FIT3142 Laboratory #3: Client-Server Applications Using Shared Memory Inter-Process Communications Part 1

Dr Carlo Kopp, SMIEEE, AFAIAA, FLSS, PEng
Faculty of IT, Clayton.
Email: Carlo.Kopp@monash.edu

August 15, 2016

Revision Status:

$Id: FIT3142-Laboratory-3.tex,v 1.3 2016/08/15 08:32:15 carlo Exp carlo $

Contents

1 Introduction

Inter-Process Communications (IPC) techniques are the foundation upon which all distributed computing schemes operate. The purpose of the this and the following three Laboratory exercises is to provide students with experience in coding and designing applications using client-server model using shared memory IPC, and then applying this using stream oriented socket IPC.

IPC techniques can be broadly divided into *signals*, *stream oriented techniques*, *message queues*, *semaphores* and *shared memory*. Each have specific advantages and disadvantages:

1. *signals* - usually transmit state change information to a task, and are limited by the operating system used.
2. *stream oriented techniques* - both BSD sockets and SVR4 streams are limited to data which is sequential and can present problems in achieving controlled timing in transmission.
3. *message queues* - will often be limited to discrete message sizes, and may also present issues in controlling timing in transmission.
4. *semaphores* - are typically used for synchronisation and essentially limit communications to state changes.
5. *shared memory* - can be used not only to transmit large blocks of data between tasks, but also to transmit state information, or to permit processes to share access to common tables of system information. Shared memory is the fastest IPC technique.

Which of these schemes is implemented in a specific operating system will depend on the design of the system in question. Some systems may provide only a subset of the full range of techniques. For IPC over networks, stream oriented techniques are used almost exclusively. Shared memory is on the other hand, limited only to host systems, in multicore machines, where two or more processor cores can address the same area of memory in the machine.

The *shared memory* technique is very popular due to its speed, flexibility in use, and in many operating systems, simplicity in implementation.

A common implementation of this technique uses the manipulation of page or segment tables in the operating system to permit more than one process (task) to read, write or read and

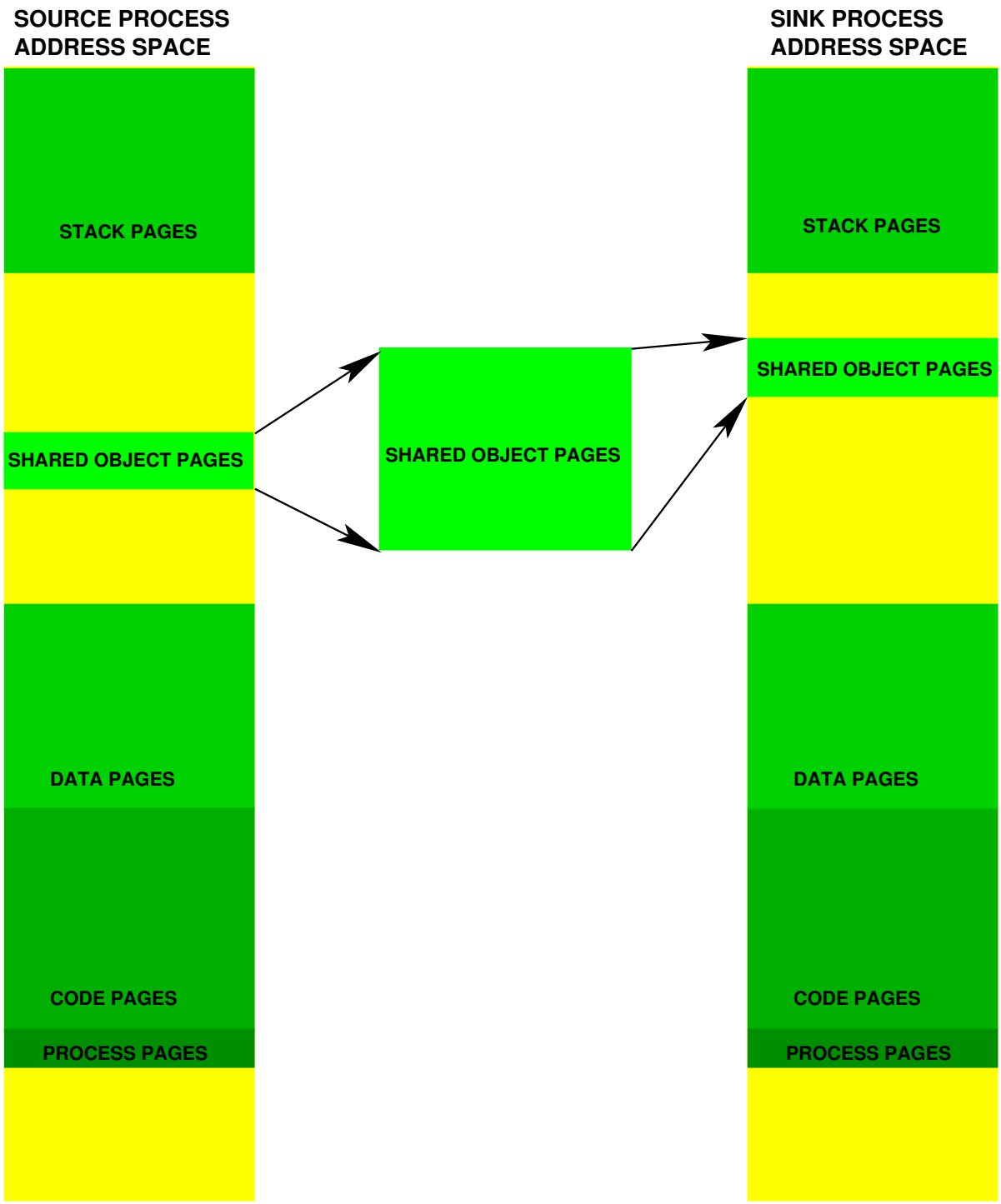

Figure 1: *Shared Memory - Two Processes on one host system.*

write a specific area of memory mapped into the address spaces of the respective processes (tasks).

