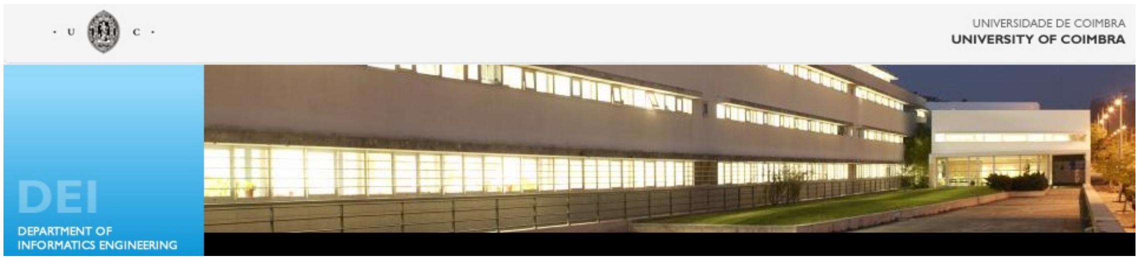


Operating Systems Project 2013

Introduction

This assignment will focus in the following topics:

- Concurrent programming with processes and threads, using the `<pthread.h>` library.
- Use synchronization primitives such as semaphores and condition variables.
- Inter-process communication using pipes and named-pipes;
- I/O multiplexing by using `select();`
- Asynchronous signals;
- Memory Mapped Files, as a way to manipulate files on disk and also share data between collaborative processes.

Objectives

At the end of this project, students should be able to:

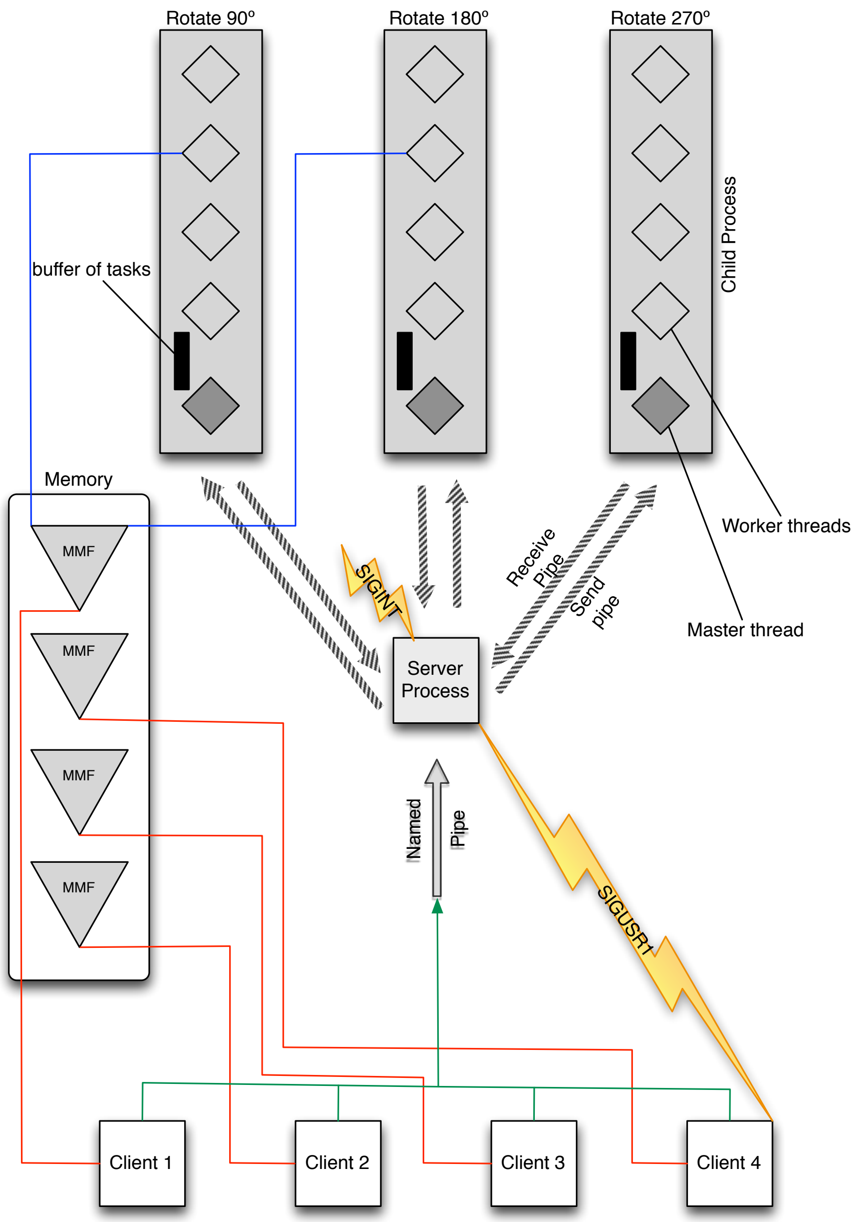
- Create and manage processes in Unix;
- Create and manage multiple threads;
- Use the library `<semaphore.h>` to create and use semaphores, *mutex* and condition variables to synchronize the access to shared resources between competing threads and processes;
- Cope with signal management;
- Master the communication between processes using pipes and named-pipes;
- Know how to use `select()` to implement I/O multiplexing;
- Map a file into a buffer in memory and manipulate its data.

