8. Storage Management

Each and every programming language requires the use of the particular storage technique.

Types of Storage Management

1. Programmer Control Storage: When a management is explicitly done by a user or programmer with the help of certain in-build function like malloc or free, that allocates and frees the memory as per requirement.

2. System Controlled Storage Management: In many programming language the storage management is implicitly done by the system only with the help of various language features

Difference between Programmer & System Controlled Memory Management

Programmer Controlled Storage Management	System Controlled Storage Management
In this, programmer explicitly control over the storage with some in build function	In this only system implicitly controlled over the storage using some language features
There is extra burden on the programmer	There is no extra burden on the programmer
It is dangerous to programmer as it may lead to the errors	No such kind of error take place
Sometime programmer may conflicts with the controlled system memory management.	No such kind of interference of programmer is allowed.
It is easily & efficiently done by programmer as knows the actual requirement of the application.	It is quite difficult for the system to determine when to allocate and when to freed

Major elements that requires Storage:

1. Code Segment
2. System Run time Program
3. Storage Management Routine
4. User Define data structure
5. Subprogram Return Points
6. Referencing Environment
7. Temporaries in expression evaluation
8. Temporaries in parameter transmission
9. Input/output buffers
10. Miscellaneous system data

Along with this some operation also requires the storage like:

- Subprogram call and return
- Data structure creation and destruction
- Component insertion and deletion operation

Objective of Storage Management:

1. To provide the memory space to enable the several processes
2. To provide the efficient use of memory
3. To protect each program resource
4. To share the memory space if required
5. To make addressing transparent for programmer as much as possible

Features of Storage Management:

1. Reallocation
2. Protection
3. Sharing
4. Logical Organization: Program are generally organized in to modules some of them modules may be shared by other some are Read only or some them can be modified. So the storage management will responsible to handle this logical organization.
5. Physical Organization: Memory divided in either fast RAM or slow secondary memory so the memory management handles the moving or accessing the information between these two levels of the memory.

Phases of Storage Management

The following the phases that come under the storage management:

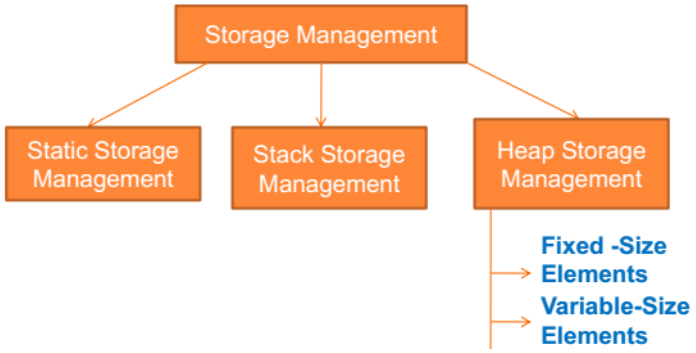
1. INITIAL ALLOCATION: In the starting of an execution, each piece of storage may be either allocated for some use or free. If it is free then it is available to implement and use it. Any storage management must have some techniques for keep track of free storage as well as allocation of free storage as per need during the execution.

2. RECOVERY: Storage that has been allocated and used and when it will become available it must be recovered by the storage management for reuse. It may perform very simple by means of repositioning stack pointer or complex by means of garbage collection.

3. COMPACTION AND REUSE: storage recovered may be immediately ready for reuse or compaction may be required to construct vary large blocks of the free storage from small pieces.

Storage Management Techniques

Following are the most basic storage technique:



Static Storage Management

This is simplest form of allocation, which is done during translation and remains fixed throughout the execution. Following are the some important point related with the static storage management:

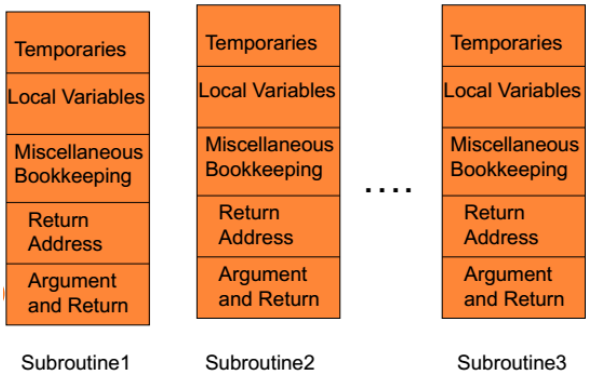
- Storage for all variables allocated in a static block
- Allocation can be done by the translator could be attached to the code segment
- Memory reference can be calculated at the translation time
- Subprogram variables use space even if subprogram never called
- Recursion not possible
- Dynamic data structure are difficult to handle
- All storage known at translation time
- Activation records directly associated with code segment
- Procedure call return straight forward

Advantage:

- It is efficient as there is no time or space is expanded for storage management during the execution
- The translator can direct generate I-value address for data items.

