Proceedings of Sixth ACM Symposium on Operating Systems Principles (November 1977) 125-131.

ANALYSIS OF DEMAND PAGING POLICIES WITH SWAPPED WORKING SETS

Dominique Potier
IRIA-LABORIA
BP 105
78150 Le Chesnay
FRANCE

The performance improvements brought by demand paging policies with swapped working-sets depend on several factors, among which the scheduling policy, the behaviour of the programs running in the system and the secondary memory latency characteristics are the more noticeable. We present in this paper a modelling approach to quantify the effects of these factors on the performance of a system running with a swapped working-sets policy. A preliminary analysis, conducted in the virtual time of the programs, shows their influence on the paging behaviour of programs. The results of this analysis are then used within a detailed queueing network of a multiprogrammed system. Computationnaly simple expressions for the CPU time spent in user state and in supervisor state are obtained for a class of paging policies ranging from pure demand paging to demand paging with swapped working-sets. Numerical examples illustrate the analysis, and these results are compared with measurements made on a real system running with swapped working-sets policies.

Key words and Phrases : paging algorithms, demand paging algorithms, working-set memory management, swapping, program behaviour, virtual memory, semi-Markov model, queueing network model.

CR categories : 4.32, 4.35, 8.1.

Introduction.

In time-sharing multiprogrammed systems using a pure demand paging policy a page is loaded into main memory only after the occurence of a page fault. An important part of the page traffic is thus incurred when the working pages of a process are loaded on demand at the start of its execution. This effect is enhanced by the influence of such factors as the scheduling policy and slow input-output requests which cause the execution of a process to be split into several memory residence intervals, thus giving rise to as many initial loading phases. Experimental evidence [1] indicates that more than fifty percent of the page traffic comes from these page faults and the results of an analysis on the influence of process loading on the page fault presented in [9] also points out the importance of this effect.

An obvious solution to this drawback of a pure demand paging memory management policy is to preload the process active pages or working-set when the process is reactivated in the multiprogrammed set. We shall call these policies demand paging policies with swapped working sets or, more briefly, SWS policies.

The main advantages associated with these policies are to reduce the number of page faults and to generate bulk requests to the secondary memory (however, the volume of pages moved remains unchanged, and may even increase because of unused preloaded pages). Since reducing the number of page faults has a direct effect on the amount of supervisor time spent in processor switching, a gain is to be expected there. The gains brought by the bulk requests depend on the secondary memory device characteristics. Preloading from a device with no latency (e.g. core-to-core transfers) would gain nothing as it was proved by Mattson et al. [8] and Aho, Denning and Ullman [3]. On the contrary, in systems using sequential access secondary memories such as drums, the average time to load a page will be reduced since preloading a working set may require only one access to the drum rather than as many as pages for pure demand paging.

It remains to analyse and quantify these effects according to the environment in which the SWS policy is implemented. It is the approach followed in this paper.

The performance analysis of demand paging policies with swapped working-sets that we present has two startpoints : the analysis presented in [9] which provides the basic modelling framework for our study ; the demand paging policy with swapped working-sets currently implemented in the Edinburgh Multi-Access system (EMAS) [2]. This policy proved to be successful, and we shall refer to it in the remainder of this paper.

The study of SWS policies that we shall develop is based on a simple probabilistic model of the paging behaviour of a program. The analysis proceeds in two steps. A preliminary analysis conducted in the virtual time of the programs shows how SWS policies reduce the average page fault rate and generate bulk requests to the secondary memory device. These results are then used within a multiclass queueing network model of a multiprogrammed system where programs are divided into classes according to the number of their pages present in main memory. The memory management policy represented in the model follows the one implemented on EMAS and the

paging drum service time characteristics and the overheads involved in the different operations are taken into account in detail. Expressions for the Central Processing Unit (CPU) time spent in user state and supervisor state are obtained for a class of demand paging policies ranging from pure demand paging (PDP) to demand paging with SWS. These results are illustrated by numerical examples and compared to the observations made on the EMAS system.

Process time analysis.

