MIT/LCS/TR-171

A MULTI-PROCESS DESIGN OF A PAGING SYSTEM

Andrew R. Huber

December 1976

The research reported here was sponsored in part by Honeywell Information Systems Inc., and in part by the Air Force Information Systems Technology Applications Office (ISTAO), and by the Advanced Research Projects Agency (ARPA) of the Department of Defense under ARPA order No. 2641 which was monitored by ISTAO under contract No. F19628-74-C-0193.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LABORATORY FOR COMPUTER SCIENCE
(formerly Project MAC)

CAMBRIDGE MASSACHUSETTS 02139

A MULTI-PROCESS DESIGN OF A PAGING SYSTEM *

by

Andrew R. Huber

ABSTRACT

This thesis presents a design for a paging system that may be used to implement a virtual memory on a large scale, demand paged computer utility. A model for such a computer system with a multi-level, hierarchical memory system is presented. The functional requirements of a paging system for such a model are discussed, with emphasis on the parallelism inherent in the algorithms used to implement the memory management functions.

A complete, multi-process design is presented for the model system. The design incorporates two system processes, each of which manages one level of the multi-level memory, being responsible for the paging system functions for that memory. These processes may execute in parallel with each other and with user processes. The multi-process design is shown to have significant advantages over conventional designs in terms of simplicity, modularity, system security, and system growth and adaptability. An actual test implementation on the Multics system was carried out to validate the proposed design.

Thesis Supervisor: David D. Clark
Title: Research Associate

*This report is a minor revision of a thesis of the same title submitted to the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, on May 19, 1976 in partial fulfillment of the requirements for the degrees of Master of Science and Electrical Engineer.

ACKNOWLEDGEMENTS

I wish to thank my advisor, Dave Clark, for his patience in what has been a rather protracted effort. The original idea for this thesis is due to him. Three people were of great help to me in implementing the design presented in this thesis: Bernie Greenberg explained many of the mysteries of Multics page control and gladly contributed his time, knowledge and enthusiasm. Bob Mabee implemented some of the code necessary to permit page control to be implemented on Multics as parallel processes, and helped in getting the design working on Multics. Doug Wells was expert at finding my programming errors and explaining the pitfalls of PL/1. Without their help, I would still be debugging. Many other members of the Computer System Research Division contributed in ways too numerous to mention.

The research reported here was sponsored in part by Honeywell Information Systems Inc., and in part by the Air Force Information Systems Technology Applications Office (ISTAO), and by the Advanced Research Projects Agency (ARPA) of the Department of Defense under ARPA order No. 2641 which was monitored by ISTAO under contract No. F19628-74-C-0193.

