MODELS OF MEMORY SCHEDULING

A. K. Agrawala
R. M. Bryant
University of Maryland

Queueing theoretic models of single and multi-processor computer systems have received wide attention in the computer science literature. Few of these models consider the effect of finite memory size of a machine and its impact on the memory scheduling problem. In an effort to formulate an analytical model for memory scheduling we propose four simple models and examine their characteristics using simulation. In this paper, we discuss some interesting results of these simulations.

Key Words and Phrases: multiprogramming, scheduling, memory fragmentation, swapping systems, dynamic memory allocation, first-fit, performance evaluation, computer system simulation, analytical models.

CR Categories: 4.3.2, 8.1

I. INTRODUCTION.

Computer science literature contains a large number of reports on analytical and simulation models of computer systems. Queueing theory has often been used in formulating these models. As a system usually has one or a few processors, a queueing theoretic model with a single server or a few servers can be used. Various realistic scheduling disciplines may then be studied.

For any program to execute in a system, in addition to access to the CPU it also requires access to memory. Therefore, the operating system has to handle memory scheduling along with the scheduling of the processors. Unlike the CPU scheduling in which a CPU may or may not be allocated to a program, a number of programs may be occupying the main memory of the system. This situation is not easily solvable by queueing theoretic methods.

The question of memory scheduling has received little attention in the literature. Such attention has been focused on the analysis of paging systems for which a number of models and disciplines have been proposed and analyzed. Most of these models concentrate on the time varying memory requirements of a program rather than the question of overall memory scheduling.

Currently the best results in this area are the so-called fifty percent rule due to Knuth [1] and some inequalities involving unused memory and compaction [2]. But these results only provide a qualitative analysis.

This paper reports on initial work aimed at determining qualitative and quantitative guidelines for memory scheduling disciplines in multiprogrammed systems. At first, we are considering a swapping environment in which a whole program has to be in core before its execution can be attempted.

It is interesting to note that in spite of a significant interest in paging systems in academic and research communities, a large number of systems in use today do not support paging. Several manufacturers offer only swapping machines, and in some machines, the operating system uses paging only to avoid external fragmentation while still operating under the swapping philosophy.

Our research effort is motivated by three types of questions:

(1) System Design. Given a workload characterization in the sense of arrival rates, memory requirements, I/O and CPU time distributions, approximately how much memory should a system have to process the workload?

(2) Memory Scheduling. In a heavily loaded system in which several users are competing for available core, which one should be loaded next? Are there general guiding rules such as shortest-processing-time first for the CPU scheduling models, which can significantly improve waiting time distributions and throughput?

(3) Quantitative Analysis of Placement Strategies. In a swapping system, determining where to place the next arrival in memory can be a very com-

This research was supported in part by the Mathematics and Information Sciences Directorate, Air Force Office of Scientific Research, Air Force Systems Command, USAF, under grant AFOSR 71-1982, and in part by the Control and Automation Branch, Engineering Division, National Science Foundation, under grant GK-41602.

plex task. Heuristics are usually employed to help solve the problem. Quantitatively how much better are such strategies than first fit, which Knuth endorses [1].

Our procedure for answering these questions consists of three basic steps. We first propose models of the simplest types of multiprogrammed systems in which memory size plays a significant role. Since even the simplest of such models are analytically difficult to solve, our second step consists of creating simulations of those models. From the simulation results, we hope to determine approximate answers to the questions above. The third step is to provide rigorous justification of the approximate results by formulating and solving suitable analytical models.

In this paper, the models used in the study are described in Section II. The simulation of these models led to a number of interesting results which are presented in Section III.

II. THE BASIC MODELS

In this study we considered four simple models of memory scheduling. In these models, a program once loaded remains in core until the program is terminated. In each model, the arrival process is assumed to be a Poisson process. At arrival time, each customer (program) is assigned a core size and CPU time. The core size is an integer distributed uniformly between 1 and 100 blocks. The CPU time is exponentially distributed and considered independent of memory size. (In some simulation runs we have also considered the effects of memory size and CPU time dependence.) If there is no memory queue and there is room in memory, the customer is loaded immediately. Otherwise, the customer is placed into the memory queue on either first-come-first-served basis or smallest-memory-size-first basis, depending on the scheduling discipline being used.

