Denning, Peter, and Jeffrey Spirn. 1973. Dynamic storage partitioning. Proc. ACM SIGOPS SOSP (January), 73-79.

DYNAMIC STORAGE PARTITIONING

Peter J. Denning
Department of Computer Sciences
Purdue University
W. Lafayette, IN 47907

&

Jeffrey R. Spirn
Division of Engineering
Brown University
Providence, RI 02912

Abstract: A model of paged multiprogramming computer systems using variable storage partitioning is considered. A variable storage partitioning policy is one which allocates storage among the active tasks according to a sequence of fixed partitions of main storage. The basic result obtained is, mean processing efficiency is increased and mean fault-rate decreased under a variable partition, provided that the curves of efficiency and fault-rate as a function of allocated space are concave up.

1 INTRODUCTION

In 1967 and 1969 Belady and Kuehner published studies of dynamic space sharing in paged multiprogrammed memory systems [1,2]. These studies included the then remarkable experimental observation that by varying the partition of memory among the active tasks, the processing efficiency of the system showed an increase in the range 10-15% as compared with the same system running the same tasks in a fixed partition. The capability of the system to process a given set of active tasks in a given memory space was increased by the simple expedient of varying the memory partition.

Figure 1 shows a pair of hypothetical tasks, each requiring 24 units of processing time to complete.+ Both tasks have the fault-rate function f(x) and the execution interval function 1/f(x) shown in the table; 1/f(x) gives the expected length of the virtual time (i.e., computation time) interval between page faults, as a function of x, the memory allocation in page frames. It is important to note that both f(x) and 1/f(x) are concave up. The basic system on which the tasks are run has a single processor, and an auxiliary memory which satisfies any page request in 4 time units. The three timing diagrams show the two tasks operating under three regimes: Regime I will be called the fixed partition, Regime II the variable partition; Regime III will be explained shortly. In Regime II, one task has 1 page, the other 3 pages for most of the time; there is a changeover of memory allocations during the interval (32,39). The table below the timing diagrams shows the following measures for each regime:

1. Real time efficiency of the system. This is the ratio of total processing time to the total real time to complete the tasks.

2. Real time memory allocation of a task. This is the mean amount of memory allocated to a task across real time.

3. Virtual time memory allocation of a task. This is computed by the formula $(\sum_x xt(x))/(\sum_x t(x))$, where t(x) is the total virtual time during which x pages are allocated.

4. Virtual time fault-rate of a task. This is the ratio of the number of paging interruptions in a task to the total virtual time of the task.

5. Real time fault-rate of the system. This is the ratio of the total number of paging interruptions (between both tasks) to the total real time required to complete both tasks.

It is important to note that Regime II has improved all the measures favorably; and that the mean amount of memory experienced by each task in its virtual time is increased in Regime II, even though the mean memory allocation in real time remained the same as in Regime I.

The explanation offered by Belady and Kuehner for this phenomenon was based on the observed concave up property of the execution interval curve. In their experiments they noted that, for a p-page program, the mean time between page faults (using a first-in-first-out replacement rule and demand

+We should like to express our appreciation to L. A. Belady for directing our attention so elegantly to the proper distinction between real and virtual time by the example of Fig. 1 (which he sent us), and for giving us permission to use the example here in this paper.

paging) was approximated by the curve $g(x)=cx^2$, where c is a constant depending on the program, and $0<x<p'<p$ for some p'. (See Fig. 2.) The flattening of the observed curve for $p'\leq x\leq p$ follows from the fact that execution intervals are bounded by the task's running time, and no page faults occur when all p pages are loaded. The concave up property of g(x) is expressed by the formula

$$(1.1) \qquad \frac{g(x_0-b)+g(x_0+b)}{2} > g(x_0)$$

for any b and x_0 satisfying $b\leq x_0\leq p'-b$. It is evident, therefore, that running a task for half its execution intervals with memory allocation x_0-b and half its execution intervals with memory allocation x_0+b will result in a mean execution interval longer than when the task is run for its entire execution in memory allocation x_0. (Figure 1 has $x_0=2$ and $b=1$.)

