INFORMATION TO USERS

This material was produced from a microfilm copy of the original document. While the most advanced technological means to photograph and reproduce this document have been used, the quality is heavily dependent upon the quality of the original submitted.

The following explanation of techniques is provided to help you understand markings or patterns which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting thru an image and duplicating adjacent pages to insure you complete continuity.

2. When an image on the film is obliterated with a large round black mark, it is an indication that the photographer suspected that the copy may have moved during exposure and thus cause a blurred image. You will find a good image of the page in the adjacent frame.

3. When a map, drawing or chart, etc., was part of the material being photographed the photographer followed a definite method in "sectioning" the material. It is customary to begin photoing at the upper left hand corner of a large sheet and to continue photoing from left to right in equal sections with a small overlap. If necessary, sectioning is continued again — beginning below the first row and continuing on until complete.

4. The majority of users indicate that the textual content is of greatest value, however, a somewhat higher quality reproduction could be made from "photographs" if essential to the understanding of the dissertation. Silver prints of "photographs" may be ordered at additional charge by writing the Order Department, giving the catalog number, title, author and specific pages you wish reproduced.

5. PLEASE NOTE: Some pages may have indistinct print. Filmed as received.

Xerox University Microfilms
300 North Zeeb Road
Ann Arbor, Michigan 48106

DYNAMIC STORAGE ALLOCATION

by

Roy S. Hallquist

A DISSERTATION

Presented to the Faculty of

The Graduate College in the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Doctor of Philosophy

Department of Electrical Engineering

Under the Supervision of Professor Don J. Nelson

Lincoln, Nebraska

May, 1973

TITLE

DYNAMIC STORAGE ALLOCATION

BY

Roy S. Hallquist

APPROVED	DATE
Don J. Nelson (Chairman)	May 15, 1973
George Nagy	May 15, 1973
Edwin Lowenberg	May 15, 1973
Wendall Robison	May 15, 1973
Allen Edison	May 15, 1973

SUPERVISORY COMMITTEE

GRADUATE COLLEGE UNIVERSITY OF NEBRASKA

GRAD 52002 300 5-66(7)

ACKNOWLEDGEMENTS

I am deeply indebted to Professor Don J. Nelson, my adviser, for his invaluable help in preparing this document and for the many long hours spent checking my work for the errors it contained;

to Professors Edwin C. Lowenberg and George Nagy for their suggestions and encouragement in this research;

to Cyrus Hall for help with the statistics and proofreading;

and most of all, to my wife, Nancy, who patiently retyped the entire document many times, who continually provided encouragement when the goal seemed beyond reach, who put up with my irritability after long nights staring at equations and computer printouts - without her help it could never have been completed.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONT'D)

TABLE OF CONTENTS (CONT'D)

LIST OF FIGURES

LIST OF FIGURES (CONT'D)

LIST OF FIGURES (CONT'D)

LIST OF FIGURES (CONT'D)

Chapter 1

INTRODUCTION

1.1 History

Storage allocation in early computer systems consisted of deciding where in storage the variables or arrays of variables were to be located or deciding the relative locations of various segments of program text. Once the locations for the variables or program text had been selected the items remained fixed throughout the execution of the program. This is now called static allocation. Programs were written in machine language or in simple assembly languages, the machines were slow relative to todays computers, and each program was written to solve a particular and restricted problem. These programs had no need for any storage allocation other than the static allocation made by the programmer when he assigned locations to the variables, arrays and sections of program text.

Soon general purpose programs began to appear, possibly because programmers grew weary of writing nearly identical programs for a number of similar problems. In some of these programs, storage was assigned by the program rather than the programmer. In one application of a program, 500 words of storage might be required for array A and 5000 words for

array B. In a slightly different application the storage assignment might be reversed.

To allow a single program to be used in both cases either all the arrays had to be set at their maximum sizes, or the space for each array had to be allocated by the program after it determined the nature of the particular problem it was to solve. Recompiling the program to adjust the sizes of the arrays to a particular problem was not desirable for programs to be used as utilities by a large number of people. The ability to determine which arrays should be set at which sizes would be beyond many users, and requiring them to do so would unnecessarily restrict the application of the program. Also recompilation took a long enough time that the program would be more efficient if it allocated the arrays itself.

An example of this is a general purpose program to solve a number of different statistical problems. Each of these problems would require different sizes for the internal arrays used by the program. The areas needed when the program was doing a factor analysis might be completely unnecessary or needed in different sizes when the program was computing correlation coefficients. The program would read a set of parameter cards indicating what it should do and then take its storage for the required areas from a common pool. This allocation is still static since the program makes the

allocation before it begins its intended job, and the allocation remains fixed throughout.