The first two models considered here are essentially models of I/O bound workloads. We assume there is no competition for the CPU, and the CPU time mentioned above is regarded as merely a memory residence time. As soon as a program's memory residence time has expired it is unloaded regardless of how many other programs are in core. The difference between these two models is that in the first model we do not consider memory fragmentation while in the second one we do. Therefore the first model is a model of a paged swapping environment in which a program may be loaded anywhere in the memory while the second is a model of a swapping machine with a single base register. A program requires contiguous space for loading in the second model. The placement strategy for the second model is first-fit starting at the bottom end of memory.

The third and fourth models which we consider are models of computation bound workloads. In these models, competition for use of the CPU determines how long each customer remains in memory. The CPU scheduling discipline is the processor sharing discipline described in [4]. This is the idealized discipline which results when the quantum of a round robin type system tends toward zero. We assume that sharing of the CPU only occurs among those programs loaded into memory. Thus, if at a given point in time there are n programs loaded and the remaining CPU time of a given program is c, then that program is scheduled to depart in nc time units. A new arrival (or departure) can change the value of n and cause rescheduling of the departure times of all currently loaded programs. The third model is again a model of a paged swapping type system, while the fourth model simulates memory fragmentation.

To distinguish between the two classes of models we have described, we will call the first two "memory-residence-time models" and the latter two "processor-sharing models".

The loading time of the programs is not considered explicitly in these models. At first glance, this seems to mean that we cannot model a round robin type system. The round robin schedules can be incorporated by artificially increasing the number of arrivals and decreasing the CPU time assigned to each arrival. Thus, if a five-second job of 50K words is to arrive in the round robin system, and the time slice is one second; we would model this single arrival as five 50K jobs of one second each. Neglecting swapping time, an observer stationed at the CPU cannot tell the difference between these two situations unless he is allowed to tag specific jobs.

III. SIMULATION RESULTS.

A. The Memory-Residence-Time Models.

The memory-residence-time models described above were executed with main memory sizes of 100, 200,...,1000 blocks. Each run was executed twice, once under the first-come-first-served discipline and once with the memory queue ordered by smallest memory size first. The arrival rate was chosen to be the quantity main memory size divided by fifty since fifty is the mean memory size of arriving customers. The service rate was chosen as unity. (These rates were chosen to guarantee a uniformly heavy overload on each of the models.)

A variety of statistics were collected during each simulation run. Tables I and II illustrate the most interesting of the statistics which we gathered. The quantities listed are:

- N_I The mean value of the time integral of the number of customers in memory at any given time.
- N_M The mean number in memory given that the memory queue was non-empty.
- M_I The mean value of the time integral of the number of blocks of memory in use at any given time.
- M_u Mean memory utilization calculated as the percentage M_I is of available core.

The first columns of each table list the values obtained under the FCFS discipline while the later columns give corresponding values for smallest-memory-size first (SMF).

Table I. Memory-residence-time paging model.

Core Size	FCFS N_I	FCFS N_M	FCFS M_I	FCFS M_u	SMF N_I	SMF N_M	SMF M_I	SMF M_u
100	1.39	1.70	71.2	71.2%	1.46	1.85	71.2	71.2%
200	3.22	3.60	164.4	82.2	3.40	3.89	164.6	82.3
300	5.14	5.47	265.2	88.4	5.61	6.14	265.2	88.4
400	7.09	7.40	364.3	91.7	7.71	8.15	365.7	91.4
500	9.10	9.35	467.0	93.4	9.84	10.24	466.5	93.3
600	11.10	11.36	565.0	94.2	10.98	11.30	545.9	94.5
700	13.00	13.29	664.1	94.9	14.12	14.48	665.3	95.0
800	14.94	15.14	673.9	95.5	15.33	15.70	763.3	95.4
900	16.90	17.11	866.9	96.3	17.15	17.56	863.6	95.9
1000	18.90	19.07	966.8	96.7	18.71	18.96	965.2	96.5

Table II. Memory-residence-time fragmentation model.

Core Size	FCFS N_I	FCFS N_M	FCFS M_I	FCFS M_u	SMF N_I	SMF N_M	SMF M_I	SMF M_u
100	1.33	1.55	66.9	66.9%	1.35	1.64	66.2	66.2%
200	2.67	2,68	136.6	68.3	2.64	2.76	138.6	69.3
300	4.30	4.32	220.4	73.4	4.23	4.30	217.3	72.4
400	5.88	5.88	303.8	78.5	6.30	6.46	305.2	76.3
500	7.47	7.49	383.5	76.7	7.93	8.09	382.0	76.4
600	9.12	9.20	466.8	77.8	9.64	9.77	462.6	77.1
700	10.83	10.90	556.7	79.5	11.31	11.49	543.7	77.7
800	12.68	12.79	651.2	81.4	13.10	13.24	631.5	78.9
900	13.92	13.94	712.6	79.0	15.20	15.42	725.8	80.6
1000	15.80	15.80	808.1	80.8	16.66	16.89	806.4	80.6

The first interesting feature of Tables I and II is that they illustrate the magnitude of wasted memory in a swapping system. While it is true that our model of memory size distribution is very simple, it is still startling to see from ten to thirty percent of memory unused depending on the model and total amount of core available. (Recall that the arrival rates are high enough to cause transient behavior of the system so no more memory will be used regardless of how high the arrival rate becomes.)