Project Assignment

Overview

The aim of this project is to provide an image manipulation service to the users of a system. This service in particular is able to rotate ppm images in three angles: 90º, 180º and 270º. Any user logged into the system must be able to use this service. Users invoke the service through a small client application that connects to the server program and sends it the identification of the image to rotate and the amount of rotation to apply.

The following image depicts the overall architecture of the software that must be developed in the scope of this project. The actual image manipulation code is available in a sample program provided with this assignment.



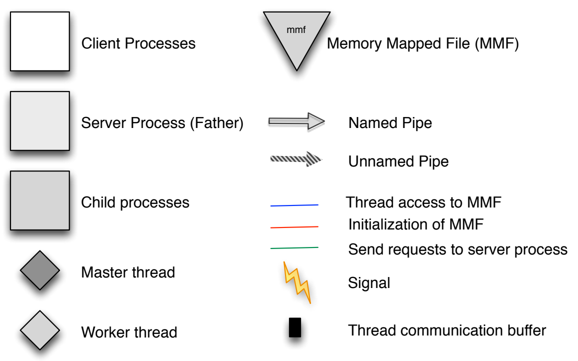


Figure 1 - Project and legend

There are three types of processes in the system:

1. **Client** Processes: Any user can invoke client processes in the system. The aim of these processes is to submit images to the server service for rotation. Consider that the name of the client program is “rotate”. Invoking `rotate` is done in the shell using “`$ rotate image-file-name rotate-angle`”, where `image-file-name` is the path to the image file and `rotate-angle` can take the values 90/180/270. This program starts by mapping the image in memory, as a memory mapped file, and then sends a rotation request to the server program through a named-pipe in the file-system. Beware that this named-pipe must be available to all users in the system, thus it probably has to be created by the system root user and have the necessary permissions to allow any user to communicate through it. The rotation request must provide information for the server program to access the memory mapping of the file and perform the rotation. After sending the request, the client process pauses until the server sends it a SIGUSR1 signal. The reception of this signal is a warning that the file has been rotated and that the client can eliminate the memory mapping and terminate. Make sure the client does a clean shutdown and prints the result of the operation (SUCCES/FAILURE) in the screen.
2. **Server** Process: The server process has to be started with root permissions. This is necessary because the process has to create a named-pipe in the file-system, for any user to access, and has to be able to manipulate any image in the file-system. The server process creates 3 other child processes that will contain the rotation worker threads. Each child process is able to perform one type of rotation only, thus we need three processes in order to provide three different rotation levels. After creating its child process and pipes, the server process blocks while waiting for new work. The server program itself is responsible for receiving the rotation request and forwarding them to the right child process. The server process sends this information to the child process through an unnamed pipe. The child process, upon completing its task, informs the server that it has rotated the image through another unnamed pipe. The Server then sends a SIGUSR1 signal to the specific client process to inform that the operation requested has been terminated. Since the server has to listen to the named-pipe and all the unnamed pipes coming from the child processes, you will need to use the `select()` function to multiplex communication. The server process and their child must be terminated with a CTRL+C (SIGINT).
3. **Child** processes (worker processes): The child processes are multithreaded. Each process has 5 threads: main thread and 4 worker threads. Both main and worker threads must be blocked while new requests don’t arrive. The main thread is responsible for listening to the unnamed pipe that links it to the server process and forwarding the received assignments to the worker threads. The communication between the main thread and the worker threads is done through an in-memory buffer using a producer-consumer algorithm. Make sure that the main thread always has priority when accessing the buffer, i.e. if the main thread needs to write to the buffer, any thread trying to consume in the same moment gets turned down (except if the buffer is full). If the main thread is writing, there can be threads reading from the buffer but new reads cannot start before the write ends. When worker threads finish the rotation job, they notify the server process through a second unnamed pipe and block until new job arrives.

