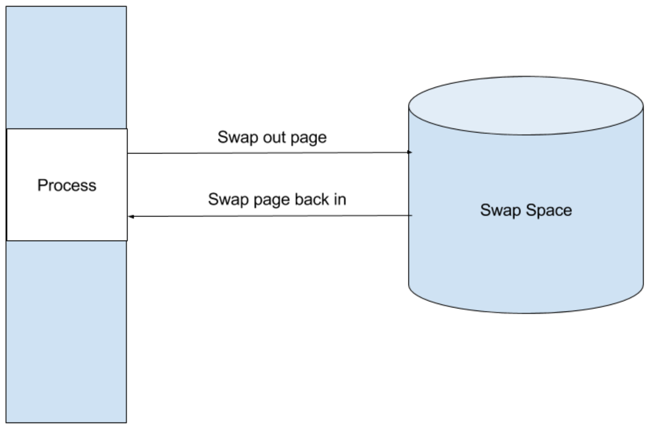
Paging to Disk

Team 18: Henry Peteet, Premkumar Saravanan, Millad Asgharneya

Problem & Motivation:

Paging to disk is a memory management scheme by which a computer stores and retrieves data from disk for use in main memory when environments require more than the available amount of main memory. Consequently, a dedicated partition called swap space is used to free up main memory by taking in relatively inactive pages.

Swapping to disk allows the operating system to extend the the amount of virtual memory available by combining the sizes of physical memory and swap space. However, disk I/O operations are far slower than RAM operations. The latency of disk operations can be thousands of times slower than accessing physical memory. Therefore, it is important to try to minimize the frequency of swapping to maximize the performance of the operating system.

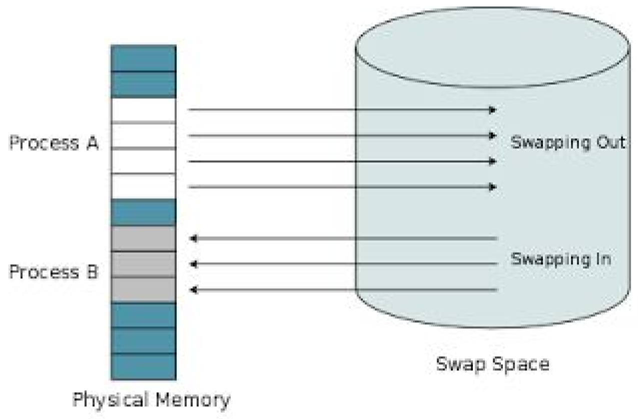


The diagram describes the simple mechanism involved in swapping memory.

Inactive pages are swapped into disk when more memory is needed and are eventually swapped back from the swap space to main memory when the pages become active and are requested again by the environment.

Design:

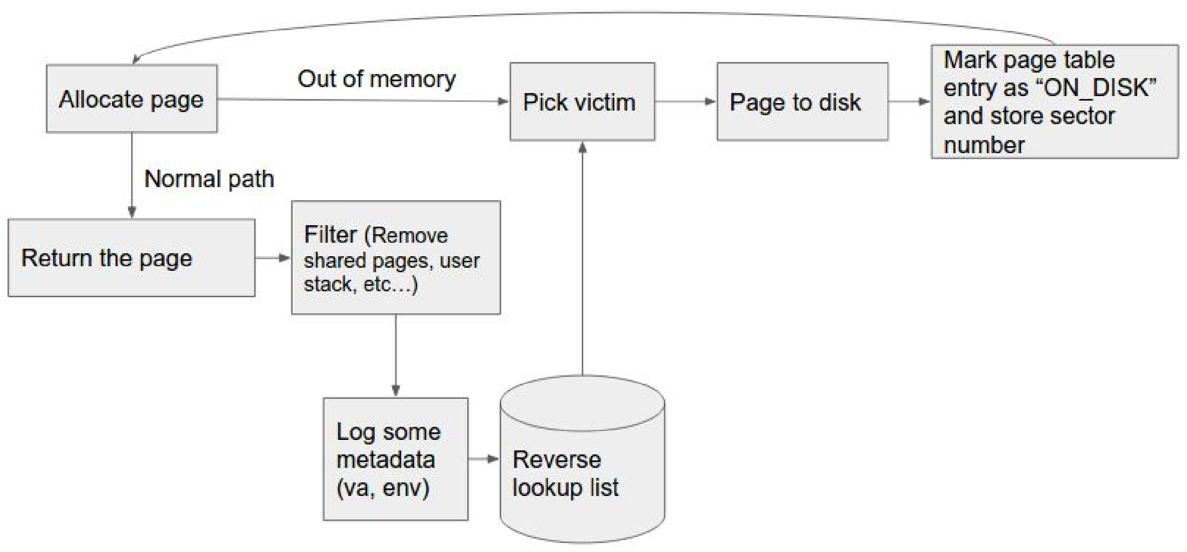
We wanted to add swapping to disk to JOS. We initially planned on creating a simple swapping functionality able to support a single environment that could be larger than 64M, which is the size of the physical memory in JOS. We were also interested in creating a shared swap space which would allow multiple environments to use the same shared space. Furthermore, it would allow us to evict relatively inactive pages from one environment which would enable the creation of new environments. For the main page eviction policy involved in selecting relatively inactive pages, we used the FIFO or First-In First-Out eviction policy. While this is a poor algorithm for minimizing page faults it is the easiest to implement allowing us to spend time on other features, mainly the ability to evict pages from other environments.