This paper extends the work described above. Before proceeding, we believe it important to emphasize a central point: the arguments to be presented depend crucially on making a proper distinction between virtual and real time. If one fails to make the distinction properly, one can be led to apparently strange and contradictory conclusions. In the case of Fig. 1, for example, one can argue, "If a task is run for half its virtual time with memory x_0-b and half its virtual time with memory x_0+b, that the fault-rate function is concave up will imply an increase in the mean fault rate, not the decrease suggested by applying the concave-up argument to the execution interval function." This argument is exact; the "contradiction" arises because, to implement this policy, one would have to force the two tasks to operate under Regime III, which leaves part of the memory unused; this regime is clearly not one we would use in practice. The Regime II of Fig. 1 corresponds to running each task for half its real time with memory x_0-b and half its real time with memory x_0+b, which does allow full use of memory. Regime III improved nothing but the real time fault rate. A simple calculation for Regime II shows that the mean virtual time fault rate is $\bar{f}=0.33$ and that $f(2.6)=0.30$, $f(2.6)$ being the fault rate were it possible to grant each task a fixed allocation of 2.6 pages of memory. However, $0.50=f(2)>\bar{f}=0.33$, so that Regime II nonetheless reduced the fault-rate in comparison to Regime I.

2 DEFINITION OF A MODEL

We shall use the operator $R(\cdot)$ to signify the expectation in real time of a quantity, and $V(\cdot)$ to signify the expectation in virtual time of a quantity. When it is important to call attention to expectations referring to a particular task i, we shall subscript the expectation operations: R_i and V_i. In particular, R(x) and V(x) will denote, respectively, the mean memory allocation of a task in real time and in virtual time, where x is the random variable of memory allocation; R(g) and V(g) will denote, respectively, the mean value of a function g in real time and in virtual time; and R(A) will denote the expected real time to complete a virtual time interval of length A, i.e., containing A references.

Let T_m denote the mean main memory access time per reference in the system. Let T_{ai} denote the mean auxiliary memory access time per page request of task i. Define the speed ratio of task i be be $a_i=T_{ai}/T_m$. Virtual (or program) time is measured as the number of (virtual) address space references completed by a task in a given interval of real time. The fault-rate function $f_i(x)$ gives the rate of page interruptions in the virtual time of task i, when the main memory allocated to it is x pages; in other words, $1/f_i(x)$ gives the expected number of references between two page interruptions. Note that $1=f_i(0)\geq f_i(x)\geq f_i(p_i)=0$ whenever task i contains p_i pages, and $0\leq x\leq p_i$. The efficiency function of a task i over an interval of execution containing A of the task's memory references is defined by

$$(2.1) \qquad e_i(x) = \frac{AT_m}{R_i(A)}$$

where $R_i(A)$ is the expected real time for task i to complete A references when x pages of main memory are allocated to it. It is easy to show that

$$(2.2) \qquad R_i(A) = AT_m+Af_i(x)T_{ai} = AT_m(1+a_if_i(x))$$

whence

$$(2.3) \qquad e_i(x) = \frac{1}{1+a_if_i(x)}.$$

Note that $e_i(x)$ measures the effect of the speed reduction due entirely to the virtual memory mechanism; it measures the ability of the task to make progress in real time (neglecting queueing delays for the processor), and it therefore provides an upper bound on the fraction of real time during which the task is available to use the Processor.

For a set of $n\geq 2$ active tasks, define a partition of main memory of size M pages among the tasks at time t (the parameter t will always be used to denote real time) to be:

$$(2.4) \qquad \underline{Z}(t) = (Z_1(t),\ldots,Z_n(t))$$

where $Z_i(t)\geq 1$ is the block (the memory allocation) of pages granted to task i, for $1\leq i\leq n$, and

$$(2.5) \qquad Z_1(t)+\ldots+Z_n(t) = M.$$

Because some of the subsequent comparisons involve hypothetical (i.e., not implementable) partitions having nonintegral blocks, we have not included the constraint that $Z_i(t)$ be an integer for each i. A partition $\underline{Z}(t)$ is a fixed partition if $\underline{Z}(t)=\underline{Z}$ for some constant vector $\underline{Z}$ for all t in the time interval under consideration; otherwise $\underline{Z}(t)$ is a variable partition. A special case of fixed partition is the equipartition $\underline{Z}=(M/n,\ldots,M/n)$.

