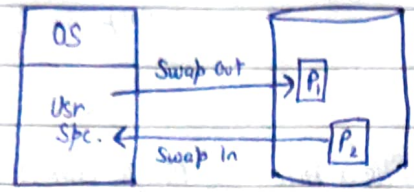
Swapping —:



- Swapping algorithm components —: (Swapping approach).
  - × Manages space on swap device.
  - × Swapping processes out of main memory.
  - × Swapping processes into main memory.

× Approaches :
- Swapping : Swap complete process in & out of memory.
- Paging : Swap pages to / from memory.

× Swap Device : Block device in a configurable disk section.
- Kernel allocates spc. one block at time for files
- Swap Device allocated in contigious blocks (suffers from fragmentation).

× Swapping (Approach) —:

- Management of Space on Swap Device :
  - Using map (data structure) (in-core table).
  - Map : Array w/ entries consisting of allocatable resource addr. & # of res. units avail.
    (Allows first-fit alloc. of contigous 'blks' of resource).

Eg :

Addr.	Units
1	10000.

Init.

Alloc 100 u :

101	9900

Map.

Alloc 50u :

151	9850

Map:

Alloc 100 u :

251	9750

Map

× Freeing Resources = Cases —:

1) Freed Resource completely fills a hole (Merge entries adj. to holes in 1)
2) Freed Resource partially fills a hole (Update adj. free entry)
3) Freed Resource partially fills hole but is discontiguous to existing (Create new entry)

Free 50u @ Addr 101.

101	50
251	9750

Free 100u @ Addr 1 :

1	150
251	9750

Free 100u @ Addr 151

1	10000

Alloc 200 u :

1	150
451	9550

Alloc 50u :

51	100
451	9550

malloc : Allocates map space :

Input : map addr.
: # units reqd.

Output : Addr. if successful, NULL (0) otherwise

- For every map entry, check if entry can fit req. units
If map entry gets completely filled, remove entry or else adjust start addr. & return addr from entry. (original addr.) If no entry satisfies, return 0.
(Also adjust memory with size of map entry if it remains).

× Map entries allocated to process (swap mem.) saved in U-area of process.

- Swapping Processes Out —:
  - Kernel performs swap out of process if memory required, owing to:
    - Creation of child processes using fork.
    - Modifying data using brk.
    - Increase in stack size (natural).
    - Swapped out process to be swapped in.
  - During swap out, only copy entries of regions containing data (non-empty) to swap device.
  - All user processes may be moved to swap area, but some bg prog., kernel proc & sys. proc. must not be moved.

× Fork Swap :
  - If parent process did not find enough memory to create child context, actions taken :
    × Kernel swaps process out w/o freeing memory occupied by in-core (parent) copy. (Child proc. is swapped out).
    × On swap-complete, child process exists on /swap, parent places child in 'ready-to-run' mode state & returns to user mode
    × Ready-to-run child will eventually be swapped to memory.

× Extension Swap :
  × Required space (incl. new requested spc.) reserved on swap device.
  × Addr. translation mapping adjusted.

- Swapping Process In —:
  - Performed by process w/ PID 0 (Swapper)
  - Infinite run wastes CPU resources., so execute only when requirement.
  - PID 0 sleeps until wakeup for swap, executed all times in kernel mode (~~wakeup~~ wakeup sent by clock handler) (no control to user).

* Swap-in candidates :
  Ready-to-run, swapped-out processes.

× Periodic Process — :
  - Search set of ready, swapped-out processes for others to swap in. & selects candidate.
  - Swaps in selected candidate after creating room.

× Conditions — :
  - No ready processes on swap device (return to sleep)
  - Process candidate cannot be accommodated (out of memory)

× Sleeping processes have higher chances of being swapped out.

× If out of memory, swapper performs swap out first.

× Swap Out Criteria :
  - Sleeping processes, criteria based on priority & time process has been in memory (choose proc. w/ low priority & high memory time).
  - If no sleeping process, choose ready to run process based on nice value & time in memory.
  - Prior swap out, ready to run process to be swapped out must be core resident for ≥ 2s.
  - Process being swapped in must have been swapped out for ≥ 2s. (Prevent immediate swap out, swap in)
  - If no process to swap-in / swap-out, swapper sleeps.

