Memory management

Memory management is a form of resource management applied to computer memory. The essential requirement of memory management is to provide ways to dynamically allocate portions of memory to programs at their request, and free it for reuse when no longer needed. This is critical to any advanced computer system where more than a single process might be underway at any time.[1]

Several methods have been devised that increase the effectiveness of memory management. Virtual memory systems separate the memory addresses used by a process from actual physical addresses, allowing separation of processes and increasing the size of the virtual address space beyond the available amount of RAM using paging or swapping to secondary storage. The quality of the virtual memory manager can have an extensive effect on overall system performance.

Contents

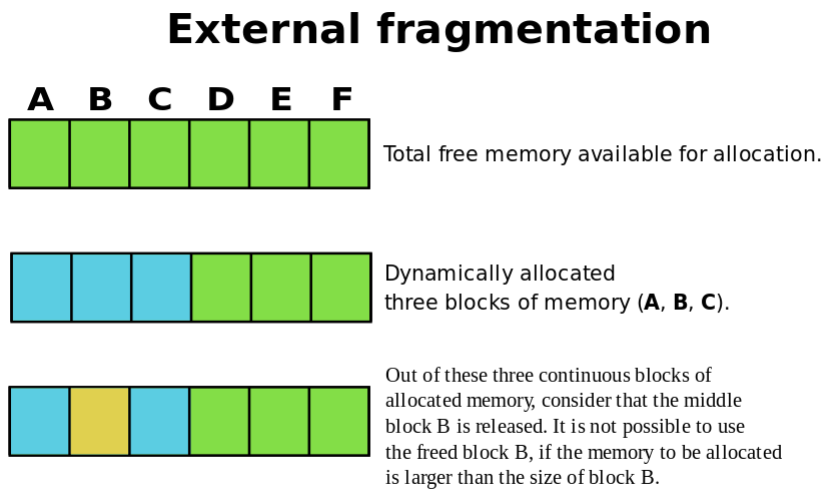
Details

Application-level memory management is generally categorized as either automatic memory management, usually involving garbage collection, or manual memory management.

Dynamic memory allocation

The task of fulfilling an allocation request consists of locating a block of unused memory of sufficient size. Memory requests are satisfied by allocating portions from a large pool of memory called the *heap* or *free store*.[a] At any given time, some parts of the heap are in use, while some are "free" (unused) and thus available for future allocations.

Several issues complicate the implementation, such as external fragmentation, which arises when there are many small gaps between allocated memory blocks, which invalidates their use for an allocation request. The allocator's metadata can also inflate the size of (individually) small allocations. This is often managed by chunking. The memory management system must track outstanding allocations to ensure that they do not overlap and that no memory is ever "lost" (i.e. that there be no "memory leak").



An example of external fragmentation

Efficiency

The specific dynamic memory allocation algorithm implemented can impact performance significantly. A study conducted in 1994 by Digital Equipment Corporation illustrates the overheads involved for a variety of allocators. The lowest average instruction path length required to allocate a single memory slot was 52 (as measured with an instruction level profiler on a variety of software).[2]

Implementations

Since the precise location of the allocation is not known in advance, the memory is accessed indirectly, usually through a pointer reference. The specific algorithm used to organize the memory area and allocate and deallocate chunks is interlinked with the kernel, and may use any of the following methods:

Fixed-size blocks allocation

Fixed-size blocks allocation, also called memory pool allocation, uses a free list of fixed-size blocks of memory (often all of the same size). This works well for simple embedded systems where no large objects need to be allocated, but suffers from fragmentation, especially with long memory addresses. However, due to the significantly reduced overhead this method can substantially improve performance for objects that need frequent allocation / de-allocation and is often used in video games.

Buddy blocks

In this system, memory is allocated into several pools of memory instead of just one, where each pool represents blocks of memory of a certain power of two in size, or blocks of some other convenient size progression. All blocks of a particular size are kept in a sorted linked list or tree and all new blocks that are formed during allocation are added to their respective memory pools for later use. If a smaller size is requested than is available, the smallest available size is selected and split. One of the resulting parts is selected, and the process repeats until the request is complete. When a block is allocated, the allocator will

start with the smallest sufficiently large block to avoid needlessly breaking blocks. When a block is freed, it is compared to its buddy. If they are both free, they are combined and placed in the correspondingly larger-sized buddy-block list.

Slab allocation

Stack allocation

Automatic variables

In many programming language implementations, all variables declared within a procedure (subroutine, or function) are local to that function; the runtime environment for the program automatically allocates memory for these variables on program execution entry to the procedure, and automatically releases that memory when the procedure is exited. Special declarations may allow local variables to retain values between invocations of the procedure, or may allow local variables to be accessed by other procedures. The automatic allocation of local variables makes recursion possible, to a depth limited by available memory.

Garbage collection

Garbage collection is a strategy for automatically detecting memory allocated to objects that are no longer usable in a program, and returning that allocated memory to a pool of free memory locations. This method is in contrast to "manual" memory management where a programmer explicitly codes memory requests and memory releases in the program. While automatic garbage has the advantages of reducing programmer workload and preventing certain kinds of memory allocation bugs, garbage collection does require memory resources of its own, and can compete with the application program for processor time.