Consider a vector $\underline{x}_j=(x_{1j},\ldots,x_{nj})$ where $x_{ij}\geq 1$ for $1\leq i\leq n$ and $x_{1j}+\ldots+x_{nj}=M$. By (2.4) and (2.5), $\underline{x}_j$ is a valid fixed partition of memory. Now, suppose the dynamic storage allocation policy assigns a sequence of partitions $\underline{x}_1,\ldots,\underline{x}_j,\ldots$ at times $t_1,\ldots,t_j,\ldots$; and suppose that $\underline{x}_j\neq\underline{x}_{j+1}$ for $j\geq 1$. By the j^{th} interval of allocation we mean (t_j,t_{j+1}) -- i.e., the real time interval in which the memory is allocated among the n active tasks according to the partition $\underline{x}_j$. For $1\leq i\leq n$, therefore, x_{ij} is the amount of memory allocated to task i in the j^{th} interval of allocation. The random variable x_i denotes

the main memory allocation of task ; it takes on values in the sequence $x_{i1},x_{i2},\ldots,x_{ij},\ldots$.

For any such dynamic partitioning policy, we shall assume that each interval of allocation is of sufficient duration that the fault rate function and efficiency function of each task i converge to the respective functions f_i and e_i. As noted earlier, the experiments of Bélady and Kuehner appear to indicate that this assumption is not restrictive in practice.

Our subsequent results concerning increases in efficiency for a given set of active tasks in a given memory of size M under a variable partition will depend on the assumption that the f and e functions of each task are concave up in the range of memory allocations under consideration. A function g is said to be (strictly) concave up in an interval (u,v) if for every choice of $x_1,\ldots,x_n$ in (u,v) and every choice of a set of weights $w_1,\ldots,w_n$ such that $w_1+\ldots+w_n=1$ and $0<w_i<1$ for $1\leq i\leq n$, it is true that

$$(2.6)\qquad \sum_{i=1}^{n} w_i g(x_i) > g\left(\sum_{i=1}^{n} w_i x_i\right) .$$

This definition is equivalent to the earlier one (1.1), and is more convenient to work with. A function is said to be concave down if the inequality in (2.6) is reversed. The function e(x) typically is concave up for $0<x<p'<p$ in a p-page program, the value of p' being closer to p as the speed ratio gets large [1,2,3,4]. The function f(x) typically is concave up for $0\leq x\leq p$ in a p-page program [4,5,6].

3 ANALYSIS OF SIMPLE VARIABLE PARTITION SYSTEM

Consider a set of n active tasks operating under a variable partition policy which induces the sequence of partitions $\underline{x}_1,\ldots,\underline{x}_j,\ldots$ of M as described earlier. We shall investigate the behavior of this system for a finite interval (t_1,t_{r+1}) where $r>1$, and in which $\underline{x}_j\neq\underline{x}_{j+1}$ for $1\leq j<r$. We shall assume that each allocation interval (t_j,t_{j+1}) is sufficiently long that the fault-rate and efficiency functions of each task i converge to the functions f_i and e_i. This system will be called the simple variable partition (SVP) system, because queueing delays (experienced by tasks waiting for the processor) and swapping delays (experienced by tasks when the changes in partition $\underline{x}_j$ to $\underline{x}_{j+1}$ are inconsistent with demand paging) will be ignored. Swapping delays will be accounted for in Section 4.

Consider a particular one of the tasks, i: for $1\leq j\leq r$ let x_{ij} and A_{ij} denote, respectively, its memory allocation and the virtual time it completes in the j^{th} allocation interval (t_j,t_{j+1}). (See Fig. 3.) The total virtual time and real time for task i in the interval (t_1,t_{r+1}) are, respectively,

$$(3.1)\qquad A_i = \sum_{j=1}^{r} A_{ij}$$

$$(3.2)\qquad T = \sum_{j=1}^{r} R(A_{ij}).$$

It is especially important to realize that the expected real time of each task in each allocation interval (t_j,t_{j+1}) is the same, since these intervals are common to all the tasks; thus $R(A_{ij})=R(A_{kj})$ for $1\leq i\leq n$, $1\leq k\leq n$, and $1\leq j\leq r$. The mean virtual and real time memory allocations for task i are, respectively,

