Lecture 11: Segmented & Paged Memory

Adam Hawley

February 26, 2019

Contents

1 Segmentation

1.1 Introduction

Memory-management scheme that supports user view of memory. A program is a collection of segments where a segment is a logical unit search as: main program, procedure, function, method, object, local variables, global variables, stack, array etc. Segments have variable sizes. This approach means that when you are in the main program, only the main program needs to be stored in memory.

1.2 Addressing with Segmentation

The logical address consists of a tuple: `<segment-number, offset>`. The addresses are stored inside a **segment table** which maps a two-dimensional logical address to a one-dimensional physical address.

`base` contains the starting physical addrss where the segments reside in memory

`limit` specifies the length of the segment

Segment table is kept in memory:

`segment-table base register` **(STBR)** points to the segment table's location in memory

`segment-table length register` **(STLR)** indicates number of segments used by a program

- segment number `s` is legal if `s < STLR`

2 Paging

2.1 Introduction

Physical address space of a process can be noncontiguous: process is alllocated physical memory whenever the latter is available:

- avoids external fragmentation
- avoids problem of varying sized memory chunks

Divide physical memory into fixed-sized blocks called **frames** (size is a power of 2, between 512 bytes and 16 Mbytes). Then divide the logical memory into blocks of the same size called **pages** (internal fragmentation is still a minor issue).

2.2 Costs & Implementation

The OS must:

- Keep track of all free frames in memory
- (to run a program of size N pages), need to find N free frames and load program

- set up a **page table** to translate logical to physical addresses — kept in memory

  Page-table base register (PTBR) Points to the page table

  Page-table length register (PTLR) indicates size of the page table

2.3 Address Resolution

Assume the logical address space is 2^m and that the page size is 2^n then the address generated by the CPU is divided into:

page number (p) used as an index into a **page table** which contains base address of each page in physical memory

- size of **p** is `m - n` bits

page offset (d) combined with base address to define the physical memory address that is sent to the memory unit

- size of **d** is `n` bits

2.4 Internal Fragmentation in Paging

Internal fragmentation happens when the process requires memory which is not a multiple of the page size, when this happens, the last page will cause internal fragmentation as it will not fill the frame. The worst case fragmentation would be equal to `1 frame - 1 byte` but average fragmentation is around half a frame size.

2.4.1 Page Size Trade-Off

- Reducing the page size → minimises internal fragmentation
- Increaseing the page size → less pages needed, reduces page table size (faster, simpler implementation of MM)

2.5 Performance Issues

If the page table is kept in main memory every data/instruction access requires two memory accesses (one for the page table and one for the data/instruction).

The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers (TLBs)**. The TLB is typically small (64 to 1,024 entries). Frequently accessed pages will have their frames stored in a TLB. On a TLB miss, the value of the (missed page-table and frame-number), is loaded into the TLB for faster access next time that address is used (if there is no free TLB entry, replacement policies must be considered). Some entries can be **wired down** for permanent fast access.

Address translation `(p,d)`:

1. If `p` is in associative register, get frame # out
2. Otherwise get frame # from page table in memory

2.6 Shared Pages

2.6.1 Shared Code

Processes that are read-only can be shared because there is no danger of modification. This means that only one copy is needed. It is similar to the idea of multiple threads sharing the same process space and is also useful for interprocess communication if sharing of read-write pages is allowed.

2.6.2 Private Code & Data

Each process keeps a separate copy of the code and data. The pages for the private code and data can appear anywhere in the logical space.

2.7 Page Table Structure

Memory structures for paging can get huge using straight-forward methods:

- Consider a 32-bit logical address space
- Page size of 1KB (2^{10})
- Page table would have 4 million entries ($2^{32}/2^{10}$)
- If each entry is 4 bytes, page table is of size 16 MB
  - Can be costly
  - Do not want to allocate that contiguously in main memory

There are several approaches to this problem:

- Exploit heirarchy
- 64-bit address spaces require even more sophisticated solutions

2.8 Hierarchical Page Tables

Break up the logical address space into multiple page tables A simple technique is a two-level page table. We then page the page-table.

A logical address (on a 32-bit machine with 4K page size) is divided into a page number consisting of 20 bits and a page offset consisting of 12 bits. Since the page table is paged, the page number is further divided into a 10-bit index p_1 into the outer page table and a 10-bit displacement p_2 within the page of the inner page table.