Lectures 5 and 6: Memory Management: Main Memory and Virtual Memory

Barnaby Martin and Stefan Dantchev

barnaby.d.martin@dur.ac.uk
s.s.dantchev@dur.ac.uk

Memory Sharing

As a result of CPU scheduling, we can improve:

- the utilisation of the CPU,
- the speed of the computer's response to its users.

To realise this increase in performance, however, we must be able to keep several processes in memory at once.

Types of Memory

Memory comes in many types:

- Cache memory / CPU cache (e.g. L1, L2, etc.)
- Main memory (volatile e.g. Random Access Memory (RAM))
- Storage memory (non-volatile e.g. flash memory in USB memory sticks, sold-state drives (SSD)
- Virtual memory

The cache and main memory are referred to as **primary storage**. The cache primarily supports, and is managed by, the CPU. It is invisible to the operating system. It uses the same memory management approaches as main memory, therefore are going to focus on **main memory**.

Logical / Physical Addressing

Memory is made up of many memory cells. A memory cell is an electronic circuit that stores one bit (0, 1) of information. The electro-mechanics is purely whether the circuit has a voltage higher than a pre-determined arbitrary threshold (1), or not (0).

Logical address: an address created by the CPU.

Physical address: an address on the physical memory.

Memory Management Unit (MMU): The MMU automatically translates virtual addresses to physical addresses.

IMPORTANT: A user (application) program deals with logical addresses; it never knows the real physical addresses.

Memory partitioning

Memory is a finite resource and as with the CPU, needs to managed as to be used as efficiently as possible.

The main memory is usually split into two partitions:

- Kernel processes are usually held in one partition.
- User processes are held in another partition.

Each partition is contained in a single contiguous section of memory.

Contiguous Allocation

If reading memory, contiguous memory can be read in sequence and is read faster. To achieve this an effective memory allocation strategy is required based on:

- Hole: block of available memory.
- Holes of various size are scattered throughout memory.
- When a process arrives, it is allocated memory from a hole large enough to accommodate it.

The operating system also needs to maintain information about each partition and its allocated and free holes.

Memory Hole Allocation

Some strategies to satisfy a request of size n from a list of free holes:

- **First-fit:** Allocate the first hole that is big enough.
- **Best-fit:** Allocate the smallest hole that is big enough.
  - Must search entire list, unless ordered by size.
  - Produces the smallest leftover hole.
- **Worst-fit:** Allocate the largest hole.
  - Must also search entire list.
  - Produces the largest leftover hole.

First-fit and best-fit are usually better than worst-fit in terms of speed (first-fit) and storage (best-fit) utilization. **Also used for file allocation in secondary storage management.*

Fragmentation

External fragmentation is memory space that is able satisfy a request but it is not contiguous. We can reduce external fragmentation by **compaction:**

- Shuffle memory contents to place all free memory in one block.

Internal fragmentation is memory that is allocated successfully but may be slightly larger than the requested amount of memory:

- This size difference is due to the memory being stored blocks divisible by 2, 4, 8, 16 etc - therefore a request for 27 bytes of memory will be allocated to a 32-byte block leaving 5 bytes unused.

Paging

Physical memory is divided into fixed-sized blocks called **frames**. Typically, the size is a power of 2, between 512 and 8192 bytes.

Logical memory is divided into blocks of the same size called **pages**.

To run a program of size *n* pages, we need to find *n* free frames and load the program. This process is called **paging**.

A **page table** is set up to help manage the mapping of logical to physical addresses, using an address translation.

Address Translation Scheme

The logical memory 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.
- Page offset (d): combined with base address to define the physical memory address that is sent to the memory unit.

The term **logical** address is often interchanged with **virtual** address. This is confusing as **virtual memory** also uses the same logical addressing - that is the mapping of secondary storage (disk space) as main memory.

Protection

Memory protection implemented by associating protection bit with each frame.

A **valid-invalid bit** is attached to each entry in the page table:

- **valid** indicates that the associated page is in the process's logical address space, and is thus a legal page.
- **invalid** indicates that the page is not in the process's logical address space.

Virtual Memory

Definition:

Virtual memory is the capability of the operating system that enables programs to address more memory locations than are actually provided in main memory.

Virtual memory systems help remove much of the burden of memory management from the programmers, freeing them to concentrate on application development.

Virtual Memory

- The **logical** address space is much larger than the **physical** address space.
- Pages are swapped (paged) in and out main memory.
- The physical address space is shared by several processes.
- Only part of the process needs to be in the physical address space for execution.
- A free frame list of the physical address space is maintained by the operating system.

Virtual Address Space

Each process views the address space as a contiguous block of memory holding the objects it needs to execute.

- Code (read-only)
- Data (read and write)
- Heap: the address space in memory that is used to hold data produced during execution of a process. This space will expand and contract during execution.
- Stack: the address space that is used to hold instructions and data known at the time of the procedure call. The contents of the stack grow as the process issues nested procedure calls and shrinks as the called procedures return.

