

Queuing Theory and Application
By: Joshua Jorenby

Submitted to: Dr. Taylor
Date Submitted: February 3, 2003
Class: CS-384-4

Abstract

The purpose of this paper is to give an overview of queuing theory and application. First, the basics of queuing theory are discussed, including basic concepts and a means to specify queues called Kendall notation. Next, general queuing algorithms are explored and specific concepts are touch upon. Finally, queuing theory is discussed briefly as it relates to computer science. To illustrate certain algorithms and concepts, examples are used that will hopefully be both informative and entertaining.

Table of Contents

Abstract	2
Table of Contents	3
Basics of Queuing Theory.....	5
Components.....	5
Concepts.....	5
Kendall Notation	6
Queuing Algorithms.....	8
Arrival Time Based.....	8
First-Come, First-Serve.....	8
Last-Come, First-Serve	10
Service-In-Random-Order.....	10
Service Time Based.....	11
Shortest-Job-First	11
Longest-Job-First	12
Round-Robin.....	12
Priority Based.....	13
Priority Queuing Concepts.....	14
Preemption	14
Starvation	14
Deadlocking	15

Applications in Computer Science.....	16
Conclusion.....	17
Bibliography.....	18

Basics of Queuing Theory

Components

A basic queuing system is comprised of two parts, servers and customers. A server is a producer process that handles customers. A customer is some unit that wishes to be processed by the server. In any given system, there may be one or more customer and one or more server. When multiple customers are waiting for a turn with a server, a queue forms [Laplante, 248].

Examples of the components of queuing systems can be found often in ordinary life. Drivers waiting in line at a carwash form a single-server queue where the cars are the customers and the carwash is the server. Persons wishing to make a deposit at a bank with several tellers may create a multiple-server queue.

Concepts

Flow is the major idea in queuing systems, as in the way that customers move through servers. The rate of flow of a queue can be steady or unsteady. Steady flow systems proceed predictably and are relatively easy to analyze. The maintainers of such a system simply implement the design in a way that ensures the interarrival time of the customers and the service time of the servers coincide. A factory assembly line is a perfect single-server example of this type of system. The given product moves down the line at a consistent speed from worker to worker and output can be maximized. With a network of servers, the system becomes more complicated to analyze, however, the possibility of maximization still exists [Panico, 12].

Unsteady flow systems (which are also called random or stochastic) are more difficult to explore because the arrival rate of customers is unpredictable. Unfortunately, most real-world queuing systems fall under this variety. In a barroom, there may be sitting one or more patrons and one or more patrons may arrive at any time. Due to this random nature, the barkeep may be very busy one moment and bored the next. Since there is no completely accurate way to determine the demands that will be placed on the servers, the bar manager must try to schedule help based on the mean customer arrival rate. Most of the study of queuing systems is in this area.

Kendall Notation

Those who work with queuing systems use a shorthand notation to identify and describe the systems called Kendall notation. Strangely enough, this is named after the statistician David Kendall who devised the notation.

Kendall notation has the form $A/B/c/K/m/Z$ where A is the interarrival time distribution, B is the service time distribution, c is the number of servers, K is the largest possible number of customers in the system, m is the number of customers in the source, and Z is the method by which the queue is serviced. In most common, real-world systems this notation can be shortened to $A/B/c$ where K and m are assumed to be infinite and Z is assumed to be first-come, first-serve (an algorithm that we will discuss later).

The common symbols representing time distributions for A and B include

- D deterministic (or constant) time distribution;
- M exponential time distribution, which corresponds with random arrivals;
- G general service time distribution;
- GI general independent interarrival time distribution.

An example of a queuing system using full Kendall notation could be an $M/G/10/30/\infty/FCFS$ queuing system that has exponential interarrival time, general service time, 10 servers, a system capacity of 30 customers (10 of which are being served), an infinite customer source, and first-come, first-serve queue type [Allen, 157].

Queuing Algorithms

A queuing algorithm is a method by which a queue is processed. Many different algorithms are used to attempt to maximize the productivity of the system.

