

**A SOLUTION TO MEAN DELAY IN THE $\sum M_c / G_{ck} / 1$ CYCLIC PRIORITY QUEUE
WITH CYCLE (k) AND CLASS (c) DEPENDENT FEEDBACK AND SERVICE TIMES**

Dragomir D. Dimitrijević

Consultant

Šajkaška 21, #40

11000 Belgrade

Serbia, Yugoslavia

Phone: +381-11-765-032

E-Mail: ddimitri@EUnet.yu

ABSTRACT

In this paper we develop an original solution to mean delay in a general $\sum M_c / G_{ck} / 1$ cyclic priority queue with class (c) and cycle (k) dependent service time and feedback. Each class has its priority assigned. There may be one or more classes with the same priority. This model is suitable for performance analysis of round robin processor sharing policies. The analysis may be used to examine the effects of quantum sizes in round robin scheduling used in operating systems. Each customer, upon entering the system, requires a number of service cycles before it leaves the system. Each service cycle is characterized by its service time distribution, and the probability of having at least one more cycle before the customer leaves the system. These characteristics of cycles may be different for different cycles of a service process. The solution is represented as a system of linear equations. It may be efficiently solved using the Gauss-Seidel iterative procedure. A general solution is developed of which, the two special cases are a non-priority (single-priority) and a one-class-per-priority non-preemptive queues. Computational complexity of the numerical procedure is between computational complexity of the two special cases, $O(C^3 K^3)$ and $O(CK^3)$ respectively. C stands for the number of classes, and K stands for the maximum number of cycles being numerically considered.

Keywords:

cyclic queues, queues with feedback, processor sharing, round robin scheduling, priority queues

1. INTRODUCTION

In this paper we consider a general $\sum M_c / G_{ck} / 1$ cyclic priority queue with cycle (k) and class (c) dependent service time and feedback. Each class has its priority assigned. There may be one or more classes with the same priority. The two special cases are non-priority (single-priority) and one-class-per-priority non-preemptive queues. Each customer, upon entering the system, must complete a number of service cycles before it leaves the system. Service times in each cycle have general distributions, which may be different for different cycles and classes. After completing one service cycle, a customer may enter the next cycle with a prescribed probability or leave the system. The feedback probabilities may be different for different cycles and classes. Such a queueing model is suitable for performance analysis of processors shared among multiple processes according to a prescribed round robin task scheduling policy of the operating system [9, 12].

Algorithms, which deal with this problem, simplify the problem by using one or more of the following assumptions. The service times in each cycle are equally distributed [1, 5]. The feedback is memoryless, i.e., the number of cycles has a Bernoulli distribution [5].

The Mean Value Analysis (MVA) [6, 7, 10] is a popular and numerically attractive approach. It implies perfect sharing of resources, e.g., processor. As we shall see, different duration of visits to a processor, while keeping total processing requirements constant, may significantly influence the performance.

A method which describes behavior of queueing system elements in terms of communicating finite state machines [3, 4] allows more precise modeling of different service times in different stages of service in various task scheduling policies. However, all service times are assumed to be exponentially distributed. This assumption allows construction of the state transition diagram of a Markov process and a more precise modeling of various scenarios in the service process. The method gradually builds the state transition diagram until a prescribed accuracy is achieved. The problem of state space explosion is overcome by pruning low probability states. This approach is used in [14] to evaluate performance of a network switch in which the processor applies various task-scheduling policies to perform different functions of the network switch.

The paper is organized as follows. A model of the $\sum M_c / G_{ck} / 1$ queue with multiple classes, and class and cycle dependent feedback and service times is described in Section 2. Unlike previous work, we allow that the service time distributions and the feedback probabilities vary from cycle to cycle. A theoretical solution is given in Section 3 in a form of a system of linear equations suitable for Gauss-Seidel iterative procedure. The convergence problem is also addressed. Numerical examples are given in Section 4 to illustrate the approach.

2. A MODEL

Let us consider a general $\sum M_c / G_{ck} / 1$ queue with feedback. There are C independent input streams of customers. New customers join the end of the queue. An arrival process of new customers in each class c is modeled as an independent Poisson arrival process. Each customer must complete a number of service cycles before it can leave the system. We assume that the time a class c customer spends in service in its k^{th} cycle has a general distribution. However, it does not depend on service times spent in previous cycles.