$$(3.3)\qquad V_i(x_i) = \sum_{j=1}^{r} \frac{A_{ij}}{A_i} x_{ij}$$

$$(3.4)\qquad R_i(x_i) = \sum_{j=1}^{r} \frac{R(A_{ij})}{T} x_{ij}.$$

The mean real time efficiency of task i is

$$(3.5)\qquad R_i(e_i) = A_i/T.$$

Define x_i^*, the effective memory allocation of task i, to be the value of x_i for which $e_i(x_i)=R_i(e_i)$; thus the task would have the same efficiency in a fixed partition which allocated it x_i^* pages. The mean virtual and real time fault rates for task i are, respectively,

$$(3.6)\qquad V_i(f_i) = \sum_{j=1}^{r} \frac{A_{ij}}{A_i} f_i(x_{ij})$$

$$(3.7)\qquad R_i(f_i) = \sum_{j=1}^{r} \frac{A_{ij}}{T} f_i(x_{ij}).$$

[In (3.7), note that $A_{ij}f_i(x_{ij})$ is the expected number of faults in the j^{th} allocation interval.] The assumptions that each function e_i and f_i is concave up over the range of memory allocations used in the partition sequence can be used to establish the results (3.8)-(3.15):+

$$(3.8)\qquad R_i(e_i) = \frac{1}{1+a_iV_i(f_i)} = e_i(x_i^*) = \frac{1}{1+a_if_i(x_i^*)}$$

$$(3.9)\qquad V_i(x_i) > x_i^* > R_i(x_i)$$

$$(3.10)\qquad e_i(V_i(x_i)) > R_i(e_i) > e_i(R_i(x_i))$$

$$(3.11)\qquad f_i(R_i(x_i)) > V_i(f_i) > f_i(V_i(x_i))$$

$$(3.12)\qquad V_i(f_i) > R_i(f_i)$$

The partition $\underline{R}=(R_1(x_1),\ldots,R_n(x_n))$ is a valid partition of M:

$$(3.13)\qquad R_1(x_1)+\ldots+R_n(x_n) = M.$$

+The proofs of these relations, which involve applying the defintion of concave-up function (2.6) to the defintions (3.1-3.7) are omitted here. See [7].

An approximation for the total system real time processing efficiency is

$$(3.14) \qquad e = 1 - \prod_{i=1}^{n} (1-R_i(e_i)) .$$

Finally, the total system page fault rate is

$$(3.15) \qquad \sum_{i=1}^{n} R_i(f_i) = \sum_{i=1}^{n} \frac{1-R_i(e_i)}{a_i} .$$

The results (3.8)-(3.11) are illustrated in Fig. 4 for n=2 and the partition sequence (x_1,x_2), (x_2,x_1). The important point to note from this figure and from (3.10)-(3.12) is that the variable partition causes each task i to operate at higher efficiency and lower fault rate than it would in a fixed partition in which it was allocated space $R_i(x_i)$.

As has been noted, the conditions assumed for (3.8)-(3.15) do not correspond exactly to those which would prevail in a real system, since the analysis fails to account for delays experienced by tasks queueing for the processor. Figure 5 shows Regimes I and II redrawn for the Fig. 1 tasks under conditions corresponding exactly to those used in the SVP model. Because there are no queueing delays in Fig. 5, each task completes sooner; thus $R_i(x_i)$ and $R_i(e_i)$ from SVP are too large, which introduces errors into the SVP estimates for $f_i(R_i(x_i))$ and $e_i(R_i(x_i))$. Also, relation (3.8) holds for Fig. 5 but not for Fig. 1. It should be noted that the approximation (3.14) giving the total system efficiency does work well for the tasks of Fig. 1, because in this case the tasks are identical (in Fig. 1, the total efficiency is 2R(e)). However, except for (3.8) the above relations do hold qualitatively. These results are of course exact when there is one processor available for each task; and they will serve as a good approximation when processor queueing delays are small compared to page wait times, or when the processor scheduling policy tends to make the total delay between task execution intervals the same in all tasks.

