```c
/*
    Author Name: Michael Royster
    Email: micaher@okstate.edu
    Date: 2/14/2021
    Program Description:
    Use POSIX IPC mechanisms in Unix: Memory Sharing and Message Sending.
    Server(main) Process loads items.txt into memory and creates shared memroy.
    Server Process forks N Customer Processes and 1 Helper Process.
    Each Customer Process asks for M random gifts selected from Shared Memory Spa
ce.
    Each Customer Process then sends the gifts via Message Queue to Helper Proces
s.
    Helper Process asks for the order in which to process the Customers.
    Helper then calculates total cost, prints receipt on screen in order
    and saves each receipt to a file.
    Server Process prints "Thank you" when Helper Process finishes.

    Compile: make -B main
    Run: ./main

    To remove all compiled files and receipts: make clean
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <time.h>
#include <sys/msg.h>
#include <errno.h>
#include <mqueue.h>
#include "assignment01_Royster_Michael_Gift.h"
#include "assignment01_Royster_Michael_Packet.h"
#include "assignment01_Royster_Michael_readInput.h"
#include "assignment01_Royster_Michael_writeReceipt.h"

#define QUEUE_NAME   "/customers"
#define PERMISSIONS 0660
#define MAX_MESSAGES 10
#define MAX_MSG_SIZE 4096
```

```c
#define MSG_BUFFER_SIZE MAX_MSG_SIZE + 10
#define GIFTS_IN_BUFFER 25

int main()
{
    // Create array of Gifts
    Gift gifts[100];

    // Read Gifts into an array
    readInput(gifts);

    // Read items.txt into shared memory as structure Gift
    Gift *ptrGifts;
    const int SIZE = sizeof(gifts) * 100;
    const char *sharedName = "items";
    int shm_fd;

    // Create shared memory space
    int parentid = getpid();
    shm_fd = shm_open(sharedName, O_CREAT | O_RDWR, 0666);
    ftruncate(shm_fd, SIZE);
    ptrGifts = mmap(0, SIZE, PROT_WRITE, MAP_SHARED, shm_fd, 0);

    // map memory
    int i = 0;
    for (i = 0; i < 100; i++)
        *(ptrGifts + i) = *(gifts + i);

    // Ask user for N number of processes
    char num_char[2];
    printf("Enter the number of customers: ");
    fgets(num_char, sizeof(int), stdin);
    int num = atoi(num_char);

    // Creating queue description and attributes for message queue
    mqd_t qd;
    struct mq_attr attr;
    attr.mq_flags = O_NONBLOCK;
    attr.mq_maxmsg = MAX_MESSAGES;
    attr.mq_msgsize = MAX_MSG_SIZE;
    attr.mq_curmsgs = 0;

    // For keeping track of pids
    int *customers = (int *)malloc(sizeof(int) * num);
```

```
    // Create N number of Customer Processes and 1 Helper Process
    for (i = 0; i < num; i++)
    {
        *(customers + i) = fork();
        if (*(customers+i) == 0) // Customer Processes
        {
            // Initialize random number generation
            time_t t;
            srand((unsigned)getpid());

            // openning shared memory space
            shm_fd = shm_open(sharedName, O_RDONLY, 0666);
            //wait(NULL);

            // Creating shared pointer and mapping to shared memory space
            Gift *ptr;
            ptr = (Gift *) mmap(0, SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);

            // Get number of random items
            char items_char[2];
            printf("Enter the number of items for process, %d:\n", getpid());
            fgets(items_char, sizeof(int), stdin);
            int items = atoi(items_char);


            // Buffer for sending message
            Packet packet;
            Packet *packet_out;
            packet_out = &packet;
            packet_out->size = items;
            packet_out->processid = getpid();

            // Generate random items
            int r = 0;
            for (i = 0; i < items; i++){
                r = (rand() - (int)(getpid()*getpid()) % 11) % 100;
                *(packet_out->gifts + i) = *(ptr + r);
            }

            // Open message queue
            if ((qd = mq_open (QUEUE_NAME, O_WRONLY | O_CREAT, PERMISSIONS, &attr
)) == -1) {
                perror ("Child: mq_open");
                exit (1);
            }else{
```

```
                //printf("sq_open successful\n");
            }
            if (mq_send (qd, (char *)packet_out, sizeof(Packet), 0) == -1) {
                perror ("Child: Not able to send message to the parent process.."
