Virtual Memory

Process Memory

\[
\begin{align*}
\text{stack} & \downarrow \\
\text{heap} & \quad \text{dynamic allocation} \\
\text{code} & \quad \text{(fixed)} \\
\text{stack} & \downarrow \\
\text{heap} & \downarrow \\
\text{BSS} & \rightarrow \text{variables initialized to 0.} \\
\text{init data} & \\
\text{lib data} & \\
\text{code} & \\
\end{align*}
\]

Meaning of Robustness in OS: Protect different parts of the system from each other, e.g., each process can only access its own memory space.

Relative addressing is needed sometimes:

\[\text{MAX} \quad \text{MIN} \quad 0\]
Base & Bound registers:

- If the address is absolute, it should be: Base < addr < Bound
- If the address is relative, add its value to the value of Base to find the memory cell.

If the memory is fragmented into small pieces, you can’t allocate a big piece for your process, e.g:

Although there are 9 cells available, you can’t allocate ‘1’ continuous cells.

Virtual address space for processes using base and bound registers:

+ simple virtual memory
- fragmentation (external)
- always physical-backed

Paged Virtual Memory

Idea: mapping between pages in virtual address space and physical address space.
The mapping is a hardware feature and cannot be done by software.

Map function: Page table (provided by HW) is the mechanism for mapping functions.

mechanisms: 1. Page table

2. Faulting: when there's no mapping or the mapping is read-only.

Difference between fault and interrupt: Fault must be handled immediately. (Can't continue executing the instruction.)

- If the address space is 32 bit and each page is 1k, then we want to map $2^{20}$ pages from the virtual memory to the physical memory.

- Single-level table

  20 - wastes memory: The table takes a large amount of memory.

- Fragmentation: We're using paging to solve external fragmentation. But by allocating 4MB for the table, we are actually adding another kind of fragmentation.
Two-level table

Page Dir.  Page Table  Page Table Entry (PTE)

offset to index in the physical page

1024-way trie

Need to map 20 bits to do the translation

10 bits at Pg. Dir.
10 bits at Pg. Table

Pseudocode for 2-level table

\[ V = \{ \text{Page-hi: 10,}
\text{Page-lo: 10,}
\text{offset: 12} \} \]

\[ \text{PT} = \text{PD}[\text{V, Page-hi}] \]
\[ \text{PTE} = \text{PT}[\text{V, Page-lo}] \]
\[ P = \text{PTE}.pg + \text{V.offset} \]

Example

Virtual Page#: 0900 001C

Virtual Page#:

\[
\begin{array}{cccc}
& \text{top 10} & \text{next 10} & \\
24 & 0 & \vdots & \\
0000 & 1001 & 0000 & 0000 & 0000 & \ldots \\
\downarrow & \downarrow & \downarrow & \\
\text{page # in PD} & \text{page # in PT} & \\
\end{array}
\]

4-byte entry at [024] in Page Directory: 0000 8090

4-byte entry at [0] in Page Table: 0000 7000

actual memory address: 0000 601C

data

Page 4
2-level table is flexible and powerful, but not efficient, since we read the memory 3 times to read the data.

- Caching

  *TLB*: Translation Lookaside Buffer

  TLB caches the address translations that are recently read from the page table. The next time the address is seen, there's no need to read from the page table again. Largest TLBs cache 1000 to 2000 entries which covers about 18 MB of TLB-mapped memory. So, you still need to read from PT many times.

  ![PTE Diagram](image)

  - **P**: If P=false, no mapping is present => Page Fault.
  - **U/S**: If we wanna use a mapping that is only for s, we'll get a fault.
  - **NX**: For execution permission
  - **R/W**: If we write on a read-only segment (e.g., code segment), we'll get segmentation fault.
  - **A**: In case of memory shortage, if a page is accessed frequently, we probably don't like to throw it away to free some memory. Since it will be needed soon and will be loaded again.
  - **D**: If a page is not dirty (has not been changed since it was loaded into the memory), then we don't need to write it back on disk and can simply throw it away when needed.