

Model of a Print Queue

Michael Huynh

AP Computer Science, Upper Darby High School

March 2004

Programming Problem Statement: To develop a model of a print queue using the Java programming language.

1. BACKGROUND DISCUSSION

Network load balancing is crucial to the seamless operation of a high traffic network. Corporations such as Google or Mastercard depend on efficient routing of network data to hundreds of servers to achieve the fastest search or transaction possible. It is no surprise that in the field of computer science, a good deal of research is dedicated to network load balancing and routing. The cutting edge of research models the way biological systems communicate. From the way brain synapses communicate (neural networks) to the collective yet individually independent nature of bee swarms (multi-agent systems), research in these areas have all been modeled and applied to network optimization^{1,2}.

On a smaller scale, a busy office building also faces challenges in networking. For an office with many printers, the question is: "How can I efficiently use these printers to balance the incoming print jobs?" Current physical solutions include utilizing a print server as the middleman to handle jobs and route them to the printer with the least amount of load³. In order to further examine a print server and connected printers, it is necessary to model a solution with software.

2. PROGRAMMING PROBLEM SOLUTION

The modeling of print servers and printers must start with the implementation of what is common to both: a print queue. This print queue object is very important as it forms the basis of how jobs are received and executed on print servers and printers. From the completion of a print queue object, the creation of print servers and printers need only to inherit the print queue class and add additional functionality such as load distributing and print speed.

The print queue was chosen to be modeled in the Java programming language because of the highly object oriented nature of the language. Since every program written in Java is treated as an object, programmers are forced to think and treat

all solutions as objects. This makes Java the ideal language for modeling real-life "objects."

The implementation of a print queue is fairly straightforward. As the name suggests, a print *queue* would require some sort of data structure that mimics a queue. A queue is an ordered collection of items in which new items are added at one end, the back, and removed at the other end, the front⁴. A queue is a first in, first out (FIFO) data structure. A very similar data structure is a stack or an ordered collection of items where items can be added and removed only at one end⁴. A stack is a last in, first out (LIFO) data structure. A stack is not an ideal way to model a print queue since the last added job would be the first one completed. In contrast, a queue models a print queue as the last added job is the last completed. A queue is much like a line of people waiting to buy tickets. Anyone who wants to buy a ticket must wait in the back of the line. If the line was like a stack, a person wanting to buy a ticket can just cut in the front, get the ticket, and leave.

A queue is based on a linked list which is a dynamic data structure that stores data in nodes with pointers to the next node, creating a "list." While a queue can be implemented using an array, it would not be the best solution since removing elements from the front requires $O(n)$ for an array while only $O(1)$ for a linked list⁴. Fortunately, Java includes a `LinkedList` class which was used to program the queue⁵.

The actual program model of a print queue is basic. For each program cycle, there is a possibility of generating a new print job and if so, the job is added to the print queue. Next, the front job in the print queue is executed. Each job has a time cycle assigned to it in order to model print times of real life print jobs. In this model, print cycles from 1 to 5 are generated. A 1 represents a simple text file that can be quickly printed. A 5 represents a graphic intense job that requires much more time to complete. Everything in between constitutes a mix between text and graphics. The execution of a print job simply means decrementing its print time cycle by 1. When the time cycle reaches 0, the job is

completed and removed from the queue. For each print cycle run, a job may only be decremented once. Consequently, the default 50% job generation rate will eventually create jobs faster than the printer can complete them. Therefore, a 15 print job limit was set on the printer so that the wait time would not be extremely high. This is where a printer server would be very important in routing future print jobs to another printer.

3. PSEUDOCODE

```
do {
    //50% prob. of generating
    //a print job.
    if(GeneratePrintJob())
    {
        //If job is generated, add
        //to print queue.
        EnqueuePrintJob();
    }
    //Process jobs.
    ProcessPrintQueue();
}while(true);

ProcessPrintQueue() {
    //Get the job first in line
    currentjob=DequeueJob();
    //If the job is not completed,
    //"process" the job by
    //decrementing.
    if(currentjob.time>0)
    {
        currentjob.time--;
    }
    else
    {
        //If currentjob time is 0,
        //the job is finished and
        //removed from the queue.
        RemoveJob(currentjob);
    }
}
```

4. CODE

Actual program code can be found attached to this document.

5. VALIDATION

Validation of the program was conducted through visual inspection. Since the print queue was fairly simple in operation, it was easy to determine that the printer model ran through its job cycles successfully: generating print jobs, decrementing each job by one each time, and refusing to take on new jobs if the queue was full.

5. SUMMARY

This implementation a print queue was an extremely basic look at computer modeling of real life problems using queues and linked lists. Since the print queue is a common feature in both print servers, the next step would be to develop a print server that inherits the print queue. The print server can then be simulated and modified for different scenarios and optimization.

REFERENCES

1. Matthews, James. "Introduction to Neutral Networks." Generation5. 14 March 2004. <<http://www.generation5.org/content/2000/nnintro.asp>>.
2. "What is Bee-Gent?" Toshiba. 14 March 2004. <<http://www2.toshiba.co.jp/beegent/whatsbge.htm>>.
3. "Network Printing Basics." FreeBSD. 14 March 2004. <http://www.freebsd.org/doc/en_US.ISO8859-1/books/corp-net-guide/printerving-network.html>.
4. Teukolsky, Roselyn. How to Prepare for the AP Computer Science Advanced Placement Examination. Barron's Educational Series, 2001.
5. "Java Technology." Sun Microsystems. 25 January 2004. <<http://java.sun.com>>.