W. Chang

Computer Interference Analysis

Abstract: This paper describes a single-server queuing model with Erlang input, which can be used for computer congestion analysis. The model is intended primarily for small computer systems in which the CPU is needed for input/output operations, which aggravates the interference problem.

The model provides such information as the queuing time distribution for channel requests and the CPU delay due to channel interference. The model is illustrated by numerical examples, and procedures for the analysis are shown.

Introduction

In an earlier paper [1], basic concepts in queuing theory and single-server queuing processes with random input were reviewed in terms of their applicability to computer system analysis. The server in a computing system can be any of a number of devices, such as a channel, CPU, or communication line. Ideally, each of these devices could be represented mathematically and the appropriate analyses applied to determine their expected behavior in a system. Practically, no such complete analysis is possible because of present limitations of the queuing theory itself. The analysis of a complete computing system would require the solution of a network of queues, which is possible only for a few idealized cases. Fruitful application of queuing analysis to computing systems is therefore limited to subsystems analysis.

Sometimes it is possible using queuing theory to study the interaction of a combination of facilities, such as is done in channel interference analysis [2], if the facilities are performing a joint task. Channel interference is defined as the proportion of computer processing time required for the control of channel operations. In the context of System/360 or 370 architecture [3], the resources of main storage, the CPU and its read-only storage, and the channel are required for the control portion of a channel operation. In this case, a single-server facility can represent a combination of these hardware elements. Service time is the time required to perform the joint task. This paper examines the congestion effect of this type of operation.

The assumption of random or Poisson input [1], which is desirable for its Markovian properties, may or may not be warranted in computer subsystem analysis. If the input to a server originates from a large number of sources (e.g., many terminals in a large system) the assumption of Poisson input is justified. But in this paper, a more general type of input, the Erlang input process, is assumed. The Erlang process includes the Poisson and regular (constant) inputs as its special cases. The assumption of such a general input process is useful for the analysis of small computing systems where a Poisson input may not be justified. Such small computing systems (including minicomputers) may be used as stand-alone computers or as controllers, concentrators, or front-end processors for the large machines.

The assumption of an Erlang process may have an additional benefit. In some computing system analyses, the approximate waiting times determined analytically may be higher than those obtained by simulation, particularly in the high utilization range [5]. The analytic results may be somewhat improved if a more general input distribution, such as an Erlang process, is assumed.

A single-server queuing process with Erlang input and general service time distribution has been investigated by F. Pollaczek [6,7], A. J. Fabens [8], L. Takacs [9] and others. Both steady-state and transient (time-dependent) solutions have been determined.

The queuing model investigated in this paper accommodates two types of service requests. The first arrives

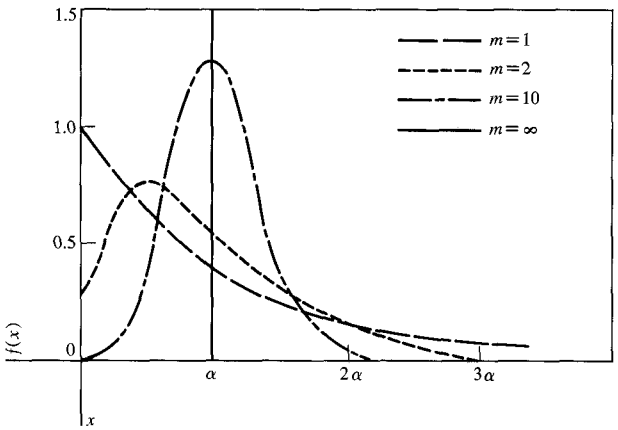


Figure 1 Erlang m functions.

at a single-server system in an Erlang process. The second is always present in the system and is served on an as-available basis by the server. The stationary queuing time and the queue size distributions are determined.

Erlang processes

Let calls arrive at a single-server system at the instants $\tau_1, \tau_2, \cdots, \tau_n, \cdots$, where the interarrival times $\tau_{n+1} - \tau_n$ $(n = 0, 1, \cdots, \tau_0 = 0)$ are identically distributed independent random variables with distribution function

$$P\{\tau_{n+1} - \tau_n \leq x\} = F_m(x) = 1 - \sum_{j=0}^{m-1} e^{-\lambda x}(\lambda x)^j/j!$$

$$(x \geq 0).$$

Then the arrival pattern of calls is an Erlang process.

Let the average interarrival time be α. Then $\alpha = m/\lambda$. The arrival rate is $1/\alpha = \lambda/m$.

The parameter m determines the shape of the Erlang distribution, as illustrated in Fig. 1. In the case of Erlang input with $m = 1$, $F_1(x)$ is an exponential distribution:

$$F_1(x) = 1 - e^{-\lambda x}.$$

This is identical with the Poisson process of density λ.

In the case of Erlang $m = \infty$, the arrival process is regular:

$$F_\infty(x) = \begin{cases} 0 & \text{if } 0 \leq x < \alpha \\ 1 & \text{if } x > \alpha. \end{cases}$$

Intermediate Erlang-m values yield a family of interarrival time distribution curves. If an arrival pattern ranges between random and regular arrivals, the input process can be treated as one of the Erlang family.

The probability density function

$$f_m(x) = dF_m(x)/dx$$

is plotted in Fig. 1 for four values of the parameter m. The rth moment of the Erlang m distribution is given by

$$\alpha_r = [(r + m - 1)!/(m - 1)!](\alpha/m)^r,$$

where $\alpha_1 = \alpha$ is the average interarrival time.

Since the Erlang process includes the Poisson process as its special case, it is useful in modeling a wide class of input environments. Consider the following input processes in a computing system.

Small number of input sources: An I/O device, such as a card reader, transmits its data to the computer in two phases. The first phase includes control of the mechanism, checking it for device-ready condition, etc.; and the second phase is the actual transmission of data. During the first phase, control information is passed through the control unit and the channel between the device and the processor. The first phase of the operation involves irregularly occurring service requests (control calls) to the system (i.e., a combination of main storage, the CPU and its read-only storage, the channel, and the control unit). The second phase is highly repetitive or periodic, with the arrival of calls for service (data transmission) being more regular.

Cox and Smith [10] show that if a system serves more than ten input sources, with each source generating calls regularly, the pooled input (the sum of all inputs) to the system can be treated as a Poisson process. However, for a small computer system where two or three sources are active, such an input cannot be treated as Poisson without introducing some error in the modeling. Thus a more general type of input is preferred.

Queues in series: A computer system can be structured as a network of queues. For example, terminals are attached to communications lines, communications lines to line control adaptors, the adaptors to a multiplexor channel, and the multiplexor channel to the computer. Queues can be formed at each of the servers or at buffering devices. The output of one server often forms the input to the next server so that calls arrive in one queue immediately after they have been serviced in the preceding queue. If the first server (with independent service times) is heavily utilized, the arrival pattern for the second queue is then nearly equal to the distribution function of the service time in the first queue. When we treat the second queue as a single-server process, the arrival pattern is approximately Poisson only if the service time of the preceding server is distributed exponentially.

Buffer effect: When bits are assembled into bytes or bytes into words, buffering is required. Although characters may arrive randomly, the service requests of assembled words are distributed as an Erlang process. This is so mathematically because the convolution of m exponential functions (assembled from m randomly arriv-

ing characters) generates an Erlang-m process (time between service requests of assembled words).

Computer interference

In computer systems, the activities of moving information back and forth between the computer main storage and an input/output device constitutes an I/O operation. The least costly design to accomplish an I/O operation in a small computer is that of the so-called "programmed-I/O." The I/O device is attached to an interface, which is connected to the processor unit, as shown in Fig. 2. The I/O operation, including device selection, mechanical control if required, and data transfer, is accomplished by an I/O program stored in main storage. During the whole I/O operation, the CPU is dedicated completely to that single I/O activity. The CPU data path is used to accomplish the data transfer between the device and main storage. Thus interference with the CPU by the I/O activity is great and its duration may be rather long with slow devices. This I/O scheme may not be practical for a high-speed device, such as a fast disk, but it is rather popular with low-activity devices, such as teletypes (TTY's).