Let us introduce the following notation:

- λ_c is the Poisson process arrival rate of class c customers;
- S_{ck} is a random variable which represents the service time of a class c customer in its cycle k ;
- s_{ck} is the mean value of the corresponding service time, i.e., $E[S_{ck}]$;
- v_{ck} is the mean square of the corresponding service time, i.e., $E[S_{ck}^2]$;
- p_{ck} is the probability that a class c customer will need at least one more cycle after it finishes its cycle k ; The customer leaves the system with the probability equal to $1 - p_{ck}$;
- ω_c is the priority of class c ; A lower number means the higher priority;
- s_c is the mean total service time required by a class c customer;
- w_c is the mean total waiting time in the queue experienced by a class c customer;
- $d_c = s_c + w_c$ is the mean total delay (time spent in the system) experienced by a class c customer;
- w_{ck} is the mean waiting time experienced by a class c customer who enters its k^{th} cycle;
- n_{ck} is the mean number of class c customers who wait in the queue in their k^{th} cycle.

Two typical cases of feedback are:

- Bernoulli feedback when $p_{ck} = p_c$, i.e., the feedback probability does not depend of the cycle index;
- Constant number ($const_c$) of cycles when $p_{ck} = 1$ for $k < const_c$ and $p_{ck} = 0$ otherwise.

In this paper we consider a general case where more than one class may have the same priority. The two special cases are non-priority (single-priority) when $\omega_c = 1$, and single-class-per-priority non-preemptive queues when $\omega_c = c$. The first case is shown in Figure 1. All customers join the same queue. They are served in a First-Come-First-Served (FCFS) order. In the second case, customers of different classes join different queues, as shown in Figure 2. Classes with lower indices have higher priorities. A customer can begin a service cycle only if all queues with lower indices are empty. Customers with the same priority are served in a FCFS order. We assume a non-preemptive service, i.e., once a service in a service cycle begins, it cannot be interrupted by a high priority customer.

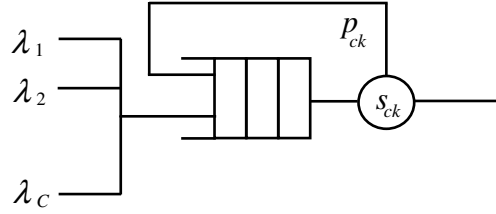


Figure 1: A Single-Priority Cyclic Queue with Multiple Classes

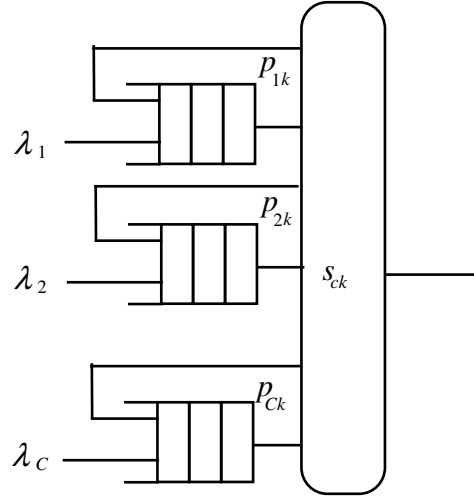


Figure 2: A Single-Class-Per-Priority Cyclic Multiple Priority Queue

Assuming that probabilities of staying in the system in different cycles are mutually independent, the probability that a class c customer will reach cycle k may be computed as

$$\pi_{ck} = \prod_{j=1}^{k-1} p_{cj} \quad (1)$$

By definition, $\pi_{c1} = 1$, i.e., each customer has to complete at least one cycle. Since each customer eventually leaves the system, it holds that

$$\lim_{k \rightarrow \infty} \pi_{ck} = 0 \quad (2)$$

The following hold:

$$s_c = \sum_{k=1}^{\infty} \pi_{ck} s_{ck} \quad (3)$$

$$w_c = \sum_{k=1}^{\infty} \pi_{ck} w_{ck} \quad (4)$$

Notice that, assuming sums (3) and (4) exist, series of their partial sums, $\sum_{k=1}^x$, increase and converge to their actual values. We may use this property to truncate calculations when the series of partial sums numerically converge. Certainly, artificial examples may be constructed where a partial sum appears to be converging while exhibiting different behavior past certain number of cycles.