TABLE OF CONTENTS

ABSTRACT 2

ACKNOWLEDGEMENTS 3

LIST OF FIGURES 7

CHAPTER 1: Introduction 8

1.1 Processes 9

1.2 Paged Systems 10

1.3 Paging Systems as Processes 11

1.4 Summary of Thesis 12

CHAPTER 2: Basic Objects and Functions of Paging Systems 15

2.1 Page Control Objects 17

2.1.1 Pages 17

2.1.2 Page Frames 18

2.1.3 Address Translation Registers 20

2.1.4 Segments and the File System 21

2.2 Page Control Functions 26

2.2.1 Memory Allocation 27

2.2.2 Memory Deallocation 31

2.2.3 Memory Reconfiguration 35

2.2.4 Memory Wiring 36

2.3 Summary 38

CHAPTER 3: Designs for Paging Systems 40

3.1 Paging System Structures 41

3.2 Multics' User Process Page Control 43

3.2.1 The Current Multics Paging System 44

3.2.2 Multics as a Single System Process Paging System 48

3.3 Multi-process Combination Paging Systems 50

3.3.1 A Two Process Paging System 51

3.3.2 Hoare's Structured Paging System 61

3.3.3 Saxena and Bredt's Hierarchical Operating System 65

3.3.4 System Versus Combination Paging Systems 68

3.4 Advantages of Multi-process Paging Systems 70

3.4.1 Simplicity 71

3.4.2 Modularity 74

3.4.3 Security 76

3.4.4 Expandability 77

CHAPTER 4: A Multics Implementation of Multi-process Page Control 80

4.1 The Multics Implementation 80

4.1.1 Size and Scope 81

4.1.2 Differences from the Model 82

4.1.3 Performance 85

4.2 The Interface with Segment Control 90

4.2.1 Necessary Segment Control Functions 91

4.2.2 Complications Introduced 92

4.3 Other Page Control Functions 94

CHAPTER 5: Eliminating the Global Page Table Lock 96

5.1 The Strategy 97

5.2 Locks on Segments 104

5.3 Multics Complications 107

CHAPTER 6: Conclusion 111

BIBLIOGRAPHY 113

APPENDIX A: Changes made to Multics standard page control 116

APPENDIX B: Components of Multi-process page control 117

APPENDIX C: Code from Multi-process page control 119

LIST OF FIGURES

Figure 2.1: Model of a multi-level hierarchical memory system 16

Figure 2.2: Translation of virtual address page i, word n 22

Figure 2.3: Virtual address translation with segmentation 24

Figure 2.4: Allocating memory to pages 30

Figure 3.1: Multics page control 45

Figure 3.2: Algorithm of the Core manager process 53

Figure 3.3: Algorithm of the page frame allocating procedure 55

Figure 3.4: Binding a page to a page frame 58

Figure 3.5: Multi-process page control 60

Figure 4.1: Performance of multi-process page control 87

Figure 5.1: Processes accessing page control data bases 98

Figure 5.2: Processes locking multiple locks 100

CHAPTER 1

Introduction

This thesis will examine a general multiple process design of a paging system. Such a design could be used in the implementation of a demand paged memory in any suitable computer operating system. As computer systems have grown in size, the operating systems have also greatly increased in size, scope, and complexity, especially so-called computer utilities and large time shared systems. The design presented here offers a method for simplifying one large component of such systems: the memory management task. The resulting system is less complex yet readily expandable to accomodate future systems growth.

There are two central concepts underlying the design presented in the following chapters. These are the concept of a process as an abstraction of a program in execution, and the concept of paging as a means of implementing a virtual memory. Before the motivation for designing a paging system as cooperating processes can be discussed, these two concepts warrant closer examination.

1.1 Processes

The essence of a process is the execution of a program. Numerous definitions of a process are given by various authors [Da68] [Ha70] [Di68a] but all include the notion of an execution point passing through the instructions of some program. Thus a process is an abstraction of the locus of control that passes through an executing procedure [De66].

The address space of a process, that is, the set of all memory addresses the process may reference, is an important component of a process. In fact, the address space of a process influences the computations the process can carry out to such an extent that we include the address space in our definition of a process. A process, then, consists of a pair: an execution point, or locus of control, and an address space.

The process abstraction provides a natural way of describing an operating system. Each user's work is viewed as a process, i.e. a task to be performed. The operating system itself is seen as a task or process manager. The various facilities the operating system provides, such as memory or device management, can themselves be implemented as processes. Two good examples of systems designed around the process concept in this manner are Dijkstra's THE system [Di68b] and a multiprogramming system described by Hansen in [Ha70].

In any multi-processor computer system, processes offer a straightforward technique for achieving multi-processing (the simultaneous execution of two or more programs). Any physical processor (CPU) in the system can execute any user or system process. This permits the operating

system to be multi-processed, i.e. different functions of the operating system may be executed in parallel. Parallel execution of the operating system, or one component of the operating system (the paging system) is a central theme in this thesis.

1.2 Paged Systems

Paging is a common strategy for solving the memory allocation problem, one of the chief tasks any operating system must perform. Examples of systems using paged memories include Multics [Da68], TENEX [Mu72], and IBM's VS systems [Wh74] [Sc73].

In a paged system the address space of a process is divided into contiguous pieces of fixed size called pages. Physical memory is partitioned in the same manner into contiguous blocks called page frames. When allocating memory to a process, any available page frame may be allocated to hold any page.