Some channel hardware is added to modern small computers to permit more than one device to be active at one time and to increase the time that the CPU can be used to perform other tasks. But the CPU is required to supervise channel activity and to control the input/output operation. However, after the input/output operation has been started, the CPU is released for other tasks during the mechanical operation of the device or, in the case of some channel facilities, during the actual data transfer. The data transfer may be completed by using another data path and by "cycle-stealing." Cycle-stealing is the use of a main storage cycle to accommodate a data transfer request, with the fetch/store request to main storage by the CPU delayed for one cycle. Cycle-stealing prolongs the service time of a CPU instruction. During the cycle-steal operation, the channel must update the storage addresses and keep an account of the actual number of bytes transferred.

Most small computing systems, such as IBM System/3, 1130, 1800, and small models of System/360 and 370 use cycle-stealing to accomplish high-speed I/O data transfer. In large systems, the cycle-stealing method is also used but the channel is made more self-contained to further reduce CPU interference.

The interference problem in a small computer like the one shown in Fig. 3 can be simplified and formulated as follows. Let the CPU with main storage be represented by a single server facility. This server handles two kinds of input requests: all channel service requests (programmed-I/O, channel control, etc.) except for cycle-stealing requests and all CPU service requests. Assume

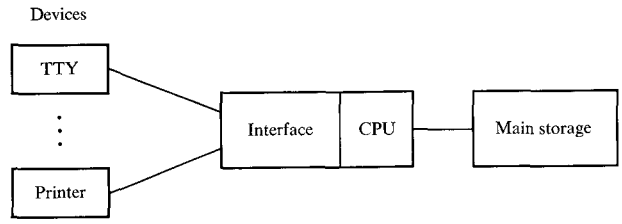
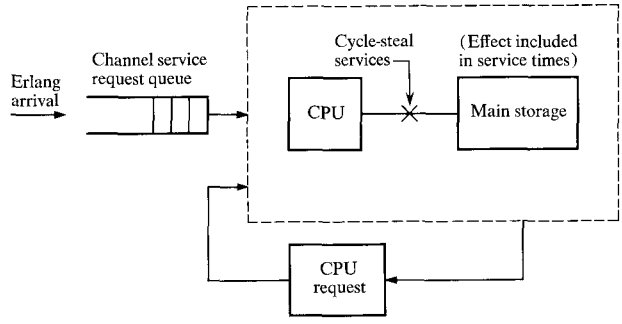


Figure 2 Simple computer with programmed I/O.

Figure 3 Model of computer interference.



that the arrival of channel service requests follows an Erlang-m process. The service discipline is assumed to be first-come, first-served. Although in reality the channel is served on a priority basis, the mathematical solution for priority queues first requires the solution of a model with no priority, with the results then modified to account for the effects of priority [1]. CPU requests are assumed to always be present and to be served on an as-available basis whenever the server is not busy servicing the first type of requests. The cycle-stealing requests are included in the service times of both types of services and prolong the required service times. As an example, assume that main storage is busy because of cycle-stealing for 20 percent of the time. A gross estimate of the prolonged service time is obtained by dividing the original service time by a factor of $1 - 0.2$ [11]. For a more accurate estimate, one can use statistical methods based on renewal theory [12,13].

In some computing systems, if special registers are not provided, the updating of control and status information for the channels requires the use of common registers. Thus the contents of the registers must be stored prior to servicing a channel request and restored after completion of channel service [2]. A single server with two service time distributions has been investigated by Welch [14].

We assume in this paper that a channel request arriving when the channel request queue is empty must wait

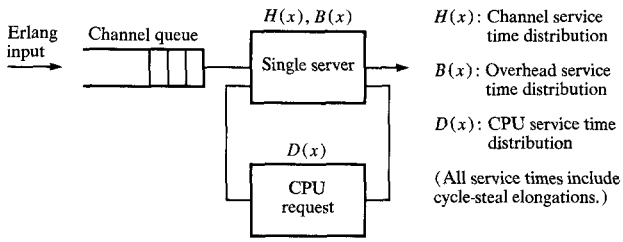


Figure 4 Queuing model with Erlang input.

an additional time increment for the system to store registers and to handle other housekeeping functions before the service of the channel can commence; a request arriving when there is a queue waits for its turn but it is not required to wait for this system set-up time.

In System/370 the system is interrupted from an in-progress CPU operation to service a channel request almost immediately, the delay being at most a memory reference cycle. However, some systems can be prepared to service a channel request only after execution of the current CPU instruction has been completed. Since the mathematical model assumes a general service time distribution for a CPU service, both of these cases can be analyzed.

Analytic model

The model assumes that all service time distributions (including the extension due to the cycle-stealing interruptions) are known. Let $H(x)$ be the channel service time distribution, $D(x)$ the CPU service time distribution, and $B(x)$ the extra set-up (overhead) time distribution, which includes the time needed to prepare the system to serve the channel requests.

The Laplace-Stieltjes transforms of these service time distributions are:

$$\psi(s) = \int_0^\infty e^{-sx}\, dH(x)\,,$$

$$\delta(s) = \int_0^\infty e^{-sx}\, dD(x)\,, \quad \text{and}$$

$$\beta(s) = \int_0^\infty e^{-sx}\, dB(x)\,.$$

We further define the rth moments as

$$h_r = \int_0^\infty x^r\, dH(x)\,,$$

$$d_r = \int_0^\infty x^r\, dD(x)\,, \quad \text{and}$$

$$b_r = \int_0^\infty x^r\, dB(x)\,.$$

The first moments, $h = h_1$, $d = d_1$, and $b = b_1$, are the mean service times.

The arrival of channel requests to the system is assumed to follow an Erlang-m process with an input rate of λ/m. The queuing system is shown in Fig. 4. CPU requests are serviced on an as-available basis and are assumed to be present continuously throughout the operation. This fact is shown in Fig. 4 by a feed-back loop from the output of the system. A CPU request is generated as soon as a previous CPU service is completed.

• *Queue length distribution*

Let ξ_n be the channel queue length immediately after the nth departure of a channel request. The stochastic behavior of the queue size with Erlang-m input can be deduced immediately if we know the stochastic behavior of the process defined in the following paragraphs [9].

Suppose that calls arrive at a counter at the instants $\tau_1', \tau_2', \cdots \tau_n', \cdots$, where the interarrival times $\tau_{n+1}' - \tau_n'$ $(n = 1, 2, \cdots; \tau_0' = 0)$ are identically distributed, independent random variables with distribution function

$$F(x) = \begin{cases} 1 - e^{-\lambda x} & \text{if } x \geq 0 \\ 0 & \text{if } x < 0\,, \end{cases}$$

i.e., $\{\tau_n'\}$ is a Poisson process. The calls will be serviced by a single server in batches of m in the order of their arrival. The server is allowed to service CPU requests if and only if fewer than m calls are present in the queue. The service times of the batches are assumed to be independent, positive random variables. The service time distribution is $H(x)$. The first batch that is to be serviced after the completion of a CPU request must wait an extra time period, which is distributed according to $B(x)$.

If we identify every mth arrival in the Poisson process with an arrival in the Erlang process, i.e., $\tau_n = \tau_{nm}'$, and if we identify the service time of the nth batch in the Poisson process with the service time of the nth channel request in the Erlang process, then the Poisson process is reduced to an Erlang process. This is so because $F_m(x)$ is the mth iterated convolution of $F(x)$ with itself.

Comparing these two processes, we find that the waiting time follows the same probability law in both processes. The departures also agree. However, the queue lengths are different but are related by

$$\xi_n = \lfloor \xi_n'/m \rfloor\,.$$

where ξ_n' is the queue size at the nth departure in the Poisson process and $\lfloor \xi_n'/m \rfloor$ means the greatest integer less than or equal to ξ_n'/m.

These two processes have the same waiting time distribution and the queue sizes are also related. If we find the solution for the Poisson process, the Erlang process can immediately be determined. It is an advantage that the Poisson process has more Markovian properties.