3. A SOLUTION

Let us consider customer A who joins the end of its queue in its k^{th} cycle. We distinguish two different cases with respect to the cycle index:

- First cycle ($k = 1$): Due to Poisson arrivals, customer A will find the system in a steady state.
- Other cycles ($k > 1$): When customer A completes its $k-1^{\text{th}}$ cycle, and joins its queue in its k^{th} cycle, the system is not in a steady state. However, knowing the state of the system at the beginning of the previous cycle, we can compute its new state at the beginning of the current cycle due to the following facts: (i) while a customer is served by the server, none of the other customers may leave the system; (ii) a customer instantaneously goes from the server to the end of its queue when it enters the next cycle so there are no new arrivals in that infinitely short period of time; (iii) arrival rates of new customers do not depend of the state of the system.

When customer A joins the end of its queue in its k^{th} cycle, it will see the following customers:

- The customer who was being processed when A entered the system for the first time and provided that this customer has not yet left the system;
- Customers found in waiting the queue(s) when A entered the system, provided that they have not yet left;
- Customers who entered the system after customer A entered and have not yet left the system.

Customer A has to wait for the above three types of customers (providing they have equal or higher priorities) before it reaches the server and starts another service cycle. We shall denote the mean total waiting time for the three types of customers as r_{ck} , w'_{ck} , and w''_{ck} respectively.

3.1 Residual Service Time

Let T be the residual time in the current service cycle of a customer found in service by a new arrival. Let t be the expected (mean) value of T . In this section we derive t using an approach similar to the one in [2], where the mean residual time for a single class M/G/1 queue and multiple classes priority M/G/1 queue with no feedback are derived. We generalize the derivation to multiple cycles with different service time distributions.

Assuming that the system is stable, each customer eventually leaves the system. Since the service is non-preemptive, a high priority customer cannot preempt a low priority customer whose service cycle is already in progress. Since Poisson arrivals see time average (PASTA), the residual service time in a service cycle for a customer found in the server by a Poisson arrival, does not depend on the class nor priority. In what follows we derive this value.

Let us consider the server perceived by a new customer just before it enters its first cycle. It will see 0 customers, if the system is empty or 1 customer currently being processed. The mean residual service time, t , perceived by a new customer may be derived from Figure 3. When a customer begins its service, its residual service time is equal to its service time in that service cycle, say $T_i = S_{ck}$. The index i stands for i^{th} service cycle (any customer, any customer's cycle) since the beginning of observation. Then, the residual service time linearly decreases until it becomes equal to 0 after S_{ck} . Due to the PASTA rule, the mean residual service time, perceived by a Poisson arrival, is equal to the mean height of the graph in Figure 3.

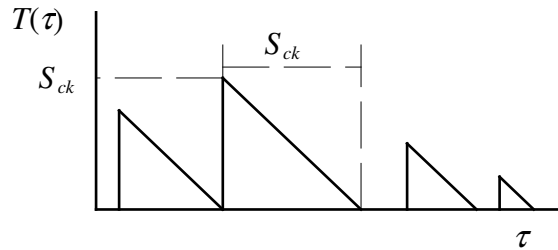


Figure 3: Residual service time perceived by a new arrival

The mean residual service time is

$$t = \lim_{\Delta \rightarrow \infty} \frac{1}{\Delta} \int_0^{\Delta} T(\tau) d\tau = \lim_{\Delta \rightarrow \infty} \frac{1}{2\Delta} \sum_{i=1}^{M(\Delta)} T_i^2 \quad (5)$$

where $|M(\Delta)|$ is the number of service cycles completed in period Δ . By grouping together terms which belong to the same class and service cycle, expression (5) may be rewritten as

$$t = \lim_{\Delta \rightarrow \infty} \sum_{c=1}^C \frac{A_c(\Delta)}{2\Delta} \sum_{k=1}^{\infty} \frac{|M_{ck}(\Delta)|}{A_c(\Delta)} \frac{\sum_{i \in M_{ck}(\Delta)} T_i^2}{|M_{ck}(\Delta)|} \quad (6)$$

where $M_{ck}(\Delta)$ is the set of k^{th} service periods of class c , $|M_{ck}(\Delta)|$ is its cardinality, and $A_c(\Delta)$ is the number of completely served (all cycles) customers of class c in period Δ . The following hold

$$\lim_{\Delta \rightarrow \infty} \frac{A_c(\Delta)}{\Delta} = \lambda_c \quad (7)$$

$$\lim_{\Delta \rightarrow \infty} \frac{|M_{ck}(\Delta)|}{A_c(\Delta)} = \pi_{ck} \quad (8)$$

$$\lim_{\Delta \rightarrow \infty} \frac{\sum_{i \in M_{ck}(\Delta)} T_i^2}{|M_{ck}(\Delta)|} = v_{ck} \quad (9)$$

