Shared Memory & Message Queue

Samuele Germiniani
samuele.germiniani@univr.it

UNIVERSITÀ di VERONA
Department of ENGINEERING FOR INNOVATION MEDICINE

ISD
Cyber-Physical & IoT Systems Design

Exercise templates

- Download the templates of the exercises

```
git clone https://github.com/SamueleGerminiani/ex_shm_msgq_templates.git
```

Message queue - Exercise 1

- Create a client-server application based on a message queue that simulates the delivery of an order from a customer to a supplier. The message exchanged between the client and server is of type "order" struct defined in **order.h** .

- The **server** program creates a message queue and prints each message deposited in the queue to standard output. If the server receives the SIGINT signal, it removes the queue and finally terminates.

- The **client** program prompts the user for an order code, a description, a quantity, and an email address for delivery. Once all the required data is collected, the client deposits the order on the message queue created by the server and finally terminates.

Hints

```
msgget(…, …, …)
S_IRUSR S_IWUSR
fgets(…, sizeof(…),
stdin);
readInt(…)
msgsnd(…, …, …, …)

sigdelset(…, …)
sigprocmask(…, …, …)
signal(…, …)

msgget(…, IPC_CREAT |
S_IRUSR | S_IWUSR)

msgrcv(…, …, …, …, …)

msgctl(…, IPC_RMID, …)
```

Message queue - Extercise 2

Extend exercise 1 to meet the following requirements.

Assume that there are two types of customers: a "normal" customer and a "prime" customer. Extend the server program so that orders from "prime" customers can always be dequeued from the queue before orders from "normal" customers.

Hint: Exploit the mtype field of the deposited message. The msgrcv system call with a negative mtype can be used to receive messages from "prime" customers first and then messages from "normal" customers.

Hints

```
Same as Exercise 1
```

Message queue - Exercise 3

Extend exercise 2 to meet the following requirement: if no message is present in the message queue, the server prints the message "No orders! Contact the marketing office" to standard output. After that, the server should suspend itself (sleep) for 5 seconds before checking the message queue again.

Hints

```
Same as Exercise 1 +

ENOMSG
IPC_NOWAIT
```

Shared memory - Exercise 4

- Develop a Client-Server application based on shared memory to read the first 100 characters of a file present on the filesystem.

- The Server process instantiates a shared memory segment SH1 large enough to contain the **Request** structure defined in shared_memory.h. As soon as an instance of Request is deposited in SH1, the Server program performs the following operations:
  - a) opens the file indicated by the pathname for reading;
  - b.1) deposits the first 100 characters of the opened file in SH2;
  - b.2) if the file does not exist, deposits the value -1 in SH2;
  - c) removes the SH1 segment
  - d) finally, terminates

- The Client process creates a shared memory segment SH2 of size 100 bytes. It asks the user for the pathname of a file and deposits the Request structure in SH1. As soon as the first 100 bytes of the requested file are deposited in SH2, the Client program performs the following operations:
  - a) prints the content of SH2 to the screen
  - b) removes the SH2 segment
  - c) removes the semaphores used for synchronization with the Server
  - d) terminates

N.B You only need to edit the shared_memory.c file; however you must understand server.c and client.c first. The repository contains a file 'file.txt' you can use to test your application.

Hints

```
shmget(…, …,…)
IPC_CREAT | S_IRUSR |
S_IWUSR
shmat(…, …, …)
shmdt(…)
shmctl(…, …, …)
```

Shared memory - Exercise 5

Extend exercise 1 in a way that allows the Client process to read the entire file using the shared memory segment SH2.

Hints

Same as exercise 4