The Poisson process can be solved as follows:

Let ν_n be the number of calls (m calls = 1 channel request) that arrive during the service of the nth batch, where ν_n is a conditional random variable that depends on the service time. Assume that the nth batch is serviced following the service of the $n-1$th batch. Let χ_n be the service time of the nth batch. The probability distribution of ν_n can be found as

$$P\{\nu_n = i\} = \int_0^\infty P\{\nu_n = i | \chi_n = x\}\, dH(x)\,. \tag{1}$$

Since the calls arrive in a Poisson stream, we have

$$P\{\nu_n = i\} = \int_0^\infty [(\lambda x)^i/i!] e^{-\lambda x}\, dH(x)\,. \tag{2}$$

The generating function for ν_n can be written as

$$\sum_{i=0}^{\infty} P\{\nu_n = i\}\, z^i = \int_0^\infty e^{-\lambda(1-z)x}\, dH(x) = \psi(\lambda(1-z))\,. \tag{3}$$

Equation (3) implies that if we replace s by $\lambda(1-z)$ in $\psi(s)$, the transform of service time distribution, we obtain the generating function of ν_n. Similarly, if we let ν_n' be the number of calls that arrive during the set-up time and the service time of the nth batch, the generating function ν_n' can be found as

$$\sum_{i=0}^{\infty} P\{\nu_n' = i\}\, z^i = \int_0^\infty e^{-\lambda(1-z)x}\, dH(x) * B(x)$$
$$= \psi(\lambda(1-z))\beta(\lambda(1-z))\,. \tag{4}$$

(The notation $*$ indicates convolution.)

The queue lengths ξ_{n+1}' and ξ_n' and the number of calls received during the service of the $(n+1)$th batch are related by

$$\xi_{n+1}' = \begin{cases} [\xi_n' - m] + \nu_{n+1} & \text{if } \xi_n' \geq m \\ [\gamma_n + \xi_n' - m] + \nu_{n+1}' & \text{if } \xi_n' < m\,, \end{cases} \tag{5}$$

where γ_n is the total number of new calls that arrived during the servicing of CPU requests after the service of the nth batch, and γ_n must be such that

$\gamma_n + \xi_n' \geq m$.

Equation (5) can be explained as follows. When $\xi_n' \geq m$, the next batch (i.e., the $(n+1)$th service) can be serviced immediately, the queue length is reduced by m (i.e., the nth batch departs) and the number of new calls increasing the queue size is ν_{n+1}. When $\xi_n' < m$, the $(n+1)$th batch cannot be serviced immediately, since fewer than m calls are in the queue. The system is then allowed to service the CPU requests. During these services, if more than $m - \xi_n'$ new calls arrive, the system can switch to the service of the $(n+1)$th batch. Since in this case the system requires a set-up period, ν_{n+1}' represents the number of new arrivals during the set-up time and the service time of the $(n+1)$th batch.

Let $P\{\xi_n' = j\} = P_j$ be the probability of the queue length in the system at the nth departure. Define the generating function

$$U(z) = \sum_{j=0}^{\infty} P_j z^j\,.$$

If the stationary solution for queue length exists, ξ_{n+1}' and ξ_n' must have the same marginal distribution. Their generating functions must also be the same.

Let $\varphi_j(\lambda(1-z))$ be the conditional generating function of γ_n, given that the nth departure leaves $m-j$ calls in the queue so that exactly j new calls must arrive before the $(n+1)$th channel service can begin. Its derivation is given in Appendix A.

From the theorem that the generating function of the sum of two independent variables is the product of two generating functions, it follows that

$$U(z) = \left[U(z) - \sum_{j=0}^{m-1} P_j z^j\right] \psi(\lambda(1-z))/z^m$$
$$+ \sum_{j=0}^{m-1} P_j \varphi_{m-j}(\lambda(1-z))\, z^j\, \beta(\lambda(1-z))$$
$$\psi(\lambda(1-z))/z^m\,. \tag{6}$$

The first term of Eq. (6) represents the condition that $\xi_n' \geq m$ and the second term, $\xi_n' < m$.

From Eq. (6),

$$U(z) = \sum_{j=0}^{m-1} P_j z^j\, [\varphi_{m-j}(\lambda(1-z))\, \beta(\lambda(1-z)) - 1]$$
$$\psi(\lambda(1-z))/[z^m - \psi(\lambda(1-z))]\,. \tag{7}$$

From Rouche's theorem [15], the equation

$$z^m - \psi(\lambda(1-z)) = 0$$

has exactly m roots, $\omega_1, \omega_2, \cdots, \omega_m$, on and inside the unit circle $|z| = 1$. In particular, $\omega_m = 1$. Since $U(z)$ is a regular function of z if $|z| \leq 1$, the numerator of Eq. (7) must also vanish at these roots. Thus

$$\sum_{j=0}^{m-1} P_j \omega_i^j [\varphi_{m-j}(\lambda(1-\omega_i))\, \beta(\lambda(1-\omega_i)) - 1]$$
$$\psi(\lambda(1-\omega_i)) = 0 \qquad (i = 1, 2, \cdots, m)\,. \tag{8}$$

Equation (8) is a set of m equations with m unknowns, $P_0, P_1, \cdots, P_{m-1}$, which can be determined uniquely by solving these m equations. In the case of $\omega_m = 1$, L'Hopital's rule can be used to obtain an equation. Since $U(1) = 1$, we have

$$\sum_{j=0}^{m-1} P_j [-\varphi_{m-j}'(0) + b]\lambda = m - \lambda h\,. \tag{9}$$

Equation (9) determines the generating function for ξ_n'. The queue length in the Erlang process can be determined by the relationship $\xi_n = \lfloor \xi_n'/m \rfloor$.

• *Queuing time distribution*

Queuing time is defined as the sum of the waiting time in the channel queue and the service time of the channel request. Let $T(x)$ be the queuing time distribution, and let the Laplace-Stieltjes transform be

$$\theta(s) = \int_0^\infty e^{-sx}\, dT(x) .$$

Then $\theta(s)$ can be determined as follows. Since the queue length at a departure point is equal to the new calls that arrived during the preceeding queuing time period [1,9], we have

$$\theta(\lambda(1-z)) = U(z) . \tag{10}$$

If we put $s = \lambda(1-z)$ in (10), we have

$$\theta(s) = \sum_{j=0}^{m-1} P_j \lambda^{m-j} (-s+\lambda)^j [\varphi_{m-j}(s)\ \beta(s) - 1]\ \psi(s) \div [(-s+\lambda)^m - \lambda^m \psi(s)] . \tag{11}$$

Since the queuing time distribution is the same in both processes, the transform given by Eq. (11) holds for Erlang input.

• *Waiting time distribution*

Let $W(x)$ be the waiting time distribution in the channel queue. Let the Laplace-Stieltjes transform be

$$\Omega(s) = \int_0^\infty e^{-sx}\, dW(x) .$$

Since the queuing time is the sum of the waiting time and the service time, $\Omega(s)$ can be obtained from $\theta(s)$ as

$$\Omega(s) = \theta(s)/\psi(s) . \tag{12}$$

Two special cases are of practical interest: deriving the Laplace-Stieltjes transform of the waiting time distribution for a Poisson input and for the Erlang-2 input. Finally, an expression for $\Omega(s)$ is given for Erlang-m input.

Case 1: Poisson input

Since the Poisson input is a special case of Erlang input with $m = 1$, we have from Eqs. (11) and (12)

$$\Omega(s) = \frac{P_0 \lambda[\varphi_1(s)\ \beta(s) - 1]}{(-s+\lambda) - \lambda\psi(s)} .$$

This can be written as

$$\Omega(s) = P_0 \frac{\lambda s}{s - \lambda + \lambda\psi(s)} \cdot \frac{1 - \varphi_1(s)\ \beta(s)}{s} .$$

Since $\Omega(0) = 1$, we have by applying L'Hopital's rule

$$P_0 = (1-\lambda h)/[\lambda(-\varphi_1'(0)) + \lambda b] ,$$

where $-\varphi_1'(0)$ can be obtained from $\varphi_1(s)$. To simplify the expression, let

$$a_{jr} = (-1)^r \varphi_j^{(r)}(0) .$$

(A few expressions for a_{jr} are given in Appendix A.) Thus $\Omega(s)$ can be written as

$$\Omega(s) = \frac{(1-\lambda h)s}{s - \lambda + \lambda\psi(s)} \cdot \frac{1 - \varphi_1(s)\ \beta(s)}{(a_{11}+b)s} . \tag{13}$$