Shared swap space allows relatively inactive pages of memory from one environment to be swapped into disk while more frequently used pages in another environment can be swapped back into main memory to improve the overall performance by reducing the number of disk operations.

Implementation

The implementation of this system has a few more moving parts than the high level explanation. Firstly we have to keep track of the pages that are in memory at any given time so that we can pick which page should be evicted. We implemented this with a linked list of on-disk pages.


Allocate page
Out of memory
Pick victim
Page to disk
Mark page table entry as "ON_DISK" and store sector number
Normal path
Return the page
Filter (Remove shared pages, user stack, etc...)
Log some metadata (va, env)
Reverse lookup list

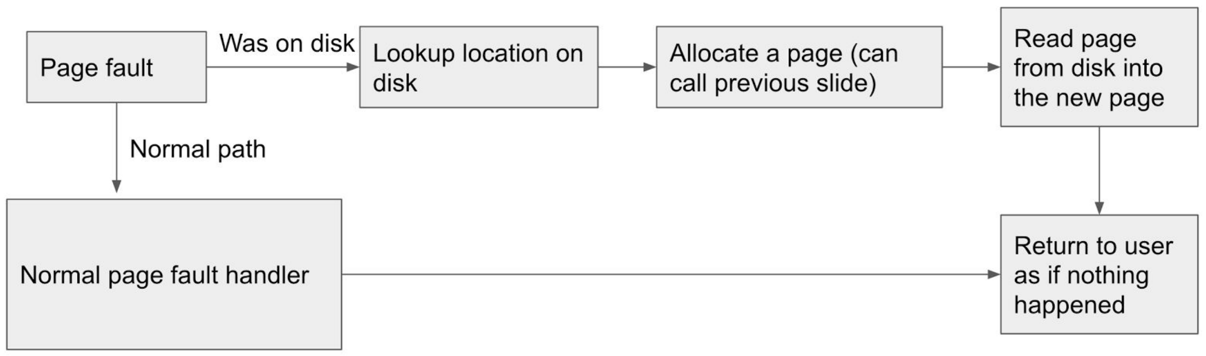
To maintain this list we had to

1. Add to the list when we allocate a new page
2. Remove from the list when we send a page to disk
3. Remove pages whenever we remap pages to be shared or unmap them from an environment.
4. Remove all pages belonging to an environment that is destroyed.

The second part of this system is a modification to the page allocation system. Whenever we try to allocate a page and there is not enough space in memory, instead of failing like we did in traditional JOS we need to evict a page. To do this we pick a free block of size 4096 on disk in our swap space, pick a victim page from the list, write the page to disk, and then record that this page is on disk. To do this we:

1. Mark the page's page table entry as not-present
2. Mark the page table entry as "on disk" (custom bit we set in the page table entry)
3. Store the block number into the page table entry

From this you can see we are keeping all of our information in the page table entry for this page, and part 1 ensures that we will get a page fault if anyone needs to use a page that is on disk. From here we simply intercept the page fault, check if the page was on disk (a bit that we set in the page table entry) and if it was, then we restore the page and retry the memory access. This allows for transparent restoration of pages.


Page fault
Was on disk
Lookup location on disk
Allocate a page (can call previous slide)
Read page from disk into the new page
Normal path
Normal page fault handler
Return to user as if nothing happened