



# Operating Systems [2020-2021]

---

## *Assignment 08 – Message queues and Memory Mapped Files*

### Introduction

Processes can exchange messages using a common system message queue, the IPC Message Queue. The sender places a message onto a queue, which can be read by the reading process. Each message is given an identifier that allows the reading processes to select the appropriate message. Processes must share the queue identifier (msgqid) in order to gain access to the queue.

Memory Mapped Files (MMF) provide a way to map a file into the address space of a process. It provides an easier access to file contents by treating file I/O through memory rather than through the use of system calls such as `read()` and `write()`. MMFs link the contents of files into pages of virtual memory and allow multiple processes to map those pages into their address space. Page faults trigger the mechanism for transporting “pages” of a file from disk to memory. All the I/O is done by the kernel. MMFs allow different processes to simultaneously access the contents of the same file in memory. However, synchronization is required between processes that are storing and fetching information to and from the shared memory pages.

### Objectives

Students concluding this work successfully should be able to:

- Use IPC Message Queues
- Use Memory Mapped Files

### Support Material

- K. A. Robbins, S. Robbins, “Unix Systems Programming: Communication, Concurrency, and Threads”, Prentice Hall:
  - Chapter 15 – POSIX IPC
- “Programming in C - UNIX System Calls and Subroutines using C” (<http://www.cs.cf.ac.uk/Dave/C/CE.html>):
  - IPC: Message Queues <sys/msg.h>
  - IPC: Shared Memory

## Exercises

**Note:** Only some of the exercises provided in this assignment will be done during the practical classes. The extra exercises should be done by the student as homework and any questions about them should be clarified with the teacher.

### 1. Priority

Write a program that generates and reads messages using a SysV Message Queue. The original process starts by creating 3 worker processes. Next, each second, it generates one message with random priority. The first worker process only gets messages with the highest priority. The second worker process gets messages with any priority but choosing the higher priorities first. The third worker process gets messages by the order they were created.

Consider:

- 3 levels of priority, each defined as an integer of your choice.
- The messages only contain the priority and a sequential message number.
- The original process produces one message every second and each worker gets one message every 5 seconds.
- The program finishes when the user presses Ctrl+C. All resources in use must be removed before leaving.

Complete the code given in file *priority.c* in order to obtain an output similar to the one in the next text box. An executable solution is also given in the assignment files.

```
user@ubuntu:~ /SO/Assignment08$ ./priority
Press ^C to finish...

=> [Message generator] Message 1 has priority 3
[high_priority_first] Received message [1]!
=> [Message generator] Message 2 has priority 3
[by_order] Received message [2]!
=> [Message generator] Message 3 has priority 1
[only_highest_priority] Received message [3]!
=> [Message generator] Message 4 has priority 1
=> [Message generator] Message 5 has priority 3
[high_priority_first] Received message [4]!
=> [Message generator] Message 6 has priority 3
[by_order] Received message [5]!
=> [Message generator] Message 7 has priority 1
[only_highest_priority] Received message [7]!
=> [Message generator] Message 8 has priority 3
^CTerminated
```

### 2. File Copier

Implement an application that copies the content of one file to another file using memory mapped files. This application should receive two arguments: the first is the path of the original file, and the second the path of the target file.

### 3. Store cashiers

Implement a simulator of store cashiers where a store has a number of cashiers open (CASHIERS\_NUM) to handle the customers. Each customer arrives at random intervals (from 2 to 10 seconds between arrivals), acquires a random number of products (from 2 to 10 products) and selects one of the available cashiers. The cashier takes 1 second to process each product.

The main process of your application should create the customer requests and add them to a common message queue. In each message, the message type identifies the cashier selected by the customer. Each cashier should be represented by a different process and handle its customers in the message queue (selecting the ones that have his message type). Customers can be simulated by using just one process that inserts values on a message queue.

Complete the code given in file *store.c* in order to obtain an output similar to the one in the next text box.

The simulation ends when the main process receives a Ctrl+C.

```
user@ubuntu:~ /SO/Assignment08$ ./store
[13644] Store is now open!
[13644] Customer 0 arrived and selected cashier 2 (3 products).
[13645] Cashier 1 open!
[13645] Cashier 1 waiting for next customer.
[13646] Cashier 2 open!
[13646] Cashier 2 waiting for next customer.
[13646] > Cashier 2 handling customer 0 (3 products)!
[13644] Customer 1 arrived and selected cashier 1 (4 products).
[13645] > Cashier 1 handling customer 1 (4 products)!
[13646] Cashier 2 waiting for next customer.
[13645] Cashier 1 waiting for next customer.
[13644] Customer 2 arrived and selected cashier 1 (5 products).
[13645] > Cashier 1 handling customer 2 (5 products)!
[13645] Cashier 1 waiting for next customer.
[13644] Customer 3 arrived and selected cashier 2 (7 products).
[13646] > Cashier 2 handling customer 3 (7 products)!
^C
[13644] Cleanup and exit!
[13645] Cleanup and exit!
[13646] Cleanup and exit!
```

### 4. News station

Implement a simulator of a news station that has several news crews gathering information on different topics across the country and reporting back to the news station. In the news station the section editors (each responsible for one news topic) receive the news, evaluate them and publish the ones considered relevant.

In your application represent each section editor and each news crew by a different process. The communication between the news crew and the section editors should be implemented using a message queue.

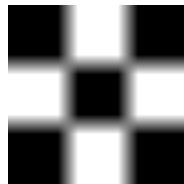
Use the following information to implement this application:

- There should be 5 topics (0: economy; 1: culture; 2: sports; 3: science; 4: technology)
- The number of section editors is the same as the number of topics
- Have at least 10 news crews
- Each crew should send a news to the station every 1 to 3 seconds
- The topic of the news created by the crews should be randomly selected
- The section editors only publish 10% of the received news

## 5. Squares image

Implement an application that creates different size PPM images ([http://en.wikipedia.org/wiki/Netpbm\\_format](http://en.wikipedia.org/wiki/Netpbm_format)) with different size black and white squares. This application should receive the destination file, the image width (in pixels, same as height) and each square width (in pixels). It creates a memory-mapped file with the PPM image P1 file format. In your program you must **use 2 processes**: a) one for producing the white squares; and b) another for producing the black squares.

An example of a 9x9 PPM P1 image with 3x3 squares is presented in the following images.



*Image 9x9 with 3x3 squares*

```
P1
9 9

1 1 1 0 0 0 1 1 1
1 1 1 0 0 0 1 1 1
1 1 1 0 0 0 1 1 1
0 0 0 1 1 1 0 0 0
0 0 0 1 1 1 0 0 0
0 0 0 1 1 1 0 0 0
1 1 1 0 0 0 1 1 1
1 1 1 0 0 0 1 1 1
1 1 1 0 0 0 1 1 1
```

*File content of image 9x9 with 3x3 squares*