Therefore

$$t = \frac{1}{2} \sum_{c=1}^C \lambda_c \sum_{k=1}^{\infty} \pi_{ck} v_{ck} \quad (10)$$

In a queue with no feedback (single service cycle), $\pi_{c1} = 1$ and $\pi_{ck} = 0$ ($k > 1$). Using this fact, we obtain $\frac{1}{2} \sum_{c=1}^C \lambda_c v_c$, $\frac{\lambda v}{2}$, and $\frac{\lambda}{\mu^2}$, which are mean residual times for priority M/G/1, single class M/G/1, and M/M/1 queues respectively, as obtained in [2].

3.2. Waiting Times

Next, we find r_{ck} . Obviously, when class c has the highest priority¹ and $k = 1$

$$r_{c1} = t \quad (11)$$

For lower priority classes and $k = 1$, a new arrival has to wait for the customer found in service to complete its current cycle. In case that customer has a higher priority, it will complete all its remaining cycles before the new customer reaches the server for the first time. In case a class i customer was being served in its j^{th} cycle, the additional time is $\sum_{l=1}^{\infty} p_{i,j} \cdots p_{i,j+l-1} s_{i,j+l}$. The throughput of such customers is $\lambda_i \pi_{ij}$, and their mean number is $\lambda_i \pi_{ij} s_{ij}$. Let us define

¹ Not necessarily 1 if there are no classes with priority equal to 1

$x_{ij} = \sum_{l=1}^{\infty} \pi_{i,j+l} s_{i,j+l}$. Then the total influence of such customers on waiting time is an increase of $\lambda_i x_{ij}$. By summing over all higher priority classes² and all cycles, we obtain the first cycle waiting time for the customer found in the service upon arrival

$$r_{c1} = t + \sum_{i:\omega_i < \omega_c} \lambda_i \sum_{j=1}^{\infty} s_{ij} x_{ij} \quad (12)$$

Obviously, $r_{c1} = r_{k1}$ if $\omega_c = \omega_k$. Using (11) as the initial value, we may rewrite (12) in the following recursive and numerically more efficient form

$$r_{c1} = r_{k1} + \sum_{i:\omega_i = \omega_k} \lambda_i \sum_{j=1}^{\infty} s_{ij} x_{ij} \quad \text{such that } \omega_k = \omega_c - 1 \quad (13)$$

Next we consider when $k > 1$. When customer A of class c enters subsequent cycles only customers of classes with the same priority, if originally found in service upon arrival of customer A, will interfere with customer A. Namely, a higher priority customer originally found in service would have already left, while lower priority customers will have to wait until customer A leaves the system. When customer A enters its first cycle, it will find on the average $\lambda_i \pi_{ij} s_{ij}$ class i customers being served in their j^{th} cycle. When customer A enters its k^{th} cycle, on the average $\lambda_i \pi_{i,j+k-1} s_j$ of those customers will still be in the system and in their $j+k-1^{\text{th}}$ cycle. On the average they will spend $s_{i,j+k-1}$ in service. By summing the waiting times over all classes with the same priority, the mean time customer A will spend waiting for the customer originally found in service is

$$r_{ck} = \sum_{i:\omega_i = \omega_c} \lambda_i \sum_{j=1}^{\infty} s_{ij} \pi_{i,j+k-1} s_{i,j+k-1} \quad k > 1 \quad (14)$$

Next we find w'_{ck} . Again, we distinguish calculation for the first and for the subsequent cycles. When a new class c customer enters its first cycle, it must wait for all higher priority waiting customers to leave the system. It also must wait for all customers with the same priority as its own to finish their current cycles. The customer does not have to wait for any of the lower priority customers (except for the one currently being served). A class i customer (higher priority) waiting in its j^{th} cycle will spend $s_{i,j} + \sum_{l=1}^{\infty} p_{i,j} \cdots p_{i,j+l-1} s_{i,j+l}$ in service before it leaves the system. From the Little's theorem, there are $\lambda_i \pi_{ij} w_{ij}$ such customers. Their overall contribution is $\lambda_i w_{ij} x_{ij}$. The overall first cycle waiting time due to the higher priority customers can be obtained by summing over all higher priority classes and all cycles, $\sum_{\omega_i < \omega_c} \lambda_i \sum_{j=1}^{\infty} w_{ij} x_{ij}$.

