Design Document for MP2: Frame Manager

Harsh Wadhawe

21/09/2025

Course: CPSC-410/611/613
Assignment: Machine Problem 2

Overview

This document describes the implementation of a contiguous frame pool manager for the kernel's virtual memory system. The frame manager handles allocation and deallocation of physical memory frames, supporting both single frame and contiguous multi-frame allocations.

Files Modified

The following files were modified for this assignment:

1. `cont_frame_pool.H` – Added private data members and method declarations
2. `cont_frame_pool.C` – Implemented all frame pool management functions

No other files were modified. The implementation works with the existing kernel infrastructure.

Design Decisions

Data Structure Design

Bitmap Approach: The implementation uses a simple bitmap where each frame's state is stored as one byte (8 bits), allowing for three distinct states:

- `Free (0)`: Frame is available for allocation
- `Used (1)`: Frame is allocated but not the head of a sequence
- `HoS (2)`: Frame is the Head-of-Sequence (first frame in an allocation)

Static Pool Management: A static array `frame_pools[]` tracks all created frame pools to support the static `release_frames()` function that must identify which pool owns a given frame.

Key Implementation Details

Constructor

- Initializes frame states to `Free`
- Handles both internal and external storage of management information
- Registers the pool in the static pool list
- Marks info frames as used when stored internally

Frame Allocation (`get_frames`)

- Sequential search for contiguous free frames
- Marks first frame as `HoS`, subsequent frames as `Used`
- Returns absolute frame number (not relative to pool)
- Returns 0 on allocation failure

Frame Release (`release_frames`)

- Static method searches all pools to find frame owner
- Validates that the frame is marked as `HoS`
- Marks the head frame and all subsequent `Used` frames as `Free`
- Stops when encountering `Free` or another `HoS` frame

Inaccessible Regions (`mark_inaccessible`)

- Similar to allocation but for predetermined frame ranges
- Converts absolute frame numbers to pool-relative indices
- Marks specified regions as permanently allocated

Memory Overhead (`needed_info_frames`)

- Calculates storage requirements for 1-byte-per-frame bitmap
- Returns ceiling of (`total_frames`/`frames_per_info_frame`)

Memory Layout

The implementation supports the kernel's memory organization:

- **0–1MB:** System reserved areas (ignored)
- **1–2MB:** Kernel code and stack (ignored)
- **2–4MB:** Kernel frame pool (512 frames)

- **4–32MB:** Process frame pool (7168 frames)
- **15–16MB:** Marked as inaccessible region

Testing Strategy

The `kernel.C` file includes basic tests that verify:

1. Single frame allocation and deallocation
2. Multi-frame contiguous allocation
3. Frame reuse after deallocation
4. Proper handling of inaccessible regions

Efficiency Considerations

Time Complexity:

- Allocation: $O(n)$ where n is the number of frames in the pool
- Deallocation: $O(p+m)$ where p is the number of pools and m is the sequence length
- Mark inaccessible: $O(1)$ for marking, $O(n)$ for the range size

Space Complexity:

- Uses 1 byte per frame for state storage
- For 32MB memory: 8192 frames $\times$ 1 byte = 8KB management overhead

The current byte-per-frame approach was chosen for simplicity and clarity, following the DRY principle with clean getter/setter abstractions.