The second interesting data presented in Table I is the magnitude of the difference between N_I and N_M. Since the system is heavily loaded one would expect closer agreement between these two numbers. Now N_M can be evaluated by a simple application of a basic renewal formula (See Appendix for details). The values which result from this calculation agree closely with those of column two of Table I.

The values for N_I are not checked so easily. Since no analytic method seems available we may take the following indirect approach. If the values for N_I for core size=100 are correct then each arrival sees 1.39 identical exponential servers in operation. Thus, while an arrival rate of 2.00 makes the system transient, an arrival rate of 1.30 or so should result in a stable system and 1.39 should be the borderline between stable and transient behavior. While our simulation results in this area are very tentative, this does seem to be the case.

The final feature of Tables I and II which merits attention is the difference between the FCFS and SMF disciplines. In most cases (except for random fluctuations) the SMF discipline serves customers at a higher rate than does the FCFS discipline. The slight differences in service rates can cause significant difference in mean waiting times between the models. It can be argued, of course, that because we are testing the models in situations which cause the memory queue length to grow with time, that the SMF discipline deposits large executions at the end of the queue and thus alters the memory size distribution at the CPU. An examination of the mean memory size as observed at departure time shows that the effect of this situation is minimal in the executions we have conducted. The effect would undoubtedly be more important in longer executions.

Table III gives some statistics from the fragmentation model under the FCFS discipline which indicate the relationships between the amount of free core and the size of the largest hole available at certain times. The statistics listed are the following:

M_A is the mean memory available as recorded at arrival time.

F_A is the mean number of holes as recorded at arrival time.

H_A is the mean size of the largest hole available as recorded at arrival time.

M_{AF} is the mean amount of memory available at fragmentation failure time. (Frag-

mentation failure occurs whenever there is enough free core available to load the next customer, but the customer cannot be loaded because the free core is fragmented.)

M_{RF} is the mean amount of core requested by the next customer at fragmentation failure time.

H_F is the mean size of the largest hole available at fragmentation failure time.

F_F is the mean number of holes which existed at fragmentation failure time.

Table III. Memory-residence-time fragmentation model under FCFS discipline.

Core Size	M_A	F_A	H_A	M_{AF}	M_{RF}	H_F	F_F
100	33.1	1.19	30.1	75.6	66.8	52.8	2.01
200	63.4	1.91	43.3	94.6	71.9	57.2	2.32
300	79.6	2.68	45.1	105.3	70.9	55.6	2.99
400	96.2	3.25	49.2	118.6	74.0	57.4	3.52
500	116.5	4.41	50.1	128.7	74.4	54.2	4.57
600	133.2	4.84	53.4	143.3	75.5	56.4	5.00
700	143.3	5.73	50.9	153.2	75.1	53.7	5.86
800	148.8	6.64	49.5	154.4	74.9	51.3	6.70
900	187.4	7.43	55.1	188.9	78.0	56.1	7.45
1000	191.9	7.78	55.4	195.2	75.6	54.5	7.89

Several interesting observations can be made about the data presented in Table III. First of all, the values of H_A, M_{RF}, and H_F are relatively independent of memory size. Also the values for H_A and H_F are near the mean (50.0) of the memory size distributions of arriving customers. This suggests that the largest hole available occurs where a single previous customer has departed, and that adjacent regions are rarely freed simultaneously. Finally the values for F_A and F_F are nearly the same for large core sizes which is surprising because F_F is observed only at fragmentation failure time. This suggests that fragmentation failure is not a very special occurrence with regard to the number of holes which exist at any given time.

Although the data is not presented in Table III, an interesting feature of the values for F_A and F_F is that they appear to be normally distributed with mean values as given in the table and with standard deviations of 1.5. This observation becomes more pronounced above core sizes of 400.

When we observed the validity of the fifty percent rule (i.e. the number of holes is approximately one-half the number of active segments), an interesting variant of this rule was also discovered. The mean hole size was slightly less than one-half the mean value of the largest hole size. All mean values in this case are those as seen by arrivals. This data is summarized in Table IV. (Since these executions took longer to run, only a few core sizes were tested.)

