Lecture 7

Memory Allocation Techniques

- How does the OS manage the free memory space ?
- How are free memory blocks allocated to processes ?
- Is there any side effect of memory allocation ?

Free space management

- The OS needs to locate free memory blocks which can then be allocated to processes.
- If the system is using pages of fixed-size, one bit/page can show its state, free or not – this solution is called *free bitmap*.
- In some cases, memory blocks are not created equally and processes need contiguous page frames. In this case, the allocation is satisfied when a set of free pages that match the request is available.
- Free memory blocks can also be represented in a *linked list*. Again, when dealing with fixed-size pages, allocation is easy – take first off the list, or when it becomes free, append it to the list.

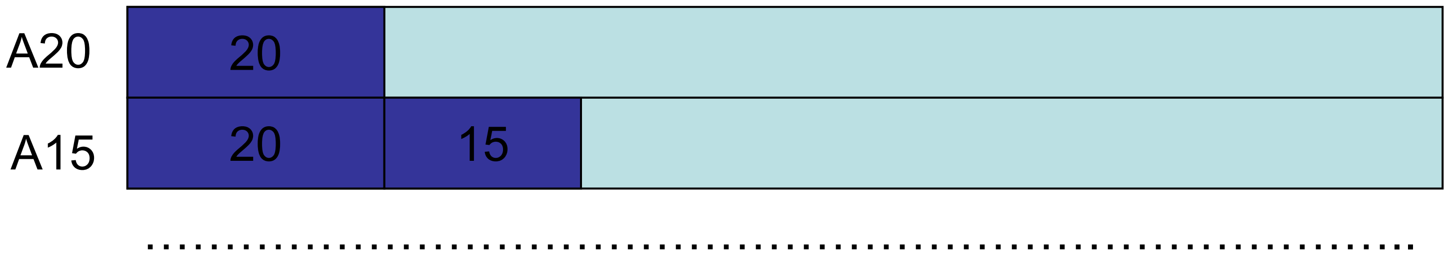
- If memory is allocated in variable-sized blocks, the list needs to be searched to find a suitable block.
- With the linked list, there is a structure which stores the first address, the size and a pointer to the next element in the list.
- To speedup the search process (O(n) for list), binary trees or hash tables can be used.
- Another important aspect is that except for *the global pointer free_list*, everything else is stored in the free blocks themselves.

Fragmentation

- If memory blocks are fix in size, the allocation process can result in waste – more memory allocated than necessary - internal fragmentation, or left not allocated – external fragmentation.
- The simplest method of allocating memory is based on dividing memory into areas with *fixed partitions*. Typically, fixed partitions between blocks of varying size are defined from the time the system starts until it shuts down.
- However, the flexibility of allocating memory in either large or small blocks is needed: e.g., a free block is selected and split into two parts – the first is allocated to the process, the second is returned as free. The allocation is done in multiples of some minimum allocation unit (OS parameter). This helps to reduce external fragmentation. Generally, the allocation unit is small.

Selection policies

- If more than one block can satisfy a request, then which one to select ?
- *First fit* takes the first block from the list which is greater than or equal to the requested size. If the request cannot be met, it fails. This policy tends to cause allocations to be clustered towards the low memory addresses – the effect is that the low memory area gets fragmented, while the upper memory area tends to have larger free blocks.
- *Example*: sequence of allocations (A) and deallocations (D): A20, A15, A10, A25, D20, D10, A8, A30, D15, A15, where n denotes the number of KB requested. Let’s assume that the memory space is 128 KB.



Other strategies

- *Next fit* starts the search with the free block that is next on the list to the last allocation. During the search the list is treated as a circular one. If returned to the initial starting point without any allocation, the process fails. This strategy leads to a more evenly allocation of free memory.
- *Best fit* allocates the free block that is closest in size to the request. Like first fit, best fit tends to create significant external fragmentation, but keeps large blocks available for requests of larger sizes.
- *Worst fit* allocates the largest block for each request. It has an advantage: if most requests are of similar size, the worst fit minimizes external fragmentation.

The buddy system

- All blocks are a power of 2 in size.
- Let *n* be the size of the request. Locate a block of at least *n* bytes and return it to the requesting process:

1. If *n* < smallest allocation unit, set *n* to be the smallest size.
2. Round *n* up to the nearest power of 2. select the smallest *k* such that $2^k \geq n$.
3. If there is no free block of size 2^k, then recursively allocate a block of size 2^{k+1} and split it into two free blocks of size 2^k.
4. Return the first free block of size 2^k in response to the request.

- Each time a block is split, a pair of buddies is created; they'll either be split or paired together.
- It is easy to determine (by looking at bit k+1) which is the buddy of a block.
- This method tends to have very low external fragmentation. The price paid is more internal fragmentation.
- The block that is deallocated is a buddy to a free block, in order to create a larger free one.

