber = FALSE;
    char itemName[100], store[100], buffer[BUFFER_SIZE];
    char tempChar;

    fd = shm_open(SHARED_MEMORY_NAME, O_CREAT | O_RDWR, PERMISSIONS);
    ftruncate(fd, SHARED_MEMORY_SIZE);
    ptr = mmap(NULL, SHARED_MEMORY_SIZE, PROT_WRITE, MAP_SHARED, fd, 0);
    outFile = fopen("items.txt", "r");

    /*Read the file line by line*/
    while (fgets(buffer, BUFFER_SIZE, outFile)) {
        /*Extremely hard regex for parsing the input file. This is basically equivalent to
tab delimiting.*/
        sscanf(buffer, "%d.\t%[^\t]\t$%lf\tat\t%[^\t]", &serial, itemName, &price, store);
        ptr[itemCount].price = price;
        ptr[itemCount].serialNumber = serial;
        strcpy(ptr[itemCount].giftName, itemName);
        strcpy(ptr[itemCount].storeName, store);
        itemCount++;
    }
    fclose(outFile);

    while (!isValidNumber) {
        printf("Enter the number of customers [1-26] >> ");
        scanf("%s", buffer);
        isValidNumber = sscanf(buffer, "%d", &N) && (N >= 1 && N <= 26);
    }

    while (!isValidOrder) {
        printf("Enter the order of processes in the form ABCD...>> ");
        scanf("%s", buffer);

        /*If there are more or fewer characters than processes, then checking further is
redundant.*/
        if (strlen(buffer) == N) {
            isValidOrder = TRUE; //Assume the ordering is valid, until proven otherwise
            for (int i = 0; i < strlen(buffer); i++) {
                tempChar = buffer[i];
                if(!isalpha(tempChar) || (toupper(buffer[i]) - 'A') > N - 1) {
                    printf("Invalid Character Sequence Encountered.\n");
                    isValidOrder = FALSE;
                    break;
                }
            }
        } else {
            printf("The number of characters in the order does not align with the number of
processes.\n");
        }
    }

    /*This portion of the shared memory is unoccupied because there are only 100 items.*/
    char *t = (char*)(ptr+100);
    strcpy(t, buffer);

    /*Iterate N+1 times to fork the customers as well as the helper*/
    for (int i = 0; i < N + 1; i++) {

        /*Fork two processes*/
        cpid = fork();

        //If the id is 0 then it is a child process.
        if (cpid == 0) {
            if (i == N) {
                helper();
            } else {
                customer(i);
            }
            exit(0);
        } else if (cpid == -1) {
            exit(1);
        } else {
            waitpid(cpid, &status, 0);
        }
    }
    /*Server resumes execution after the Helper finishes*/
    printf("\nThank you\n");
    return 0;
}

/**
 * This method is invoked by a customer process, it selects M random items to purchase. It
then sends this list to helper.
 * @param i Denotes the ith process.
 */
void customer(int i)
{
    mqd_t qd;
    int M = 0;
    struct mq_attr attr;
    bool isValidNumber = FALSE;
    char tempQueueName[BUFFER_SIZE] = {0}, outBuffer[MAX_MSG_SIZE] = {0},
tempString[MAX_MSG_SIZE] = {0}, buffer[BUFFER_SIZE] = {0};

    attr.mq_flags = 0;
    attr.mq_maxmsg = MAX_MESSAGES;
    attr.mq_msgsize = MAX_MSG_SIZE;
    attr.mq_curmsgs = 0;

    srand(time(0));
    /*Continue to prompt the user until they enter an integer ranging in [1,100]*/
    while (!isValidNumber) {
        printf("Enter the number of items [1-100] for %c >> ", 'A' + i);
        scanf("%s", buffer);
        isValidNumber = sscanf(buffer, "%d", &M) && (M >= 1 && M <= 100);
        if (!isValidNumber) {
            printf("An invalid amount was entered. Please try again.\n");
        }
    }

    snprintf(tempQueueName, BUFFER_SIZE, "/%c", 'A' + i);
    qd = mq_open (tempQueueName, O_WRONLY | O_CREAT, PERMISSIONS, &attr);
    snprintf(tempString, MAX_MSG_SIZE, "%d,", getpid());
    strcat(outBuffer, tempString);

