My malloc: mylloc and mhysa

ID1206 OPERATING SYSTEMS

malloc(size)

- Used to allocate memory dynamically (at runtime) on the heap
- Needed when you do not know how much memory will be used until the program runs
- Return value:
  - a pointer (variable that holds an address) to the beginning of the allocated memory
  - or NULL if the memory can’t be allocated

malloc() - example

- Declaring an array with n number of elements
  - You do not know the number of elements from the beginning
  - It is given as an input from the user
  - The input n is taken at runtime and that much memory is allocated using malloc

sbrk(size)

- Ask for more memory on the heap by using the program break
- The program break points to the end of the heap
- Increments the program break by the defined size in bytes
- Return value:
  - the previous program break
  - or -1 on error
- sbrk(0) – returns current location of the program break

brk(address)

- Similar to sbrk() but takes an address instead
- Sets the program break to the specified address
- Return value:
  - 0 on success
  - or -1 on error

free(pointer)

- Deallocates the memory that malloc() points to
- Takes the pointer that was returned from malloc to find the memory
- It is the user's job to deallocate the space when it is not needed anymore
- To know how much memory that is supposed to be freed, the memory block keeps the size in a header block that lies just before the actual memory

Free list

- Data structure used for dynamic memory allocation
- Contains blocks of the available free space on the heap
- There exist different strategies used to select the appropiate memory that is requested from the user: best fit, worst fit, first fit, next fit, buddy allocation

Splitting and coalescing

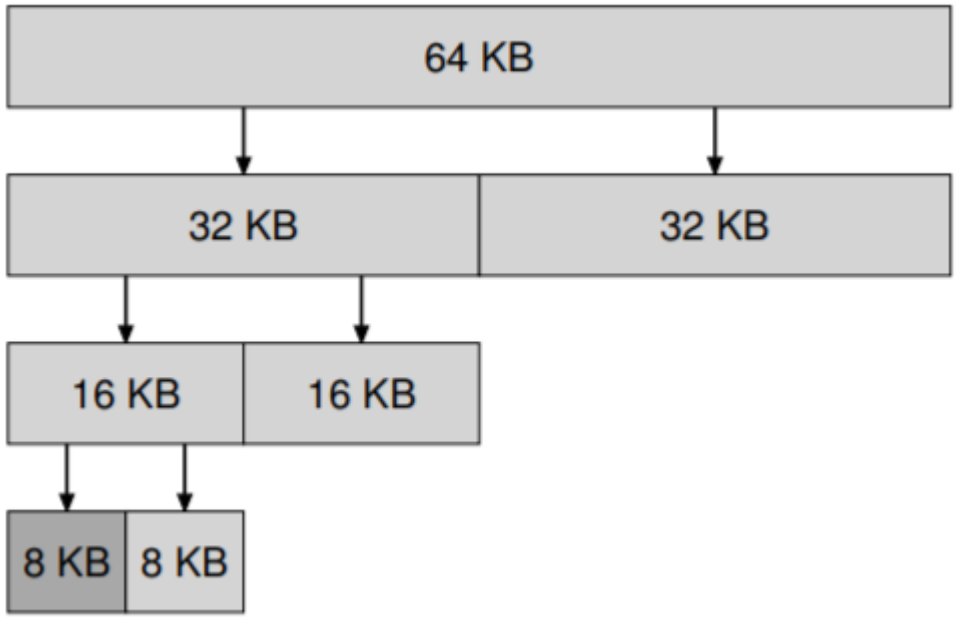
- Common techniques used in memory allocation
- Split a memory block in the free list
  - Return the needed amount to the user
  - Keep the rest
- When coalescing you merge two blocks of free memory that lies next to each other
  - Good to make sure that the heap is not divided into several small blocks of memory

Buddy allocation

- Used to make coalescing easier
- The free memory is seen as 2^n
- When a request for memory is made the memory is recursively splitted in two until an appropiate size is found
- Pros – Easy to determine if two blocks can be merged together
- Cons – Internal fragmentation as only blocks of size 2^n can be given

Buddy allocation - example

- 64 KB free space
- User request for 7 KB block of memory



Best fit

- Searches through the free list and finds a memory block that is equal or larger than the requested memory
- Returns the requested amount to the user and keeps the rest
- Pros – Reduces the amount of wasted memory
- Cons – A lot of performance is required to search through the whole list

Worst fit

- Searches through the free list and finds the largest amount of free memory
- Returns the requested amount to the user and keeps the rest
- Pros – Reduces the amount of small memory, leaves big pieces of memory free
- Cons – A lot of performance, the whole list needs to be searched through, takes away large chunks of memory that could be needed

First fit

- Finds the first block of memory that is large enough
- Returns the requested amount to the user and keeps the rest
- Pros – Quick, no need to search through the whole list
- Cons – Could leave small blocks of memory in the beginning of the list

Next fit

- Similar to first fit but has an extra pointer that keeps track of where you were last time and begins the search from there next time
- Performance is similar to first fit

Free memory strategies - example

Which block will each strategy choose for a request of size 15?

- Best fit: 20
- Worst fit: 30
- First/Next fit: 30