Operating Systems: Shortnotes

Hardik Rajpal

August 28, 2023

Contents

Chapter 1

Introduction

Chapter 2

CPU Virtualization

Terms

- Process Control Block: A per-process data structure that tracks process meta-data.

Chapter 3

Memory Virtualization

Terms

1. Address Space: The running program's view of the memory available to it. Every program only sees the memory available to it, which is split between the code, (static) data, heap and stack.
2. Sparse Address Space: An address space where most of the memory between heap and stack is unused.
3. Memory Management Unit: The hardware responsible for V2P translation.
4. Protection Bits: Bits in VA that represent the process' access permission for the PA to which it points, including read/write/execute permissions.
5. Fragmentation: Wastage of physical memory space.
   - Internal: Wastage within a contiguous block of physical memory allocated to a process. An example is the unused space between heap and stack in a simple base-bounds V2P map.
   - External: Wastage outside of contiguous blocks that have been allocated. This is because of the gaps of available physical memory being too small to fit any new contiguous segments.
6. Page: Unit of address space.
7. Page Frame: Unit of physical address space to which a page can be mapped. Same unit, but **pages** exist in the virtual address space whereas **page frames** exist in the physical address space.

3.1 Address Space

The program is actually loaded into random physical addresses, but due to the address space abstraction, the program sees the memory available to it as a contiguous chunk, starting at 0. Every address the program sees is virtual and the OS (with some translation mechanism of the VA to PA) uses the PA whenever the program references the VA.

3.2 Address Translation

All address translation is hardware-based for efficiency. Each memory access (load/store) is intercepted by the hardware and the VA is translated to a PA. The OS helps setup the hardware for the right translation (per process), and manages memory.

3.2.1 Dynamic Relocation a.k.a. (Base, Bound)

- Two registers for this in the CPU: base and bound.
- Translation:

```
V2P(VA v){
    if(v > bound || v < 0){
        throw "Out of Bounds";
    }
    return (base + v);
}
```

- Values of base and bounds for each process are stored in its PCB.
- Base and bound registers are privileged: if access is attempted from user mode, the OS terminates the access-requesting process.

3.2.2 Segmentation

- Generalized base and bounds for each logical segment of each process: code, heap and stack.
- Maintain base and size for each segment:

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

- Use VA[0:2] to represent the segment type.
- Better handles **sparse address spaces**, where the program often has very little heap and stack data and thus a lot of the in-between space in a contiguous allocation is wasted.
- Translation is more involved:

```
V2P(VA v){
    segment = v[0:2];//or bit operations.
    offset = v[2:];//or bit operations.
    if(offset < 0 || offset > segdata[segment]["bound"]){
        throw "Out of Bounds"
    }
    else{
        return segdata[segment]["base"] + offset;
    }
}
```

- OS responsibilities:
  - On each context switch the **segmentation table** for the process is replaced by the incoming process' table.
  - On a receiving new process (and its accompanying address space), the OS has to find space in the physical memory for its segments. If the segments are of varying sizes (bounds) this is more involved.
  - Variable size segments lead to external fragmentation.
  - A solution to external fragmentation is periodic compaction:
    1. Stop running processes.
    2. Copy their data to a contiguous chunk of memory.
    3. Update their segment register values.

This creates available contiguous chunks of physical memory but compaction is expensive (copying segments is memory-intensive) and would take up a fair amount of CPU time.

  - Alternative approaches involve using a free-list management algorithm like:
    * best-fit
    * worst-fit
    * first-fit
    * buddy algorithm
  - External fragmentation will always exist in this scheme, however. The algorithms above simply aim to minimize it. The real solution is to disallow variable sized segments.

- Some systems merge code and heap segments to use only one bit to represent segment.
- There are **implicit** approaches to identify the segment too, where the program infers the segment by identifying how the VA was conceived:
  - from a PC $\implies$ use code segment.
  - from ebp $\implies$ use stack segment.
  - else, use heap segment.
- We also need an additional bit to identify the direction of growth of the segment, if this varies across the segments. The translation function is modified to take this into account.

Sharing Segments

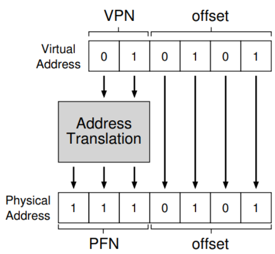
- With some VA bits used to represent permissions that a process has for the segment **protection bits**, we can factor include segments with varying permissions.
- Translation function is appropriately modified to raise an exception whenever a process attempts to violate permissions for a PA.
- Read-only code can be shared across multiple processes, without the worry of harming isolation.

Fine-grained Vs. Coarse-grained

- Segments of code, heap and stack are relatively large (coarse-grained segmentation).
- Fine-grained segmentation further splits up each of these logical segments and identifies each segment using a **segment table**. This allows for better memory management by the OS (by moving unused segments to disk).

3.2.3 Paging

- Split the address space (abstraction visible to the process) into fixed-size units called pages. The corresponding contiguous units in the physical address space are called page frames.
- To record each page's mapping for each process, the OS maintains a **page table** per process.
- Translation: (neglecting bits other than VPN and offset)



Inside The Page Table

- The OS stores the page table in memory, in units of **page table entries** (PTEs).
- Each PTE has a physical address to which it maps and a collection of metadata bits.
- The PTEs are stored in a (nested) array-like structure, such that a VPNs can be used as indices into the structure, pointing at the PTE that contains their corresponding PFN.
- The metadata bits:
  1. Valid bit:indicates whether PTE is valid.
     - When a program starts running, all the (virtual) space between the heap and stack is unused and hence unmapped. Hence entries at pagetable [VA] for those addresses are invalid.
     - If a program tries to access invalid memory, a trap is generated and the program is likely terminated.
     - The valid bit is crucial for supporting **sparse address spaces**.
     - Invalid PTEs don't have any physical memory mapped to them, this lets us use only the physical memory we need.
  2. Protection bits: indicate whether the page can be read from, written to or executed from. Invalid accesses generate a trap to the OS.
  3. Present bit: indicates whether page is present in memory (or has been swapped out to disk).
  4. Dirty bit: indicates whether the page has been modified since being loaded into memory.
  5. Reference (a.k.a. accessed) bit: used to track if the page has been accessed since the bit was last reset. This information helps in page replacement, to keep popular pages around.
  6. User/Supervisor bit: indicates if processes can access the page in user mode.
  7. Hardware caching bits: help determine how hardware caching works for the page.

Advantages of paging

- Flexibility is an important advantage of paging: it works regardless of how the process uses the address space, how the heap and stack grow or how they are used.
- Simplicity: when new pages are requested, the OS simply traverses a free list and returns the first pages that it finds.

3.3 Free Space Management

Chapter 4

Concurrency

Chapter 5

Misc.

Chapter 6

References

1. OSTEP