Usually the memory of a large computer utility is organized into several physical levels L1, L2, ... Ln. The access time and capacity of a level increases with n, and each level is normally a different type of memory device. For such devices, the smaller the access time the higher the cost per bit and therefore the smaller the capacity. By combining such components with widely varying speeds and size into a multi-level memory an overall memory system can be constructed whose capacity equals that of its largest component yet whose effective speed approaches that of its fastest component.

In such multi-level memories a process may reference only pages residing in the primary (level 0) memory. Referencing a page not allocated a page frame at the lowest level results in a page fault, an event which causes the necessary operating system mechanisms to be invoked to allocate a level 0 page frame to the page and cause the page to be read into that page frame. The operating system modules and the data bases these modules use to perform this task are called the paging system, or page control. Page control is a resource manager; page frames being the resource page control manages.

1.3 Paging Systems as Processes

There are many alternative methods for organizing and implementing the paging system functions. The most widely used is to have the user process itself perform the necessary memory management functions when needed, just as with any other system call. That is, the code that carries out the necessary operations to allocate page frames is executed in the user's address space just like a user program.

This thesis will examine several ways for organizing paging systems as processes. The paging system can be broken down into several activities, for example, removing pages from primary memory when it becomes full to make room for other pages. In such a system, each activity of the paging system can be made a separate process, with its own address space. Thus the paging system becomes a set of cooperating sequential processes, running in parallel and asynchronously. Such

systems will be called multi-process paging systems, and this thesis will argue that such systems offer significant advantages in simplicity, modularity, security and expandability over more conventional designs.

The work described in this thesis differs from a multiple process paging system proposed by Hoare [Ho73] in the number of processes used and the function assigned to each. The model developed by Saxena and Bredt [Sa75] is closer to what is described here. However Saxena and Bredt use a multi-level paging system that distinguishes user page faults from page faults caused by system processes, a distinction found unnecessary in the design presented in Chapter 3. These differences and similarities are considered in more detail in section 3.3.

1.4 Summary of Thesis

The remainder of this thesis will examine the design and implementation of paging systems for a large computer utility as several cooperating processes. The Multics system will be used as a model of such a computer utility. Multics was chosen because it is typical of large, sophisticated time shared systems and incorporates both of the prerequisite ideas already mentioned: a multi-level, demand paged memory, and processes. Therefore the basic concepts are already present and need not be added.

Currently a major research effort is being made to engineer a security kernel for Multics [Sc75]. Redesigning the paging system contributes to the certification of such a kernel by reducing both the

size and complexity of the code that must be verified. The original impetus for the work described in this thesis was the need for simplifying kernel mechanisms such as paging.

Chapter 2 discusses the basics of paging systems in detail. The objects page control uses to implement a large demand paged virtual memory are examined. Functions which the paging system must provide to the rest of the operating system are listed and discussed.

In Chapter 3 paging systems are classed into three groups based on their organization. User process paging systems, illustrated by Multics, are those where the paging functions are performed in the user's process. System process paging systems utilize special system processes to implement the paging functions. Combination paging systems, using features of both of the other two types, include designs appearing in the literature due to Hoare [Ho73] and Saxena and Bredt [Sa75]. The author's design for a combination multi-process paging system in presented, in which memory allocation is performed in the user's process but other page control functions are done in system processes. The significant advantages of both types of multi-process paging systems are considered at some length.

A test implementation of the design on the Multics system is presented in Chapter 4, concentrating on the difficulties arising in an actual implementation and the insights gained from such an effort. The results of this test implementation are compared with the current implementation to see how well the goals of a multi-process implementation can be realized.

Techniques for exploiting fully the parallelism available in a

multi-process paging system by eliminating global locking strategies are examined in Chapter 5.

Chapter 6 concludes the thesis by summarizing the important results and drawing some final conclusions and observations.

The three appendices present additional information on the implemented multi-process page control described in Chapter 4. Appendix A compares the design to the standard Multics page control. Appendix B lists the components of the implemented design, and Appendix C contains some of the actual PL/1 code from important portions of the design.

CHAPTER 2

Basic Objects and Functions of Paging Systems

In Chapter 1 the paging system, or page control, was loosely defined to be those procedures and data bases necessary to resolve page faults and provide the memory allocation task. This chapter will focus on exactly what functions and services page control must provide to the rest of the system and what objects page control must implement in providing these functions. Such a description will help suggest how the parts of page control can best be divided along functional lines into several processes.