Arrival Time Based

The first group of algorithms we will be discussing are arrival time based. These algorithms are relatively intuitive and therefore easy to understand and implement.

Customers enter the queue and are serviced according to the time they enter. This is the most common that way real-world queues are handled.

First-Come, First-Serve

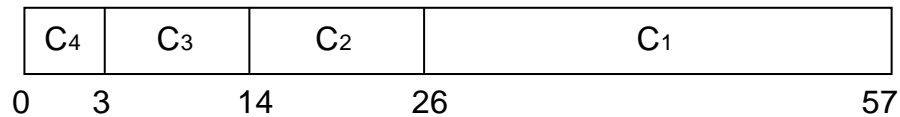
Probably the most frequently used queuing algorithm in the world is first-come, first-serve (FCFS). This method simply means that as customers come in to the queue system, the first customer is uses the server first and does not give it up until the customer is finished. Then the next customer gets a chance with the server, leaves when finished, and the process continues until the queue is empty.

This algorithm can be seen in most of the queuing systems mentioned above since most involved lines of waiting people, however, there does not necessarily need to be a lineup to use the method. Some shops employ service numbers in which the customer picks the number, goes someplace else, and waits for the customers with earlier numbers to be waited on before that customer's number is called (this same sort of approach used in hospital waiting rooms). This is brought up now because below computer scheduling is discussed where similar algorithmic approaches are utilized, but there is certainly no physical line making the queue.

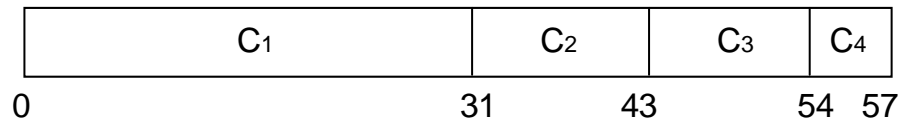
The FCFS algorithm is used so extensively because it is simply and people perceive it as very fair, on the downside, though, this algorithm can be very poor at minimizing the wait time of individual customers. Take, for instance, the following four-customer queue:

<u>Customer</u>	<u>Amount of service-time needed</u>
C ₁	31
C ₂	12
C ₃	11
C ₄	3

If the order of customers is starting with C₄ and ending with C₁ then the wait times are quite tolerable, as shown by this *Gantt chart*:



Here the queue is served from left to right and the numbers at the bottom are the wait times (in seconds) for each customer. The average waiting time for this ordering is $(0+3+14+26)/4 = 10.75$ seconds. On the other hand, if the queue is C₁ through C₄ then the waiting times are considerably longer:



This ordering gives an average waiting time of $(0+31+43+54)/4 = 32$ seconds. So the point is that one especially slow customer can stall the queue process considerably, thus FCFS is not always be the best option [Silberschatz 157].

Last-Come, First-Serve

The last-come, first-serve (LCFS) queuing algorithm is basically the same as FCFS, the difference being that after the server finishes with a customer the most recent arrival to the queue is processed next. This means that new (and possibly more important) entries are handled with very little wait. The obvious shortcoming, however, is that a customer that has been on the queue for a while may never get a chance with a server if new customers keep coming in.

An interesting example of this sort of method in the real world is expressed in the description of a person with a certain personality type. Here the author discusses the difficulty he has in keeping the attention of a coworker,

“One [Extroverted Intuitive Feeling Judging] colleague always welcomes me into his office regardless of his own circumstances. If another person comes to the door, he allows them to interrupt our conversation with their need. While discussing that need, the phone rings and he stops to answer it. Others drop in with a 'quick question.' I finally get up, go to my office and use the call waiting feature on the telephone. When he hangs up, I have his undivided attention!”
[Butt]

The system used in this case is using preemption, a concept we will cover later.

Service-In-Random-Order

The service-in-random-order queuing algorithm is again similar to FCFS and LCFS, though selection of the next customer to process is at random. This system has all the benefits and detriments of the other arrival time based algorithms, and attempts to split the difference. Sometimes a recent addition to the customer queue will be selected for a server and other times an older member of the queue will be chosen.