Disadvantage:

- It is incompatible with recursive subprograms calls and data structure whose size is dependent on input data.



Stack-Based Storage Management

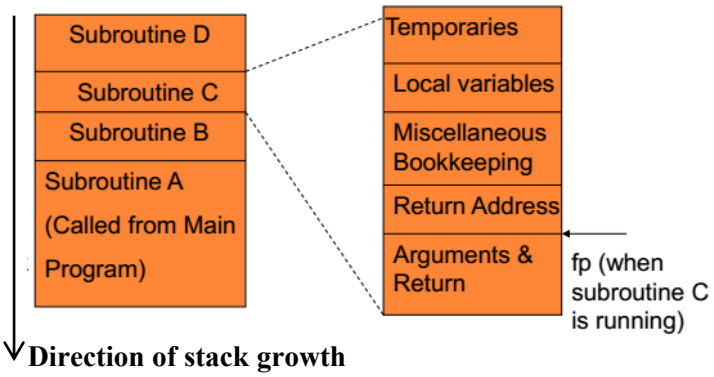
Stack based storage management is based on stack data structure that is LIFO. It is needed when language permits the recursion. The following are some important points:

- First allocated- Last freed
- Simple storage recovery, compaction, and reuse
- In stack based allocation object are allocates in Last-in, first-out.

It could be useful in language without recursion as it could save space. Each subroutine invocation creates a frame or Activation record which consist of:

- Arguments
- Return Address
- Local variables
- Temporaries
- Bookkeeping information

Stack maintained by: Calling sequence (as under)



Heap Storage Management

It is based on the heap data structure. Heap is a block of storage within which data are allocated/freed in an arbitrary manner. In this the problem of allocation, recovery, compaction and reuse may be complicated. Heap storage management can be of two types:

1. **Fixed Size Elements:** This is quite simple. Compaction is not a problem as all the elements are of same size.
2. **Variable Size Elements:** This storage technique is programmer control allocation more difficult than the fixed size elements. The major difficulties with variable size element are reuse of recover space.

As they are not allocated in stack, the life time of the objects allocated in the heap is not confined to the subroutine where they are created. They can be assigned to parameters and they can be returned as value. In then heap based allocation (heap) objects can be allocated/de allocated at arbitrary times. The heap is region of storage in which sub block can be allocated/de allocated.

Properties of Heap storage

1. Generally it is used as storage for object with an unrestricted lifetime
2. Maintains a free list -list of free space
3. On allocation - memory manager finds space and marks it as used
4. On de-allocation – memory manager marks space as free
5. Memory fragmentation – the memory fragmentation into small block over the lifetime of program
6. Garbage collection – coalesce fragments, possibly moving objects

The need for heap storage arises when a language permits storage to be allocated and freed at arbitrary points during program execution, as when its language allows creation, destruction, or extension. Three ways to allocation are:

1. Best fit
2. Worst-fit
3. First-fit

Explicit return by Programmer or System

In this method, when an element that has been in use becomes available for reuse, it must be explicitly identified as free and returned to the free-space list. But it is not always feasible because of the following two reasons:

a) Dangling reference : In context of heap storage management , a dynamic reference is a pointer to an element that has been returned to free space list means dangling pointer , so that the has been returned to free.

Space list means dangling pointer arise when an object is deleted or de-allocated, without modifying the value of the pointer so that the pointer still points to the memory location if the de allocated memory. As the system may reallocate the previously freed memory to another process, if the original program then dereferences the dangling pointers, unpredictable behavior may result, as the memory now may contain completely different data. This is especially the case if the program writes data to memory pointed by dangling pointers, as silent corruption of unrelated data may result, leading to subtle bugs that can be extremely difficult to find.

