OS Assignment 6 Report

name: 刘乐奇

sid: 12011327

1. How do the CPU hardware and the operating system cooperate in the procedure of address translation?

The CPU hardware uses a virtual address space that is independent of the physical memory available in the system. When an application running on the CPU needs to access memory, it generates a virtual memory address. The CPU then sends this address to the memory management unit (MMU), which translates the virtual address to a physical address that corresponds to a location in physical memory.

The operating system manages the mapping between virtual addresses and physical addresses using a page table. The page table is a data structure that maps virtual addresses to physical addresses. The operating system maintains this mapping in main memory and updates it as needed.

When the MMU receives a virtual address from the CPU, it consults the page table to translate the virtual address to a physical address. If the page table indicates that the memory location is not currently in physical memory, the operating system will allocate a new page of physical memory and update the page table to reflect the new mapping.

2. Comparing segmentation and paging.

- Size of chunks:
  Segmentation divides the virtual address space into variable-sized segments. Each segment represents a virtual unit of data, such as a code segment, a heap segment, or a stack segment. Paging divides the virtual address space into fixed-size pages, typically 4KB or 8KB in size.
- Management of free space:
  In segmentation, free space is managed by tracking the boundaries of segments and allocating new segments as needed, which can lead to external fragmentation. In contrast, paging manages free space by dividing memory into fixed-size pages, which can be allocated and deallocated as needed. This reduces external fragmentation, as the system can allocate pages of different sizes to fit the available memory more efficiently. However, both of them cannot get rid of internal fragmentation.
- Context switch overhead:
  Segmentation can be more expensive in terms of context switch overhead because each process has its own set of segment descriptors, and switching between processes requires updating the segment descriptors. Paging requires less overhead because the page tables can be shared between processes, and switching between processes only requires updating the page tables.
- Fragmentation:
  Segmentation suffers from external fragmentation, where there are small gaps between segments that are too small to be used. Paging much suffers from internal fragmentation, where a page is allocated to a process, but only a portion of the page is actually used, leaving the remaining space unused.
- Status bits and protection bits:
  Both segmentation and paging use status bits and protection bits to manage memory access. In segmentation, each segment has its own protection bits that determine whether the segment can be read, written, or executed. Paging uses protection bits at the page level, where each page can be marked as read-only, no-execute, or other access restrictions.
- Flexibility:
  Segmentation offers more flexibility in terms of memory allocation, as each segment can grow or shrink independently of other segments. This can be useful for applications that require a large amount of memory for one particular task. Paging provides a simpler memory management scheme that is easier to implement and more efficient in terms of memory usage.

3. How many levels of page tables would be required to map the entire virtual address space?

Since the page size is $8K$ (2^{13}) bytes, the offset size is 13-bit. Since the page table entry size is 4 bytes, a page table has at most $\frac{8K}{4} = 2^{11}$ page table entry, i.e., the index size is 11-bit.

Therefore, the 46-bit virtual address space can be divided into 11-bit L1, L2, L3 index and a 13-bit offset. Thus, it needs 3 level of page tables.

4. Consider a system with following pecifications and answer questions.

(a) What is the page size? What is the maximum page table size?

Since the offset is 12-bit, the page size is $2^{12} = 4KB$.

20-bit PTE index can determine at most 2^{20} PTE. With a PTE size of 4 bytes, the maximum page table size is $4 \times 2^{20} = 4MB$,

(b)

- 1-st level page number: 780
  offset: 770
- 2-nd level page number: 614
  offset: 1707