CS4540 - Operating Systems

Assignment 5. Printer Spooling Simulation

1. INTRODUCTION: SPOOLING

In this assignment you will create a printer spool simulation using POSIX threads, which are synchronized by POSIX mutexes or semaphores.

In computer science, the term *spooling* refers to transferring data by a program and placing it in a *buffer* (a temporary working area), where another program may access it for processing at a later point in time. Spooling occurs in parallel with other work.

Spooling is useful because devices access data at different rates. The buffer provides a waiting station where data can rest while the slower device catches up [1]. Data stored in the buffer is modified only through addition or deletion at the ends of the buffer, i.e., there is no random access or editing of data in the buffer [1].

Spooling is used in printing. Documents formatted for printing are stored in a buffer (usually an area on a disk) by a fast process, and retrieved and printed by a slower *printer spooler* at its own rate. As soon as the fast process wrote a document into the buffer, it has finished its printing job.

Using spooling, one or more processes may rapidly write several documents into a print buffer without waiting for each document to print before writing the next one into the buffer. The printer spooler may allow priorities to be assigned to print jobs, notify users when the printing is finished, distribute printing jobs among several printers, allow stationery to be changed or select it automatically, generate banner pages to identify and separate print jobs, etc. [1].

REFERENCES

[1] http://en.wikipedia.org/wiki/Spooling

2. REQUIREMENTS

2.1. Design Requirements and Implementation Hints (70 Points)

You will simulate printer spooling. The following list specifies the requirements:

1) The **size** of the **printer buffer** is 2048 KB. (The buffer size must be defined as a macro identifier at the beginning of your code: `#define BUFSIZE 2048`).
2) **Simulate spooler** as a thread. It accepts incoming print jobs as long as there is a free space in its buffer. Initially, the printer buffer is empty.
3) Assume that the **printer speed** is 100 KB/sec.
4) In addition to the spooler thread, you must have several other **threads requesting** for **printing** their jobs. Assume that each *printing thread* has exactly one print job.

   The **number of printing threads** must be given as a command line argument.
5) The **sizes** of the requested **print jobs** should vary (a uniform random distribution) from 50 KB to 1000 KB. (50 KB must be defined in your program as the minimum size, and 1000 as the maximum size.)
6) A thread can put its print job into the buffer only if there is **enough space in the buffer**; otherwise, it sleeps until enough space becomes available. In more detail, if the currently used buffer space plus the size of the print job is bigger than `BUFSIZE`, the thread has to sleep till a sufficient available space becomes available.

   As an example, if the available buffer space is 500 KB, a job that needs 400 KB can be placed in the buffer; however, a job that needs 600 KB has to wait. More specifically, the job that needs 600 KB can be buffered only after the printer completes printing one or more jobs that together use at least 100 KB of the buffer space.
   (You may use the following code snippet to keep a thread asleep but waking up every second to check if a sufficient buffer space became available:

```
while((bufferUsage + size) > BUFFSIZE)
```

```
        {
            sleep(1);
        }
```

`bufferUsage` refers to the buffer space currently occupied (by all jobs placed in the buffer).

7) The printing requests and buffer allocation by the spooler must be correctly **synchronized** with semaphores or mutexes.

8) The **output** must show all relevant events properly. This must include (but is not limited to) the events listed explicitly below.

   The following message types are required, each *starting with a timestamp* (showing the current clock time):

   a) Print a message when a *job arrives*.

   b) Print a message when a *job* is *buffered*, or when it is *denied buffering* due to insufficient buffer space.

   c) Print a message when *job printing starts*.

   d) Print a message when *job printing is finished*.

   Your output must be *clear*. Be *creative* in its implementation (e.g., providing more information, improving messages, etc.). New ideas are always welcome—as long as they are good. ☺

   Output Example: An example *fragment* of an output is given below.

```
dursunoglu@dursunoglu:~/$ ./printSim
Usage:
./printSim MAXNUMBEROFTHREADS
dursunoglu@dursunoglu:~/$ ./printSim 3
Mon Apr 13 16:20:01 2015
Simulation has started...
Free buffer size 2048 Kb.
----------------------------
Printer has started...
Mon Apr 13 16:20:01 2015 - Job 0 of size 926 KB arrived.
Mon Apr 13 16:20:01 2015 - Job 0 stored in buffer.
Mon Apr 13 16:20:01 2015 - Job 0 starts printing.
Mon Apr 13 16:20:01 2015 - Job 1 of size 975 KB arrived.
Mon Apr 13 16:20:01 2015 - Job 1 stored in buffer.
Mon Apr 13 16:20:01 2015 - Job 2 of size 917 KB arrived.
Mon Apr 13 16:20:01 2015 - Insufficient buffer space for Job 2.
Mon Apr 13 16:20:10 2015 - Job 0 completed printing.
Mon Apr 13 16:20:10 2015 - Job 1 starts printing.
Mon Apr 13 16:20:11 2015 - Job 2 stored in buffer.
Mon Apr 13 16:20:20 2015 - Job 1 completed printing.
Mon Apr 13 16:20:20 2015 - Job 2 starts printing.
Mon Apr 13 16:20:29 2015 - Job 2 completed printing.
```

9) **<u>Optional (15% extra credit):</u>** Run this simulation for different, gradually increasing buffer sizes (e.g., 1024KB, 2048KB, 3072KB, etc.). By studying the outputs for different sizes, you might understand better the simulation principles as well as the activities being controlled by the threads.

2.2. Report (30 points)

Write a report that describes the design, implementation (include one or two top-level pseudo-code refinements), and presents and discusses the results produced by your program.

2.3. Other Requirements

Remember about the proper programming style and submission guidelines (following both of them will be graded), including comments, blank lines, indentations, spaces, etc. Also, provide a *makefile* and, if needed README file explaining how to compile and run your program.

Deliver a file with your program output (produced with the *script* command).

Submit your assignment via eLearning.