Fundamentos de Sistemas de Operação

MIEI 2013/2014

Laboratory session 5

Overview

Pthreads: condition variables

Examples of the use of Pthread´s condition variables

Please see chapters 26 and 29 of the OSTEP book. There are also several tutorials on *POSIX threads* available namely this one: *https://computing.llnl.gov/tutorials/pthreads/*

Try the following example adapted from the book *Pthreads Programming*, B. Nichols et al., O´Reilly. In the program *condvar.c* (source file available in the same CLIP page) the main routine creates three threads. Two of the threads perform work and increment a *count* variable. The third thread waits until the *count* variable reaches a specified value (using a *condition*) and then adds 125 to the *count* variable. The other threads will continue incrementing *count*.
Compile the program (*gcc –o condvar condvar.c –lpthread*) and observe its behaviour.

Programming Assignment 1

Overview

Printing with threads: 1st part of programming assignment 1.

Programming assignment 1 (PA1) will be developed in the classes of the weeks starting 14th October and 21st October – The deadline for delivering PA1 (report and code) is October, 30th – 9am. Please see the instructions for delivering the code in the guide of next week.

Printing with threads (first part)

The main objective of this assignment is to build a Linux program that simulates a printing system allowing the user to submit several files for printing without having to wait for printing conclusion. The program also allows the user to obtain a list of files waiting for printing.

Step 1: Simulating a printer

Usually printers are connected to a computer using some peripheral device interface (like parallel or USB ports). Those devices are presented by the OS as special files, which can be read or written by the programs that need to use those devices. A program with the correct permissions can use a printer by writing the data to the device file representing the connection to that printer. For our assignment we are going to simulate the printer using a virtual device, in our case, the device representing a terminal.

Use the `tty' command at a terminal window and take note of its device name (like /dev/pts/1). From another terminal write to the first one by using the name you obtained. You can do that, for example, by using the echo command and redirecting the output to the device file (like */dev/pts/1*):

echo Hello > /dev/pts/1

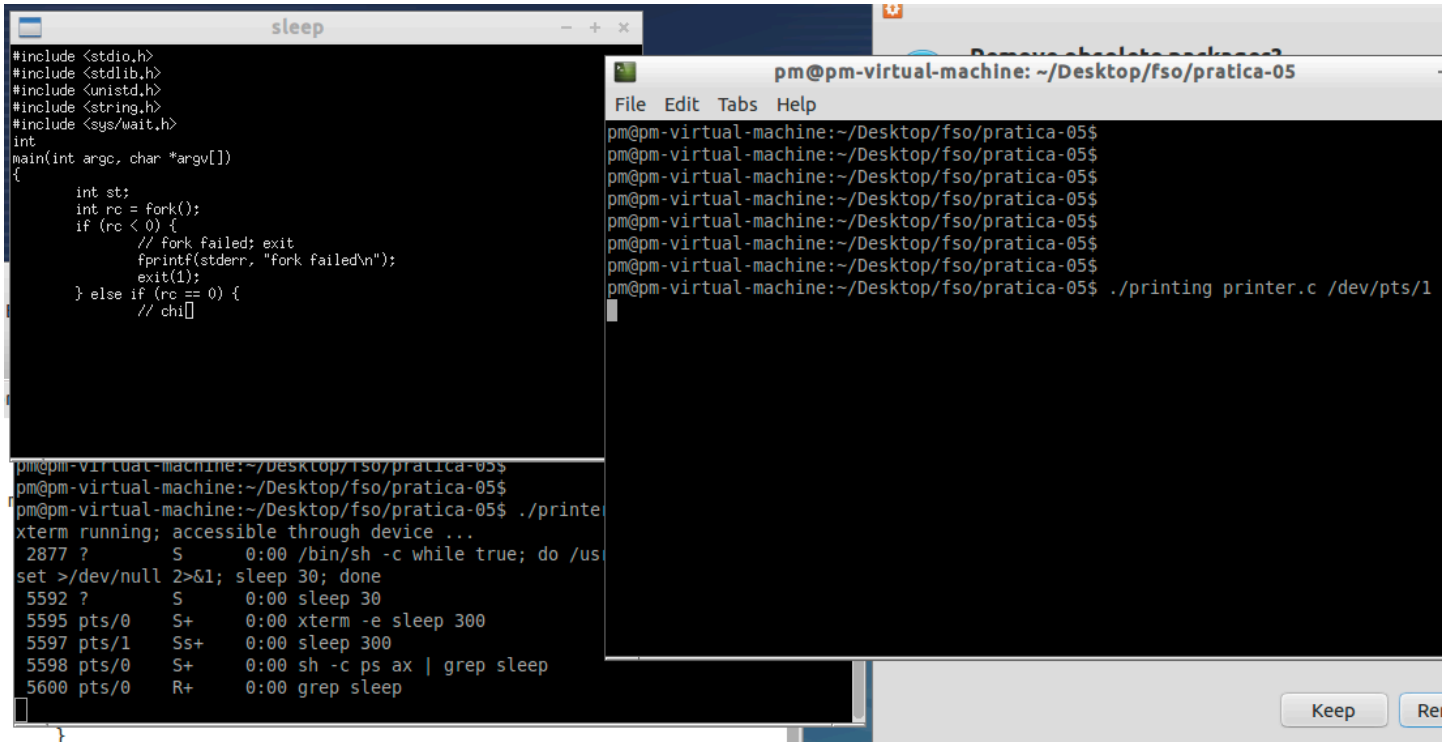
Repeat several times and see that "Hello" appears on the first terminal.

