Memory Management

…an important aspect where data structures play an important role

Most data structures are designed to store and access objects of uniform size. A typical example would be an integer stored in a list or a queue.

Some applications require the ability to store variable-length records, such as a string of arbitrary length. One solution is to store in list or queue a bunch of pointers to strings, where each pointer is pointing to space of whatever size is necessary to old that string.

- The basic model for memory management is that we have a (large) block of contiguous memory locations, which we will call the **memory pool.**

- Periodically, **memory requests** are issued for some amount of space in the pool.

- A **memory manager** has the job of finding a contiguous block of locations of at least the requested size from somewhere within the memory pool. Honoring such a request is called **memory allocation**.

- At some point, space that has been requested might no longer be needed, and this space can be returned to the memory manager so that it can be reused. This is called a **memory deallocation.**

Internal and External fragmentation

Internal Fragmentation:

- In internal fragmentation, fixed-sized memory blocks are allotted to processes.

External Fragmentation:

- In external fragmentation, variable-sized memory blocks are allotted to processes.
- External fragmentation happens when the method or process is removed.

Sequential-Fit Methods

When memory is requested, a decision needs to be made about which block of memory is allocated to the request.

Sequential-fit methods attempt to find a “good” block to service a storage request.

In order to discuss which method is best, we need to investigate how memory might be managed.

The sequential-fit methods described here assume that the free blocks are organized into a doubly linked list.

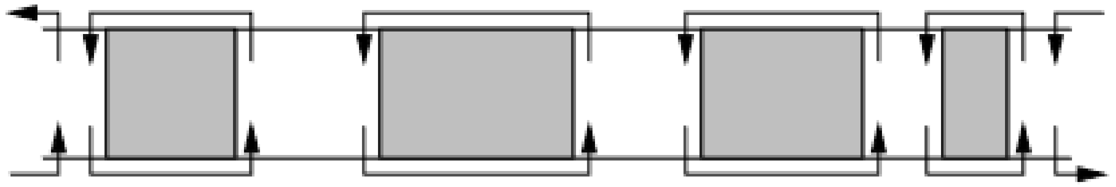


Fig: A doubly linked list of free blocks as seen by the memory manager.
Shaded areas represent allocated memory. Unshaded areas are part of the freelist

When memory is allocated or returned, the list is rearranged, either by deletion or insertion.

Sequential Fit Methods

1. First Fit Algorithm
2. Best Fit Algorithm
3. Worst Fit Algorithm
4. Next Fit Algorithm

1. **First Fit**:

- In the first fit, the partition is allocated which is the first sufficient block from the top of Main Memory.
- It scans memory from the beginning and chooses the first available block that is large enough.
- Thus, it allocates the first hole that is large enough.

2. Best Fit

- Allocate the process to the partition which is the first smallest sufficient partition among the free available partition.

- It searches the entire list of holes to find the smallest hole whose size is greater than or equal to the size of the process.

3. Worst Fit

- Allocate the process to the partition which is the largest sufficient among the freely available partitions available in the main memory.
- It is opposite to the best-fit algorithm. It searches the entire list of holes to find the largest hole and allocate it to process.

4. Next Fit:

- Next fit is similar to the first fit but it will search for the first sufficient partition from the last allocation point.

Comparing Sequential Fit Methods

- **First Fit** is most efficient, comparable to the Next Fit.

  Disadvantage: There can be more external fragmentation.

- **The Best Fit** algorithm leaves very small blocks of practically unusable memory.

- **Worst Fit** tries to avoid this fragmentation, by delaying the creation of small blocks.

Non-Sequential Fit Methods

- In reality, with large memory, sequential fit methods are inefficient.
- Therefore, non-sequential fit methods are used where memory is divided into sections of a certain size.
- Buddy system is an example of Non-Sequential Fit strategy.

Buddy Systems

- In buddy systems memory can be divided into sections, with each location being a buddy of another location.
- Whenever possible, the buddies are combined to create a larger memory location.
- If smaller memory needs to be allocated the buddies are divided, and then reunited (if possible) when the memory is returned.

Binary Buddy Systems

- In binary buddy systems the memory is divided into 2 equally sized blocks.

- **Suppose we have 8 memory locations**;
  {000,001, 010, 011, 100, 101, 110, 111}

- Each of these memory locations are of size 1, suppose we need a memory location of size 2.