Service Time Based

One of the biggest problems with arrival time based queuing algorithms is that waiting times for customers may or may not be very long. To attempt to alleviate this problem a different approach to queue processes may be used that is service time based.

Shortest-Job-First

The shortest-job-first (SJF) queuing algorithm is used to get around the main obstacle in FCFS, namely that waiting times can be poor because of the order of the queue. The SJF method basically reorders the queue so that the customers that can be served most quickly will be first in line so average waiting times will be minimized, just as the first ordering of the FCFS example showed.

The difficulty with the SJF algorithm is having a prior knowledge of how long it will take any given customer to process in a server. One solution to this is to have the customer notify the system as to the length of time it will need for processing. This method causes the customer to want to be as accurate as possible for processing time because overestimating will result in more customers getting a place ahead in the queue, while underestimating will cause the server to stop processing prematurely so the customer will have to reenter the queue and wait even longer. Another solution is to predict the processing time for each customer based on previous ones. The simple formula

$$t_{next} = \alpha t_{last} + (1 - \alpha) t_{past},$$

where t_{next} is the predicted time for the next customer, t_{last} is the time of the previous customer, t_{past} is the historical time for customers, and α is ratio value between zero and

one that is in charge of the influence of t_{last} and t_{past} in predicting the current customers processing time. This latter method gives adequate, if relatively inaccurate, time forecasts that lead to better efficiency than arrival time based algorithms [Silberschatz, 159].

Longest-Job-First

The counterpart to SJF is the longest-job-first (LJF) queuing algorithm. Here the queue is reordered so that customers that will take the longest amount of time to process in a server go first. LJF has the same difficulty predicting processing time as SJF and can create very lengthy waiting times, so it is not used nearly as often.

A somewhat trivial example of the LJF method at work is what a student may do who has several homework assignments that are due soon. A hard working student will likely complete the more difficult, time-consuming assignments first so the easier one can be done at their leisure. Conversely, a less hard working student may work on the easier assignments first, procrastinating on the other ones. This student would be using the SJF method.

Round-Robin

Round-robin (RR) is another common queuing algorithm, especially in computer systems. It works in fundamentally the same way as FCFS, but each customer is only given a constant, limited time quantum with the server and if that customer is not finished being processed then it is sent back to the queue to await more time. This ensures that each customer will be processed even, but customers that have long processing times will take a very long time to complete. Another problem is the amount of overhead associated

with a customer entering a server will be multiplied by the number of times the customer needs to begin processing again with a server.

A real-world example of this queuing behavior could be the way a line of fans is processed at an autograph booth. A FCFS queue is formed and each fan in turn gets a certain allotted time with the famous person. If, for some reason, one of the fans needs more time to get autographs they must start again at the back of the line.

Priority Based

Priority based queuing algorithms use ideas from many of the previously mentioned algorithms to attempt to maximize the efficiency of a complex system. Each customer is given a numerical priority and they are processed according to that priority. This approach adds flexibility by allowing different factors to play a larger or smaller role in the assignment of priority depending upon the needs of the system. Possible factors in the allocation of priority include processing and arrival time, though other factors play a large role in algorithms used in computer processing [Levi, 151].

A hospital emergency room employs a sort of priority queuing algorithm to take care of patients. Here the doctors are the servers and the patients are the customers. Normal operation is done using the FCFS method, however the severity of individual customers' cases also play a role in who is processed first. A particularly serious case may result in a doctor being called away from another patient. This is called preemption, which we will discuss shortly.

Priority Queuing Concepts

Algorithms that use priority and other methods to process queues can bring on certain pitfalls of which the manager of such a system must be aware and have means to avoid and overcome such hazards.

Preemption

A method utilized by the RR queuing algorithm and sometimes priority or service time based queuing algorithms is preemption. Preemption is when a server stops processing the current customer, sends it back to the queue, and begins processing a different customer. This is used to stop customers with long service times from keeping the server busy indefinitely and can also guarantee that important customers get immediate attention. If used properly, preemption can improve efficiency [Schwan, 737]. In real-world queues preemption is generally frowned upon, especially by the currently processing customer who gets snubbed, except in special cases like the emergency room example above.