In the context of multiprogrammed page on demand computer systems, the execution of a program consists of a sequence of memory residence intervals (MRI), the number and the duration of which depend on factors such as the I/O behaviour of the program and the memory management and scheduling policies implemented in the system. Within each memory residence interval, the execution is interrupted by page faults, the interval of time between two consecutive page-faults depending on the internal behaviour of the program, the paging policy and the number of pages of the program present in main memory.

Let X(t) be the number of pages of a program present in main memory at the instant t of its virtual time. The behaviour of X(t) can be described in two steps. We specify in the first place the variations of X(t) within a MRI and then the transitions of X(t) between two consecutive MRI's. The first step consists in representing the paging behaviour of the program under a page on demand policy ; the second step in describing the swapping policy.

The analysis of the process X(t) follows closely the one developped in [9] and we shall recall it briefly.It is conducted under the following assumptions :

- H_0 : the programs executing on the system have identical behaviour.
- H_1 : the length of the consecutive memory residence intervals are i.d.d. exponential random variables with mean T.
- H_2 : the maximum number M of main memory page frames allocated to a program is fixed and, unless otherwise specified, pages are loaded in main memory on a page on demand basis.
- H_3 : the intervals of time between two consecutive page faults of a program are i.i.d. exponential random variables with mean q_i when i pages of the program are present in main memory.
- H_4 : the transitions between the number i of pages of a program present in main memory at the end of given MRI and the number j of pages of the same program in main memory at the beginning of the next MRI are described by a first order Markov chain with transition matrix (α_{ij}). Thus (α_{ij}) characterizes the swapping policy implemented by the system.

The set of states E of the process X(t) is then: E = {1,2,... M}. An example of a realization of X(t) is represented in figure 1.

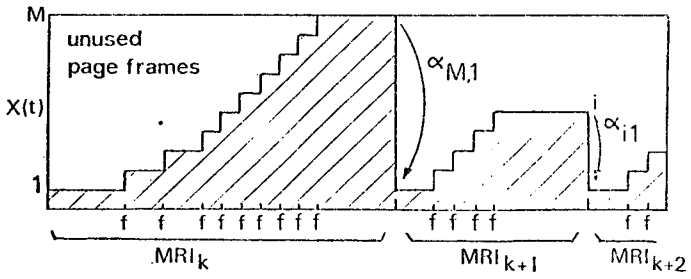

Figure 1.

Under assumptions H_0-H_4, X(t) is a semi-Markov process. Let $p = (p_{ij})$ be its transition matrix which is computed as follows. The probability $\mathcal{O}_i$ that the interruption which occurs is a page fault when the process is in state i is given by

$$(1) \qquad \mathcal{O}_i = \frac{1/q_i}{(1/q_i + 1/T)} ,$$

and the mean uninterrupted CPU interval ℓ_i in state i is

$$(2) \qquad \ell_i = \frac{1}{(1/q_i + 1/T)} = T(1 - \mathcal{O}_i) ,$$

From equation (1) and assumption H_4, we obtain

$$(3) \quad p_{ij} = \begin{cases} \mathcal{O}_i & i=1,\ldots,M-1 \ ; \ j=i+1 \\ (1 - \mathcal{O}_i)\alpha_{ij} & i=1,\ldots,M-1 \ ; \ j=1,\ldots,i \\ (1 - \mathcal{O}_i)\alpha_{Mj} & i=M \ ; \ j=1,\ldots,M-1 \\ (1 - \mathcal{O}_M)\alpha_{MM} + \mathcal{O}_M & i=j=M \end{cases}$$

Let $\bar{\omega}_i$ represent the equilibrium probability that a program is activated on the CPU in state i. Denoting by π the vector :

$$(4) \qquad \pi = (\bar{\omega}_1,\ldots,\bar{\omega}_M)$$

we have, from the definition of P

$$(5) \quad \begin{cases} \pi = \pi P \\ \sum_{i=1}^{M} \bar{\omega}_i = 1 \end{cases}$$