Systems with virtual memory

Virtual memory is a method of decoupling the memory organization from the physical hardware. The applications operate on memory via *virtual addresses*. Each attempt by the application to access a particular virtual memory address results in the virtual memory address being translated to an actual *physical address*. In this way the addition of virtual memory enables granular control over memory systems and methods of access.

In virtual memory systems the operating system limits how a process can access the memory. This feature, called memory protection, can be used to disallow a process to read or write to memory that is not allocated to it, preventing malicious or malfunctioning code in one program from interfering with the operation of another.

Even though the memory allocated for specific processes is normally isolated, processes sometimes need to be able to share information. Shared memory is one of the fastest techniques for inter-process communication.

Memory is usually classified by access rate into primary storage and secondary storage. Memory management systems, among other operations, also handle the moving of information between these two levels of memory.

See also

- Dynamic array
- Out of memory

Notes

a. Not to be confused with the unrelated heap data structure.

Notes

1. Gibson, Steve (August 15, 1988). "Tech Talk: Placing the IBM/Microsoft XMS Spec Into Perspective" (https://books.google.com/books?id=ZzoEAAAAMBAJ&pg=PA34). *InfoWorld*.
2. Detlefs, D.; Dosser, A.; Zorn, B. (June 1994). "Memory allocation costs in large C and C++ programs" (http://www.eecs.northwestern.edu/~robby/uc-courses/15400-2008-spring/spe895.pdf) (PDF). *Software: Practice and Experience*. **24** (6): 527–542. CiteSeerX 10.1.1.30.3073 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.30.3073). doi:10.1002/spe.4380240602 (https://doi.org/10.1002%2Fspe.4380240602).

References

- Donald Knuth. *Fundamental Algorithms*, Third Edition. Addison-Wesley, 1997. ISBN 0-201-89683-4. Section 2.5: Dynamic Storage Allocation, pp. 435–456.
- Simple Memory Allocation Algorithms (http://buzzan.tistory.com/m/post/view/id/428)Archived (https://web.archive.org/web/20160305050619/http://buzzan.tistory.com/m/post/428) 5 March 2016 at the Wayback Machine. (originally published on OSDEV Community)
- Wilson, P. R.; Johnstone, M. S.; Neely, M.; Boles, D. (1995). "Dynamic storage allocation: A survey and critical review". *Memory Management*. Lecture Notes in Computer Science. **986**. pp. 1–116. CiteSeerX 10.1.1.47.275 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.47.275). doi:10.1007/3-540-60368-9_19 (https://doi.org/10.1007%2F3-540-60368-9_19). ISBN 978-3-540-60368-9.
- Berger, E. D.; Zorn, B. G.; McKinley, K. S. (June 2001). "Composing High-Performance Memory Allocators". *Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation* (http://www.cs.umass.edu/%7Eemery/pubs/berger-pldi2001.pdf) (PDF). pp. 114–124. CiteSeerX 10.1.1.1.2112 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.1.2112). doi:10.1145/378795.378821 (https://doi.org/10.1145%2F378795.378821). ISBN 1-58113-414-2.
- Berger, E. D.; Zorn, B. G.; McKinley, K. S. (November 2002). "Reconsidering Custom Memory Allocation". *Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (http://people.cs.umass.edu/~emery/pubs/berger-oopsla2002.pdf) (PDF). pp. 1–12. CiteSeerX 10.1.1.119.5298 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.119.5298). doi:10.1145/582419.582421 (https://doi.org/10.1145%2F582419.582421). ISBN 1-58113-471-1.

Further reading

- Wilson, Paul R.; Johnstone, Mark S.; Neely, Michael; Boles, David (September 28–29, 1995), *Dynamic Storage Allocation: A Survey and Critical Review* (http://www.cs.northwestern.edu/~pdinda/icsclass/doc/dsa.pdf) (PDF), Austin, Texas: Department of Computer Sciences University of Texas, retrieved 2017-06-03

External links

- "Generic Memory Manager" C++ library (http://memory-mgr.sourceforge.net/)
- Sample bit-mapped arena memory allocator in C (https://code.google.com/p/arena-memory-allocation/downloads/list)
- TLSF: a constant time allocator for real-time systems (http://www.gii.upv.es/tlsf/)
- Slides on Dynamic memory allocation (https://users.cs.jmu.edu/bernstdh/web/common/lectures

/slides_cpp_dynamic-memory.php)
- Inside A Storage Allocator (http://www.flounder.com/inside_storage_allocation.htm)
- The Memory Management Reference (http://www.memorymanagement.org/)
    - The Memory Management Reference, Beginner's Guide Allocation (http://www.memorymanagement.org/articles/alloc.html)
- Linux Memory Management (http://linux-mm.org/)
- Memory Management For System Programmers (http://www.enderunix.org/docs/memory.pdf)
- VMem - general malloc/free replacement. Fast thread safe C++ allocator (http://www.puredevsoftware.com/)
- A small old site dedicated to memory management (http://www.memorymanagement.org/)
- Dynamic Memory in IEC61508 Systems (https://www.embedded-office.com/en/blog/dynamic-memory-iec-61508.html)