To construct a particular activation of a subprogram it is required to split into parts:

A static part: Also known as code segment made with the help of constant and executable code. It should be invariant during execution of a subprogram and so a single copy may be shared by all activation.

A dynamic Part: Known as activation record made with the help of parameter, function results and local data some other point like temporary storage areas, return point. The size and structure of activation record for a subprogram can be found out at translation time.

If program attempts to modify through a dangling reference a structure that has already been freed, that contains of an element on the free-space list may be modified inadvertently. If this modification overwrites the pointer linking the element to the next free-space-list element, the entire reminder of the free-space list may become defective.

Event worse, a letter attempt by the storage allocator to use the pointer in the overwritten element leads to completely unpredictable results.

b) Garbage: In opposite to dangling references discussed above, if the last access path to storage is without the structure itself being destroyed and the storage recovered, then the structure becomes garbage. A garbage element is one that is available for reuse but not on the free-space list, and thus it has become inaccessible.

Reference Count: The most straightforward way to recognize garbage and make its space reusable for new cells is to use reference count. We augment each heap cell with a count field that records that total number of pointers in the system that point to the cell. When an element is initially allocated from the free-space list, its reference count is set to 1. Each time a new pointer to the element is created, its reference count is increased by 1. Each time a pointer is destroyed, the reference is counter decreased by 1. If the reference count ever goes to 0, we can reuse the cell by placing it on a free list.

Advantages

1. Conceptually simple
2. Immediate reclamation of storage

Disadvantages

1. Extra space for storing reference counter.
2. Extra time(every pointer assignment has to Check change/check count)
3. Cost of maintaining reference counter
4. Decrease in execution efficiency because testing incrementing and decrementing occurs continuously throughout execution.

Garbage collection

Data that cannot be referenced is generally known as garbage. When the entire access path to a data object is destroyed but the data object continues to exist, the data object is said to be garbage. Garbage collection is the reclamation of chunks of storage holding objects that can no longer be accessed by a program. Many high-level programming languages remove the burden of manual memory management from the programmer by offering automatic garbage collection, which de-allocates unreachable data. Garbage collection dates back to the initial implementation of Lisp in 1958. Other significant languages that offer garbage collection include Java, Perl, ML, Modula-3, Prolog and Smalltalk.

Because garbage collection is done only rarely (when the free-space list is exhausted) it is allowable for the procedure to be fairly costly. Two stages are involved:

1. Mark: In the first stage each element in the heap which is active, i.e. which is part of an accessible data structure, must be marked. Each element must contain a garbage collection bit of each active element “OFF”.

2. Sweep: Once the marking algorithm has marked active element, all those remaining, whose garbage – collection bit is “ON” are garbage and may be returned to the free-

space list. A simple sequence can of the heap is sufficient the garbage collection bit of each element is checked as it is encountered in the scan. If the element is “OFF”, it is passed over. If the element is “ON”, it is linked into the free space list. All garbage collection bits are reset to “ON” during the scan.

Advantages

Garbage collection frees the programmer from manually dealing with memory allocation and de-allocation. As a result, certain categories of bugs are eliminated or substantially reduced.

Dangling pointer bugs, which occur when a piece of memory is freed while there are still pointers to it, and one of those pointers is used.

Double Free bugs, which occur when a program attempts to free a region of memory that is already free. Certain kinds of memory leaks, in which a program fails to free memory that is no longer reference by any variables, leading overtime memory exhaustion.

Disadvantage:

A potential disadvantage of a garbage collected heap is that it adds an overhead that can affect program performance. This activity likely require more CPU time then would have been required if the program explicitly freed unnecessary memory. In addition, programmer in a garbage collected environment has less control over the scheduling of CPU time devoted to freeing objects that are no longer needed.

Garbage Collection Algorithms

1. Reference counting
    a. Deferred reference counting
2. Trace-based garbage collector
    a. Basic Mark- and-Sweep
    b. Baker's Mark- and-Sweep
    c. Basic Mark-and- Compact
    d. Cheney's Copying Collector
3. Short-Pause garbage collection
    a. Incremental collection
    b. Partial collection
    c. Generational garbage collection
    d. Train algorithm