$\Omega(s)$ is the product of two Laplace-Stieltjes transforms, the first of which is identical to the Laplace-Stieltjes transform of the waiting time distribution in a single-server system with a Poisson input and general service times $H(x)$, and the second has the form

$$[1 - \varphi_1(s)\ \beta(s)]/[(a_{11}+b)s] .$$

Let W_1 and W_2 be the first and the second moment of the waiting time distribution. They can be obtained as follows:

$$W_1 = -\Omega'(0) = \frac{\lambda h_2}{2(1-\lambda h)} + \frac{a_{12}+b_2+2a_{11}b}{2(a_{11}+b)} ,$$

$$W_2 = \frac{\lambda h_3}{3(1-\lambda h)} + \frac{\lambda^2 h_2^{\,2}}{2(1-\lambda h)^2} + \frac{a_{13}+b_3+3a_{12}b+3a_{11}b_2}{3(a_{11}+b)} + 2\,\frac{\lambda h_2}{2(1-\lambda h)} \times \frac{a_{12}+b_2+2a_{11}b}{2(a_{11}+b)} .$$

Case 2: Erlang-2 input

For the case of Erlang input with $m = 2$, $\Omega(s)$ can be determined from

$$\Omega(s) = \frac{P_0\lambda[\varphi_2(s)\ \beta(s) - 1] + P_1(-s+\lambda)[\varphi_1(s)\ \beta(s) - 1]}{(-s+\lambda)^2 - \lambda^2\psi(s)} .$$

Let $R_1 = P_1/P_0$. R_1 can be obtained from Eq. (9) as

$$R_1 = \frac{\lambda[1 - \varphi_2(s_1)\ \beta(s_1)]}{(-s_1+\lambda)[\varphi_1(s_1)\ \beta(s_1) - 1]} ,$$

where $s_1 = \lambda(1-\omega_1)$.

Since P_0 can be determined from $\Omega(0) = 1$ and L'Hopital's rule, we have

$$\Omega(s) = \frac{(2-\lambda h)\ \lambda(-s)(1-s/s_1)}{(-s+\lambda)^2 - \lambda^2\psi(s)} \times \frac{[\varphi_2(s)\ \beta(s) - 1] + (R_1/\lambda)(-s+\lambda)[\varphi_1(s)\ \beta(s) - 1]}{[a_{21} + b + R_1(a_{11}+b)](-s)(1-s/s_1)} . \tag{14}$$

Again $\Omega(s)$ is the product of two Laplace-Stieltjes transforms. The first one is identical to the Laplace-Stieltjes transform of the waiting time distribution of a single-server queue with Erlang-2 input and a general service time distribution. Thus one can conclude that the

waiting time in the channel queue consists of two parts: the first part is the same as that of a single-server queue with an Erlang input and a general service time distribution, and the second part is a delay that depends on the CPU service time and the switching time.

Case 3: Erlang-m input

An expression for $\Omega(s)$ for Erlang-m input is:

$$\Omega(s) = \frac{(m-\lambda h)\,\lambda^{m-1}(-s)\prod_{i=1}^{m-1}\left(1-\frac{s}{s_i}\right)}{(-s+\lambda)^m - \lambda^m \psi(s)} V(s)\,. \qquad (15)$$

$V(s)$ can be determined as follows:

$$V(s) = \frac{\varphi_m(s)\,\beta(s) - 1 + \sum_{j=1}^{m-1} R_j\left(\frac{-s+\lambda}{\lambda}\right)^j [\varphi_{m-j}(s)\,\beta(s) - 1]}{\left[a_{m1} + b + \sum_{j=1}^{m-1} R_j(a_{j1}+b)\right](-s)\prod_{i=1}^{m-1}\left(1-\frac{s}{s_i}\right)},$$

where $R_j = P_j/P_0$ for $j = 1, 2, \cdots, m-1$ can be obtained from the $m-1$ equations of Eq. (15) and $s_i = \lambda(1-\omega_i)$.

Thus $\Omega(s)$ is the product of two Laplace-Stieltjes transforms. The waiting time moments can be obtained by differentiating Eq. (15). (See the section on numerical examples for details.)

● *An estimate of CPU delay*

In this paper, we assume that a CPU request is serviced only when there are no channel requests. Channel service has a higher priority than CPU service, and nonpreemptive service discipline is assumed [1]. If a CPU service is delayed, the delay time is equivalent to the length of time that the system is busy servicing the channels. The delay ends when the system is free again to service CPU requests and there are no channel requests pending.

The exact mathematical solution for the CPU delay is rather complicated, since such a solution requires the transient analysis of the system. In the following paragraphs, we derive an expression for the mean CPU delay assuming that a stationary condition exists.

Let $G_{nk}(x)$ be the probability that the delay period consists of at least n channel services, the total service time of the first n batches is at most x, and at the end of the nth service k calls are present in the queue (m calls = 1 channel request). Then

$$G_n(x) = \sum_{k=0}^{m-1} G_{nk}(x)\,.$$

If $G_n(x)$ can be determined, then the conditional probability $G(x)$, the CPU delay distribution given that a CPU request is delayed, can be found by

$$G(x) = \sum_{n=1}^{\infty} G_n(x)\,.$$

Let the Laplace-Stieltjes transform be

$$\Gamma_{nk}(s) = \int_0^\infty e^{-sx} dG_{nk}(x)\,.$$

Then

$$\Gamma_n(s) = \sum_{k=0}^{m-1} \Gamma_{nk}(s)\,.$$

For $n = 1$, i.e., the delay period consists of at least one channel service, we have

$$G_{1k}(x) = \int_0^x e^{-\lambda y}[(\lambda y)^k/k!]\,dH(y) * B(y)\,, \qquad (16)$$

where the notation $*$ denotes convolution.

$G_{nk}(x)$ can be obtained from Takacs [9] as follows:

$$G_{nk}(x) = \sum_{r=m}^{m+k} \int_0^x G_{n-1r}(x-y)e^{-\lambda y}[(\lambda y)^{k-r+m} \div (k-r+m)!]\,dH(y)\,. \qquad (17)$$

Taking the Laplace-Stieltjes transforms of Eqs. (16) and (17), we have

$$\Gamma_{1k}(s) = \int_0^\infty e^{-(\lambda+s)y}[(\lambda y)^k/k!]\,dH(y) * B(y) \qquad (18)$$

and

$$\Gamma_{nk}(s) = \sum_{r=m}^{m+k} \Gamma_{n-1r}(s) \int_0^\infty e^{-(\lambda+s)y}[(\lambda y)^{k-r+m} \div (k-r+m)!]\,dH(y)\,. \qquad (19)$$

If we introduce the generating function

$$C_n(s,z) = \sum_{k=0}^{\infty} \Gamma_{nk}(s) z^k\,,$$

then we have from Eqs. (18) and (19)

$$C_1(s,z) = \psi(s+\lambda(1-z))\;\beta(s+\lambda(1-z))$$

$$z^m C_n(s,z) = \psi(s+\lambda(1-z))\left[C_{n-1}(s,z) - \sum_{r=0}^{m-1}\Gamma_{n-1r}(s)z^r\right](n = 2, 3, \cdots).$$

Hence, the generating function is:

$$\sum_{n=1}^{\infty} C_n(s,z)w^n = w\psi(s+\lambda(1-z))\left[z^m\beta(s+\lambda(1-z)) - \sum_{n=1}^{\infty}\sum_{r=0}^{m-1}\Gamma_{nr}(s)z^r w^n\right] \div [z^m - w\psi(s+\lambda(1-z))]\,. \qquad (20)$$

The left hand side of Eq. (20) is a regular function of z if $|z| \le 1$, $\mathscr{R}(s) > 0$, and $|w| < 1$. In this domain the denominator of the right hand side has exactly m roots,

$z = \mu_r(s, w)$, $(r = 1, 2, \cdots, m)$. These must also be roots of the numerator. Thus the numerator, which is a polynomial of degree m, is determined uniquely,

$$z^m \beta(s + \lambda(1 - z)) - \sum_{n=1}^{\infty} \sum_{r=0}^{m-1} \Gamma_{nr}(s)\, z^r\, w^n = \prod_{r=1}^{m} [z - \mu_r(s, w)]\,. \tag{21}$$

Since we are interested in a stationary solution for the delay distribution, we can put $z = 1$ in Eq. (21),

$$\sum_{n=1}^{\infty} \Gamma_n(s)\, w^n = \beta(s) - \prod_{r=1}^{m} [1 - \mu_r(s, w)] \tag{22}$$

for $|w| < 1$, and this is also true for $|w| = 1$, which can be shown by analytical continuation. If we make $w = 1$ in Eq. (22), we have

$$\Gamma(s) = \sum_{n=1}^{\infty} \Gamma_n(s) = \beta(s) - \prod_{r=1}^{m} [1 - \mu_r(s)]\,, \tag{23}$$

where $\mu_r(s)$ is the root with an absolute value less than or equal to 1 for $\mathscr{R}(s) \geq 0$ of the equation

$$[\mu_r(s)]^m = \psi(s + \lambda(1 - \mu_r(s))) \qquad r = 1, 2, \cdots, m\,. \tag{24}$$