Starvation

Priority and service time based queuing methods have the possibility of keeping certain customers at the end of the queue, thus never getting processed. This predicament is called starvation and it occurs when new, high-priority customers continually enter the queue. These customers dominate the servers and force their lower-priority counterparts back. When preemption is used by the system this becomes particularly problematic since even customers that make it to a server may be pushed out and possibly neglected further.

The most common approach to circumvent starvation and ensure well-fed customers is incrementally increasing the priority customers based on the length of time they have spent in the queue. This way even an originally low-priority customer will eventually beat out high-priority newcomers and be processed by a server [Shilbershatz, 162].

Deadlocking

Another difficulty that comes up with priority based queuing algorithms is deadlocking. Deadlocking occurs when one customer needs another to finish processing in a server in order to free or create some resource that the first customer must have to process, but the first customer has higher priority. The result is the system freezes and nothing can enter the servers. Deadlocking is somewhat akin to what happens to a pilot named Orr in the book *Catch-22*:

“There was only one catch and that was Catch-22, which specified that a concern for one's own safety in the face of dangers that were real and immediate was the process of a rational mind. Orr was crazy and could be grounded. All he had to do was ask; and as soon as he did, he would no longer be crazy and would have to fly more missions. Orr would be crazy to fly more missions and sane if he didn't, but if he was sane he would have to fly them. If he flew them he was crazy and didn't have to; but if he didn't want to he was sane and had to [Heller, 55].”

To overcome problems caused by deadlocks system managers can use priority inversion, in which the lower-priority customer that is making the higher-priority customer wait has its priority temporarily boosted. This way the customers can be processed in an order that keeps the queuing system running smoothly [Laplante, 155].

Applications in Computer Science

Queues are employed extensively in many aspects of computer science. CPU task scheduling, network load, and printer job scheduling have all been improved and effectively realized by use of queue theory.

The main benefit of studying computer science through queue theory is that real-world results and algorithms can be mathematically analyzed. With advances in computer hardware more efficient queuing algorithms can make an even large mark in computer science. By improving the speed and applicability of queue processes computer scientist can maximize the potential of these new systems.

Conclusion

Hopefully this paper has been engaging and informative, since queue theory is an interesting topic. The study of queuing systems is ongoing because advancements in theory can have an immediate impact on real-world results.

Bibliography

Allen, Arnold O. Probability, Statistics, and Queuing Theory with Computer Science Application. New York: Academic Press, 1978.

Anderson, James, Kevin Jeffay, Arun Moorthy, and F. Donelson Smith. "Proportional Share Scheduling of Operating System Services for Real-Time Applications." University of North Carolina at Chapel Hill.

Heller, Joseph. *Catch-22*. New York: Simon & Schuster Inc., 1996.

Kleinrock, Leonard. Queueing Systems. 2 Vols. New York: Wiley, 1975-1976.

Laplante, Phillip A. Real-time systems design and analysis : an engineer's handbook. 2nd Ed. New York: Institute of Electrical and Electronics Engineers, Inc, 1997.

Levi, Shem-Tov and Ashok K. Agrawala. Real Time System Design. New York: McGraw-Hill Publishing, 1990.

Panico, Joseph A. Queueing Theory. New Jersey: Prentice-Hall, Inc., 1969.

Schwan, Karsten and Hongyi Zhou. "Dynamic Scheduling of Hard Real-Time Tasks and Real-Time Threads." *IEEE Transactions on Software Engineering*. Vol. 18, No. 8. August 1992. P. 736-748.

Silberschatz, Abraham, Peter B. Galvin, and Greg Gagne. Operating System Concepts. 6th Ed. New York: John Wiley & Sons, 2003.

Tanenbaum, Andrew S. Modern operating systems. New Jersey: Prentice-Hall, 2001.

White, J.A., J. W. Schmidt, and G. K. Bennett. Analysis of Queueing Systems. New York: Academic Press, 1975.