The $\bar{\omega}_i$ can be simply computed using the following recursive equation obtained from (3) and (5) and starting with ω_M

$$(6) \quad \begin{cases} \bar{\omega}_{j-1} = \frac{1}{\mathcal{O}_{j-1}} \left[\bar{\omega}_j - \sum_{i=j}^{M} (1-\mathcal{O}_i)\alpha_{ij}\bar{\omega}_i\right] , \\ \qquad\qquad\qquad\qquad j=2,\ldots,M-1 \\ \bar{\omega}_{M-1} = \frac{1}{\mathcal{O}_{M-1}} \bar{\omega}_M (1 - \mathcal{O}_M)(1 - \alpha_{MM}). \end{cases}$$

The vector of steady-state probabilities $\Gamma = (\gamma_1,\ldots,\gamma_M)$, where γ_i is the equilibrium probability that $X(t) = i$, is obtained from π and ℓ_i, $i=1,\ldots,M$, by

$$(7) \qquad \gamma_i = \frac{\bar{\omega}_i \ell_i}{\sum_{j=1}^{M} \bar{\omega}_j \ell_j} \quad , \quad i=1,\ldots,M$$

Hence, in the virtual time of the process, we can now compute the average time r_M between two consecutive page faults for a given memory allocation of M pages, and the memory utilization ratio ε_M. We have

$$(8) \qquad 1/r_M = \sum_{i=1}^{M} \gamma_i/q_i \ ,$$

$$(9) \qquad \varepsilon_M = \frac{1}{M} \sum_{i=1}^{M} i\,\gamma_i \ .$$

The model of swapping policy.

We consider the following class of SWS policies. A program which has i pages in main memory when it is swapped out at the end of the current memory residence interval, has i pages preloaded at the beginning of the next memory residence interval with probability $1-\beta$, and only one page with probability β, $(0 \leq \beta \leq 1)$. The parameter β takes into account different factors such as the proportion of new processes for which no preloading can be achieved and the way the active pages of the process are identified. For $\beta=1.$, pure demand (PDP) is followed, whereas for $\beta=0.$, the SWS policy is always successful. Thus β defines a continuum of paging policies ranging from pure demand paging to demand paging policies with swapped working sets.

It should be noted that more complex models of swapping policies could be used. Indeed, any policy which can be described in terms of a first order Markov chain (assumption H_4) may be imbedded in the model. However, for the sake of clarity, we shall restrict our attention in the remainder of the paper to the class of SWS policies with one parameter defined above.

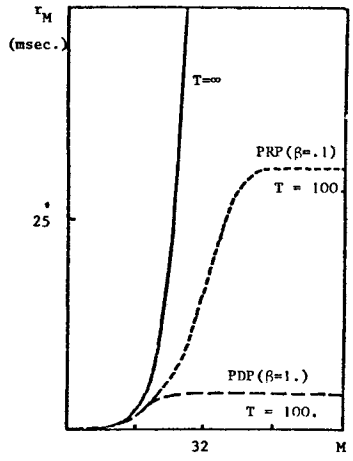

Figure 2a

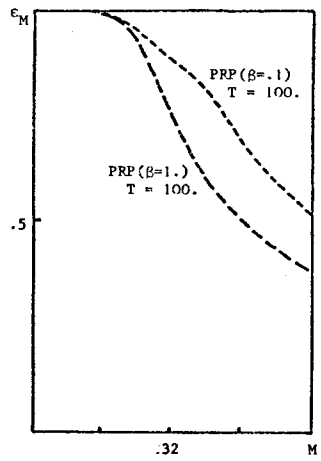

Figure 2b

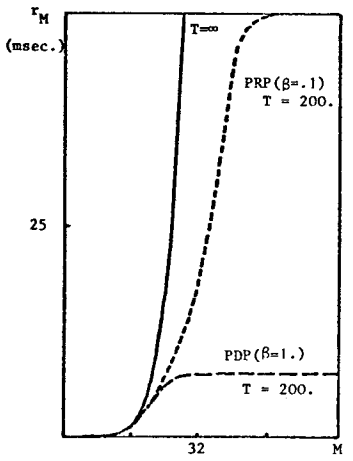

Figure 3a

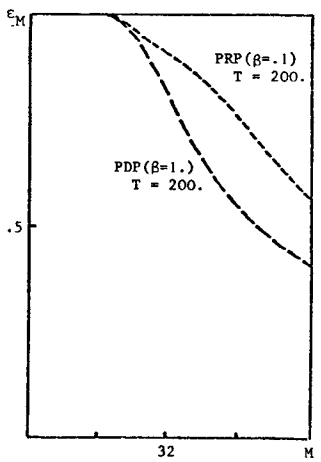

Figure 3b

Numerical examples.

