Operating Systems: Project #3

Introduction

This assignment will focus in the following topics:

- Concurrent programming with threads, using the `<pthread.h>` library.
- Use synchronization primitives such as semaphores and condition variables

Objectives

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

- Create and manage multiple threads inside a process.
- Use the library `<semaphore.h>` to create and use semaphores, *mutex* and condition variables to synchronize the access to shared resources between competing threads.

Support Material

- K. A. Robbins, S. Robbins, "Unix Systems Programming: Communication, Concurrency, and Threads", Prentice Hall:
  - Chapter 12 – "POSIX Threads"
  - Chapter 13 – "Thread Synchronization"
  - Chapter 14 – "Critical Sections"
- "Programming in C and Unix", available at http://gd.tuwien.ac.at/languages/c/programming-dmarshall/:
  - Threads: Basic Theory and Libraries
  - Processes and Threads
  - The POSIX Threads Library: `libpthread`, `<pthread.h>`
  - Further Threads Programming: Synchronization
  - Threads and Semaphores
- Online Unix manual
  - Man pages of `sem_post`, `sem_wait`, `sem_open`, `sem_init`.
  - Man pages of `pthread_create`, `pthread_exit`, `pthread_sigmask`, `sigwait`.
- Course materials:
  - Notes on Unix basic commands
  - Notes on reading and writing to files
  - Notes on `Makefile`

Project Assignment

Overview

In this project we will be revisiting an example that was presented and discussed in the beginning of the semester, the Virtual Printer. In this example, one process used the `select()` instruction to read from 3 different named pipes and print the data on screen. In this project, each part of the input should be formatted with a special header.

- `job <job_id> page <page_id>:` the content that follows belongs to page `<page_id>` of job `<job_id>`
- `job <job_id> end:` this header represents the end of the file to be printed, and is not followed by any more data. When this command is received, the printer has all content and can print the received data on screen.

Your program should work as described:

- **Use a multi-threaded approach (only one process, several threads);**
- The main thread on your program will read data from 3 named pipes, representing 3 virtual printers, and select a new thread (from a pool of worker threads) to assemble the pieces of data for each job as they arrive;
- Each worker thread will only handle data for one job. In the event of multiple concurrent jobs being received in several virtual printers or on the same virtual printer, one thread must be assigned to each job;
- Each virtual printer can receive different pieces of different jobs (beware of the problem of non-atomic reads/writes on pipes). Each piece must carry the `<job_id>`. The main thread uses the `<job_id>` to forward data to the correct worker thread;
- The worker threads will collect the data of each document in memory and, upon reception of the `job <job_id> end` command, they will correctly print the document on screen with each page clearly identified and numbered;
- Assume that you can have several worker threads active and waiting for data at the same time. Each thread will only get the data for its job and will completely ignore data destined to other jobs/threads. There should not be busy-waiting.

Testing your program

In order to test if your program is working correctly, you must write another program that will send multiple jobs concurrently to the available virtual printers. This program will also be evaluated and will have impact on your grade. You should impose as much “stress” as possible in your virtual printer driver program.

Quality requirments

Your application must make use of:

(a) Multi-threaded process;

(b) POSIX synchronization primitives (semaphores, mutex and condition variables);

(c) Solutions using Condition Variables will get higher scores;

(d) Use the `select()` instruction;

(e) Provide a test application that conveniently verifies the required functionality.

To fulfil the objectives of this assignment you should be as creative as you want, provided you have included this list of features in your solution.

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 and stop all processes before exiting the Master process;
- Always release the OS resources that are not being used by your application;
- 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 instructions

1. 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).
2. Any external libraries must be provided with the source code file, including the usage instructions. Use a ZIP file to archive multiple files.
3. You must provide a short REPORT explaining the design and implementation of your solution and justifying the design options that you have made.
4. The project should be submitted at http://moodle.dei.uc.pt.
5. The submission deadline is December 6, 2011.