Figure 2.1 illustrates the model of a memory system that will be assumed in the remainder of this thesis. The memory system is a hierarchical, multi-level memory consisting of three levels: 1. Primary memory, in which any data referenced by a processor must reside. 2. The paging device, or backing store (which need not be a single device) which acts as a large, high speed buffer between primary and secondary memory. 3. Secondary storage, which provides long term storage of data and programs. For example, in such a system primary memory is often high speed core memory; the paging device is often a drum (or a bulk store device in the case of Multics); and disks and perhaps tape normally

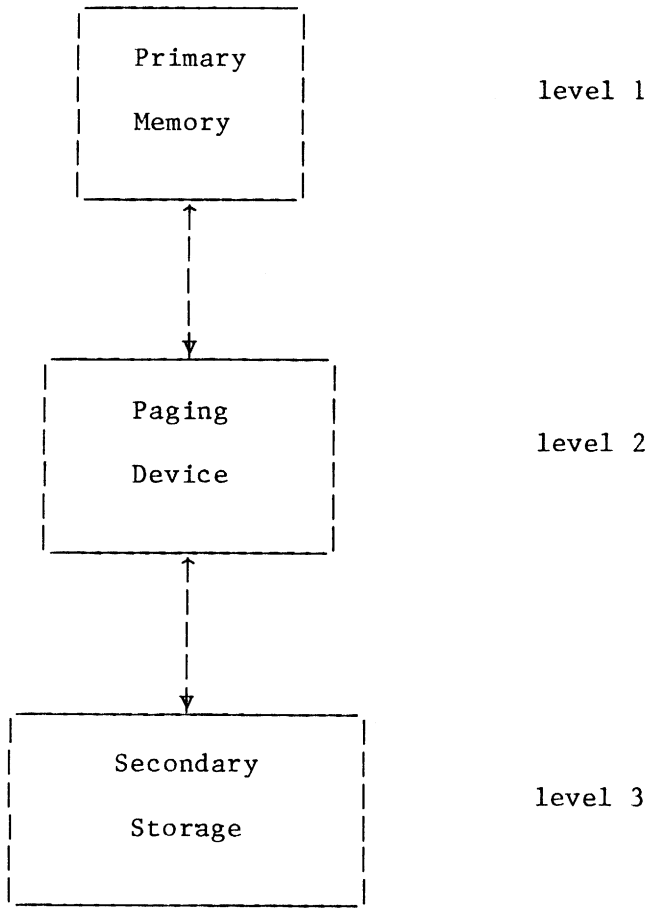

Figure 2.1

Model of a multi-level hierarchical memory system

provide secondary storage.

While the model shown incorporates three levels of memory, more or fewer are possible. The actual number of levels should not be crucial in a well designed system. Indeed, the design presented in Chapter 3 will be seen to adapt easily to a multi-level memory with any number of levels.

Pages are moved from level to level by the paging system. It is assumed that a page may reside in any or all levels of the memory at any given time; however only one copy of the page may exist in each level. If multiple copies of a page do exist in the memory hierarchy, they may not all be identical. The most up to date version of a page will be the copy in primary memory (if there is one), then the paging device copy (if there is one).

2.1 Page Control Objects

There are three objects of fundamental importance to page control: pages, the basic allocatable unit of virtual memory; page frames, the corresponding unit of physical memory; and address translation registers, which translate virtual addresses into absolute physical memory addresses.

2.1.1 Pages

In paged systems, the address space of a process is divided into units called pages, or sometimes virtual pages. A page is an abstraction of a portion of a process's address space, a set of consecutive virtual

addresses (hence the term "virtual page"). Procedures and data are both broken into pages, although this division into pages is invisible to the programmer.

The number of consecutive virtual addresses (locations) in a page is the page size. The page size is typically fixed at a power of two, and generally ranges from 128 to 4,096. The page size is usually determined by characteristics of the hardware in order to optimize performance of secondary memory. The virtual address space of a process is restricted only by the hardware's limits on the number of pages the process may reference.