² $i:\omega_i < \omega_c$ denotes all i such that $\omega_i < \omega_c$.

The delay due to classes with the same priority may be derived as follows. Due to the PASTA rule, the new arrival will see n_{ij} class i customers waiting in their j^{th} cycle. The total throughput of these customers is $\lambda_i \pi_{ij}$. Using the Little's formula [8], we have that $n_{ij} = \lambda_i \pi_{ij} w_{ij}$. When customer A enters its k^{th} cycle, class i customers initially found waiting in their j^{th} cycle, will be in their $j+k-1^{\text{th}}$ cycle. Only $p_{i,j+1} p_{i,j+2} \dots p_{i,j+k-1}$ of those customers will still be in the system. By summing over all classes with the same priority as customer A and all cycles, we obtain $\sum_{i:\omega_i < \omega_c} \lambda_i \sum_{j=1}^{\infty} w_{ij} x_{ij}$. This expression may be simplified as $\sum_{i:\omega_i = \omega_c} \lambda_i \sum_{j=1}^{\infty} n_{ij} p_{i,j+1} p_{i,j+2} \dots p_{i,j+k-1} s_{i,j+k-1}$ and therefore the total first cycle waiting time w'_{c1} is

$$w'_{c1} = \sum_{i:\omega_i < \omega_c} \lambda_i \sum_{j=1}^{\infty} w_{ij} x_{ij} + \sum_{i:\omega_i = \omega_c} \lambda_i \sum_{j=1}^{\infty} w_{ij} \pi_{ij} s_{ij} \quad (15)$$

In the subsequent cycles ($k > 1$), a class c customer must wait only for customers with the same priority found in the system in the first cycle. Therefore

$$w'_{ck} = \sum_{i:\omega_i = \omega_c} \lambda_i \sum_{j=1}^{\infty} w_{ij} \pi_{i,j+k-1} s_{i,j+k-1} \quad k > 1 \quad (16)$$

Next we find w''_{ck} . Again we distinguish first and subsequent cycles. In the first cycle, during the waiting time w_{c1} , $\lambda_i w_{c1}$ new higher priority class i customers entered the system and must be served completely before the class c customer starts its first service cycle. For that reason, the class c customer has to wait additional s_i units of time. The overall contribution of higher priority customers is

$$w''_{c1} = w_{c1} \sum_{i:\omega_i < \omega_c} \lambda_i s_i \quad (17)$$

In the subsequent cycles, customer A must wait for

- Higher priority customers who came during the waiting time in the current cycle, and the service time in the previous cycle; These customers will leave the system before the customer is allowed to start its current service cycle. The contribution of all higher priority customers is $\sum_{i:\omega_i < \omega_c} \lambda_i (s_{c,k-1} + w_{ck}) s_i$;
- Same priority customers who entered the system after the customer being considered entered the system; The contribution of all such customers is $\sum_{i:\omega_i = \omega_c} \lambda_i \sum_{j=1}^{k-1} (w_{ij} + s_{ij}) \pi_{i,k-j} s_{i,k-j}$.

Therefore, for $k > 1$, it holds

$$w''_{ck} = (s_{c,k-1} + w_{ck}) \sum_{i:\omega_i < \omega_c} \lambda_i s_i + \sum_{i:\omega_i = \omega_c} \lambda_i \sum_{j=1}^{k-1} (w_{ij} + s_{ij}) \pi_{i,k-j} s_{i,k-j} \quad (18)$$

Waiting times for all classes and all cycles can be solved as a system of linear equations

$$w_{ck} = r_{ck} + w'_{ck} + w''_{ck} \quad (19)$$

where parameters in the right-hand side of (19) are derived in the above discussion.

