4BSD UNIX Virtual Memory

- 4BSD UNIX use a virtual memory with demand paging.
- A global replacement policy and the clock (second chance) algorithm is used.
- The replacement algorithm is implemented by the **pageout daemon** process.
- Furthermore, there is a **swapper** process.
- The *pageout daemon* and *swapper* processes are executing in kernel mode but have its own process structure and kernel stack to be able to use kernel synchronization mechanisms such as sleep.
- All physical memory available for paging is divided into page frames.
- There is a **core map table (cmap)** with one entry for each page frame.
- For a process to be executable its page table and u-block need to be in main memory. Other pages are fetched via a page fault the first time they are referenced.

Page Replacement Algorithm

The clock algorithm is implemented by the *pageout daemon* process.

A software clock hand repeatedly sweeps all entries in the *core map*.

For each entry in the core map, the following is performed:

- If the frame is unused, advance the clock hand to the next entry.
- If I/O is active on the page, leave it as it is.
- If the reference bit is set, reset it.
- If the the reference bit is reset (because the page have not been referenced since last time the bit was reset), release the page by entering it into free list. If the page was modified it must first be written to the paging disk.

Clock Algorithm with two Clock Hands

- If a large main memory is used, the clock algorithm with one hand do not work very well because the time to scan all page frames is too large.
- On some systems an algorithm with two clock hands is used.
- The first clock hand is used to reset the reference bit, and the second clock hand releases pages with the referenced bit still reset. (they have not been referenced in the time span between the scan of the first and second clock hands)

Page Replacement Algorithm

- The goal for the *pageout daemon* is to assure that there are a sufficient number of free frames available at all times.
- The *pageout daemon* is awakened every 250 ms by a timeout to check that there is at least *lotsfree* (system parameter) free page frames. If this is the case the pageout daemon continues to sleep.
- If the number of free frames is below *lotsfree*, the clock hand will sweep a number of steps. How many steps depends of the number of pages needed to reach *lotsfree* free frames.
- There is a maximum number of steps the *pageout daemon* is allowed to sweep at one occasion. This is adjusted so that the *pageout daemon* should not use more than 10% of the CPU time.

Swapping

If it is discovered that the paging system is overloaded, the swapper process will be started to swap out some processes entirely (including page table and u-block).

The reason for using swapping is to try to avoid that the system enters a *thrashing* condition.

The *swapper* process is started only if the following conditions are fulfilled:

1. Load average is high (many processes in the ready queue).
2. The number of free page frames is below a low value *minfree*.
3. The average number of free frames is less than *desfree*. (lotsfree $>$ desfree $>$ minfree).

Swapping

Swapout

The swapping algorithm worked as follows:

- If some process has been idle more than 20 seconds, swap it out.
- Else select the one among the four biggest processes that has been in main memory the longest time.

The algorithm is repeated until a sufficient number of frames is available or no more candidate processes for swapping can be found.

Swapin

with a few seconds interval, the swapper process will check if some swapped out process can be swapped in again.

Only the page table and the u-block is brought in by the *swapper* process. The pages are not fetched until they generate a page fault.

Page Fault Handler

When a process references a page that is not in main memory, a page fault is generated by hardware.

In principle the system will allocate a page frame for the page, read it in from the paging disk and update the page table so that the process can be restarted.

In some cases the page need not be fetched from the disk:

- If the page have been used earlier, it may remain unmodified in the free list. In this case the page can be re-found by a hashing search. The page table is updated and the process restarted.
- Pages in uninitialized data or stack areas do not need to be fetched from disk. A free frame is allocated and initialized with zeros, and the process is restarted.