    /*Iterate over the number of items purchased to select them randomly. Append the string 
to
    the output for the helper to order*/
    for (int j = 0; j < M; j++) {
        snprintf(tempString, MAX_MSG_SIZE, "%d", rand() % 100);
        strcat(outBuffer, tempString);
        if (j != M - 1) {
           strcat(outBuffer, ",");
        }
    }

    /*Send a message containing the indexed list of items.*/
    mq_send(qd, outBuffer,  strlen(outBuffer) + 1, 0);
}

/**
 * This methods is called by the Helper process. It receives the order of the processes from 
the Server process.
 *  Based on the order of processes, it receives the list of items via message passing.
 */
void helper()
{
    mqd_t qd;
    FILE *outFile;
    struct Item *ptr;
    char *order, *cptr;
    double runningTotal = 0;
    int fd = 0, arrIndex = 0, itemCount = 0, delimmitedIndex = 0, customerPID = 0;
    char inBuffer [MAX_MSG_SIZE], orderBuffer[MAX_MSG_SIZE], tempQueueName[BUFFER_SIZE] = 
{0}, tempFileName[BUFFER_SIZE] = {0}, tempString[BUFFER_SIZE] = {0};

    /*Access the shared memory to retrieve the list and items from main*/
    fd = shm_open(SHARED_MEMORY_NAME, O_RDONLY, PERMISSIONS);
    ptr = mmap(NULL, SHARED_MEMORY_SIZE, PROT_READ, MAP_SHARED, fd, 0);
    order = (char*)(ptr+100);
    strcpy(orderBuffer, order);

    /* Handle the orders from each customer, based upon the order specified from the Server. 
*/
    for (int j = 0; j < strlen(orderBuffer); j++) {

        snprintf(tempQueueName, BUFFER_SIZE, "/%c", toupper(orderBuffer[j]));
        snprintf(tempFileName, BUFFER_SIZE, "%cReceipt.txt", toupper(orderBuffer[j]));
        qd = mq_open (tempQueueName, O_RDONLY);
        outFile = fopen(tempFileName, "w");
        mq_receive (qd, inBuffer, MSG_BUFFER_SIZE, 0);
        cptr = strtok (inBuffer,",");
        /*Iterate over the comma delimmited list */
        while (cptr) {

            /*The first integer is the customer pid*/
            if (delimmitedIndex > 0) {
                /*After the customer pid has been read, start displaying receipt details*/
                if (delimmitedIndex == 1) {
                    sprintf(tempString, "Receipt for %c (Process %d):\n", orderBuffer[j], 
customerPID);
                    printf("\n%s", tempString);
                    fprintf(outFile, "%s", tempString);
                }

                ++itemCount;
                arrIndex = atoi(cptr);

                /*Write to both stdin as well as an output file.*/
                sprintf(tempString, "%d) $%.2lf - %s at %s", itemCount,  ptr[arrIndex].price, 
ptr[arrIndex].giftName, ptr[arrIndex].storeName);
                printf("%s", tempString);
                fprintf(outFile, "%s", tempString);
                runningTotal += ptr[arrIndex].price;
            } else {
                customerPID = atoi(cptr);
            }
            /*Continue along the list*/
            cptr = strtok (NULL, ",");
            delimmitedIndex++;
        }

        /*Close the current file and queue as it is no longer needed. Unlink the queue as 
well*/
        mq_close(qd);
        mq_unlink(tempQueueName);
        sprintf(tempString, "Total: $%.2lf\n", runningTotal);
        printf("%s", tempString);
        fprintf(outFile, "%s", tempString);
        fclose(outFile);

        /*Reset stats for the next process*/
        itemCount = 0;
        runningTotal = 0;
        delimmitedIndex = 0;
    }
}
```