Operating Systems

Lecture #2b: Early Paradigms of Memory Management II

Written by David Goodwin
based on the lecture series of Dr. Dayou Li
and the book *Understanding Operating Systems* 4^{th} *ed.*
by *I.M.Flynn and A.McIver McHoes* (2006)

Department of Computer Science and Technology,
University of Bedfordshire.

Operating Systems, 2013

11^{th} February 2013

Outline

1. Relocatable Dynamic Partitions
2. Example - assembly language
3. Bounds & Relocation Registers
4. Key Terms

Relocatable Dynamic Partitions

Relocatable Dynamic Partitions

- Fixed and dynamic memory allocation schemes share unacceptable fragmentation characteristics
- This had to be resolved before the number of jobs waiting became excessively large
- Also, these bacme a growing need to use all slivers of memory often left over
- The solution to both problems was to use **relocatable dynamic partitions**
    - The memory manager relocates programs to gather together all of the empty blocks
    - Compacts them to make one block of emory large enough to accommodate jobs waiting

Compaction

- The **compaction** of memory (also called garbage collection, or defragmentation) is performed by the OS to reclaim fragmented sections of the memory space
- This can be viewed as a disadvantage - it is an overhead process i.e. while compaction is taking place, everything else must wait.
- Compaction is not an easy task
  - First every program in the memory must be relocated so they are contigious
  - Second, every address, and every reference to an address, within each program must be adjusted to take into account the program's new location in the memory
  - Third, all other values than addresses within the program must be left alone

Example - assembly language

Relocation example

- Here we show an example of an assembly language program instruction. The instruction to add the integer 1 to I is coded as:

  ADDI I, 1
- However, after it has been translated into actual code it could look like this:

  000007 271 01 0 00 000001
- It is not immediately obvious which are addresses and which are instructions or data values

Relocation example

- The address is the number on the right (000007)
- The instruction code is next (271)
- and the data value is on the right (000001)

Relocation example

- OS can tell the function of each group of digits by its location in the line and the operation code
- However, if the program is to be moved to another place in the memory, each address must be identified (or flagged)
- The amount of memory displacement must also be added or subtracted to all of the original addresses in the program
    - This is particularly important in loop, decision, or branching sequences, as well as data references
    - If every address were not adjusted by the same value, the program could:
        - branch to the wrong section of the program
        - branch to a section of another program
        - reference the wrong data

Questions

- The discussion of compaction raises three questions:
  1. What goes on behind the scenes when relocation and compaction take place?
  2. What keeps track of how far each job has moved from its original storage area?
  3. What lists have to be updated?
- The answer to the last question is simply the free lists and the busy lists
  - The free list is changed to show the partition for the new block of memory
  - The busy list is changed to show the new locations for all jobs already in process, now relocated
- To answer the other two questions we must learn more of the registers, specifically the **bounds register** and the **relocation register**

Bounds & Relocation Registers

Bounds Register & Relocation Register

Bounds Register is used to store the highest location in memory accessible by each program. This ensures that a program will not try to access memory locations that do not belong to it i.e. those *out of bounds*

Relocation Register contains the value that must be added to each memory address referenced in the program ($= 0$ if the program is not relocated)

Compaction/Relocation

- In effect, by compacting and relocating, the memory manager optimizes the use of memory and improves throughput (a measure of system performance)
- However, more overhead is incurred than with previous schemes.
- The crucial factor is the timing of compaction; when and how often should it be done? There are three options:
  1. when a certain percentage of memory becomes busy (with the disadvantage that an unnecessary overhead would be incurred if no jobs are waiting)
  2. only when there are jobs waiting (may slow down the system as constant checking of the entry queue would be needed)
  3. after a certain amount of time has elapsed (if the time is too small then more time will be spent compacting than processing, too large and jobs will congregate in the waiting queue losing the benifits of compaction)

Key Terms

Key terms

address a number that designates a particular memory location.

best-fit memory allocation a scheme that considers all free blocks and selects one with least waste.

bounds register a register used to store the highest location in memory legally accessible by each job.

compaction the process of collecting fragments of available memory space into contigious blocks by moving programs and data in memory (also called garbage collection of defragmentation).

deallocation the process of freeing an allocated resource, whether memory space, a device, a file, or a CPU.

external fragmentation a situation in which the dynamic allocation of memory creates unusable fragments of free memory between blocks of busy, or allocated memory.

first-fit memory allocation a scheme that searches from the beginning of the free block list and selects the first block of memory large enough to fulfill the request.

fixed partitions a scheme in which main memory is sectioned off, with portions assigned to each user.

Key terms

internal fragmentation a situation in which a fixed memory partition is only partially used by the program; the remaining space within the partition is unavailable to any other job and is therefore wasted.

kilobyte (K) a unit of memory or storage space equal to 1,024 bytes.

null entry an empty entry in a list.

relocatable dynamic partitions a memory allocation scheme in which the system relocates programs in memory to gather together all of the empty blocks and compact them to make one block.

relocation

1. the process of moving a program from one area of memory to another.
2. the process of adjusting address references in a program to allow the program to execute correctly when loaded into different sections of memory.

relocation register a register that contains the value that must be added to each address referenced in the program so that it will be able to access the correct memory addresses after relocation.

static partitions another term for *fixed partitions*