A final set of executions was run to test the effect of correlation between a program's memory size and CPU time. In these executions the memory

Table IV. Mean largest hole size (H_A) and mean hole size (H_M) as observed at arrival time in the memory-residence-time fragmentation model.

	FCFS		SMF	
Core Size	H_A	H_M	H_A	H_M
600	53.4	27.5	51.8	25.8
700	50.8	24.9	54.5	25.0
800	49.5	22.4	53.8	25.1
900	54.3	24.9	54.2	22.5
1000	55.4	24.7	55.0	22.8

size distribution was unchanged but the CPU time distribution was changed so that larger programs were assigned longer CPU times. This was done as follows: If the memory size selected was m, then the CPU time was selected from an exponential distribution with mean m/50. Thus a program with the mean memory size would get an exponentially distributed CPU time chosen from a distribution with mean 1.0. Larger programs would have larger mean CPU times and smaller programs would have shorter mean CPU times.

The results of these executions for a selection of core sizes was given in Table V. A quick comparison of Table V with Tables I and II shows that the N_I values are markedly smaller in Table V. The M_I values are relatively the same, although for the fragmentation model more core is in use than previously was the case. Returning to Table V we note that the SMF discipline is now less efficient than FCFS in the sense that fewer people are now in service. Clearly, the problem of optimally ordering the memory queue does not have a simple solution.

Table V. Memory usage statistics for the memory-residence time models with CPU time and memory size dependence.

	FCFS			SMF		
Core Size	N_I	M_I	M_u	N_I	M_I	M_u
	Paging Model					
500	7.13	464.0	94.8 %	6.99	463.9	92.8 %
800	11.70	761.7	95.2	11.58	764.5	95.5
1000	14.82	965.6	96.6	14.62	964.3	96.4
	Fragmentation Model					
500	6.22	400.8	80.1 %	6.07	399.3	79.8 %
800	10.10	652.6	81.5	9.94	652.0	81.5
1000	12.79	823.8	82.4	12.63	833.3	83.3

B. The Processor-Sharing Models.

The processor-sharing models were executed with main memory sizes of 100,200,...,1000 blocks. Each memory size was run twice, once under the FCFS discipline and once under the SMF discipline. In this case, however, we chose an arrival rate of 0.999999 and a service rate of 1.0. This choice was made because the statistics of interest in this case were mean time in queue and mean time in system. But these numbers are finite only when the queueing system is stable. Due to our exponential interarrival and service time distributions it follows that the processor-sharing model is stable when the

corresponding M/M/1 queueing system is stable. Thus, we must choose an arrival rate less than the service rate. In order to make the results interesting, we have chosen the rates as close together as possible.

Table VI summarizes the statistics for the two processor sharing models under the FCFS discipline. The results are intuitive and not very surprising. As main memory size is increased, more programs can share the CPU at any given time so that the mean time in queue decreases. On the other hand, it takes longer for a program to obtain its CPU time requirement and depart, this being demonstrated by the increasing values for mean time in system.

Table VI. Mean time in queue (V) and mean time in system (W) for the processor-sharing models under FCFS discipline.

	Paging Model		Fragmentation Model	
Core Size	V	W	V	W
100	12.16	13.92	12.26	13.93
200	10.51	13.95	10.96	13.97
300	9.72	14.74	10.30	14.65
400	9.06	15.64	9.93	15.69
500	7.18	15.14	8.25	15.14
600	6.51	15.93	6.68	15.89
700	5.95	16.72	7.49	16.74
800	3.97	16.32	5.63	16.41
900	3.09	16.70	4.55	16.55
1000	3.15	18.36	4.74	18.41

We will not present a similar table for the processor-sharing models under the SMF discipline. The reason is that the sample variances of the quantities "time in queue" and "time in system" in this case are so large that meaningful results cannot be observed from such a table. In spite of the fact that we made several executions of these models with different random number sequences, we could discern no easily identifiable trends such as the one present in the FCFS case. More sophisticated statistical methods need to be used in both the simulation and analysis of these results.

The SMF discipline is clearly the cause for the large variations. To see why, note that under the SMF discipline the characteristics of a later arrival can change the time in queue of a previous arrival. This is because the later arrival may possibly be inserted ahead of previous arrivals in the memory queue, thus increasing in a random way the time in queue of all those previous arrivals.