Since the CPU services are not delayed when the system is free from channel service, we have

$$Q(0) = \sum_{j=0}^{m-1} P_j \tag{25}$$

as the probability that a CPU request is not delayed. If a CPU service is delayed, then the delay distribution $G(x)$ is determined by inverting $\Gamma(s)$ from Eq. (23):

$$G(x) = \frac{1}{2\pi j} \int_{-\infty}^{\infty} \Gamma(s)\, e^{sx}\, ds\,. \tag{26}$$

Let G_1 be the mean CPU delay time for those CPU services delayed by the channel operation. By definition,

$$G_1 = \int_0^{\infty} x\, dG(x)\,. \tag{27}$$

Let C_1 be the mean busy time of the system as the CPU services instructions. Since CPU instructions can be processed when the system is free from the channel service, we can determine C_1 from the relation,

$$Q_0 = C_1/(C_1 + G_1)\,. \tag{28}$$

The average number of CPU instructions serviced during C_1 is C_1/d.

The mean CPU delay time can be found by taking the derivative of Eq. (23) and then letting $s = 0$:

$$G_1 = b + \sum_{r=1}^{m} (-1)\, \mu_r'(0) \prod_{r \neq k} [1 - \mu_k(0)]\,. \tag{29}$$

Note that when $s = 0$, Eq. (24) yields

$$\mu_1(0) = \omega_1,\ \mu_2(0) = \omega_2, \cdots, \mu_m(0) = \omega_m = 1\,,$$

which were used before to determine P_j for $j = 0, 1, \cdots, m - 1$.

Since $\omega_m = 1$, Eq. (29) can be simplified into

$$G_1 = b + (-1)\, \mu_m'(0) \prod_{r \neq m} (1 - \omega_r)\,, \tag{30}$$

where $\mu_m'(0)$ can be found by taking the derivative of Eq. (24) as

$$m[\mu_r(s)]^{m-1}\, \mu_r'(s) = \psi'(s + \lambda(1 - \mu_r(s)))[1 - \lambda\mu_r'(s)]\,. \tag{31}$$

With $r = m$ and $s = 0$, we have

$$-\mu_m'(0) = h/(m - \lambda h)\,. \tag{32}$$

Finally, we have the mean CPU delay due to channel interference:

$$G_1 = b + \prod_{r<m} (1 - \omega_r) h/(m - \lambda h)\,. \tag{33}$$

If the variance of the CPU delay is needed, we must have an expression for the second moment of the CPU delay time distribution, i.e., G_2. This can be obtained from the second derivative of Eq. (28) as

$$G_2 = b_2 + \mu_m''(0) \prod_{k=1}^{m-1} (1 - \omega_k) + (-\mu_m'(0)) \left[\sum_{r=1}^{m-1} (-\mu_r'(0)) \right] \prod_{k \neq r}^{m-1} (1 - \omega_k)\,. \tag{34}$$

From Eq. (31),

$$\mu_r'(0) = \psi'(\lambda(1 - \omega_r))/[m\omega_r^{m-1} + \lambda\psi'(\lambda(1 - \omega_r))] \qquad \text{for } r = 1, 2, \cdots, m\,. \tag{35}$$

Taking the derivative of Eq. (31), and setting $r = m$, we obtain

$$\mu_m''(0) = [m^2 h_2 - m(m - 1)h^2]/(m - \lambda h)^3\,. \tag{36}$$

Substituting Eqs. (35) and (36) into Eq. (34), we obtain G_2.

Numerical examples

Two numerical examples are used to illustrate how the model can be applied to the analysis of some computer system problems. In the first example, the channel interference problem for a small computer system is examined. The second example shows how the model can be used to analyze an interference problem for large systems – the interference of real-time jobs with batch jobs.

• *Computer channel interference*

A small computer executes its instructions sequentially, as shown in Fig. 5. Because execution of an instruction

takes a variable amount of time, i.e., some instructions require two memory cycles, some require four, etc., the service time for an instruction can be treated as a random variable. Let X be this variable. By the definition of the model,

$$D(x) = P\{X \leq x\}.$$

At τ_1, the first channel request arrives. The computer in this example completes execution of the current instruction, prepares the system to service the channel, and switches to the channel service mode. The channel request is serviced by one of several channel programs consisting of a sequence of CPU and channel instructions. Since different I/O requests require different system actions, execution time of a channel program is another random variable, Z. Therefore,

$$H(x) = P\{Z \leq x\}.$$

After the channel request has been serviced, the computer determines whether there are other channel requests pending. If there are, the system continues to service channel requests. If there are not, the system switches back to the CPU mode. Let Y_1 be the time required to prepare the system to return from the channel mode to the CPU mode. This time is needed to restore the CPU registers, which may have been used by the channel program. Let Y_2 be the time to prepare the system to service a channel request. This time is needed to store the current CPU information so that when the interrupt has been cleared, the system can resume normal processing. Assuming that Y_1 and Y_2 are small compared to Z, we can simplify the problem somewhat by combining them into one overall "overhead" time Y, which is the sum of Y_1 and Y_2. Then,

$$B(x) = P\{Y_1 + Y_2 \leq x\} = P\{Y \leq x\}.$$

If Y_1 and Y_2 are not small, we can extend the mathematical model to cover two switching time distributions, as discussed in Appendix B.

Assume that the arrival of channel requests follows an Erlang-2 distribution with $\lambda = 0.1$, $m = 2$.

Let the execution time distribution for the CPU instructions be Erlang-3 with a mean service time of 1 time unit.

$$d = 1,\ m = 3,$$

$$D(x) = 1 - \sum_{j=0}^{m-1} e^{-(m/d)x}\,[(mx/d)^j/j!]$$

$$= 1 - \sum_{j=0}^{2} e^{-3x}\,(3x)^j/j!,$$

$$d_2 = [(2+3-1)!/(3-1)!](1/3)^2 = 4/3,$$

$$\delta(s) = [3/(s+3)]^3,$$

$$\delta'(s) = 3[3/(s+3)]^2\,[-3/(s+3)^2],$$

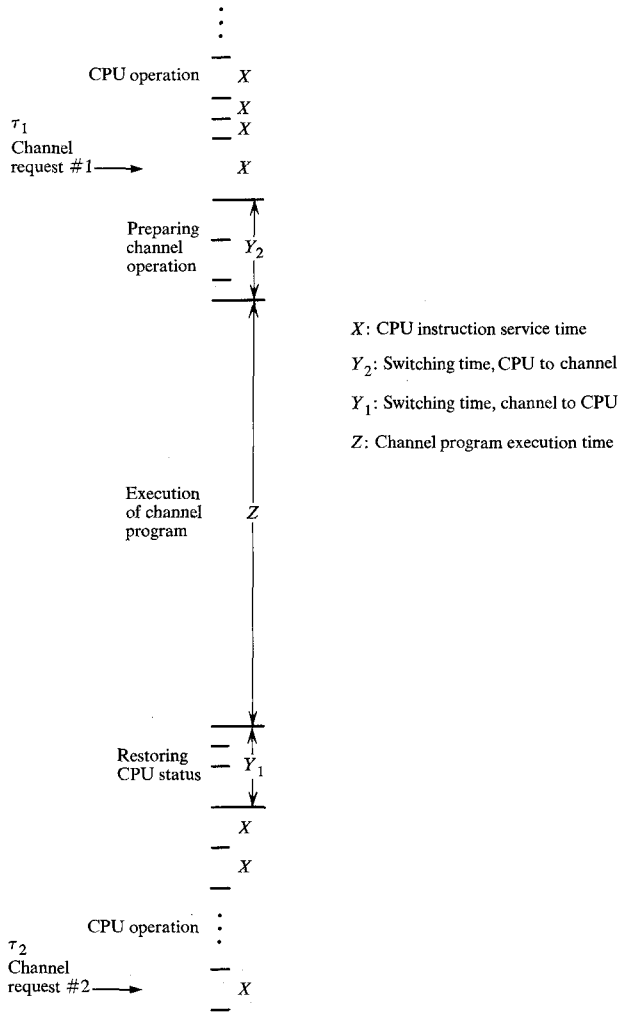


Figure 5 Computer execution sequences.