Figures 2 and 3 present ε_M and r_M for different values of β and T. The function q_i has been estimated by $q_i = ai^k$ with $a = 10^{-7}$, $k = 6$, using the experimental results reported in [7]. Unless otherwise specified, the unit of time is 0.001 sec.. The graphs show the effect of the paging policy on the memory utilization ratio ε_M and on the average interval of time between page faults r_M for two values of the mean residence time (obviously, for $T = \infty$, we have $r_M = q_M$ and $\varepsilon_M = 1$). The improvements brought by a SWS policy are clearly pointed out in figures 2a and 3a. There improvements are the more noticeable the larger the memory allocation M. This illustrates why in paged systems using a PDP policy, allocating main memory to a process beyond a certain point does not improve its paging behaviour very much.

An other way to illustrate the effects of a SWS policy is to compare the rate u_r of I/O requests generated to the rate u_p of pages transfered. For a demand paging policy with no SWS we have $u_r = u_p$, whereas for a SWS policy we expect, as noted in the introduction, to have $u_r < u_p$. Thus, the comparison of u_r and u_p gives an indication on the gain which can be obtained by the reduction of device accesses. Following [5],we shall draw an $u_r - u_p$ plot in order to provide a visual comparison of the performance of the SWS policies we have defined. The derivation of u_r and u_p is straight forward. Assuming that no page is modified during an MRI, we have

$$(10) \qquad u_r = \frac{1}{r_M} + \frac{1}{T},$$

$$(11) \qquad u_t = \frac{1}{r_M} + \frac{1}{T}[\beta + M\varepsilon_M(1-\beta)]$$

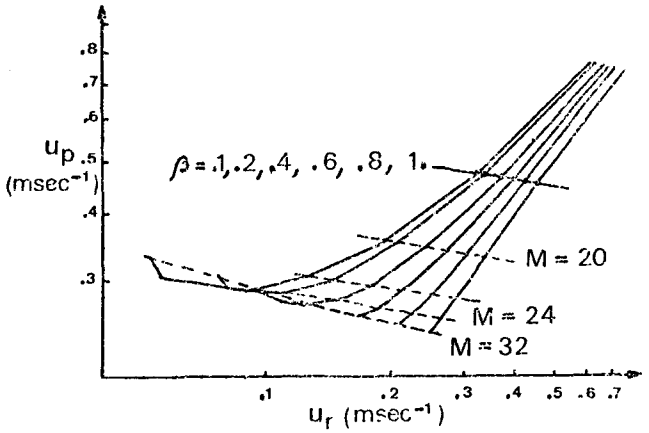

Figure 4.

The u_r - u_t plot is represented in figure 4 for different values of β and M. The others parameters are set to the same values as above with T = 100. The graph shows clearly that as β increases the rate u_r decreases significantly whereas u_t increases.

These observations indicate how an SWS policy which recorded the set of active pages of program when it is suspended and preloaded this set of pages at the reactivation of the program, reduces the page fault rate of the programs, increases the usage of main memory and reduces overheads due to latency owing to bulk requests. It remains to evaluate the effects of a SWS policy on system performance such as throughput and supervisor activity. This will be done in the next section.

Real time analysis.

The model of the system.

The model of the system is represented in figure 5. It consists of a central processing unit (CPU) station and a secondary memory (SM) station. A new program is queued at the SM station to have its first page loaded into main memory and then put in the CPU queue. A page fault causes the execution of the running program to be suspended, and the program to be queued at the SM station to have its missing page transfered. At the end of a memory residence interval, the execution of the program is suspended and the program leaves the CPU-SM loop and the multiprogramming set.