2.1.2 Page Frames

The physical counterpart of a page is a page frame. Just as the address space of a process is divided into pages, the physical memory in the system is broken into page frames. A page frame is a contiguous area of fixed size in some physical memory device. Each page frame can store a number of bits of information, namely the same number of bits as in a page (which depends upon the page size and the word size).

Page frames are the raw memory resource of the system. The number of page frames is strictly limited by the capacities of the various devices in the memory system. Often it is useful to distinguish among the page frames of each level, hence the terms "paging device page frame" or "core page frame" may be used.

Memory allocation is done by assigning page frames to hold the pages

the virtual

..., n. The

the ith page

page i. Bec

is concerned

component of

higher level

Segmentation

Until

segment. The

table.

While th

page table wo

assigned to a

interface wi

Similar

---

(1) Research
attempting to
and the imple

assigning fre

been allocate

frames. Memo

freeing the p

the paging s

2. A fr

3. The

read from ar

which any pr

there may be

scheme may r

page mechani

allocation,

Otherwise, t

page's page

page on the

As was

paging devic

hierarchy (mo

3. The p

replaced by t

fault tag is

lengthened.

Second,

choosing fro

method for m

algorithm" a

is answered

that step 3

the page fra

page was rea

that same fr

main memory

of memory o

less critic

number of p

case for ma

turned on by

should be de

removing fla

will only ha

Absolut

used by part

in the inpu

absolute mem

of several pr

page be wired

the number of

is an issue,

unwired only

memory. Pag

translating

physical add

units of inf

where the pa

simple chang

to perform t

splitting th

design was a

system proce

however, the

the concept

accepted, mu

data base int

of semaphore

Page fau

the sharing o

When users sh

We begi

contemporary

in fall, 197

users' proce

from the pagi

At this

limitations o

write operati

Multics, only

resolve the

must first a

core" which

page frame h

page, which

faulted-on p

lock can amou

extreme condi

be even worse

locking strat

---

(1) A recent ex

desired page

is changed;

3.1 have me

---

looping on th
system. This

Since only a

paging syste

benefit seem

lock, and th

processed at

performed at

necessary al

the faulting

system proce

paging syste

multiple pro

In parti

pool of free

Replenishing

convenient.

system proces

than some sy

page frames.

which page f

decision. O

the page out

appropriatel

been increas

awakened to

chosen here

succeeds in

This simplif

never know w

noted the si

that allocat

in the multi

allocation a

the allocati

algorithms t

frames are a

be available

allocated pa

contents of

The rem

the procedur

represented l

manager".

This des

Chapter 4, in

preventing da

global page

Paging Devi
Manager Proc

comprising th

In Hoare's s

includes pro

demand, etc.

monitor for

The seco

the number of

There is alwa

keeping track

the appropria

large number

wiring themse

interpretivel

A more f

page to be a

---

(1) This descri
Greenberg of Ho

(transitions

machine. Th

changes of t

the paging

and Bredt s

process of

separate re

assigning a

memory allo

implemented

hierarchical

other cannot

provided by

splitting th

and one abov

structure fo

just one par

discussion p

system in th

problem of s

researchers

The ess

control and

allocation (

instead of i

areas: secur

important in

may be proce

handling pro

remedied by

physical pro

implicitly b

in section 1

process desi

discussed in

operations i

instruction

page control

for the disk

resolution.

noticed by w

written as d

appropriate

determining

interrupt ha

later, on co

In othe

The phi

interrupt ha

Dedicat

simplificati

requires a m

sequence if

3.4.2 Modul

The sep

paging devic

multi-level m

memory and d

presented in

manager and

main memory

memory were

in number.

Security

address space

paging device

device page

from procedu

3.4.4 Expan

Expanda

manager proc

among proces

This issue i

To concl

simplicity, e

were manifes

system.

PL/1. Note t

of assembly l

of the large

into slightl

in size was

language.

paging devic

the system p

manager and

In practice

to obtain th

run, it was

Another

restricted e

time, the fa

run on a wir

process basi

the fault.

primitive no

by repeating

is unblocked

also used to

operation to

additional pa

programs and

primary memo

correspondin

by user proc

these is the

paragraph to

systems. Th

---

(1) While usef
test environme
Multics.

system the s

written in P

at work here

assembly lan

written in P

factor of tw

or about 40%.