$$\delta(\lambda) = 27/(3.1)^3 = 0.903,$$

$$\delta'(\lambda) = -0.873.$$

Let the switching time $(Y = Y_1 + Y_2)$ be a constant of six time units. The Laplace-Stieltjes transform is

$$\beta(s) = e^{-6s}.$$

The moments are

$$b = 6 \quad \text{and} \quad b_2 = 36.$$

Let the channel program service times be Erlang-2 distributed with a mean of ten time units. Then

$$h = 10,\ m = 2,$$

$$H(x) = 1 - e^{-0.2x} - 0.2x\,e^{-0.2x},$$

$$h_2 = 150,$$

$$\psi(s) = [0.2/(s+0.2)]^2.$$

The problem is to determine the waiting time distribution and the mean waiting time of the channel request. The Laplace-Stieltjes transform of the waiting time distribution is given by Eq. (14), which is the product of two transforms. The first one is identical to the Laplace-Stieltjes transform of the waiting time distribution of a single-server queue with Erlang-2 input and a general service time distribution of $H(x)$. Denote it by $\Omega_E(s)$. Then,

$$\Omega_E(s) = [(2-\lambda h)\lambda(-s)(1-s/s_1)] \div [(-s+\lambda)^2 - \lambda^2\psi(s)]$$
$$= [0.1(-s)(1-s/s_1)(s+0.2)^2] \div [(-s+0.1)^2(s+0.2)^2 - 0.01(0.2)^2],$$

where s_1 is a root to be determined from the denominator equation, which is

$$(-s+0.1)^2(s+0.2)^2 - 0.0004 = 0.$$

This equation yields

$$s(s^3 + 0.2\,s^2 - 0.03\,s - 0.004)$$
$$\approx s(s-0.155)(s^2 + 0.355\,s + 0.025) = 0.$$

Thus, $s_1 = 0.155$.

Substituting the value of s_1 and simplifying, we have

$$\Omega_E(s) = [0.645(s^2 + 0.4\,s + 0.04)] \div (s^2 + 0.355\,s + 0.025).$$

The first moment can be found by

$$-\Omega_E'(0) = -0.645[(0.025)(0.4) - (0.04)(0.355)] \div (0.025)^2$$
$$= 4.34 \text{ time units.}$$

The second Laplace-Stieltjes transform from Eq. (14) is

$$V(s) = \{[\varphi_2(s)\,\beta(s) - 1] + (R_1/\lambda)(-s+\lambda)[\varphi_1(s)\,\beta(s) - 1]\} \div \{[a_{21} + b + R_1(a_{11} + b)](-s)(1 - s/s_1)\}.$$

Let $V(s)$ be written in the form

$$V(s) = N(s)/D(s).$$

Since $V(0) = 1$, from L'Hopital's rule we have

$$-V'(0) = [V(0)D''(0) - N''(0)]/[2D'(0)]$$
$$= \{a_{22} + 2\,a_{21}b + b_2 + (R_1/\lambda)[\lambda(a_{12} + 2\,a_{11}b + b_2) + 2(a_{11} + b)] - [a_{21} + b + R_1(a_{11} + b)](2/s_1)\} \div \{2[a_{21} + b + R_1(a_{11} + b)]\},$$

where

$$R_1 = \lambda[1 - \varphi_2(s_1)\,\beta(s_1)] \div \{(-s_1 + \lambda)[\varphi_1(s_1)\,\beta(s_1) - 1]\}.$$

Substituting the numerical values given by this example and using the formulas given in Appendix A, we have

$$\beta(s_1) = e^{-6(0.155)} = 0.395,$$

$$\varphi_1(s_1) = -0.050/0.097 = -0.515,$$

$$\varphi_2(s_1) = (-0.515)(0.055)(-0.873)/0.097 + 0.853 - (0.055)(-0.873) - 0.903$$
$$= 0.252.$$

Thus, $R_1 = 1.35$.

In addition,

$$a_{11} = 10.3,$$

$$a_{12} = 13.35,$$

$$a_{21} = 19.6, \quad \text{and}$$

$$a_{22} = 187.35.$$

From these values, we find

$$-V'(0) = 520.35/95.2 = 5.5.$$

Finally, the mean waiting time at the channel for the Erlang-2 input is

$$W_1 = -\Omega_E'(0) - V'(0) = 4.34 + 5.5 = 9.84.$$

The mean delay to CPU instruction processing due to channel interference can be computed as follows. First, we determine P_0 from Eq. (8) or from Eq. (14) as

$$P_0 = (2 - \lambda h)/\{\lambda[a_{21} + b + R_1(a_{11} + b)]\}$$
$$= 1/4.76 = 0.21.$$

P_1 can be determined as

$$P_1 = R_1\,P_0 = 1.35(0.21) = 0.284.$$

The probability that the channel queue is empty is

$$Q_0 = P_0 + P_1 = 0.494.$$

The root ω_1 can be determined from s_1 as

$$\omega_1 = (\lambda - s_1)/\lambda = -0.55.$$

The mean delay to CPU instruction processing is determined from Eq. (33) as

$$G_1 = 6 + 1.55(10)/(2-1) = 21.5$$

It is of interest to compare these results with the case of an equivalent Poisson input. Assuming that we have the same channel input rate in the Poisson case, we have $\lambda = 0.05$ and $m = 1$.

Assuming that the service time distributions are the same, we have

$$\delta(\lambda) = (3/3.05)^3 = 0.95\,,$$

$$a_{11} = 20\,, \quad \text{and}$$

$$a_{12} = 26.66\,.$$

The mean waiting time for the channel is

$$W_1 = 0.05(150)/[2(1 - 0.05 \times 10)] + (26.66 + 36 + 2 \times 20 \times 6) \div [2(20 + 6)] = 7.5 + 5.8 = 13.4\,.$$

For the Poisson input case, the probability that the channel is empty is

$$P_0 = (1 - \lambda h)/[\lambda(a_{11} + b)] = 0.385\,.$$

The mean delay to CPU instruction processing due to channel interference is

$$G_1 = 6 + (10/0.5) = 26\,.$$

Thus the mean waiting time at the channel and the mean delay to the CPU are all higher than for Erlang-2 input.

• *Computer task interference*

Consider a computer system processing two types of jobs. The first type consists of real-time tasks, which must be processed on a higher priority basis, and the second consists of batch jobs, which can be processed by the CPU on an as-available basis. Since some processing time may be lost in switching between job types, it is desirable to maintain a balance in some systems between real-time processing and batch processing that will minimize the time spent in switching.

Consider that real-time messages arrive randomly at 3 messages/second. Suppose that it takes a computer system an average of 200 ms to process a message with a service time distribution of Erlang-3 type. Thus, we have

$$\lambda = 3, \qquad m = 1 \qquad \text{(Poisson input)}\,,$$

$$h = 0.2, \qquad h_2 = [(3 + 1)/3]\, h^2 = (4/3)(0.04)\,.$$

Suppose that an average real-time response is satisfactory to the customer if the mean response time T for the messages is 500 ms. Thus, the mean waiting time satisfactory to the customer is

$$W_1 = T - h = 300 \text{ ms}\,.$$

Suppose further that the switching time is a constant of 30 ms. Assume that we can break up a batch job into small tasks and that each task takes d seconds to complete. The problem here is to determine the time to allow the batch task to proceed (batch task service time) for such a message rate. In this case, we can use the Poisson interference model to obtain a solution:

$$b = 0.03, \qquad b_2 = 0.0009\,,$$

$$\delta(s) = e^{-ds}, \qquad d_2 = d^2, \qquad \delta(\lambda) = e^{-3d}\,.$$

From the mean waiting time formula based on Poisson input, we have

$$W_1 = \lambda h_2/[2(1 - \lambda h)] + [d^2 + 2db + b^2(1 - \delta(\lambda))] \div [2\,d + 2\,b(1 - \delta(\lambda))]\,.$$

Substituting the numerical values, we have

$$0.3 = 0.2 + [d^2 + 2(0.03)\,d + 0.0009(1 - \delta(\lambda))] \div [2\,d + 0.06(1 - \delta(\lambda))]\,.$$

We can determine the value of d numerically from the above equation. Since the switching time here is rather small compared to d, the first approximation for d is $0.1 = d/2$. Thus, $d = 0.2$.

