Lecture 11

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?

A. 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.

B. 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.

C. Selection policies for variable size blocks

- If more than one block can satisfy a request, then which one to select ?
- *First fit* takes from the list the first block 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.

D. 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 search for a free block of size 2^{k+1} and split it into two 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 de-allocated is a buddy to a free block; they are merged in order to create a larger free one.

E. 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 blocks in use 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 presence (P) bit is cleared, causing a page fault if 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*.

Questions and problems

1. How can the free bitmap and the linked list of free pages be stored for the fastest decision on the free memory blocks?
2. Compare the best-fit and worst-fit strategies stressing the benefits and disadvantages of each technique.
3. Consider that main memory size is 4 GB and it is organized of blocks of pages of variable size. A page is 4 KB, and a block can have one page, two pages, four pages, eight pages, or sixteen pages. The number of blocks of a certain size is up to you but their aggregated size can't be more than 4 GB. Consider a sequence of requests, 3.5KB, 4.1KB, 7.5KB and show how best-fit, next-fit and the buddy algorithm work.
4. Would it be beneficial for the kernel to create blocks of memory of variable size at run-time? Explain your answer.