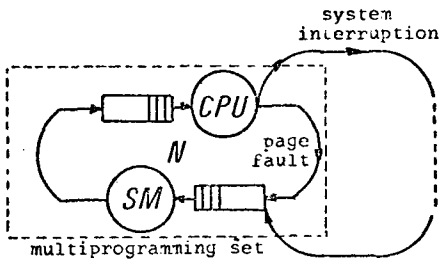

Figure 5.

The memory management policy follows the one currently implemented in the EMAS system, with the main simplification that all programs have identical behaviour. The analysis is performed under the assumption that the degree of multiprogramming N is fixed : a program which leaves the CPU-SM loop due to a system interrutpion is immediately replaced by another program. When a program enters the multiprogrammed set, it is allocated a fixed number of main memory page frames m = M/N, where M is the total number of main memory page frames. The working-sets of programs belonging to the multiprogramming set are periodically recorded so that when a program is reactivated in the multiprogramming set to start a new MRI its working set can be preloaded into main memory by the SM station.

It should be noted that a new program which replaces a terminating program will have an empty working set and only its first page can be loaded. Moreover, due to fact that working sets are measured at relatively large periods of time if no proper mechanism is available, some fraction of the preloaded pages will not actually be used. In order to take into account these two factors, we assume that when a program is suspended with i pages present in main memory, the program which immediately replaces it is preloaded with one page with probability β , and with i pages with probability 1-β. This define (α_{ij}) and the ω_i as above.

We now define a classification of the programs being multiprogrammed according to their state (number of pages in main memory), the station where they are queues or serviced, and the kind of service they are receiving (page transfer, swapping out, preloading) from the SM station.

i) a program is in class (1,i) if it is in state i, i=1,...,M and in the CPU station,
ii) a program is in class (2,i) if it is in the SM station after having page faulted in state i,
iii) a program is in class (3,i) if it is in the SM station to have its pages swapped out after having completed the current MRI in state i. Only modified pages are swapped out on the SM and we define p_{mod} as the proportion of pages which have been modified during the current MRI,
iv) a program is in class (4,i) if it is in the SM station to have its pages preloaded after having completed the previous MRI in state i.

With each SM service is associated a CPU time overhead. Let (2,i)', (3,i)', (4,i)' denote the class of a program suspended in state i and receiving services from the supervisor for processing a page fault, a swap out, a preload. We denote by $\tau_{2,i}$, $\tau_{3,i}$, and $\tau_{4,i}$ the length of the corresponding overheads.

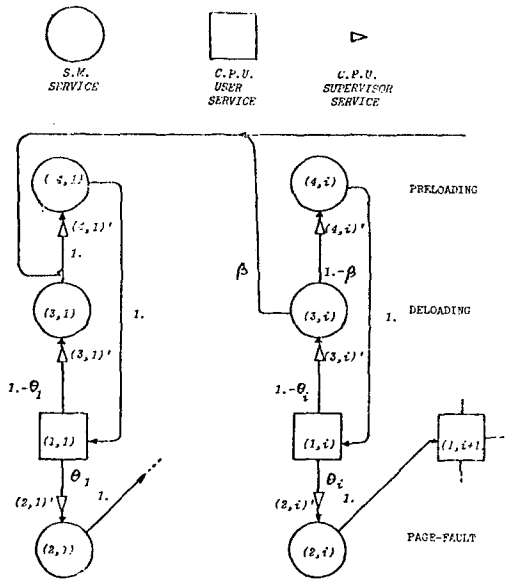

Figure 6.

From these definitions and previous assumptions, the class of a program is a Markov chain whose transition diagram is represented in figure 6. The relative arrival rates $X_{k,i}$ of a program (k,i) are simply obtained by noting that we have from definition (i)

$$X_{1,i} = C\bar{\omega}_i, \quad i=1,M, \tag{12}$$

where C is a multiplicative constant. The transition diagram yields :

$$\begin{cases} X_{4,1} = X_{3,i} + \beta \sum_{i=1}^{M} X_{3,i} \\ X_{2,i} = \theta_i X_{1,i} & i=1,\ldots,M, \\ X_{3,i} = (1-\theta_i) X_{1,i} & i=1,\ldots M, \\ X_{4,i} = (1-\beta) X_{3,i} & i=2, M, \end{cases} \tag{13}$$

and the overheads associated with a page fault, a swap out, and a preload occur with the respective frequencies $X_{2,i}$, $X_{3,i}$ and $X_{4,i}$.