Note: if there are any errors during the communication between processes or the manipulation of images, the clients must receive a SIGUR2 signal that indicates the failure of the operation. All processes must ignore signals that are not used in the system.

Checklist

shared memory

server.c

client.c

Application	Task	Amount of work
Server process	Use `fork()` to create child processes	5%
	Create unnamed pipes	4%
	Create named pipe	3%
	Perform communication between server and child processes	10%
	Receive requests from clients	5%
	Use `select()` to manage I/O	10%
	Send signal to clients indicating the success or failure of their requests.	2%
	Handle the expected signals and ignore unexpected signals	5%
	Provide good error handling code	5%
	Perform a clean shutdown	2%
Child Processes	Receive information from the unnamed pipe	5%
	Create the worker threads	5%
	Correctly implement the synchronization of the buffer communication	10%
	Write in the unnamed pipe the result of the operation	5%
	Correctly manipulate the image and image mapping	5%
	Perform a clean shutdown using signals and ignore the unexpected signals	2%
Client Process	Send requests through the named pipe	5%
	Handle signals received from the server and inform the user of the results of the requests	5%
	Perform the image mapping/un-mapping in memory correctly	5%
	Perform a clean shutdown using signals and ignore the unexpected signals	2%

Notes

- **Plagiarism or any other kind of fraud will not be tolerated**. Attempts of fraud will result in a ZERO and consequent failure in the OS course.
- Do not start coding right away. Take enough time to think about the problem and to structure your design;
- Implement the necessary code for **error detection and correction**;
- **Avoid busy-waiting** in your code!!
- Assure a **clean shutdown** of your system. Release all the used resources;
- Use a `Makefile` for simplifying the compilation process;
- Be sure to have some debug information that can help us to see what is happening.

Example:

```
#define DEBUG  //remove this line to remove debug messages
(...)
#ifdef DEBUG
  printf("Creating shared memory\n");
#endif
```

Submission and defenses

1. **Midterm presentation**
    a. During the **week starting on November 25th**, students have to present their work in the PL classes. You must prepare a **DEMONSTRATION** of what you have accomplished until this moment, and it should roughly represent 25% of the complete project.
2. **Final submission**
    a. The names and student ids of the elements of the group must be placed at the top of the **source code file(s)**. Include also the time spent in this assignment (sum of the time spent by the two elements of the group).
    b. Any external libraries must be provided with the source code file, including the usage instructions. Use a **ZIP** file to archive multiple files.
    c. You must provide a short **REPORT** (maximum 2 A4 pages) explaining the design and implementation of your solution and justifying the design options that you have made.
    d. The project should be submitted to InforEstudante.
    e. The submission deadline is **December 13th, 2013 (23h55)**
3. **Defense**
    a. **Functional group defenses** are scheduled to the PL classes of the **week starting on December 16th**
    b. **Written individual** defense is scheduled to **December 18th, 2013 (14h00)**