Introduction

The purpose of this design document is to describe the design and implementation of a memory manager in C. This memory manager will be able to emulate the allocation and deallocation of memory in a computer system. The program will implement two eviction policies: FIFO (First-In-First-Out) and Third Chance Replacement (a variation of Second Chance Replacement). A linked list will be used to track the state of physical frames.

Overall Design

The memory manager program will consist of several main components:

1. A table element called ‘v_Page’ struct that represents a physical frame of memory. Each v_Page will contain a **resident** property that indicates whether the frame is currently in use or free, as well as a **next** field that points to the next v_Page in the linked list. It will also contain a property **frame_number** and **virt_page** that represents its physical frame number and virtual page number respectively. It will also store a **wb** write bit, **modified** bit, and a **referenced** bit to indicate its current state in regards to the third chance eviction policy described in the project specifications.

2. A **Table_Stack** struct that represents the linked list of v_Page. This struct will contain a **head** field that points to the first frame in the list, as well as a resident_count field that indicates the number of frames currently allocated in physical memory.

3. A set of functions that implement the core functionality of the memory manager. These functions include:

- **add_frame ()**: Allocates a new v_Page in physical memory, using the specified eviction policy if necessary.

- **evict ()**: Frees a v_Page from memory, and adds a new v_Page according to the eviction policy. This is done using either evict_FIFO or evict_THIRD.

- **get_frame ()**: Finds a frame in the physical memory if allocated, calls the add_frame() function

Detailed Design

FIFO Eviction Policy

The FIFO (First-In-First-Out) replacement policy works by selecting the frame at the head of the linked list as the best candidate for eviction. When a new element is added to the tail of the linked list, it becomes the new candidate for eviction. This policy assumes that the frame at the head of the linked list has been in memory the longest, and therefore is the least likely to be needed again in the near future. When allocating a new frame of memory, the memory manager will first check if there are any free frames available. If there are, it will simply allocate one of those frames and mark it as in use. If there are no free frames available, the memory manager will use the eviction policy to select a frame to evict, and then allocate that frame to the requesting process. In the case of FIFO, the frame at the head of the linked list will be selected, and the head will be updated to point to the next frame in the list.

Third Chance Eviction Policy

The Third Chance Replacement policy is a variation of the Second Chance Replacement policy. Like in the FIFO policy, each new element is added to the tail of the linked list, and the head of the list represents the frame that is the best candidate for eviction. However, in this policy, each frame also keeps track of a "referenced bit" and a "modified bit". When a frame is allocated, the referenced bit is set to 1 and the modified bit is set to 0. When a process accesses the frame, the referenced bit is set to 1. When a process modifies the contents of the frame, the modified bit is set to 1.

When allocating a new frame of memory, the memory manager will first check if there are any free frames available. If there are, it will simply allocate one of those frames and mark it as in use. If there are no free frames available, the memory manager will use the eviction policy to select a frame to evict, and then allocate that frame to the requesting process. In the case of Third Chance Replacement, the memory manager will start at the head of the linked list and iterate through the list until it finds a frame with both the referenced bit and the modified bit set to 0. If it finds such a frame, it will immediately evict that frame and allocate it to the requesting process.

If it does not find a frame with both bits set to 0, it will instead look for a frame with the referenced bit set to 1 and the modified bit set to 0. If it finds such a frame, it will decrement the replacement bit and give the frame a second chance. If after a second pass through the list, the memory manager still does not find a frame with both bits set to 0, it will select the frame with the replacement bit and modified bit set to 0 and evict it.

If all frames have both the referenced bit and the modified bit set to 1, the memory manager will again iterate through the list and decrement the modified bit of each frame. If after a second pass through the list, the memory manager still does not find a frame with both bits set to 0, it will select the frame with the replacement bit and modified bit set to 0 and evict it. This gives each frame up to three chances before it is evicted.

Conclusion

In conclusion, the FIFO policy was successfully implemented in the memory manager program. It was a simple and straightforward policy to implement, and it worked well in selecting the best candidate for eviction when allocating new frames of memory.

However, there were some troubles in implementing the Third Chance Replacement policy. This policy was more complex and required the memory manager to keep track of additional information for each frame, including the referenced bit and the modified bit. Additionally, the eviction algorithm was more complicated and required multiple passes through the linked list in some cases.