In the context of multi-class queueing networks [4] , the $X_{k,i}$ represent the relative arrival rates to the different stations of the network. It remains to characterize the mean service times of the different classes of program at each station. At the CPU station, the mean service time of programs of class (1,i) is obviously ℓ_i given by equation (2). Processes asking for supervisor services when they are in classes (2,i)', (3,i)' and (4,i)' require a mean CPU service time $\tau_{2,i}$, $\tau_{3,i}$ and $\tau_{4,i}$.

At the SM station, we denote by s(j) the mean time to access and transfer j pages. The function s(j) takes into account service time characteristics of the SM station. For a typical paging drum, s(j) would be approximated by :

$$s(j) = x_a + j\, x_t , \tag{14}$$

where

x_a = mean access time,

x_t = mean transfer time.

Solution of the model.

We assume that the discipline of service is processor-sharing at the CPU and the SM station. Although the processor-sharing assumption is not realistic for the SM station, experiments we have conducted indicate that in a closed two server queueing model, the utilization factors of the servers are not sensitive to this assumption. Let ρ_{CPU} be the CPU utilization rate, including overheads, ρ'_{CPU} utilization rate, overheads not included, and ρ_{SM} the SM utilization rate. Following [4] , we then have :

$$\rho_{CPU} = \frac{1 - A^N}{1 - A^{N+1}} , \tag{15}$$

$$\rho'_{CPU} = \delta\, \rho_{CPU} , \tag{16}$$

$$\rho_{SM} = \frac{1 - A^N}{1 - A^{N+1}} . \tag{17}$$

where

$$A = \frac{\sum_{i=1}^{M} \{X_{1,i}\ell_i + X_{2,i}\tau_{2,i} + X_{3,i}\tau_{3,i} + X_{4,i}\tau_{4,i}\}}{\sum_{i=1}^{M} \{s(1)X_{2,i} + X_{3,i}s(p_{mod}.i) + X_{4,i}s(i)\}} \tag{18}$$

$$\delta = \sum_{i=1}^{M} X_{1,i}\ell_i \Big/ \sum_{i=1}^{M} [X_{1,i}\ell_i + X_{2,i}\tau_{2,i} + X_{3,i}\tau_{3,i} + X_{4,i}\tau_{4,i}]. \tag{19}$$

The derivation of equations (15) and (17) is straightforward. The coefficient A can be simply interpreted as the ratio of the average CPU service time for all classes requiring a service from the CPU to the SM average service time for all classes requiring a service from the SM. It can be noted that the parameter p_{mod} is used in the computation of the average SM service time in order to determine the number of pages actually swapped out. The parameter δ is the ratio of the CPU time spent in user state to the CPU time spent in user state and supervisor state. Without further computations, δ provides an important indication on the performance of the memory management policy.

To summarize, the model has been built in order to take into account the main factors which determine the performance of a prepaging policy :

- the intrinsic program behaviour described by the function q_i,
- the mean memory residence interval T,
- the probability β of not being able to preload,
- the proportion p_{mod} of modified pages during an MRI,
- the overheads associated with the SM service. These overheads take explicitly into account the number of page transfers demanded by each request,

- the service time characteristics of the SM. Since bulk requests generated by a SWS policy result in reduced access time on the drum for each transfer, the dependence of the service time to the number of pages j to be transfered is explicitly considered by the function s(j).

The influence of these factors is illustrated in the next section by several numerical examples.

Numerical examples.

The model has been run using measurements made on the EMAS system [1]. The EMAS system makes use of a SWS policy but this policy can be easily turned off so that data on the performance of the system running the same benchmark with SWS turned off and on are available for a variety or hardware configurations. These measurement indicate that the runs using SWS show consistently that the CPU time spent in the user state is significantly increased by more than 10 % and that the time spent in supervisor state is decreased. It has also been observed that the utilization of the drum channel is greater with SWS.