3.3. Numerical Considerations

As we have seen, the problem can be solved as a system of linear equations (19). The number of unknowns is approximately equal to CK , where K is the maximum number of cycles numerically being considered in any class. The “numerically being considered” means that low probability cycles with large indices are ignored, i.e., the calculation of sums with an infinite number of terms/cycles (e.g., Bernoulli feedback) is truncated when the series of partial sums numerically converge with a prescribed accuracy. The worst case computational complexity of solving (19) is $O(C^3K^3)$.

We can notice that waiting times of a class do not depend on waiting times of lower priority classes. Therefore, we can find waiting times of the highest priority classes as a sub-system of linear equations with a smaller number of unknowns. Then we can use those values as constants, and find waiting times for classes with the next highest priority etc. In such a way, the computational complexity in one-class-per-priority case is $O(CK^3)$. In general, computational complexity is between the two special cases and increases as the maximum number of classes per priority increase.

The system of linear equations (19) is in a form suitable for Jacobi and Gauss-Seidel algorithms for iterative solution of systems of linear equations. Generally, examples can be made where one or the other algorithm exhibits better convergence. Similarly, cases may be found when either or both algorithms have problems with convergence. A detailed discussion on the convergence criteria may be found in [11] and [13].

In our examples, we used the Gauss-Seidel algorithm, which is simpler for programming, and has lower memory requirements, as both old and new values need not be stored. We used r_{ck} as an initial value of w_{ck} . Using this as the initial value, we did not experience any problems with convergence. Certainly, the algorithms do not converge in case of unstable systems ($\sum_{i=1}^C \lambda_i s_i \geq 1$) but this situation can be detected before the procedure starts. The system may be stable for higher priority classes, and unstable for lower priority classes for which $\sum_{\omega_i \leq \Omega} \lambda_i s_i \geq 1$.

Generally, queues with lower utilization exhibit better convergence. In our numerical examples for moderately low utilization (below 75%), the number of iterations was below 50 and between 10 and 20 in most of the cases. The relative error was 10^{-3} . It may be shown [13] that for the same system of linear equations, the number of iterations increases as a linear function of logarithm of the relative error.

At the end of this section, we discuss how to choose service times in each cycle so that the total service time has a given distribution. Let $S(\Theta)$ be the moment generating function of the total service time where Θ is the parameter of the moment generating function. In this paper we assume that service time in each cycle is independent of service times in other cycles. Let $a_k \leq 1$ be the portion of the total service time spent in cycle k . then, the moment generating function of the service time in cycle k is $S^{a_k}(\Theta)$. The mean service time in each cycle is $a_k s$. The corresponding squared coefficient of variation is $(E[S^2] - s^2)/(a_k s^2)$.

4. NUMERICAL EXAMPLES

In this section we present some numerical examples as an illustration of the developed numerical procedures.

The first example shows the influence of number of cycles in round robin process scheduling [12] of a single class queue. This is an important parameter of an operating system. We keep the total service time equal to 0.25. Here, we neglect the context-switching overhead spent whenever the processor switches from one customer to another. We vary the number of cycles from 1 to 150, In Figure 4, we show the results only up to 5 cycles. We vary also the processor utilization (25%, 50%, 75%). We notice that the delay increases as the number of cycles increase despite the constant total service time. The increase is more noticeable when the utilization is higher. As the number of cycles increase from 1 to infinity, the overall delay increases from the one corresponding to the M/D/1 queue to the one corresponding to the equivalent M/M/1 queue (as shown by the dotted lines). This effect may be explained using the following reasoning. When the number of cycles is sufficiently large, all customers in the system are effectively served in parallel (perfect processor sharing). The processor time available to each customer is inversely proportional to the number of customers in the system. On the average, the processor speed available to each customer is proportional to $1 - \rho$ where ρ stands for the processor utilization.