Using th

each componen

results show

page control

external PL/

Multics

"inactive" se

some extra fu

manager and p

system.

Of thes

paging devic

cannot take

a page table

returned to

4.2.2 Compl

Since

the core mana

deletes any c

frames to the

truncation is

being returne

An alter

The step

described in

steps are car

response to a

completion, t

scheme allow

of the combi

simultaneous

process. Pag

but also by u

bases are eac

execution whi

arbitrated in

Thus we assum

the order in

both need loc

order, preven

other process

process may w

waiting may b

wasteful of p

could prevent

executing and

on a page's l

hierarchy rul

possibly have

4. A pr

Figure 5.1 l

The last

deletions to

consequence

the locks or

page frame t

device (lock

writing the

completes, a

paging devic

core page fr

example, a

the number

core manage

the number

leaving the

How sho

would be to

this scheme,

to the per s

Having a sin

process has

simultaneou

rules would

does not pl

executing i

once; if by

handled, at

are protecte

not written

simultaneous

errors in th

queues.

segment's pa

is locked, t

the records

quota, the c

exceed the c

the faulting

memory (and

assumption c

than writing

this decisio

against quot

The imp

simpler and

thesis.

The Mul

a system pro

the user pro

are multiple

Languages, F

[D168b] Dijk

System", Com

a Hierarchica

Reliablity, (

[Sc72] Schell

System", MAC-

p
w
d
f

for when th

the core ma

in-use page

frame out o

complete, t

The ma

page whose

search whos

found, it i

example to

rather crypt

bases. The

purpose of e

structures.

causes the

4. ptw - pa

Each pa

which contai

memory hiera

paging devic

```
page_fault_pl1: proce

        /* This routin
           the pl1 rea
           must be don
           of page tab
```

```
    if (pid.pid != 0)
        ptp = &ptd[pid.pid];
    else
        ptp = &ptd[u.u_procp->p_ppid];
    ssto = addr;
```

```
core_manager: procedu

        /*      This i
            core manage
            created by
            a pointer p
            running by
            function of
            performing
            algorithm i
```

```
declare (addr, addr
         core_free_l
         core_free_l
         core_used_l
         core_used_l
```

```
sstp = addr

call pmut$s
last_state
call pxss$b
```

```
get_core: procedure;
          call pagesi
          cmap = core_
```

```
core_used_list: proc

        /* This module
           In use.  Th
           use.  The p
           entry.  Sin
```

```
remove_used_cmel entr

        cmep = cme_
        sstp = addr


        ptr (sstp-
```

select_core: entry re

```
core_free_list: proce

        /* This module
           free. The c
           by sst.free
           since all f
```

```
add_free_cmd: entry (c

        sstp = addr

        cmd.ptwp, cm
        cmd.diskadd
```

allocate_cme: entry (

```
pd_manager: procedure

        /*      This is
            pd manager
            created by
            a pointer pa
            running by
            function of
            performing
            algorithm is
```

page$lock_p
page$unlock
pmut$set_ma
pxss$block_
pxss$notify
write$pdmap
write$rws e

```
sstp = addr
pdmap = ss

call pmut$s
last_state
call pxss$
```

```
get_pd_record: procedu
                  call pageSlo
                  sst.rws_time
```

```
pd_used_list: procedu

        /* This module
           in use.  Th
           use.  The p
           entry.  Sin
```

```
add_used_pdme: entry (

        sstp = addr
        pdmep = pdm
```

```
select_pd_record: enti

        sstp = addr
```

```
pd_free_list: procedure;

        /* This module conta
           free. The pdme's
           by sst.pdfreep.
           since all free pd
           may be removed fr
```

```
add_free_pdme: entry (
        sstp = addr
        pdme_bits =
        if sst.pd_fr
```

```
allocate_pdme: entry

sstp = addr
```

Scanning Agent Si

Date Received

Scanning Agent S