Relations (3.8)-(3.12) can be used to compare the efficiency of the SVP system with three (hypothetical) fixed partition systems:

1. Mean real time memory allocations, $\underline{R}=(R_1(x_1),\ldots,R_n(x_n))$.

2. Effective memory allocations, $\underline{X}^*=(x_1^*,\ldots,x_n^*)$.

3. Mean virtual time memory allocations, $\underline{V}=(V_1(x_1),\ldots,V_n(x_n))$.

These partitions are in general infeasible, since their blocks are not necessarily integral numbers of pages. Relation (3.10) tells us that the SVP system increases processing efficiency over the fixed partition $\underline{R}$, and the fixed partition $\underline{V}$ has higher processing efficiency than the SVP system. Similarly, the partition $\underline{X}^*$ would perform the same as the SVP system. The utility of the fixed partitions $\underline{V}$ and $\underline{X}^*$ is, the $\underline{V}$-partition provides an upper bound on the improvement of the variable partition system, and the $\underline{X}^*$-partition provides a fixed partition equivalent to the variable partition.

A further interpretation of (3.8)-(3.12) is the following. Suppose $\underline{S}=(s_1,\ldots,s_n)$ is a given fixed partition, and we are able to find a sequence of partitions $\underline{x}_1,\ldots,\underline{x}_t$ and a SVP system for which $\underline{R}=\underline{S}$; then we know that the SVP system will provide higher efficiency (relation (3.10)) and lower fault rate (relations (3.11) and (3.12)) in each task than the fixed partition system using $\underline{S}$. Moreover, if $\underline{S}$ happens to maximize the figure of merit

$$e_1(s_1)+\ldots+e_n(s_n)$$

which represents the expected number of processors that could be fully utilized by the active tasks, then any variable partition system in which $\underline{R}=\underline{S}$ will increase this figure of merit and would therefore improve over the best fixed partition system.

We have not investigated how to find a SVP system such that $\underline{R}=\underline{S}$ for some given arbitrary $\underline{S}$. If, however, $\underline{S}$ is an equipartition, an answer is easily obtained. Define a round robin partition to be a SVP which cycles a set of memory sizes $x_1,\ldots,x_n$ among the tasks using equal real time intervals of allocation — i.e., $R(A_{i,j})=T/n$.+ Other cases in which round robin partitions will improve over equipartitions include the case of n identical tasks where the length of the allocation interval is determined by the passage of K page faults, or by K units of virtual time, in the task having the largest memory allocation (K is a parameter). This corresponds very closely to the experiment reported by Belady and Kuehner, and thus corroborates their observation.

4 SWAPPING EFFECTS

The foregoing analysis has not accounted for the possibility that the partition sequence induced in a SVP system is inconsistent with demand paging — e.g., memory allocations are changed at non-page-fault times, or a task's new allocation differs in size by more than one page from its previous allocation. Thus with two tasks, for example, it is possible to conclude that the partition sequence (0,M)(M,0) repeated indefinitely by a round robin partition — which corresponds to a pure swapping strategy — would maximize the gain in efficiecy, $R_i(e_i)-e_i(R_i(x_i))$. That this need not be so can be established by correcting our previous analysis for the effect of swapping. Let F_i denote the effective number of page faults induced in task i by swapping in the time interval (t_1,t_{t+1}) under consideration. The actual efficiency for this task is thus

$$(4.1) \qquad R_i^*(e_i) = \frac{A_i}{T+a_iF_i} = (1-q_i)\frac{A_i}{T}$$

where

$$q_i = \frac{a_iF_i}{T+a_iF_i} .$$

+To put this more precisely, let $\underline{x}=(x_1,\ldots,x_n)$ be a valid partition. Define the permutation operator P such that $P(\underline{x})=(x_2,\ldots,x_n,x_1)$; define also $P^k(\underline{x})=P(P^{k-1}(\underline{x}))$ for $k>1$. One possible round robin partition would use the partition sequence $\underline{x}_1,\ldots,\underline{x}_n$ cyclically, where $\underline{x}_j=P^{j-1}(\underline{x})$ for $1\le j\le n$; another would use the sequence $\underline{x}_1,\ldots,\underline{x}_n,\underline{x}_n,\ldots,\underline{x}_1$.