Q Processes : 5 (same size, ready to run)
  Memory : 2 processes at max
  Time Quantum : 1s. (For context switch)
  Assume no syscall, no preemption (no new processes)

T↓	A	B	C	D	E
	Runs Swap	Runs Swap	Swap Out	Swap Out	Swap Out
0	0 0 ↓1	0 0 ↓1	0 0 ↓1	0 0 ↓1	0 0 ↓1
1	1 0 ↓2	1 0 ↓2	0 ↓2	0 ↓2	0 ↓2
	Swap Out	Swap Out	Swap In	Swap In	Swap Out
2	0 0 ↓1	0 0 ↓1	0 0 ↓1	0 0 ↓1	0 2 ↓3
3	0 1 ↓2	0 1 ↓2	1 0 ↓2	1 0 ↓2	0 3 ↓4
	Swap In		Swap Out	Swap Out	Swap In
4	0 0 ↓1	0 2 ↓3	0 0 ↓1	0 0 ↓1	0 0 ↓1
5	1 0 ↓2	0 3 ↓4	0 1 ↓2	0 1 ↓2	1 0 ↓2
	Swap Out	Swap In	Swap In		Swap Out
6	0 0	0 0	0 0	0 2	0 0

- Issues during Swap In -:
  - × Swapped out process to swap in process not enough to fulfil ~~create~~ memory requirement ~~fulfil~~ (size of swapped out process lower than that of process to swap in).
  - × If swapper sleeps when no memory avl. to swap in process, must research for process to swap in even when process was already selected.
  - × If ready process selected to swap out

  - × If swapper attempts swap out but encountered deadlock as:
    - All procs. in memory are asleep.
    - Ready to run processes swapped out.
    - No room in swap device to swap out.
    - No room in main memory to swap in.

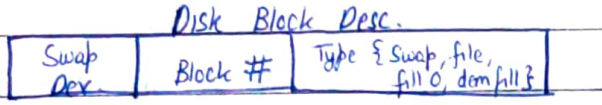
Swapper :
in an unconditional loop ; check swapped out, ready to run processes & select one waiting for the longest. If no process, sleep until swap in regd. & reloop.

× Demand Paging (Approach) —:

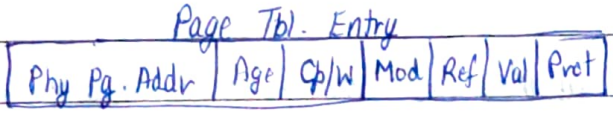
× Locality : Processes tend to exec. instructions in small portions of their text space such as loops & subroutines, and their data refs. tend to cluster in small subsets of total data space. Thus, nearby pages are more likely to be used.
× Working Set : Set of pages used by a process in last n memory refs. (n = window of work. set).
: Fraction of complete process.
× Multiple processes may simultaneously fit in memory in a paging system, potentially improving throughput by reducing swapping of complete processes.
× Implementation :
- Swap rarely used pages to swap device
- Handle page faults.
× Data Structures :
- Page Table Entries
- Disk Block Descriptors
- Page Frame Data Table (pfdata)
- Swap Use Table

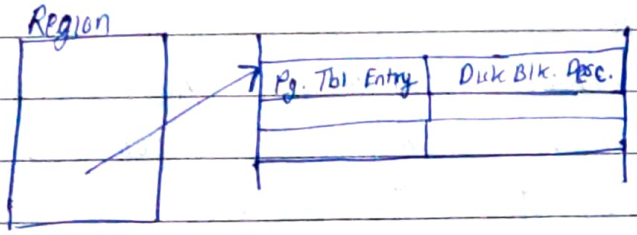
× Regions contain page tables to access physical memory.
× Page Table Entry :
- Physical Address of Page
- Protection Bits for r, w, x from page
- Demand Paging Bits :
  - Valid : Page Data Legality (Not related to valid page ref.)
  - Reference : Whether page was recently referred by processes.
  - Modify : Whether process recently modified page content
  - Copy on Write : Whether new copy must be created upon modification
  - Age : Duration of membership in working set (increased on every clock hck. on idle state)

-! A page reference may be to an invalid page, but such ref. is legal.
-! Copy on Write used in conjunction with the fork() syscall.
-! Pages maintained by the page stealer. (Aging one of criteria for stealing).
× Disk Block Descriptor : Describes disk copy of virtual page.