A similar static allocation is made by a compiler as it goes through the lists of variables and allocates each to a fixed storage location. In this case the allocation is made from an imaginary storage pool since the real storage is in use by the compiler itself. The compiler may be allocating storage for data and storage for instructions at the same time if the language is one which permits intermixing instructions with either implicit or explicit data declarations such as used by FORTRAN. If the compiler is to do as much as possible on a single pass of the source program and the operating system does not support a link editor for combining programs with subprograms, the data items may be allocated by placing them one after the other at one end of storage and the instructions allocated by placing them one after another starting at the other end of storage.

Both of these examples show a general purpose program allocating storage which is fixed and will not change throughout the lifetime of the process for which it is being allocated. The allocation process is different from the storage allocation done by a programmer working in a simple assembly language or in machine language only because it is done by a program rather than a human. Although the method

used by the program and the person is probably similar in this case, writing a program to do what a person does is often difficult, especially the first time.

As more complex and more general programs were written the ability to allocate storage and to free it as circumstances changed while the program was running became necessary. The process of allocating and freeing storage during the execution of a program is called dynamic storage allocation. An example of this requirement occurs in the list processing languages such as LISP, SLIP, or SNOBOL which form, modify and destroy character strings, lists, arrays, and tree structures while the program runs. These languages are designed and used for problems in symbol manipulation involving complex and often large data structures such as compilers, theorem provers, and simulators. Since the structures formed depend on the data presented at run time, static allocation at compile time is not possible.

One early attempt to resolve the problem of insufficient storage on computers, which takes an approach different from dynamic storage allocation, is called paging. The goal of paging is to present to the programmer the appearance of a memory as large as he would ever need. If this is done then the programmer is not concerned with whether his data will fit in storage, or what parts of it are needed at the

same time. The programmer or a compiler simply produces a program for the largest possible amount of storage which might be needed at any time and the operating system translates the program into real storage when the program is run.

To accomplish the appearance of unlimited storage the large imaginary storage, called virtual storage, is broken up into small blocks called pages. Usually these blocks are all of one size but in some systems there are several different sizes. These pages are placed in a file called the page file which is usually located on a disk or a drum. The device holding the page file is called the paging device. During an interval of a program's execution, the only blocks which need to be in real storage are the ones actually referenced within that interval. All other blocks can remain in the page file until needed. The paging system must arrange to have the various blocks in real memory as they are needed.

In order to have the blocks in memory as they are needed during the execution of a program, a paging system must know whether an address generated by an executing program is in one of the pages located in memory or is in one of the pages of the page file. If the address refers to a page which is not in memory, the system must arrange to

have it brought in. If the address refers to a page which is in memory, the address generated by the program must be translated to a real address before the data or instruction can be retrieved. This is because the original addresses used in the program were virtual addresses and may extend beyond the size of the real memory. Special hardware is usually used to translate the addresses since real addresses must be generated for every data reference and every instruction fetch.

When the system needs to bring a new page into the real memory some page already there must usually be replaced. It is not usual to remove a page as soon as it has been referenced because it might be needed again in the near future. As a result the replacement problem reduces to deciding which block is least likely to be needed in the near future. If this least likely block has not been modified since it was brought in from the page file it may be deleted because a copy remains on the page file. If it has been modified then it must be copied to the page file before it is deleted so that the modifications made while it was in real storage will not be lost.

A complete list of the future references to all pages would provide the information necessary to minimize page movement between the page file and real storage. Such

information is seldom available. As a result pages are brought in when an executing program makes reference to some data or instruction in a page which is not resident in real storage at the time. The reference to a non-resident page is called a page fault and is detected by the address translation hardware. A system which does not pre-load pages but waits until a page fault occurs to load them is said to page on demand.

The decision of which page to replace when a page fault occurs may be made with reference to the previous statistical behavior of the programs. Various methods of page replacement have been tried including random, least recently used, longest resident page, and the working set model (Denning 1968).

Recent interest in dynamic storage allocation is due to multi-programming operating systems and the impact of interactive computing from remote terminals. A multi-programming operating system must allocate storage for each of the several programs it executes concurrently. The programs are placed in an input queue when they are submitted to the operating system, and as sufficient resources for each are freed they are taken from the queue and executed. Often there will be more programs in the queue than can be executed concurrently and the system will operate for long

periods of time at its maximum rate, limited only be the number of programs which will concurrently fit in storage. As is common with queuing problems of this type a small increase in the number of jobs run concurrently may make a large difference in the average turnaround time for jobs.