Using this value, we obtain

$$\delta(\lambda) = e^{-0.6} = 0.55\,.$$

Substituting this value into the equation for a second iteration, we have

$$0.1 = [d^2 + 0.06\,d + 0.0009(0.45)] \div [2\,d + 0.06(0.45)]\,,$$

which leads to

$$d^2 - 0.14\,d - 0.0023 = 0\,.$$

The root for this equation is

$$d = 0.156\,.$$

If we substitute this value for another iteration, we find

$$d = 0.152\,.$$

We conclude that if we allow the batch job to be processed for a constant 152 ms, during which no interruption is allowed for real-time processing, we can maintain a message response time satisfactory to the customer's real-time processing requirement.

Acknowledgment

I would like to thank P. H. Seaman of the IBM Data Processing Division for many helpful comments.

References and Footnotes

1. W. Chang, "Single-server queuing processes in computer systems," *IBM Systems Journal* **9**, No. 1, 36–71 (1970).
2. W. Chang and D. J. Wong, "Computer channel interference analysis," *IBM Systems Journal* **4**, No. 2, 164–170 (1965).
3. "The Structure of System/360," *IBM Systems Journal* **3**, 2 (1964).

4. L. Takacs, "Priority Queues," *Operations Research* **12,** No. 1, 63–74 (1964).
5. P. H. Seaman, R. A. Lind and T. L. Wilson, "On Teleprocessing System Design–Diskfile System Analysis," *IBM Systems Journal* **7,** No. 3 (1968).
6. F. Pollaczek, "Ueber eine Aufgabe der Wahrscheinlichkeitstheorie I," *Mathematische Zeitschrift* **32,** 64–100 (1930); ibid II, **32,** 729–750 (1930).
7. F. Pollaczek, "Ueber das Warteproblem," *Mathematische Zeitschrift* **38,** 429–537 (1934).
8. A. J. Fabens, "The Solution of Queuing and Inventory Models," Technical Report No. 20, Dec. 7, 1959, Dept. of Statistics, Stanford University.
9. L. Takacs, "Transient Behavior of Single-server Queuing Processes with Erlang Input," *Trans. American Math. Soc.* **100,** 1–28 (1961).
10. D. R. Cox and W. L. Smith, "The Superposition of Several Strictly Periodic Sequences of Events," *Biometrika* **40,** 1–11 (June 1953).
11. Let ρ be the utilization factor of the main memory for the service of the cycle-steal requests. Let x be a given service time when the main storage is entirely available. Let x be the extended service time, which includes the cycle-steal operation, then x can be computed as

$$x = x* + \lambda_1 x \quad c + \lambda_1(\lambda_1 x* c)\ c + \cdots$$
$$= x*\,(1 + \rho + \rho^2 + \cdots)$$
$$= x*/(1-\rho)\,,$$

where λ_1 is the rate of the cycle-steal requests, c is the cycle time, and $\rho = \lambda_1 c$.
12. J. Riordan, *Stochastic Service Systems*, John Wiley and Sons, New York, New York (1962).
13. W. L. Smith, "Renewal Theory and Its Ramifications," *J. Roy. Statist. Soc. Ser. B*, **Vol. 20,** 243–302 (1958).
14. P. D. Welch, "On a General M/G/1 Queuing Process in which the first customer of each busy period receives exceptional service," *Operations Research* **12,** No. 5, (1964).
15. L. Takacs, *Introduction to the Theory of Queues*, Oxford University Press, New York, New York, 1962.
16. L. Takacs, "The limiting distribution of the virtual waiting time and the queue size for a single-server queue with recurrent input and general service times," *Sankhya, the Indian Journal of Statistics*, Series A, **Vol. 25,** Part 1 (1963).

Received July 26, 1972

The author is located at the IBM Data Processing Division, Poughkeepsie, New York 12602.

Appendix A: Derivation of the generating function

The expression for $\varphi_j(\lambda(1-z))$ can be determined step by step. First, if after the nth departure, there are exactly $m-1$ calls left in the queue, we need one more arrival to start the $(n+1)$th service. Thus, the condition

$$\gamma_n \geq 1$$

must be met for the $(n+1)$th channel service to be started. Such an event can occur in the following way. After the nth service, the system returns to CPU processing. Assume that i services of CPU requests have been completed and that there are no arrivals in the queue, but that during the $(i+1)$th CPU service, one or more calls arrive.

The probability of no arrivals during a CPU service is

$$P\{\gamma_n = 0\} = \int_0^\infty e^{-\lambda x}\, d\,D(x) = \delta(\lambda)\,. \tag{A1}$$

The generating function for one or more calls arriving during a CPU service is expressed by

$$\sum_{k=1}^{\infty} P\{\gamma_n = k\}\, z^k = \sum_{k=1}^{\infty} z^k \int_0^\infty [(\lambda x)^k/k!]\, e^{-\lambda x}\, d\,D(x)$$
$$= \delta(\lambda(1-z)) - \delta(\lambda)\,. \tag{A2}$$

Thus, considering all possible values of i initial services, the generating function for one arrival after the nth departure is

$$\varphi_1(\lambda(1-z)) = \sum_{i=0}^{\infty} \left[\int_0^\infty e^{-\lambda x}\, d\,D(x)\right]^i \sum_{k=1}^{\infty} z^k \int_0^\infty [(\lambda x)^k \div k!]\, e^{-\lambda x}\, d\,D(x)$$
$$= \sum_{i=0}^{\infty} [\delta(\lambda)]^i\, [\delta(\lambda(1-z)) - \delta(\lambda)]$$
$$= [\delta(\lambda(1-z)) - \delta(\lambda)]/[1-\delta(\lambda)]\,. \tag{A3}$$

Next, suppose that after the nth departure $m-2$ calls are left in the queue. The $n+1$th service cannot begin until at least two more calls arrive in the queue. Then,

$$\varphi_2(\lambda(1-z)) = \sum_{i=1}^{\infty} i[\delta(\lambda)]^{i-1} \left[z \int_0^\infty (\lambda x)\, e^{-\lambda x}\, d\,D(x)\right]$$
$$\times \sum_{k=1}^{\infty} z^k \int_0^\infty [(\lambda x)^k/k!]\, e^{-\lambda x}\, d\,D(x)$$
$$+ \sum_{i=0}^{\infty} [\delta(\lambda)]^i \sum_{k=2}^{\infty} z^k \int_0^\infty [(\lambda x)^k \div k!]\, e^{-\lambda x}\, d\,D(x)$$
$$= [\delta(\lambda(1-z)) - \delta(\lambda)](-\lambda)\,\delta'(\lambda)\, z \div [1-\delta(\lambda)]^2$$
$$+ [\delta(\lambda(1-z)) - \delta(\lambda) - (-\lambda z)\delta'(\lambda)] \div [1-\delta(\lambda)]\,. \tag{A4}$$

Equation (A4) is obtained by considering two mutually exclusive situations. The first is that there is one arrival during the initial i CPU services, which may occur during any one of the services, leaving no arrivals occurring during the remaining $i-1$ service times. This is followed by one or more arrivals during the $(i+1)$th CPU service. The queue length is now increased to more than m calls and the next service will be a channel operation. The second situation is that there are no arrivals during the first i CPU services, but at least two more calls arrive during the $(i+1)$th CPU service.