);
                exit(1);
            }else{
                //printf("message sent\n");
            }

            // Done with shared memory: close, unmap, and unlinking
            munmap(ptr, SIZE);
            close(shm_fd);
            shm_unlink(sharedName);

            // Free memory and return
            free(customers);
            return 0;
        }
    }


    // Ask for the order in which these processes will be executed
    if (parentid == getpid())
    {
        if (fork() == 0) // Helper Process
        {
            // Continue trying to open the queue until successful
            while((qd = mq_open(QUEUE_NAME, O_RDONLY)) == -1){
            }

            // Variables for the buffer
            char in_buffer[sizeof(Gift) * GIFTS_IN_BUFFER];
            Packet *packet_in;
            Packet packets[20];

            int j = 0;
            // Keep checking for queue messages until Helper receives N customer
messages
            while(j < num){
                if(mq_receive(qd, in_buffer, MSG_BUFFER_SIZE, NULL) > 0){
                    // Cast the buffer to a Packet type
                    packet_in = (Packet *)in_buffer;
                    packets[j] = *packet_in;
                    j++;
```

```
            }
        }

        // Get the order in which the packets should be processed.
        // Also load the order
        char order[32];
        char start = 'A';
        for (i = 0; i < num; i++){
            printf("PID %d is %c\n", *(customers+i), start);
            for (j = 0; j < num; j++){
                if(packets[j].processid == *(customers+i))
                    packets[j].order = start;
            }
            start++;
        }

        // Get the order in which to process
        printf("Enter the order: (ex: C B A):\n");
        printf("Order: ");
        fgets(order, 32, stdin);

        // Compute cost of all gift items
        for (i = 0; i < num; i++){
            for (j = 0; j < packets[i].size; j++){
                packets[i].totalCost += packets[i].gifts[j].price;
            }
        }

        int k = 0;
        // Print customer pid, gift items, total cost and save to file
        for(i = 0; i < strlen(order); i++){
            for(j = 0; j < num; j++){
                if(order[i] == packets[j].order){
                    printf("==================================\n");
                    printf("Customer PID: %d : %c\n", packets[j].processid, p
ackets[j].order);
                    printf("_____________________________________\n");
                    for(k = 0; k < packets[j].size; k++){
                        printf("%d %s %.2f %s\n", packets[j].gifts[k].serialN
umber,
                                packets[j].gifts[k].giftName, packets[j].gifts[k]
.price, packets[j].gifts[k].storeName);
                    }
                    printf("_____________________________________\n");
                    printf("Total Cost: %.2f\n", packets[j].totalCost);
```

```
                        // Write to file
                        writeReceipt(packets[j]);
                    }
                }
            }

            // mq_getattr(qd, &attr);
            // printf ("messages are currently on the queue: %ld.\n", attr.mq_cur
msgs);

            // Close the Queue
            if (mq_close (qd) == -1) {
                perror ("Helper: mq_close");
                exit (1);
            }
            // Unlinke the Queue
            if (mq_unlink (QUEUE_NAME) == -1) {
                perror ("Helper: mq_unlink");
                exit (1);
            }
            // Free memory and return
            free(customers);
            return 0;
        }
    }
    for (i = 0; i < num + 1; i++)
        wait(NULL);

    // Free memroy
    free(customers);

    // Print Thank you
    printf("\n Server: Thank you.\n");
    return 0;
}
```