Consider now the following program (printing.c):

```
#include <sys/time.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
```

```c
#define BUFSIZE 1024
char buf[BUFSIZE];

void realPrint( int ch, char *c){
  write( ch, c, 1);
  usleep( 50000 ); /* simulating a slow device ... */
}

int main(int argc, char *argv[ ]) {
      int pw, xt, nr, i;
      if( argc!= 3){
          printf("Usage: %s <file> <pseudo terminal>\n", argv[0]);
          exit(1);
      }
      pw = open( argv[1], O_RDONLY);
      if( pw < 0){
          printf("Could not open file to print %s\n", argv[1]);
          exit(2);
      }

      xt= open( argv[2], O_WRONLY);
      if( xt < 0){
          printf("Could not open pseudo terminal %s\n", argv[2]);
          exit(3);
      }

      do{
        nr = read( pw, buf, BUFSIZE);
        for (i=0; i <nr; i++)
                realPrint(xt, buf+i);
      }while (nr == BUFSIZE);

      close(pw);
      close(xt);
      return 0;
}
```

What the program does can be easily guessed looking at the code and observing the following screenshot, where a file named printer.c is being printed from the right terminal to the left one:



Reproduce the situation above. Verify that printing is slow, simulating a command that sends a file directly to a printer. Try using this command from a third terminal to print to the “same printer” and confirm that the output will be jumbled.

Step 2: Using pthreads

For this version, we will write a multi-threaded program *mtPrint*, using Pthreads. This program should receive commands from the user, like for printing several files, at the same time that is printing one file. *mtPrint* will be invoked with a parameter indicating the pseudo terminal used for printing. An example would be:

./mtPrint /dev/pts/2

After initialization, *mtPrint* will enter a loop attending user commands, where:

- Writes its prompt and gets a user command
- Recognizes the following three commands:
  - *print filename* - will launch a thread for printing the given file. That new thread will wait for exclusive access to the "printer" and then execute something similar to what is in file *printing.c.* The main thread, after launching the printing thread will loop writing the prompt and receiving more user commands, while a file is being printed.
  - *status* – will reply BUSY if the printer is busy and FREE if no printing is underway
  - *quit* – terminate the program (all threads)

Assume that only one mtPrint process will execute in your computer, and that all printing will be done through it.