Note that

$$\int_0^\infty (\lambda x)\, e^{-\lambda x}\, d\,D(x) = (-\lambda) \int_0^\infty (-x)\, e^{-\lambda x}\, d\,D(x)$$
$$= (-\lambda)\, \delta'(\lambda)$$

and in general

$$\int_0^\infty (\lambda x)^k\, e^{-\lambda x}\, d\,D(x) = (-\lambda)^k \int_0^\infty (-x)^k\, e^{-\lambda x}\, d\,D(x)$$
$$= (-\lambda)^k\, \delta^{(k)}(\lambda)\,. \qquad \text{(A5)}$$

Equation (A4) can also be written by comparison with Eq. (A3) as

$$\varphi_2(\lambda(1-z)) = \varphi_1(\lambda(1-z))(-\lambda z)\, \delta'(\lambda)/[1-\delta(\lambda)]$$
$$+ [\delta(\lambda(1-z)) - \delta(\lambda) - (-\lambda z)\, \delta'(\lambda)]$$
$$\div [1-\delta(\lambda)]\,. \qquad \text{(A6)}$$

In general, $\varphi_j(\lambda(1-z))$, for $j = 2, 3, \cdots, m$, can be obtained from the following recursive formula:

$$\varphi_j(\lambda(1-z)) = \sum_{k=1}^{j-1} \varphi_k(\lambda(1-z))[(-\lambda z)^{j-k}$$
$$\div (j-k)!]\, \delta^{(j-k)}(\lambda)/[1-\delta(\lambda)]$$
$$+ [\delta(\lambda(1-z)) - \sum_{k=0}^{j-1} [(-\lambda z)^k$$
$$\div k!]\, \delta^{(k)}(\lambda)]/[1-\delta(\lambda)]\,. \qquad \text{(A7)}$$

Equation (A7) can be explained as follows: To have at least j new arrivals during the CPU services, either of two conditions may exist. Exactly k new calls ($k < j$) may have arrived during the initial CPU services and at least $j-k$ calls during the last CPU service. Or, there may have been no new calls during any CPU service except the last, during which j or more calls arrived.

Putting $s = \lambda(1-z)$ into Eq. (A7), we have

$$\varphi_j(s) = \sum_{i=1}^{j-1} \varphi_i(s)[(s-\lambda)^{j-i}/(j-i)!]\, \delta^{(j-i)}(\lambda)$$
$$\div [1-\delta(\lambda)]$$
$$+ \left[\delta(s) - \sum_{i=0}^{j-1} [(s-\lambda)^i/i!]\, \delta^{(i)}(\lambda)\right]$$
$$\div [1-\delta(\lambda)]\,. \qquad \text{(A8)}$$

Taking the derivatives of (A8) and putting $s = 0$, we have the moments

$$a_{11} = -\varphi_1'(0) = d/[1-\delta(\lambda)]\,, \qquad \text{(A9)}$$

$$a_{12} = \varphi_1''(0) = d_2/[1-\delta(\lambda)]\,, \qquad \text{(A10)}$$

$$a_{13} = -\varphi_1'''(0) = d_3/[1-\delta(\lambda)]\,, \qquad \text{(A11)}$$

$$a_{21} = -\varphi_2'(0) = [d - \lambda(-\varphi_1'(0))\, \delta'(\lambda)]/[1-\delta(\lambda)]\,, \qquad \text{(A12)}$$

$$a_{22} = \varphi_2''(0) = [\varphi_1''(0)(-\lambda) - 2(-\varphi_1'(0))]\, \delta'(\lambda)$$
$$\div [1-\delta(\lambda)]$$
$$+ d_2/[1-\delta(\lambda)]\,, \qquad \text{(A13)}$$

$$a_{23} = -\varphi_2'''(0) = [-\varphi_1'''(0)(-\lambda) - 3\varphi_1''(0)]\, \delta'(\lambda)$$
$$\div [1-\delta(\lambda)]$$
$$+ d_3/[1-\delta(\lambda)]\,. \qquad \text{(A14)}$$

Appendix B: An extended model with two switching time distributions

In the main body of this paper, we assumed that the switching times are small compared with the channel service times and they can be combined into one switching time function and studied in the model. However, the mathematical model can be modified so that two switching time distributions can be included in the analysis. Here we discuss briefly how this can be achieved.

Let $B_1(x)$ and $B_2(x)$ be the two switching time distributions from the channel mode to the CPU mode and from the CPU mode to the channel mode, respectively. Let $\beta_1(s)$ and $\beta_2(s)$ be their Laplace-Stieltjes transforms. $B_2(x)$ can be easily included in the analysis by replacing $\beta(\lambda(1-z))$ in Eq. (6) by $\beta_2(\lambda(1-z))$. However, the generating function $\varphi_{m-j}(\lambda(1-z))$ will be more complicated, since the effect of $B_1(x)$ must be included in the analysis.

The problem can be treated as follows. Assume that when the system switches from the channel mode to the CPU mode, at least one CPU service is provided. Since a CPU service is in general very short in duration, this will not cause any significant error in the analysis. In this case, we can consider that the first CPU service after the system has switched from the channel mode has a special service time distribution that includes the switching time needed to prepare the system to serve the subsequent CPU requests. Thus, the first CPU service time distribution is a convolution of two distributions, $B_1(x) * D(x)$. The next and subsequent CPU requests have a normal CPU service time distribution, $D(x)$.

The problem is now reduced to determining a set of new generating functions, $\varphi^*_{m-j}(\lambda(1-z))$, which allows the first CPU request to have a special service time distribution.

Consider the case that after the nth channel service, there are exactly $m-1$ calls left in the queue. The $(n+1)$th channel service can be processed if there is at least one more arrival. After the nth channel service, the system returns to CPU service mode. There are two possibilities. During the first CPU service period, which includes the switching time from the channel mode to the CPU mode, there are no arrivals but there is at least

one arrival during the subsequent CPU services, or there is at least one arrival during the first CPU service period and the system is to switch back to the channel mode.

The probability that there are no arrivals during the first CPU service period is

$$P\{\gamma_n = 0\} = \int_0^\infty e^{-\lambda x}\, d[B_1(x) * D(x)] = \beta_1(\lambda)\ \delta(\lambda)\,. \tag{B1}$$

The generating function that there is at least one arrival during the first CPU service period is

$$\begin{aligned}\sum_{k=1}^{\infty} P\{\gamma_n = k\}\ z^k &= \sum_{k=1}^{\infty} \int_0^\infty [(\lambda x)^k/k!]\ e^{-\lambda x}\, d[B_1(x) \\ &\quad * D(x)] \\ &= \beta_1(\lambda(1-z))\ \delta(\lambda(1-z)) \\ &\quad - \beta_1(\lambda)\ \delta(\lambda)\,. \end{aligned} \tag{B2}$$

Thus $\varphi_1^*\ (\lambda(1-z))$ is obtained as

$$\begin{aligned}\varphi_1^*\ (\lambda(1-z)) &= \beta_1(\lambda)\ \delta(\lambda)\ \varphi_1(\lambda(1-z)) \\ &\quad + \beta_1(\lambda(1-z))\ \delta(\lambda(1-z)) \\ &\quad - \beta_1(\lambda)\ \delta(\lambda)\,, \end{aligned} \tag{B3}$$

where $\varphi_1(\lambda(1-z))$ is given by Eq. (A3).

In general, we obtain $\varphi_j^*\ (\lambda(1-z))$ by the following equation:

$$\begin{aligned}\varphi_j^*\ (\lambda(1-z)) &= \beta_1(\lambda)\ \delta(\lambda)\ \varphi_j(\lambda(1-z)) \\ &\quad + \sum_{i=1}^{j-1} [(-\lambda)^i/i!]\ \beta_1^{(i)}(\lambda)\ \delta^{(i)}(\lambda)\,. \\ &\qquad \varphi_{j-i}(\lambda(1-z)) \\ &\quad + \beta_1(\lambda(1-z))\ \delta(\lambda(1-z)) \\ &\quad - \sum_{i=0}^{j-1} [(-\lambda)^i/i!]\ \beta_1^{(i)}(\lambda)\ \delta^{(i)}(\lambda)\,. \end{aligned} \tag{B4}$$

The generating function $\varphi_j^*\ (\lambda(1-z))$ in Eq. (B4) is obtained by considering all possibilities. First, there are no arrivals during the first CPU service period but there are at least j arrivals during subsequent CPU services. Second, there are exactly i arrivals during the first CPU service period, but there are at least $j-i$ arrivals during subsequent CPU services for all possible i values. Finally, there are more than j arrivals during the first CPU service period.

The queue-size generating function that includes two switching time distributions is given as follows:

$$\begin{aligned}U(z) &= \sum_{j=0}^{m-1} P_j\, z^j [\varphi_{m-j}^*\ (\lambda(1-z))\ \beta_2(\lambda(1-z)) \\ &\quad - 1]\ \psi(\lambda(1-z)) \\ &\quad \div [z^m - \psi(\lambda(1-z))]\,. \end{aligned} \tag{B5}$$

The waiting time and queuing time formulas can be similarly obtained.