Remote terminal systems are often designed to operate with certain expected numbers of active terminals, and do not reserve sufficient storage for all possible terminals to be processing concurrently. In many systems each terminal which is processing commands will need a storage area for the command and data text transmitted to and from the terminal, file buffers for program or data storage, and work areas for intermediate results and status information. In addition, these items may be required in different sizes and numbers depending on the function being performed for the terminal user and the type of terminal. A terminal which is inactive may require no storage areas or only a small amount of storage. To conserve storage these areas may be allocated from a single dynamic storage allocation region as needed and returned to it when no longer needed.

Even if complete and general solutions to the problems of paging systems were to be found, dynamic storage allocation would continue to be interesting and useful. Efficient paging systems require that a considerable amount

of additional hardware be built into the processor and many small inexpensive computers do not have and probably will not have this additional hardware for some time. There are many computers in use today which are large, expensive and fast but do not have the hardware necessary to economically support paged operation. The hardware associated with page replacement will usually be designed to maximize the overall utilization of the computer system and not to favor any particular subsystem or program. But if a remote terminal or real time program is being run, it may be desirable to give up some efficiency in the total computer system in return for faster response time for that application. If the application could potentially use a large amount of storage it would be better to use a dynamic storage allocation system. A real time system may not be able to tolerate the delays in transferring data to and from the page file and may have to keep it all in core. If the amount of data is variable and potentially large, dynamic storage allocation may be essential.

1.2 Fundamental Principles

The general allocation problem consists of selecting one or more items from a group of available items. Usually there are constraints which must be satisfied in selecting the number of items requested. If the selection is to be made from a pool of equipment such as cars, one such constraint might be the requirement to equalize the rate of wear on all the cars so they can be replaced at one time to yield a better quantity discount and at the same time use the entire pool for the longest possible time between replacement periods. Another example is the allocation of classrooms to courses where one constraint might be that the maximum number of students are to have all their classes between 9 A.M. and 2 P.M. to allow as much unbroken time as possible for independent study.

A special type of allocation process for computers called the storage allocation process is used to select a block of units for storing either programs or data. The block is selected from a pool of equal size storage units and the block must be composed of adjacent units. When selecting blocks for programs the units are usually much smaller than the size of the block being selected; an allocation of an entire program may require 10^4 to 10^6 total units. In other cases where storage is being allocated for

single variables within a program, the block may be composed of a single unit. A common constraint in storage allocation is that the areas be arranged with relation to one another in such a way that there are few unused units between them.

While static allocation is used when all blocks will remain in use for the same length of time, dynamic storage allocation refers to a type of storage allocation where the blocks selected are used to store something for various lengths of time. When a block is no longer needed its storage units are returned to the pool of available units to be used in some subsequent allocation. Constraints which may be used in dynamic storage allocation problems are: to keep the total size of the pool required as small as possible; to make the routines or programs which perform the allocation operate as quickly as possible; to provide the highest probability that an allocation can be successfully made; or any combination of these.

If the pool of storage units is considered to be a finite set of elements then the dynamic storage allocation problem is concerned with selecting contiguous subsets of these elements in such a manner that no element is a member of any two selected subsets. The pool of storage M, or dynamic allocation region, is a set of k elements well ordered by an index i, $M = \{M_i \mid 1 \leq i \leq k\}$. When an

allocation is made a contiguous subset B_j of size n_j is selected from those elements of the region M which have not been selected for any other subset. After a length of time t_j the elements of the subset are returned to the pool and the subset is deleted, making the elements available for further allocation. Each selection is performed at some point in time and no two allocations are made simultaneously so the allocated subsets may be ordered by an index j indicating the order in which they were constructed. Subset B_j was selected before B_{j+1}. Each subset is considered to exist only for the time the allocated block is in use.

$$B_j \subset M,$$

$$B_j = \{M_i \mid s \leq i \leq r,\ n = r - s + 1\}, \text{ and}$$

$$B_j \cap B_k = 0 \ \forall\ j \neq k.$$

Dynamic storage allocation may also be represented by an allocation function, a, which produces the allocated subsets. The allocation function is dependent on the total region M from which the allocation is made, the number of elements n_j which are to comprise the allocated block B_j and the length of time t_j the allocated block will exist and be used.

$$B_j = a(M, n_j, t_j).$$