Although the model does not describe the actual EMAS configuration, since it is mostly restricted to the description of the operations associated with the memory management policy, we shall see that, as far as the influence of SWS is concerned, it satisfactorily reproduces the performance improvements brought by SWS as they have been observed on the real system. Moreover, it will allow us to investigate the effects of the variation on some factors such as overheads or SM service time characteristics.

Input parameters have been obtained from measurements made on the EMAS system. The following values have been estimated

$\tau_{2,i} = 4.5$ (msec.)

$\tau_{3,i} = \tau_{4,i} = 1.5 + 1.7\ i$ (msec.)

$s(j) = 10(.66 + .33\ i)$ (msec.),

$T = 100$ (msec.)

$p_{mod} = 1/3$

For all the experiments we have assumed that M = 96 main memory pages are available to users. The degree of multiprogramming has been varied from 1 to 5. For each set of parameters the performance measures δ, ρ_{CPU}, $\rho'_{SPV} = \rho_{CPU} - \rho'_{CPU}$ and ρ_{SM} have been computed as a function of the degree of multiprogramming N. The parameter β has been set to 1. for pure demand paging and .1 for demand paging with SWS.

	δ		ρ_{CPU}		ρ'_{CPU}		ρ_{SPV}		ρ_{SM}	
N	PDP	SWS	PDP	SWS	PDP	SWS	PDP	SWS	PDP	SWS
1	.44	.58	.43	.41	.19	.24	.26	.17	.57	.59
2	.44	.59	.57	.54	.25	.32	.31	.22	.76	.78
3	.44	.57	.63	.61	.28	.35	.35	.27	.85	.86
4	.44	.42	.67	.62	.29	.29	.37	.34	.90	.92
5	.35	.36	.63	.59	.22	.21	.41	.39	.95	.96

Table 7

Table 7 shows the way the measures of performance are influenced by the paging policy. The main observation is that the ratio δ is steadily increased by more than 10 % with SWS as long as the system is not thrashing. Depending on the degree of multiprogramming and the utilization factor of the SM, this causes the CPU time spent in user state to increase by 2 % to 6 %. It can be noted that the maximum improvement is obtained for the optimum values of the degree of multiprogramming, i.e. the degree which maximizes ρ'_{CPU}. As already observed on measurements made on the EMAS system, the optimum degree of multiprogramming is smaller with SWS(N=3) than with PDP (N=4). Despite the service time characteristics of the SM, the utilization factor of the SM is slightly increased with SWS, which indicates that the total amount of page transfers is greater with SWS than with PDP.

	δ		ρ_{CPU}		ρ'_{CPU}		ρ_{SPV}		ρ_{SM}	
N	PDP	SWS	PDP	SWS	PDP	SWS	PDP	SWS	PDP	SWS
1	.55	.70	.38	.37	.21	.26	.17	.11	.62	.63
2	.55	.69	.49	.48	.27	.33	.22	.14	.81	.83
3	.55	.67	.54	.54	.30	.36	.24	.17	.90	.90
4	.55	.58	.56	.52	.31	.30	.26	.22	.94	.95
5	.45	.44	.51	.48	.23	.21	.28	.26	.98	.99

Table 8

The effect of overheads has been investigated in more detail by setting $\tau_{2,i} = 3$ msec., $\tau_{3,i} = \tau_{4,i} = 1+i$ (msec.). The results obtained from this new set of experiments are presented in table 8. We can observe that the reduction of overheads causes an increase in the SM utilization. This is an indirect consequence of the increase of CPU time spent in user state, although the total CPU utilization is significantly decreased by 5 % to 10 %. This observation partly explains why SWS which causes a reduction of the amount of supervisor activity, also causes an increase of the SM utilization. As in table 7, it can be noted that the improvements brought by SWS vanish when the system starts running in a thrashing zone. The explanation is that in this case programs are allocated a small number of page frames so that the influence of the initial loading phases (see figure 2) on the paging activity diminishes. Preloading is then more useful since the supplementary page transfers brought by SWS are not compensated by a sufficient decrease of the supervisor activity.