Over-allocation techniques

- Up to now, the assumption was that allocation deals only with free blocks. However, not all allocated blocks are in use all the time. Seldom used blocks can be transferred to disk.
- Swapping consists in transferring one blocked process memory space on disk. Then, when the process becomes active, it'll be restored.
  - If the process shares the code with other processes, only data and stack will be swapped.
  - The second issue is the amount of memory that is freed by swapping and the usage patterns of that process.

Segment/page swapping

- If the hardware supports only a limited number of segments (code, data, stack), only those can be swapped. A larger number of segments allow for more memory space to be freed by swapping.
- Paging is similar as technique but involves pages, which correspond to a finer-grained level. When a request for memory space is received, only the necessary number of pages is swapped.
- If a page was swapped out to disk, the present (P) bit is cleared, causing a page fault it there is an access attempt.
- When a page fault occurs, the OS must decide if the process tries to access a page not allocated to it or a swapped one. The loading of pages when they are needed is called *demand paging*.
- Generally, the system doesn’t load only one page but a set of pages (e.g., *n* pages of code) – this is called *pre-paging/pre-loading*.

Page replacement policies

- One option is to replace only pages of the process itself (local replacement) in contrast to selecting a page from all processes in the system (global replacement).
- *Belady's Min*: pick the page that will be used least soon. If t_i is the time at which page *i* will next be accessed, the selected page corresponds to $\text{argmax}_i t_i$. This strategy is optimal as it assures that the number of page faults is minimized. Of course, it is not a practical one (can't anticipate the future), but it can be used as a reference for simulations.
- *First In First Out* starts from the idea that pages are used for a finite amount of time after which they become "old". The page selected here is the one that has been in memory the longest. Implementation is done by a queue – all new pages are added to the tail of the queue.

- *Second chance* is an extension of FIFO: when a page is pulled off the head of the queue, the accessed (A) bit is examined. If it’s 0, the page is swapped out, else the bit is cleared and the page is reinserted at the tail of the queue. A second examination of the queue will produce available pages.
- *The clock algorithm* is similar to the second chance: two clock hands are moving synchronously above pages; the distance between them determines how long the page is given to be accessed. If it has not been accessed within that time, it can be swapped out.
- In many implementations of the two-handed clock, the hands are not moved only when a page fault occurs. The pair of hands periodically advances some number of pages and keep a record of those with A = 0.

- *Not recently used* refers not only to pages with A=0 but also to pages which were not written into (M=0); therefore, if they have a copy on the disk, they don’t need to be swapped out. Only one read brings the new page. This policy works well in conjunction with two-handed clock which analysis the bit pair AM – 00, and the lowest value is preferred.
- *Least recently used* is based on the time passed since a page was last time used. This concept is rarely used in practice because of the difficulty to implement it (software, hardware ?). One possibility is to periodically mark pages as not present (P=0). When a page is accessed, it generates a page fault. The OS checks to see if the page is actually present, and if it is, the access time is recorded; P=1 to avoid further page faults.

- *Not frequently used* is based on counting memory references. Periodically, all pages in the memory are swept and for each one with A=1, A is cleared and a page counter is incremented. Then, the page with the smallest value of the counter is selected for swapping.
- This strategy penalizes newly loaded pages and keeps heavily used pages longer than wished.
- This policy can be improved by weighing recent references more heavily than older ones – this is called *aging*.
- Question: how can aging be implemented ?

- *The working set* of pages corresponds to the number of pages a specific process is using at a time. There are variations, and two thresholds (called also watermarks) can be considered. They are the upper and lower bounds of the working set.
- If a process has allocated more pages than its upper threshold for the working set, it is a good candidate for page swapping. Contrary, if by taking a page the lower threshold is passed, it makes sense to swap all the pages.
- The two thresholds can be selected based on the page fault frequency. If a process generates page faults too often, it needs more pages; if it doesn't generate page faults for a time, probably it has too many pages.

Segments and pages

- Many systems work with segments that are made of pages.
- In this case, one option for overallocation management is to simply ignore segments and work only with pages.
- Sometimes segments can contain the full working set of pages, or critical pages of the process (e.g., libraries) and they can be dealt at the segment level – swap or not the entire segment.
- However, page swapping gives more flexibility in satisfying requests.

Exercises

- Consider a 64-bit system. If the page size is 64 KB, how many levels must the paging system have so that each table fits within a page ?
- Suppose a system with 128 MB of memory and no memory initially allocated. Given the sequence of allocations/deallocations: A(10), A(20), A(15), D(20), A(12), A(30), D(15) and A(17), all expressed in MB, show the free list at each stage for each of first fit, next fit, best fit and worst fit.
- Does using paged memory management hardware have any value even if pages are never swapped to disk and back ?