In (4.1), q_i is the fraction of time during which swapping renders the task idle. Since $R_i(e_i)=A_i/T$, our previous results permit us to conclude only that $R_i^*(e_i)=(1-q_i)R_i(e_i) > (1-q_i)e_i(R_i(x_i))$, whence it is possible for

$$(4.2) \quad e_i(R_i(x_i)) > R_i^*(e_i) > (1-q_i)e_i(R_i(x_i)).$$

Evidently, the maximum amount of swapping which can be tolerated in the interval (t_{r-1},t_r) corresponds to the larges q_i for which $e_i(R_i^{-1}(x_i)) \leq (1-q_i)R_i(e_i)$, which is equivalent to the condition

$$(4.3) \quad q_i \leq 1 - \frac{e_i(R_i(x_i))}{R_i(e_i)} .$$

A proposed SVP system using swapping would have to be evaluated against (4.3) to determine if in fact a gain of processing efficiency would result.

5 CONCLUSION

The simple variable partition (SVP) system model considers an arbitrary sequence of valid partitions induced on a multiprogrammed memory shared by the active tasks. Under the assumptions that the efficiency and fault rate functions both are concave up -- assumptions which the Belady and Kuehner experiments indicate are restrictive in practice -- the SVP model predicts an increase in real time processing efficiency and in virtual-time memory allocation in each task, compared with a fixed partition in which each block's size is the same as the corresponding mean block size in the variable partition. The SVP has two limitations: it fails to account for possible losses of efficiency, due to swapping induced by the variable partitioning scheme and to queueing of active tasks for the processor. The first limitation is easily removed by a simple extension of the arguments.

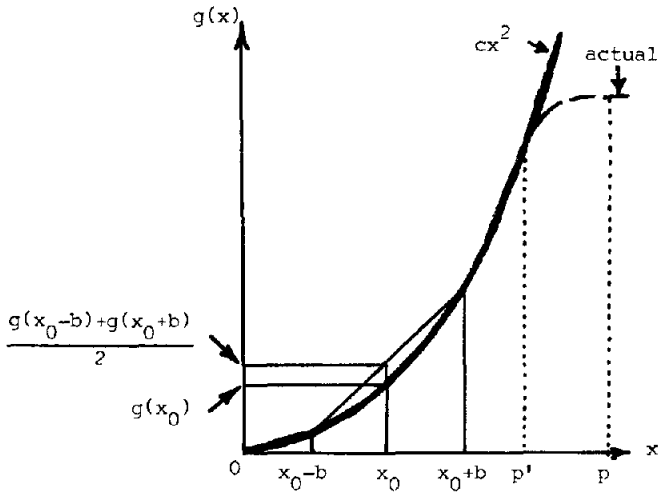


Fig. 2. Mean execution interval curve.

The second limitation can be removed by considering a queueing network model in which the tasks cycle between the processor and the auxiliary memory device, each task's service time at the processor being a function of the memory size allocated to it. Our investigations of queueing network models have for reasons of tractable analysis been limited to the rather elementary case of two identical tasks; we have not completed a study to determine whether or not the concavity properties the model requires for variable partitioning to be more efficient are met in practice. We are also investigating how severe a limitation the omission of queueing effects from the SVP model is. Since the results are preliminary and so far are inconclusive, we have not included them here.

As noted before, the SVP results are exact when there is one processor available for each task; and they will serve as a good approximation when processor queueing delays are small compared to page wait times, or when the processor scheduling policy tends to make the total delay between task execution intervals the same in each task.

REFERENCES

1. L. A. Belady, "Biased replacement algorithms for multiprogramming." IBM T. J. Watson Research Center, Research Note NC697 (March 1967).