Demand Paging

Bring a page into memory only when it is required by the process during its execution

- Reduces
  - Number of pages to be transferred
  - Number of frames allocated to a process
- Increases
  - Overall response time for a collection of processes
  - Number of processes executing

A Process's Page Table

- When reference is made to a page's address
  - Invalid reference $\rightarrow$ abort
  - Not-in-memory $\rightarrow$ bring to memory
- A valid/invalid bit is associated with each process's page table entry to indicate if the page is already in memory
  - 1 $\rightarrow$ in-memory
  - 0 $\rightarrow$ not-in-memory
- Address translation
  - when valid/invalid bit in page table entry is 0 $\rightarrow$ page fault trap
- Initially the valid/invalid bit is set to 0 on all entries

Handling Page Fault Traps

- A process requests access to an address within a page:
- Check the validity of the process's access to that address
- If an invalid request
  - terminate process
  - clear all previously allocated main memory
  - throw error message "invalid memory access"
- Else if a valid request
  - If page allocated a frame in main memory
    - break
  - Else if page fault trap
    - identify a free frame from the free frame list
    - read the page into the identified frame
    - update the process's page table
    - restart the instruction interrupted by the page fault trap (break)

Performance of Demand Paging

- Page Fault Rate $0 \leq p \leq 1$
  - if p = 0 no page faults
  - if p = 1, every reference is a fault
- Effective Access Time = (probability of a page being in memory X time to access the memory) + (probability of a page not being in memory X page fault overhead)
- Page fault overhead includes:
  - Context switching when relinquishing the CPU
  - Time waiting in the paging device queue
  - Time waiting back in the ready queue
  - Context switching when regaining the ready queue

No Free Frame?

This occurs when all main memory frames available to a process are currently allocated.

Page replacement.
Find some page in memory, but not really in use, and swap it out.

- Page replacement algorithms to decide the page.
- We want to minimise the number of page faults.

Basic Page Replacement

- Find the location of the desired page on the disk.
- Find a free frame:
  - If there is a free frame, use it.
  - Else if there is no free frame, use a page-replacement algorithm to select a victim page.
  - Write the victim page to the disk; update the process's page table and the free frame list accordingly.
- Read the desired page into the (newly) freed frame.
  - Update the process's page table and the free frame list.
- Restart the user process.

Replacement Algorithms

How to **evaluate the algorithms**:

- Run each algorithm on a particular string of memory references (reference string).
- Compute the number of page faults per algorithm on that string.
- Compare the number of page faults: We are aiming for the lowest possible number of page faults given the reference string.

First in First out (FIFO) Algorithm

Associated with each page the time it was brought into memory.

The victim page is the oldest page.

Simple to implement as a FIFO queue. Actually tracking ageing (time) is not necessary: we only need the order.

Belady's Anomaly

Belady's anomaly: "For some page fault algorithms, the page fault rate may increase as the number of allocated frames increases."

Counterintuitive: Usually increasing the size of memory will improve the memory response and throughput however, some combinations of page demands (reference strings) will have fewer page faults with a smaller number of available frames.

FIFO suffers from Belady's anomaly!

Optimal Algorithm

Replace the page that will not be needed for longest period of time.

This algorithm will always have the fewest page faults when compared with any other algorithm.

It is used to demonstrate how close to the ideal any other algorithm has progressed.

However, this requires to know the future: the OPT algorithm is therefore **impossible to implement**!

Least Recently Used (LRU)

- Associate with each page the time of that page's last use.
- Victim page: the page that has not been used for the longest period of time.
- LRU belongs to a class of page replacement algorithms known as stack algorithms.
- Stack or counter algorithms can **never exhibit Belady's Anomaly**.
  - A set of pages in memory for n frames is always a subset of the set of pages that would be in memory with $n + 1$ frames.

LRU - Implementations

Counter implementation

- Every page entry has a counter.
- Every time a page is referenced, record the time.
- When a page needs to be changed, look at the counters to determine which are to change.

Stack implementation Keep a stack of page numbers in a double link form:

- Page referenced:
  - move it to the top (requires pointers to be altered).
- No search for the replacement: the tail pointer points to the bottom of the stack, which is the LRU page.

LRU Approximation Algorithms (1)

Additional Reference Bit

- With each page associate a bit (initially = 0).
- When page is referenced bit set to 1.
- Replace one of the pages with the bit set to 0 (if one exists).
- Replace a random page when none of the pages are set to 0.
- Problem: No way of knowing if a page is a least recently used page, a most recently used page, or somewhere in between.

Page	0	1	2	3	4	5	6	7
AR bit	1	1	0	1	1	0	0	1

LRU Approximation Algorithms (2)

Second chance algorithm

- Implemented as a variation on a FIFO queue: it can be viewed as a circular queue.
- When a page enters main memory, it joins the tail of the queue and its reference bit is set to 1.
- When the queue is full and a victim page needs selecting:
  - Start at the head of the queue.
  - If the page's reference bit is set to 0: Select this page as the victim page.
  - Else if the page's reference bit is set to 1: Set the reference bit to 0 and move on to the next page in the queue.

Allocation of Frames

- Global replacement
  - Select a replacement frame from the set of all frames; one process can take a frame from another
- Local replacement
  - Select from only the process's own set of allocated frames
- Fixed allocation
  - Equal allocation
  - Proportional allocation: Allocate according to size of process
- Priority allocation
  - If process P_i generates a page fault, select for replacement a frame from a process with lower priority number

Thrashing

Thrashing occurs when a process is spending more time paging then doing actual work.

If a process does not have enough pages, the page-fault rate is very high.

Prepaging

Prepaging: Page into memory at one time all the pages that will be needed.

In a system using the working set model,

- We keep with each process a list of pages within its working set model.
- If a process is suspended we remember working set for that process.
- When the process resumes we automatically bring back into memory its entire working set before restarting the process.

Prepaging is a means to prevent thrashing. However, we must ensure that the cost of prepaging is less than the cost of servicing the page faults.

Page-Fault Frequency Scheme

A **Page-Fault Frequency Scheme** is an alternative approach to prevent thrashing.

Scheme: firstly establish an “acceptable” page-fault rate. then:

- If the actual rate is too low, i.e. the process has got too many frames
  $\Rightarrow$ The process loses frames.
- If the actual rate too high, i.e. the process has not got enough frames
  $\Rightarrow$ The process gains frames.