

# MP2 Design Document — Contiguous Frame Pool (ContFramePool)

Name: Noaa Gomez

Assignment: Machine Problem 2 (Frame Manager)

Completion: Main assignment only (no bonuses)

Date: 2026-02-02

## 1. Assigned Tasks

- Implemented the contiguous frame pool manager in **cont\_frame\_pool.C** and updated required data structures in **cont\_frame\_pool.H**.
- Main assignment completed.
- Bonus options: not attempted.

## 2. System Design

The frame manager provides contiguous allocation and release of physical frames (pages) inside a fixed pool range.

Each **ContFramePool** manages a continuous interval of frame numbers  $[base\_frame\_no, base\_frame\_no + n\_frames)$ .

The allocator supports requests for  $n$  contiguous frames and returns the first frame number on success.

To track frame usage efficiently, the implementation uses a compact bitmap with **2 bits per frame**.

Each frame is in one of four states: **Free** (available), **Used** (allocated, not first), **HoS** (Head-of-Sequence; first frame of a contiguous allocation), and **Inaccessible** (reserved/unavailable). Contiguous allocations are represented by marking the first frame as HoS and the remaining frames as Used.

Release begins at a HoS frame and frees subsequent Used frames until the run ends.

## 3. Management Information Storage

The bitmap can be stored **internally** or **externally**:

If  $info\_frame\_no == 0$ , the bitmap is stored at the start of this pool (internal). The bitmap's own frames are marked **Inaccessible** so they are never allocated.

If  $info\_frame\_no != 0$ , the bitmap is stored starting at that external frame number.

The helper **needed\_info\_frames( $n\_frames$ )** computes how many frames are required to store the 2-bit-per-frame bitmap (rounded up to whole frames).

## 4. Code Description

- **cont\_frame\_pool.H**: added private members for pool range, bitmap location/size, and a static registry used by `release_frames()`.
- **get\_state / set\_state**: read/write 2-bit state values from the bitmap (4 frames per byte).
- **get\_frames(n)**: first-fit scan for n consecutive Free frames; marks first frame HoS and the remainder Used; returns the first frame number or 0 on failure.
- **mark\_inaccessible(base,n)**: marks frames in the given range as Inaccessible if they belong to this pool.
- **release\_frames(first)**: static entry point; finds the owning pool from the registry and frees the run that starts at first (must be HoS).
- **needed\_info\_frames**: computes number of frames required for the bitmap based on 2 bits per frame and `FRAME_SIZE`.

## 5. Bonus Options

No bonus options were attempted for this submission.

## 6. Testing

Testing focused on correctness of allocation, contiguity, and release:

Allocated several runs of different sizes and verified returned frame numbers were within the pool range.

Checked that allocations do not overlap and that contiguous runs are returned.

Released allocated runs and verified the space could be re-allocated afterward.

Marked ranges as inaccessible and verified allocations never returned frames from those ranges.

The build expectation is: `unzip` → **make** → run kernel in the provided environment.