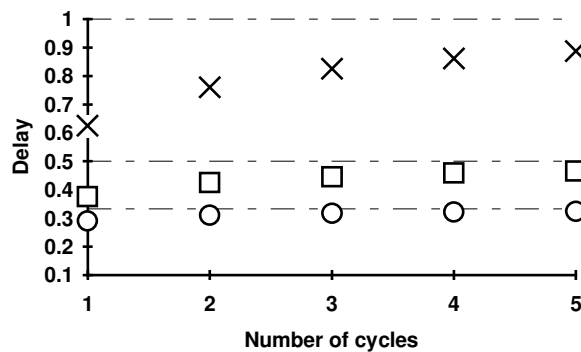


Figure 4: Delays in a Single-Class Queue vs. Number of Cycles and Utilization

In the next example, we introduce a context-switching overhead, which is a time needed to switch processor from one customer (or task in the operating systems terminology) to another. It is well known that when an operating system performs too many context switches, the system becomes congested due to "trashing" [9, 12]. We assume that it takes s' time units to complete service of a customer (useful work) and s'' time units just to switch processor between two customers. Assuming that a customer is completely served in k cycles, the processing time in each cycle is equal to $s'' + s' / k$ and the total processing time per customer is equal to $ks'' + s'$. Applying this to the previous example, we assume $s' = 0.2$ and $s'' = 0.05$. In case of a single cycle, the minimum total service time is 0.25 as the total service time in the previous example. The maximum number of cycles k_{\max} , may be determined by solving equation $\lambda(k_{\max}s'' + s') = 1$. For the arrival rates equal to 1, 2, and 3, it is equal to 16, 6, and 2 respectively. Figure 5 shows the results.

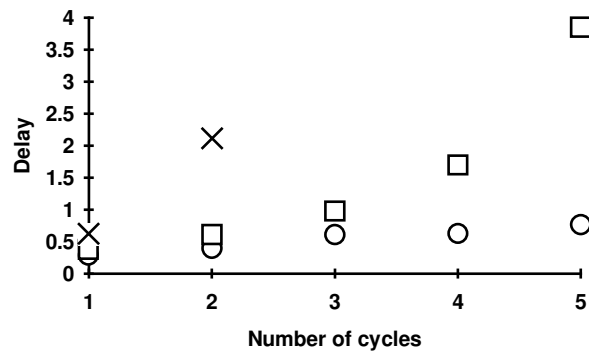


Figure 5: Delays in a Single-Class Queue vs. Number of Cycles and Utilization

Next, we discuss how the order of operations may affect the delay. Let the arrival rate be equal to 1, and there are two cycles (stages) which take constant number of time units, 0.2 and 0.4. If we schedule the shorter cycle first, the delay is 1.14286. If the longer cycle is scheduled first, the delay is 1.33333. This agrees with the intuitive reasoning that it is better to schedule shorter cycles first. The less a customer spends in a cycle, the shorter waiting time in subsequent cycles due to a smaller number of new customers which arrived after the customer being considered.

Figure 6 shows a single class queue with the arrival rate equal to 0.5, and the total service time equal to 1. We vary the length of the service period per cycle (quantum size) and the results are shown in Figure 6. When the quantum size is greater than 1, there is only one cycle and the queue behaves as an ordinary M/D/1 queue. When the quantum size is smaller than one and greater than 1/2, there are two cycles. When the quantum size is closer to 1, the difference in processing between the first and the second cycle is larger. Note a sharp increase when the quantum size becomes smaller than one. At that point, the processing is almost completed in the first cycle, and the customer has to spend one waiting period in the second cycle to receive an insignificant amount of processing. As the quantum size decrease, processing times in the two cycles become more balanced, and the delay decreases. As before, we notice a sharp increase in the delay when the third cycle is introduced.

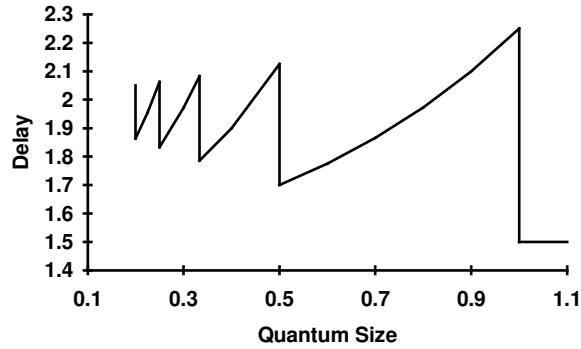


Figure 6: A Single Class Queue - Delay vs. Quantum Size

Next example considers a priority queue with three classes. Arrival rates and total service times are equal to 2, and 0.1 respectively for each class. We varied the number of cycles from one to five. Figure 7 shows the results. We can notice that when the number of cycles is increased, the delay for the high priority class (1) decreases. This is due to the decreased residual service times of low priority classes observed by a new high priority arrival. Recall that the queue is non-preemptive, i.e., a customer currently being served by the server cannot be preempted by a high priority arrival. As we may have expected, the delays of classes 2, and 3 increase as the number of cycles increase.