  {000, 010, 100, 110}

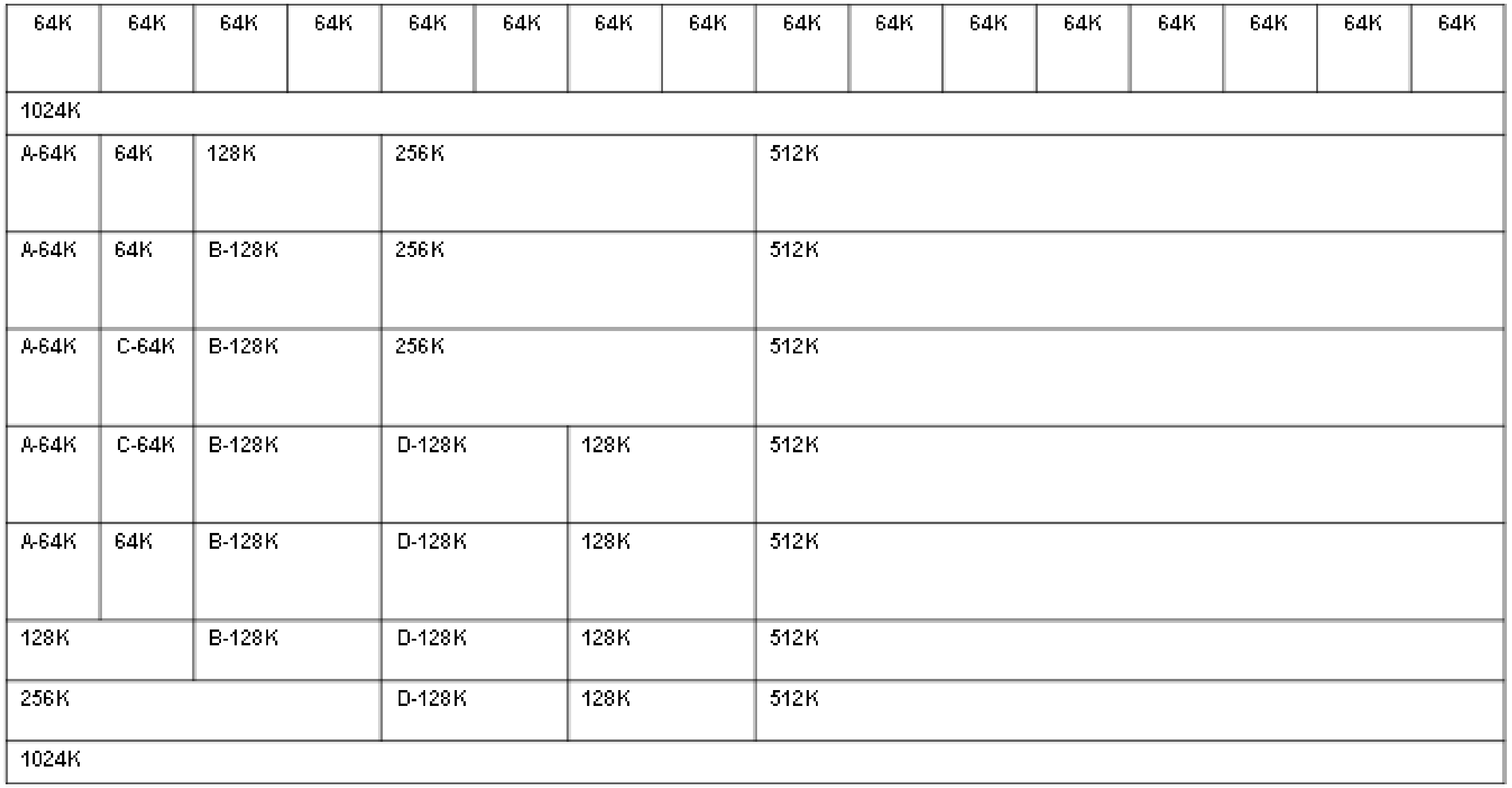
- Or of size 4,
  {000, 100}

- Or size 8.
  {000}

- In reality, the memory is combined and only broken down when requested.

Example:

Buddy System in 1024KB memory


64K 64K 64K 64K 64K 64K 64K 64K 64K 64K 64K 64K 64K 64K 64K 64K
1024K
A-64K 64K 128K 256K 512K
A-64K 64K B-128K 256K 512K
A-64K C-64K B-128K 256K 512K
A-64K C-64K B-128K D-128K 128K 512K
A-64K 64K B-128K D-128K 128K 512K
128K B-128K D-128K 128K 512K
256K D-128K 128K 512K
1024K

Sequence of Requests.

- Program A requests memory 34K..64K in size
- Program B requests memory 66K..128K in size
- Program C requests memory 35K..64K in size
- Program D requests memory 67K..128K in size
- Program C releases its memory
- Program A releases its memory
- Program B releases its memory
- Program D releases its memory

Buddy System (contd..):

Steps: If memory is to be allocated

- Look for a memory slot of a suitable size
  - If it is found, it is allocated to the program
  - If not, it tries to make a suitable memory slot. The system does so by trying the following:
    - Split a free memory slot larger than the requested memory size into half
    - If the lower limit is reached, then allocate that amount of memory
    - Go back to step 1 (look for a memory slot of a suitable size)
    - Repeat this process until a suitable memory slot is found

Buddy System (contd..):

Steps: If memory is to be FREED

- Free the block of memory
- Look at the neighbouring block - is it free too?
- If it is, combine the two, and go back to step 2 and repeat this process until either the upper limit is reached (all memory is freed), or until a non-free neighbour block is encountered

Some important points about Buddy Systems

- Unfortunately, with Buddy Systems there can be significant internal fragmentation.
  - Case ‘Program A requests 34k Memory’ – but was assigned 64 bit memory.
- The sequence of block sizes allowed is;
  - 1,2,4,8,16…2^m
- An improvement can be gained from varying the block size sequence.
  - 1,2,3,5,8,13…
- Otherwise known as the Fibonacci sequence.
  - When using this sequence further complicated problems occur, for instance when finding the buddy of a returned block.

Garbage Collection

- Another key function of memory management is garbage collection.
- Garbage collection is the return of areas of memory once their use is no longer required.
- Garbage collection in some languages is automated, while in others it is manual, such as through the delete keyword.

Garbage Collection

- Garbage collection follows two key phases;
  - Determine what data objects in a program will not be accessed in the future
  - Reclaim the storage used by those objects

Mark and Sweep

- The Mark and Sweep method of garbage collection breaks the two tasks into distinct phases.
  - First each used memory location is marked.
  - Second the memory is swept to reclaim the unused cells to the memory pool.

Mark

- A simple marking algorithm follows the pre order tree traversal method;

```
marking(node)
    if node is not marked
        mark node;
    if node is not an atom
        marking(head(node));
        marking(tail(node));
```

Sweep

- Having marked all used (linked) memory locations, the next step is to sweep through the memory.

- Sweep() checks every item in the memory, any which haven’t been marked are then returned to available memory.