Operating Systems Lab

Lab - 1: Bootloader & Physical Memory Management

Adib Sakhawat & Yasir Raiyan

ID: 210042106 & 210042152
Software Engineering
Dept. of Computer Science and Engineering
Islamic University of Technology

September 20, 2024

What is a Bootloader?

- A small program that runs when a computer starts.
- Loads the operating system into memory.
- Begins the execution of the OS.
- Essential for the OS to function.

Introduction to Physical Memory Management

- **Physical Memory:** The hardware memory (RAM) used by programs.
- **Role of OS:** Manages memory allocation and ensures efficient use.
- **Allocation:** Decides how memory is divided and organized.

Memory Allocation

- **Fixed Partitioning:** Predefined memory blocks, simple but inefficient.
- **Dynamic Partitioning:** Flexible memory allocation based on process size.
- **Contiguous Allocation:** Memory blocks assigned sequentially.
- **Non-contiguous Allocation:** Allows memory blocks scattered across RAM.

Paging

- **Pages:** Memory divided into fixed-size pages.
- **Page Table:** Maps logical pages to physical frames.
- **Page Frames:** Fixed-size blocks in physical memory.

Segmentation

- **Segments:** Memory divided based on logical divisions (code, data, stack).
- **Segment Table:** Maps segments to physical addresses.
- **Logical vs. Physical Address:** Logical address is used by the program, physical address by hardware.

Memory Protection and Fragmentation

- **Memory Protection:** Prevents one process from accessing another's memory.
- **Fragmentation:**
  - **Internal Fragmentation:** Unused memory within allocated space.
  - **External Fragmentation:** Free memory scattered across.
- **Solutions:** Compaction, Paging, and Segmentation.

MATIntro Layer: Memory Management Overview

- The code defines a Memory Allocation Table (MAT) to manage physical memory pages.
- Pages are represented as 4KB units, and permissions are assigned to each page.

Key Components:

- `NUM_PAGES`: Number of physical pages available in the system.
- `struct ATStruct`: Represents each page with permission and allocation status.
- `AT[1 << 20]`: Array storing information for each physical page (up to 4GB memory)

Core Functions:

- `get_nps()`, `set_nps()`: Get/set number of available pages.
- `at_is_norm()`: Checks if a page has normal permission.
- `at_set_perm()`: Sets page permission and marks it as unallocated.
- `at_is_allocated()`: Checks if a page is allocated.
- `at_set_allocated()`: Sets allocation status of a page.

Introduction to `pmem_init()`

- Initializes physical memory and allocation table (AT).
- Configures permissions for memory pages based on the memory map.
- Pages are 4KB in size.
- `VM_USERLO`/`VM_USERHI`: Define user-space memory boundaries.

Calculating Physical Memory Pages

- `nps`: Total number of physical pages.
- Pages calculated as: `nps = (highestAddr + 1) / PAGESIZE`.
- Fetch memory map rows with `get_size()`.
- Determine highest address using `get_mms()` and `get_mml()`.

Initializing the Physical Allocation Table

- **Kernel-reserved addresses:**
  - Pages ¡ `VM_USERLO_PI` or ¿= `VM_USERHI_PI` are reserved.
  - Set permission to 1 for these pages.
- **User-space pages:**
  - Pages within [`VM_USERLO`, `VM_USERHI`] can be used if marked available.
  - Permissions are based on memory map.

Kernel-Reserved Pages

- *Pages* $<$ `VM_USERLO_PI` and *Pages* $>=$ `VM_USERHI_PI` are reserved.
- Set permission to 1.

User-Space Page Initialization

- Pages within [VM_USERLO, VM_USERHI] are checked.
- Permissions set based on memory map.
- Pages are marked as:
    - **2**: Usable.
    - **0**: Unavailable (partial pages considered unavailable).

Final Page Permission Setup

- Loop through the memory map.
- Set permission based on usability:
  - 2: Usable pages.
  - 0: Unavailable or partially usable pages.

Introduction to Page Allocation

- **Page Allocation**: Managing physical memory by allocating and freeing pages.
- **Key Functions**:
  - `palloc()`: Allocates a physical page.
  - `pfree()`: Frees an allocated page.

Understanding Physical Pages

- **Physical Page Size**: Defined as 4KB (`PAGESIZE = 4096`).
- **User Space Limits**:
  - `VM_USERLO`: Start of user-space memory (0x40000000).
  - `VM_USERHI`: End of user-space memory (0xF0000000).
- **Page Index Range**:
  - `VM_USERLO_PI` to `VM_USERHI_PI` determines valid page indices.

Overview of `palloc()`

- **Purpose**: Allocate a physical page.
- **Process**:
  1. **Check Availability**: Ensure pages are available in the allocation table (AT).
  2. **Scan for Unallocated Pages**: Look for the first unallocated page with normal permissions.
  3. **Mark as Allocated**: If found, mark the page and return its index.

Initialization and Scanning

- **Starting Point**: The allocation starts from the variable `next`, initialized to `VM_USERLO_PI`.
- **Loop Logic**:
  - Scan from `next` to `VM_USERHI_PI`.
  - Wrap around to `VM_USERLO_PI` if the end is reached.
- **Return Value**: Returns the index of the allocated page or 0 if none are available.

Optimizing with Memoization

- **Memoization Concept**: Store the last allocated page to avoid scanning the entire AT repeatedly.
- **Efficiency**: Reduces overhead by starting the scan from the last allocated page.

Overview of `pfree()`

- **Purpose**: Free a physical page.
- **Process**:
  - Takes an index (`pfree_index`) of the page to be freed.
  - Calls `at_set_allocated(pfree_index, 0)` to mark the page as unallocated.

Conclusion

- Efficient memory management is crucial for the performance and stability of operating systems.
- Understanding the role of bootloaders and memory allocation techniques is essential.
- Key functions such as `palloc()` and `pfree()` play a vital role in managing physical memory.
- Ongoing optimization techniques can enhance memory allocation efficiency and system performance.

Thank You!

Thank you for your attention!