2. L. A. Belady and C. J. Kuehner, "Dynamic space sharing in computer systems." Comm. ACM 12, 5 (May 1969), 282-288.

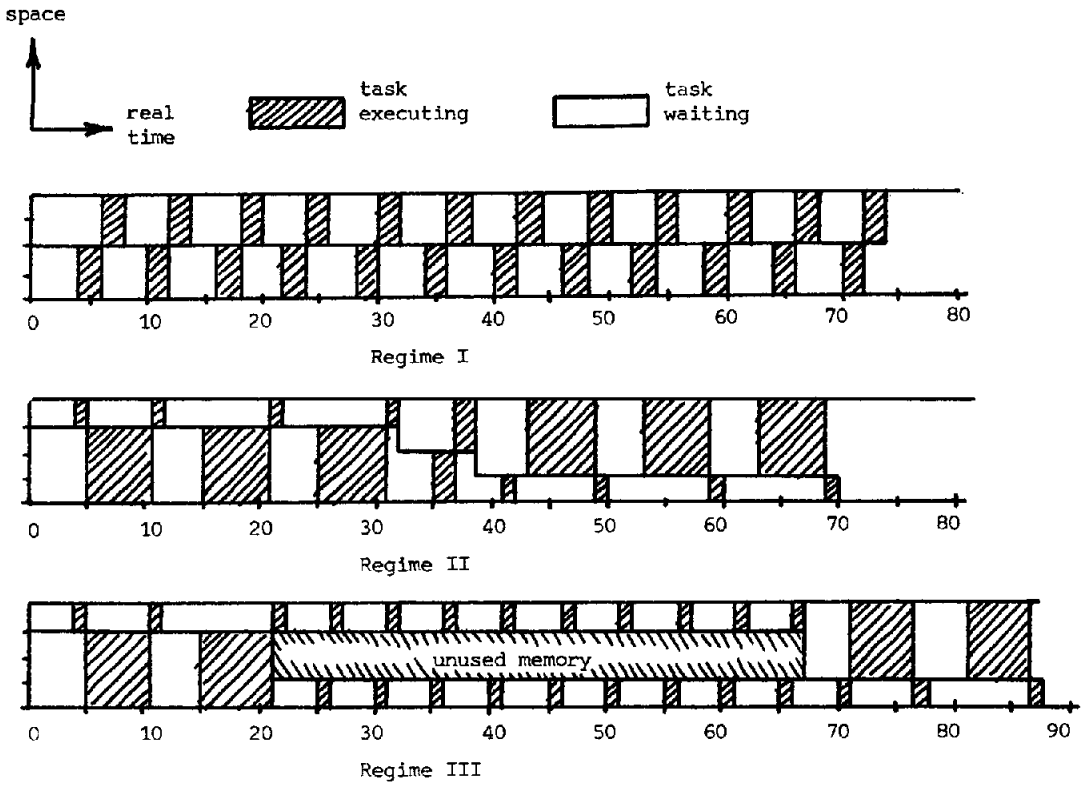
3. W. W. Chu and H. Opderbeck, "The page fault frequency replacement algorithm." AFIPS Conf. Proc. 41 (1972 FJCC), 597-609.

4. J. R. Spirn, "Program locality and dynamic memory management." Ph.D. thesis, Dept. Elec. Eng'g, Princeton University (March 1973).

5. L. A. Belady, "A study of replacement algorithms for virtual storage computers." IBM Sys. J. 5, 2 (1966), 78-101.

6. E. G. Coffman and L. C. Varian, "Further experimental data on the behavior of programs in a paging environment." Comm. ACM 11, 7 (July 1968), 471-474.

7. P. J. Denning and J. R. Spirn, "Behavior of variable storage partitions." Computer Science Dep't, Purdue University, TR-102 (Aug 1973).



Memory Size x (pages)	Fault Rate f(x)	Execution Interval 1/f(x)
1	1.00	1
2	0.50	2
3	0.17	6

Measures	REGIMES I	II	III
1. R time eff, syst	0.65	0.69	0.55
2. R time mem, task	2.0	2.0	1.5
3. V time mem, task	2.0	2.6	2.0
4. V time flts, task	0.50	0.33	0.58
5. R time flts, syst	0.324	0.23	0.318

Fig. 1. Example.