	δ		ρ_{CPU}		ρ'_{CPU}		ρ_{SPV}		ρ_{SM}	
N	PDP	SWS	PDP	SWS	PDP	SWS	PDP	SWS	PDP	SWS
1	.55	.70	.39	.42	.21	.29	.17	.13	.61	.58
2	.55	.69	.51	.55	.28	.39	.23	.17	.83	.77
3	.55	.67	.56	.61	.31	.41	.25	.20	.89	.86
4	.55	.58	.58	.57	.32	.32	.26	.24	.94	.92
5	.45	.44	.52	.49	.24	.22	.28	.27	.98	.98

Table 9

As already noted, service time characteristics of the SM play an important part in the improvements brought by SWS policies. Table 9 presents the results obtained by setting s(j) = 10(.5 + .5i). The measure of performance δ is not modified since it does not depend on the SM service time characteristics. Since the new function s(j) gives a more favourable treatment to bulk requests generated by SWS, the performance of these policies are increased.

More detailed observations reveal that the SM utilization is then smaller with SWS than with PDP. This indicates that in this case the increase of the number of page transfers is compensated by the reduction of the mean service time per page of the SM due to bulk requests.

As a whole, these results illustrate how the different factors which have been listed above interact and influence the performance of a SWS policy. As observed from the experiments we have presented, no factor is in itself determinant, but it is their combination which defines the final result. The observations we have made on the results obtained from the model corroborate the measurements made on the EMAS system. Moreover, they provide some understanding of the behaviour of a SWS policy, and indicate how this policy would work with other hardware configurations and in other situations. As an application, it is intended to use the model to study the performance of a SWS policy between the disks and the drum in the EMAS system. Although the model involves only one category of users, it could be readily extended to take into account different categories of users by associating with each category a different life-time function and mean residence time. This would merely enlarge the number of classes involved, and the derivation of the solution would remain unchanged.

Conclusion.

We have presented a model for the analysis of the performance of a variety of swapping policies in page-on-demand multiprogramming systems. This analysis is based on a detailed characterization of the paging behaviour of programs and of the system and hardware configuration. The model is computationally simple, and we have shown that it can be used to evaluate the "trade-offs" involved in the implementation of a class of swapped working-sets policies. Moreover, the approach we have followed provides a framework for analysing the influence of program behaviour on global performance and it should be useful for the performance analysis of other memory management policies.

Acknowledgements.

We wish to thank Colin Adams, from the Computer Science Group at the University of Edinburgh, for many helpful discussions on the EMAS system.

References.

[1] ADAMS, J.C., "Evaluation of performance of the EMAS System", Séminaires Modélisation et Mesures IRIA-LABORIA (1976).

[2] ADAMS, J.C. , MILLIARD, G.E., "Performance measurements of the Edingburgh Multi-Access System", Proceedings of the International Computing Symposium 1975, ACM/AFCET, Antibes (June 1975).

[3] AHO, A.V., DENNING, P.J., ULLMAN, J.D., "Principal of Optimal Page Replacement", J.ACM 18, 1 (January 1971), 80-93.

[4] BASKETT, F., et al., "Open, Closed and Mixed Networks of queues with different classes of customers", J.ACM 22, 2 (April 1975).

[5] CHOY, D.M., "A graphical tool for the evaluation of prepaging and of paging with non-uniform pages", Report RJ - 1789, IBM Research Laboratory, San Jose, Calif. (May 1976).

[6] DENNING, P.J., "Virtual memory", Computing Surveys 2, 3 (September 1970).

[7] LEROUDIER, J., BURGEVIN, P., "Characteristics and models of program behaviour", Proceedings of the Annual Conference ACM'76, Houston, (October 1976).

[8] MATTSON, R.L., et al., "Evaluation techniques for storage hierarchies", IBM Sys. J. 9, 2 (1970), 78-117.

[9] PARENT, M., POTIER, D., "A note on the influence of program loading on the page fault rate", 2nd International Workshop on Modelling and Performance Evaluation of Computer Systems, EURATOM-ISPRA, Stresa (October 1976), North-Holland Publishing Company (to appear in Acta Informatica).