Student ID: 230018374 **Date:** 03/11/2025

Practical 2 - Buddy Allocator Algorithm

Algorithm Implementation:
The allocate_pages() function implements a three layer allocation strategy. It will first attempt to check a per-order cache with an O(N) worst-case complexity for the retrieval of the recently freed blocks. Then, if the block cannot be found, it will attempt to find a block by searching through the free lists starting by the requested order and then splitting larger blocks if necessary. Moreover, if it is unable to find a block in any order bigger than the one which was requested, it will trigger a cleanup function which will complete all of the deferred merges to create larger contiguous blocks. Finally, after the final cleanup and merging, it will attempt to search for a fitting block, returning a nullptr if it fails again.

The free_pages() function expands upon the aforementioned idea by conducting a three-stage freeing process as well. Initially, it will attempt to add the recently freed blocks into the cache for fast future allocations of the same order (as the cache works by order), returning early if this can be done. If the cache is full however, it will then check if the buddy is also in the cache and evict it to merge it immediately. Finally, it also employs lazy or deferred coalition when the cache is full by marking the block with a pending merge mark through a bitmap, and only performing the merging when both buddies are freed or when the memory pressure triggers the cleanup function mentioned above.

Assumptions:
- allocate_pages() **should only log a message if there's a failure when allocating**. This was based on the self-test logs which were not expecting any other behaviour.
- It is possible that blocks of the same order are frequently requested. **Assuming that this temporal locality exists**, my cache is capable of exploiting it for an increase of performance.
- Deferred coalition will provide an increase in performance, **assuming that incoming page requests are of smaller pages rather than the opposite.** However, for workloads with frequent large page requests or high fragmentation, the allocator may trigger cleanup_pending_merges() more often, reducing but not eliminating the performance benefit as the cache still provides value for small pages, while cleanup serves as a necessary fallback to decrease fragmentation. Performance degradation would occur primarily when cleanup frequently fails to satisfy large requests. The concept of deferred coalition came from this page.

Optimisations:
- **Temporal locality cache,** which exploits frequently consistent requests of the same order for an increase of performance. By using a static array, it caches recently freed pages and returns them immediately if a request with the same order comes.
- **Lazy Coalescing,** by postponing the merging process by marking the block with a pending mark, lazy coalition reduces the performance overhead of freeing, since it only has to merge when needed (in cases of significant fragmentation or when a large enough request comes).

Challenges and Solutions:
- **Debugging:** Without a step-by-step debugger and only having access to print statements, it is difficult to walk through the functioning of the code. Especially when considering how many components are abstracted away from us as the programmer. There was no easy solution aside from implementing a conditional `DEBUG_MODE` to enable logging on the testing stages.