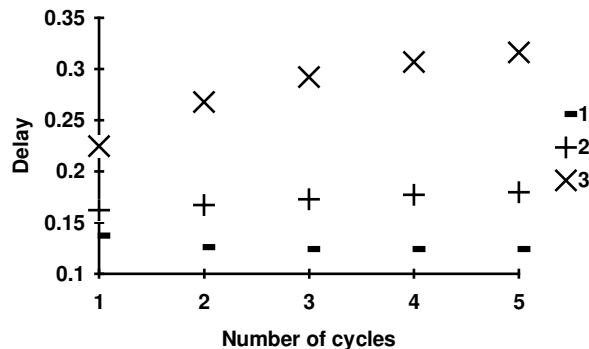


Figure 7: Delays in a Priority Queue with 3 Classes vs. Number of Cycles

Finally, Figure 8 shows the delay for a 2 classes non-priority queue vs. the quantum size. The arrival rates for both classes are equal to 0.2. The total service times are constant and equal to 1 and 2 time units. The quantum size is varied between 0.3 and 2. As we can see, the first class is completely served in a single cycle when the quantum size is greater or equal to 1. The difference in the delay in that region is due to smaller residual service time as the two service periods of the second class become more balanced. We also notice sharp increases in delays as new cycles are introduced.

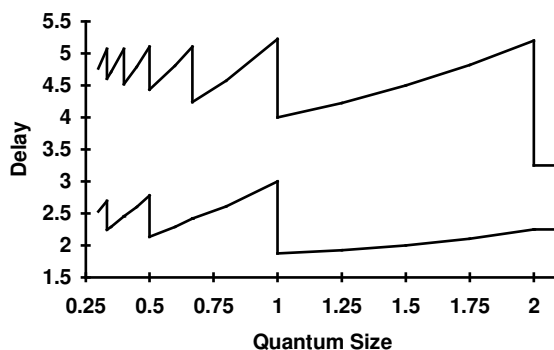


Figure 8: A Two Class Non-Priority Queue - Delay vs. Quantum Size

REFERENCES

- [1] A. O. Allen, *Probability, Statistics, and Queueing Theory*, Academic Press, 1978.
- [2] D. Bertsekas, and R. Gallager, *Data Networks*, Prentice-Hall, Inc. 1987.
- [3] D. D. Dimitrijević, and M.-S. Chen, "A Procedure for Probabilistic Protocol Verification and Evaluation," *IEEE Transactions on Communications*, Volume 40, Number 7, July, 1992, pp 1183-1191.
- [4] D. D. Dimitrijević, "Approximate Analysis of Markovian Queueing Systems: An Approach and a Tool," *GTE Technical Memorandum TM 0497-01-92-414-01*, January, 1992.
- [5] R. L. Disney, D. Konig, and V. Schmidt, "Stationary Queue Length and Waiting-Time Distributions in Single-Server Feedback Queue," *Advances in Applied Probability*, Volume 16, 1984.
- [6] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley & Sons, Inc., 1991.
- [7] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevick, *Quantitative System Performance*, Prentice-Hall, Inc., 1984.
- [8] J. D. C. Little, "A Proof of the Queueing Formula $L = \lambda W$," *Operational Research*, 9, 1961, pp 383-387.
- [9] S. E. Madnick, and J. J. Donovan, *Operating Systems*, McGraw-Hill, Inc., 1978.
- [10] D. A. Menasce, V. A. F. Almeida, and L. W. Dowdy, *Capacity Planning and Performance Modeling*, Prentice-Hall, Inc. 1994.
- [11] W. H. Press, B. P. Flannery, S. E. Tenkolsky, and W. T. Vetterling, *Numerical Recipes in C The Art of Scientific Computing*, Cambridge University Press, 1988.

- [12] A. S. Tanenbaum, *Operating Systems: Design and Implementation*, Prentice-Hall, Inc., 1987.
- [13] D. Tošić, *Uvod u Numeričku Analizu (Introduction to Numerical Analysis)*, Naučna Knjiga, Beograd, 1982.
- [14] Vučetić, D. D. Dimitrijević, and H. O. Beća, "Task Scheduling in a Multi-Domain Packet Switching Network Station," Proceedings of IEEE ICC'90, New Delhi, India, November, 1990.