(dem : demand)





× Page Frame Data Table —
- Describes each page of memory & is indexed by page #.
- Fields : Page State : Whether in swap device or exec file.
  : # Proc. referring page :
  Ref Ct. : # valid page tbl entries ref. page.
  : Logical Dev. (Swap/fs) & blk # containing copy.
  : Pointers to other pfdata entries on list of free & hashed pages.

- Maintains free list containing free pages (similar to buffer cache)
- Created only once, global in nature, only updated upon use.
- Entries of pfdata linked onto free & hashed list.
- Free List :
  - Cache of pages avl. for reassignment.
  - On page fault, page may be avl on free list.
  - New pages alloc'd in LRU order.
  - pfdata entry hashed in swap as per block & dev #.

× Swap-Use Table – :
- Contains entry for all pages on swap device.
- Entry contains reference count of number of page table entries pointing to page on swap device.

× Fork-ing in Paging Systems – :

- Kernel dups every region of parent process during fork syscall & attaches to child.
- Page copy avoided by manipulating region tbls., page table entries & pfdata tbl entries.
- For shared regions, ref count of page frame incremented.
- For private regions (data, stack), new page table entry alloc'd.
- Each parent page tbl entry examined :
  - Valid page => Reference count in pfdata table entry incremented (indicating access via separate region)
  - Page on Swap Device => Swap-use table reference count incremented.
- On modification by proc to shared page :
  - Copy page to ensure private versions.
  - Implementation :

- Set copy-on-write bit on page during fork().
- On write, incur protection fault.
- On handling protection fault, copy page, add new entry.
- Physical copying of page deferred until process requirement.

× Exec-ing in Paging Systems —

- On demand paging systems
  - Read exec. file into memory from fs.
  - If exec. file too large for avl. memory, memory not preassigned to exec. file, but faulted in (adding pages and assigning memory as required).
  - Initially, page tables & disk block descriptors assigned, with page tbl. entries marked 'demand zero' or 'demand fill'. Then file read in memory.
  - Fault handler notes when page is (during incur of validity fault):
    - Demand fill: Content immediately overwritten w/ exec file contents so it need not be cleared.
    - Demand Zero: Clear contents
  - If process not accomodable, page stealer periodically swaps pages from memory to make space.
- Inefficiencies:
  - Page fault incurred upon read of each page of exec. file, even when page is never accessed.
  - Page stealer may swap out pages before exec is done, leading to multiple swaps (2+) per page if page regd. early.
- To improve efficiency, demand page directly from exec. file if data is properly aligned (indicated by a spl. magic #).

× Page-Stealer Process —:

- Kernel process to swap out non-working set pages of processes.
- Launched during system init, invoked during system lifetime when low on free pages (thrashing, no pages in free list).
- Active unlocked regions examined & age field of valid pages incremented.
- Locked regions left as-is in hope of examination in next pass (once unlocked).
- Regions locked upon page fault for region page to prevent theft of page faulted in.
- Paging states for page in memory:
  - Aging page, not eligible for swapping.
  - Page eligible for swap & reassignment.
- Page age incremented upon every examination, swapped (or becomes eligible for swap) when age reaches n (size of window). Age of only non-referred pages are incremented. (Ref bit indicates presence in working set).
- Processes sharing regions update refs. of same set of pg. tbl. entries
- Only pages not in working set of any process eligible for swap, else it remains in memory.
- Page stealer invoked (wake-up) by kernel when avl. free memory touches low water-mark.
- Pages swapped out (stolen) until free memory exceeds high water-mark.
- Thrashing: Condition when system spends major portion of time servicing page faults (with little to no actual processing).
- Low & High Water-Marks set to reduce thrashing.
- Page Swap Out Situations —: (Considering copy on swap device)

1 - No copy on swap device
2 - Copy on swap device, In-core contents non-modified (intact)
3 - Copy on swap device, In-core contents modified. (modify bit set).

1) Update entries, create/alloc. new page in swap area/device.
2) No-op required except update entries in pg. tbl.
3) Update page content on swap device. (Freeing old used space on swap).

- Page stealer fills a list of pages to be swapped, possibly from various regions & swaps them to swap device when list is full.
- On write to swap device (by: kernel) -:
  - Turn off valid bit in page table entry.
  - Decrement use count in pfdata entry.
  - If count hits 0, pfdata entry moved to end of free list.
  - Allocate swap space, copy content.
  - Save swap address in disk block descriptor.
  - Increment swap-use count.

× Page Faults -:

× Validity Faults
× Protection Faults.