The aim of this lab will be to familiarise you with the behaviour of shared memory, using the Linux system and the C language.

It is important that you do this work independently. Collaboration, copying and plagiarism will attract zero marks in this subject, without exception.

2 Lab Session 1

You will expend at least 30 minutes examining and testing the operation of the demonstration shared memory handling code, in the client-server programs `shm-server-test.c` and `shm-client-test.c`. To build these under Linux, run `make test`.

Read through the code in the C language source files, and the manual pages for the shared memory system calls named `shmctl.pdf`, `shmget.pdf`, `shmop.pdf`. Explore the use of arguments in the demonstration programs.

To run the demonstration programs, you need to first start the server in one window, and then the client in another. This is what you should observe in the server window:

```
deathstar[carlo]1221% shm-server
Waiting for client
Task completed
deathstar[carlo]1222%
```

This is what you should observe in the client window:

```
deathstar[carlo]1005% shm-client
Reading from Server Process SHM
abcdefghijklmnopqrstuvwxyz
Writing to Server Process SHM
Task completed
deathstar[carlo]1006%
```

Once you have become familiar with the design of the program, you can proceed to write your own client application to interface with the server application named `status-server (status-shm-server.c)`. This application continuously cycles through a table of values in shared memory, changing their values once per cycle. The client application is to read these values and write them to the screen. The user interface for the client checks the user response and exits if the user types in '1', or continues if the user types in '0'. You are provided with the source code and executable for the `status-server (status-shm-server.c)` and an executable for the client `MyID-client`. You must create the source file and your own executable for MyID-shm-client.c. A template called `MyID-client-template.c` is provided for you.

1. You must not edit `status-shm-server.c` or you will break it.

2. You must edit `Makefile` to replace MyID with your ID number.
3. You must edit `MyID-shm-client.c` to incorporate your own code.
4. You are provided with the executable file `MyID-client` to show you how your client program should operate.
5. You will submit by email the source file `MyID-shm-client.c`.
6. If your program does not compile you will be awarded zero marks for this prac.
7. You must demonstrate your program to the demonstrator before the end of the second prac session. If your program does not work at all you will be awarded zero marks for this prac. If it has bugs, marks will be deducted accordingly.
8. Assessment criteria:
   (a) `Makefile` updated and template file renamed (10%);
   (b) `shm` client and server compile with no warnings from `-Wall` option (10%);
   (c) Return values from shared memory functions checked for failure (10%);
   (d) Client code is well commented (10%);
   (e) Server exits correctly (10%);
   (f) Client correctly outputs status and reads user input correctly (50%);
9. Cheating, collaborating and plagiarism will attract zero marks.
10. **Save all of your program files as you will need them in future labs.**

3 Server and Client Function

```
deathstar[carlo]1223% status-server

STATUS DUMP
UP Status        = 0
Exit Status      = 0
RPM              = 3400
Crank Angle      = -1
Throttle Setting = 69
Fuel Flow        = 49
Engine Temp      = 79
Fan Speed        = 29
Oil Pressure     = 69
Waiting for client

STATUS DUMP
UP Status        = 0
Exit Status      = 0
RPM              = 3300
Crank Angle      = -2
Throttle Setting = 68
Fuel Flow        = 48
Engine Temp      = 78
Fan Speed        = 28
Oil Pressure     = 68
Waiting for client

STATUS DUMP
UP Status        = 0
Exit Status      = 0
RPM              = 3200
Crank Angle      = -3
Throttle Setting = 67
Fuel Flow        = 47
Engine Temp      = 77
Fan Speed        = 27
Oil Pressure     = 67
Waiting for client

UP Status        = 0
Exit Status      = 0
```

```
RPM              = 3100
Crank Angle      = -4
Throttle Setting = 66
Fuel Flow        = 46
Engine Temp      = 76
Fan Speed        = 26
Oil Pressure     = 66
Waiting for client

STATUS DUMP
UP Status        = 0
Exit Status      = 0
RPM              = 3000
Crank Angle      = -5
Throttle Setting = 65
Fuel Flow        = 45
Engine Temp      = 75
Fan Speed        = 25
Oil Pressure     = 65
Waiting for client

STATUS DUMP
UP Status        = 0
Exit Status      = 0
RPM              = 2900
Crank Angle      = -6
Throttle Setting = 64
Fuel Flow        = 44
Engine Temp      = 74
Fan Speed        = 24
Oil Pressure     = 64
Waiting for client

STATUS DUMP
UP Status        = 0
Exit Status      = 1
RPM              = 2800
Crank Angle      = -7
Throttle Setting = 63
Fuel Flow        = 43
Engine Temp      = 73
Fan Speed        = 23
Oil Pressure     = 63
```

```
Waiting for client
Task completed
deathstar[carlo]1224%


deathstar[carlo]1025% MyID-client

CLIENT STATUS DUMP
RPM              = 3200
Crank Angle      = -3
Throttle Setting = 67
Fuel Flow        = 47
Engine Temp      = 77
Fan Speed        = 27
Oil Pressure     = 67
Enter Command (1 to exit, 0 to continue): 0


CLIENT STATUS DUMP
RPM              = 3000
Crank Angle      = -5
Throttle Setting = 65
Fuel Flow        = 45
Engine Temp      = 75
Fan Speed        = 25
Oil Pressure     = 65
Enter Command (1 to exit, 0 to continue): 1

Task completed
deathstar[carlo]1026%
```