IV. CONCLUSIONS.

We have argued for more analytic study of models of memory scheduling and have presented some models which we feel are realistic and still should be analytically solvable. While our current results are of the simulation type, we hope to place our observations on a more rigorous foundation.

Further simulation work could be done in combining the memory-residence-time and processor-sharing models into a single model which would be more representative of the computational and input-output activities which a real program performs. This could be done by assigning each customer a CPU and a memory residence time. Then the CPU scheduling could be modeled as in the processor sharing model, and a program could not leave memory until both its CPU time and memory residence time were completed. The time spent waiting for memory residence time to expire after computing requirements had been satisfied could be interpreted as I/O time.

Thus, in addition to our analytic analysis of the present models, we may be able to develop an inexpensive simulation model of real-life program behavior in a swapping system. We are currently considering the latter approach to determine how accurate such models might be.

REFERENCES

1. Knuth, D.C. The Art of Computer Programming, Vol.I., Addison-Wesley, Reading, Massachusetts, 1968, pp.445-448.

2. Denning, P.J. "Virtual Memory", Computing Surveys, 10,2 (Sept.1970), pp.153-189.

3. Ross, S.M. Applied Probability Models with Optimization Applications, Holden-Day, San Francisco, California, 1970, pp.32-35.

4. Coffman, E.G., and Denning, P.J. Operating Systems Theory, Prentice-Hall, Englewood Cliffs, N.J., 1973, pp.114-116.

APPENDIX

This appendix describes a simple method of evaluating N_M (The mean number of customers in memory given that the memory queue was non-empty) for the memory-residence-time paging model. The definitions and notation used here are taken largely from [3].

We first need some basic definitions:

Def. A stochastic process $\{N(t), t \geq 0\}$ is said to be a counting process if N(t) represents the number of events which have occurred up to time t.

Def. A counting process for which the inter-arrival times are independent and identically distributed random variables is called a renewal process.

Thus a Poisson process is a renewal process for which the inter-arrival times are exponentially distributed.

Def. If X and Y are random variables with distributions F and G respectively, then the distribution of the random variable X+Y is given by the convolution of F and G, written F*G, and defined as:

$$(F*G)(t) = \iint_{x+y \leq t} dF(x)dG(y) = \int_{-\infty}^{\infty} G(t-x)dF(x).$$

Def. Let $\{N(t), t \geq 0\}$ be a renewal process. Then the expected number of events which have occurred at time t will be given by $m(t) = E\{N(t)\}$.

We may now state a basic proposition:

Proposition: Let $\{N(t), t \geq 0\}$ be a renewal process with inter-arrival time distribution $F(t)$. Then

$$m(t) = \sum_{n=1}^{\infty} F_n(t)$$

where $F_n = F * F_{n-1}$ and $F_1 = F$.

Further $m(t) < \infty$ for all $t \geq 0$ provided $F(0) < 1$.

Pf: See [3], pages 32-33.

This proposition allows us to estimate N_m. Let us suppose we have a core size of 100 blocks which is initially empty. customers 1,...,n will be loaded provided

$$\sum_{i=1}^{n} X_i \leq 100 \quad ,$$

where X_i is the memory size of the ith customer. (Here we are neglecting departures since we are interested in estimating the mean number in memory at a given time.) If

$$\sum_{i=1}^{n+1} X_i > 100$$

then precisely n customers will fit into memory.

Now consider a renewal process with inter-arrival times X_i. If

$$\sum_{i=1}^{n} X_i \leq 100$$

and

$$\sum_{i=1}^{n+1} X_i > 100$$

then precisely n events occur before time 100. Thus the number of customers which fit into memory of size 100 and the number of events which occur before time 100 are the same.

Therefore N_m when core size is 100 is given by m(100). Similarly we may estimate N_m when core size is 200,...,1000 by calculating m(200),...,m(m(1000).

But this would be of little use unless $F_1, F_2, \ldots, F_n, \ldots$ could be calculated. Fortunately if X_i are integer valued random variables (and we have assumed that they are), then only $F_1, F_2, \ldots, F_k$ are required for calculation of m(k) since all other terms will be zero. Actually convergence of the series for m(t) is much faster than this, and m(k) can be evaluated by a simple computer program. These values (for the case where X_i are uniformly distributed on 1 to 100) along with the corresponding N_I values are listed in Table A-1.

Table A-1.

k	m(k)	M_I
100	1.70	1.70
200	3.64	3.60
300	5.61	5.47
400	7.60	7.40
500	9.57	9.35
600	11.60	11.36
700	13.50	13.29
800	15.50	15.14
900	17.50	17.11
1000	19.50	19.07