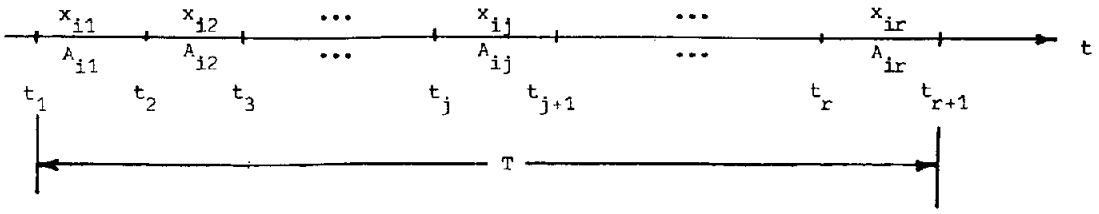
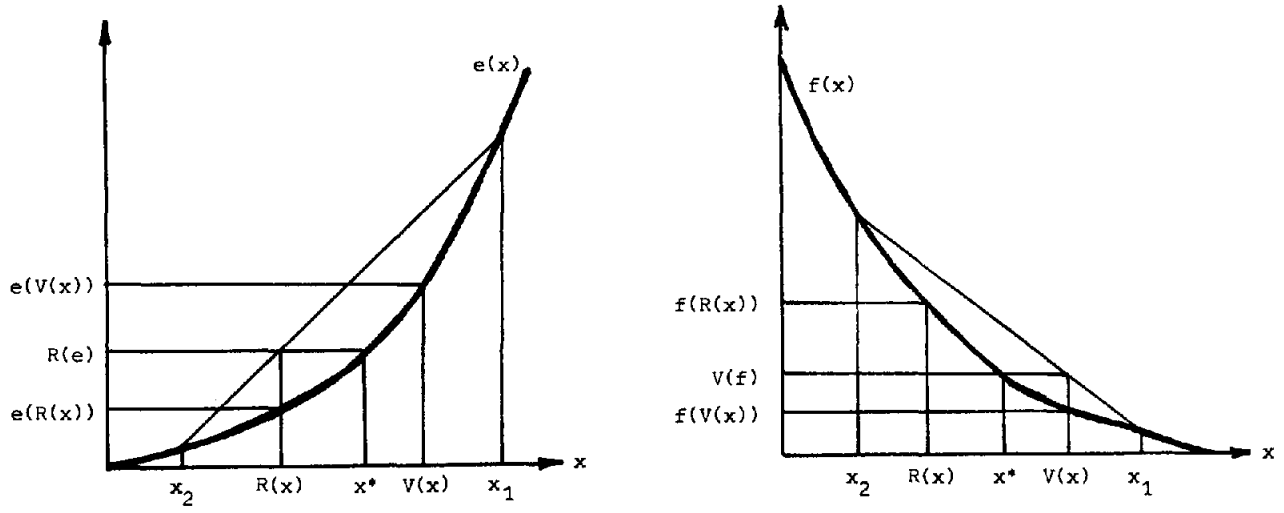


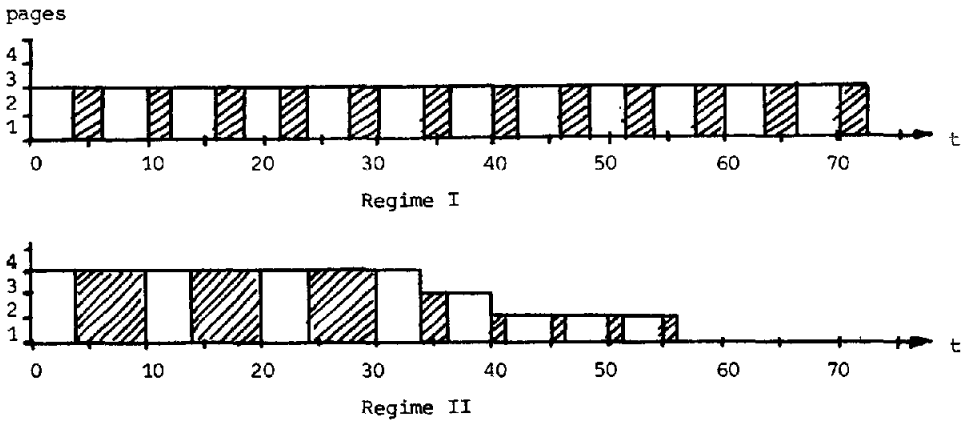
Fig. 3. One task under variable partitioning.



(a) efficiency of a task

(b) fault rate of a task

Fig. 4. Interpretation of results for n=2.



MEASURES FOR ONE TASK

	Fig. 5	Fig. 1
V(x)	2.6	2.6
R(x)	2.3	2.0
f(R(x))	0.40	0.50
V(f)	0.33	0.33
f(V(x))	0.30	0.30
e(V(x))	0.45	0.45
R(e)	0.43	0.34
$\frac{1}{1+aV(f)}$	0.43	0.43
2R(e)	-	0.69
$1-(1-R(e))^2$	0.69	-

Fig. 5. Example.