Memory Manager

Madiba Hudson-Quansah

Contents

Chapter 1

Early Memory Management Systems

1.1 Single-User Contiguous Scheme

- Entire program is loaded into memory
- Entirely contiguous allocation of memory space
- Jobs are processed sequentially
- The memory manager performs minimal work
  - Evaluates incoming process size, loading jobs if small enough to fit in memory
  - Monitors occupied memory space and clears entire memory space when a job is completed

In this scheme before a job can be executed it must be loaded in its entirely into memory and is allocated as much contiguous memory space in memory as it requires. If the program is too lage to fit into the available memory space it cannot be begin execution.

Single-user systems in a non-networked environment allocated to each user, access to all available main memory for each job. To allocate memory the memory manager performs the following steps:

1. Evaluate the incoming job to see if it small enough to fit into the available memory space. If it is load it into memory, else reject it and evaluate the next incoming process. A rejected job is never reconsidered as it can never fit into the available memory space.
2. Monitors the occupied memory space. When the resident job ends its execution and no longer needs to be in memory, indicate that the entire amount of main memory space is now available and return to step 1.

1.1.1 Advantages

- Simple to implement

1.1.2 Disadvantages

- Multiprogramming and networking is not possible, as only one job can be in memory at a time
- Not cost effective, as memory is often idle

1.2 Fixed / Static Partition Scheme

- Memory is divided into fixed number of partitions, where each partition handles one job and reconfiguration requires system shutdown
- This partitioning scheme requires protecting each job's memory space, and matching jobs sizes with partition sizes

- The memory manager allocates memory space to jobs with job information stored in a table
  - Multiprogramming is possible
  - Uses the first available partition with required size method for allocating memory
  - Requires contiguous loading of entire program
  - To work well all jobs should have similar size and memory size known in advance
  - Arbitrary partition sizes can lead to internal fragmentation, i.e. wasted space within a partition

Definition 1.2.1: Internal Fragmentation

Unused space inside a partition. Less than complete use of memory space within a partition

The first attempt to create a scheme that allows multiprogramming. The memory is divided into a fixed number of partitions, where the entirety of each partition is assigned to a single job, this allows multiple jobs to execute at the same time. To allocate memory the memory manager performs the following steps assuming there are two partitions but this generalizes to any number of partitions:

1. Check the incoming job's memory requirements. If it's greater then the size of the largest partition, reject the job and go to the next waiting job else go to step 2.
2. Check the job size against the size of the first available partition. If the job is small enough to fit, see if the partition is free. If it is, load the job into that partition else go to step 3.
3. Check the job size against the size of the second available partition. If the job is small enough to fit, check to see if that partition is free. If it is available, load the incoming job into that partition else go to step 4.
4. No partition is available now, so place the incoming job into the waiting queue for loading at a later time.

In order to allocate memory spaces to jobs the memory manager must have a table showing each partition's size, address, access restrictions, and its current status (free or busy). For example:

Partition Size	Memory Size	Access	Partition Status
100K	200K	Job 1	Busy
50K	300K		Free

1.2.1 Advantages

- More flexible than the single-user scheme as it allows multiple jobs to execute concurrently

1.2.2 Disadvantages

- Requires the entire program to be loaded contiguously into memory

1.3 Dynamic Partition Scheme

- Memory is partitioned dynamically as jobs arrive, i.e. a partition conforms to the size of the job
- Jobs are allocated on a first come, first served basis
- After the first partition sizing and allocation, all subsequent jobs are allocated using those partitions that are free and large enough to hold the job

Definition 1.3.1: External Fragmentation

Unused space between partitions. Less than complete use of memory space between partitions

Memory is rationed to an incoming jobs in one contiguous block, with each job given the exact amount of memory it requires. Works well when the first jobs are loaded and partitions form based on the sizes of those first jobs, but when the next jobs arrive, which are not the same size as those just deallocated, they are allocated space in the available partition spaces on a priority basis. Jobs allocated in memory are said to be in a partition exactly the size of the job. This means that there can only be unused space between partitions, i.e. internal fragmentation is not possible here.

1.3.1 Advantages

- Reduces internal fragmentation, as partitions are sized to fit jobs exactly
- More flexible than fixed-partition scheme, as partitions are created dynamically

1.3.2 Disadvantages

- Full memory utilization only occurs when the first jobs are loaded or if a job exactly fits a free partition
- External fragmentation can occur between partitions

1.4 First-Fit Allocation

Definition 1.4.1: First-Fit Allocation

Free and Busy lists are organized by memory locations, from low to high order memory addresses

- Jobs are assigned to the first available partition large enough to hold it
- Fast, as it searches from the beginning of memory and stops when a large enough partition is found

For both fixed and dynamic partition schemes, the operating system must keep track of each memory location's status, i.e. free or busy. This is done using lists that track memory partitions and their corresponding memory locations. These are the called the Free and Busy lists. For the first-fit allocation method both lists are organized by memory locations, from low to high order memory addresses, i.e. from 0 (if not reserved) to the highest memory address.

When a job comes in the memory manager compares jobs sizes to the free list, allocating the first partition that is large enough to hold the job. If the entire list is searched and finds no memory block large enough to hold the job, the job is placed into a waiting queue, and the memory manager fetches the next job in the queue.

1.4.1 Advantages

- Faster allocation

1.4.2 Disadvantages

- Can lead to many small unusable partitions at the beginning of memory

1.5 Best-Fit

Definition 1.5.1: Best-Fit

Free and Busy lists are organized by partition size, from smallest to largest

- Jobs are assigned to the smallest available partition large enough to hold it
- More efficient use of memory, as it searches the entire list to find the smallest

For the best-fit allocation method both lists are organized by partition size, from smallest to largest. When a job comes in the memory manager compares jobs sizes to the free list, allocating the smallest partition that is large enough to hold the job. If the entire list is searched and finds no memory block large enough to hold the job, the job is placed into a waiting queue, and the memory manager fetches the next job in the queue.

1.5.1 Advantages

- Best use of memory space

1.5.2 Disadvantages

- Slower allocation, as it searches the entire free list

1.6 Deallocation

Definition 1.6.1: Block

A contiguous region of memory (a contiguous range of addresses) treated as a unit by the allocator; it can be either a free hole or an allocated partition.

Definition 1.6.2: Deallocation

Releasing allocated memory space

1.6.1 Fixed and Dynamic Partition Deallocation

For a fixed-partition system deallocation is trivial as partition sizes are fixed so the partition's busy flag is set to free.

For a dynamic-partition system, the goal of deallocation is reduce external fragmentation, there are three dynamic partition system cases depending on the location of the to-be-freed block:

1. Adjacent to another free block
2. Between two free blocks
3. Isolated from other free blocks

1.6.2 Joining Two Adjacent Free Blocks

The block to be freed is adjacent to another free block. In this case the two blocks are joined to form a larger free block, with the new block's beginning address being the smallest beginning address. For example this free list:

Beginning Address	Memory Block Size	Status
7560	20	Free
(7600)	(500)	(Busy)
*8100	100	Free
9000	200	Free

Where the block at address 7600 is to be freed. The resulting free list is:

Beginning Address	Memory Block Size	Status
7560	20	Free
7600	510	Free
9000	200	Free

1.6.3 Joining Three Adjacent Free Blocks

The block to be freed is between two other free blocks. In this case the three blocks are joined to form a larger free block, with the new block's beginning address being the smallest beginning address. For example this free list:

Beginning Address	Memory Block Size	Status
7560	20	Free
*7600	500	Free
(8100)	(100)	(Busy)
*8200	200	Free
10000	50	Free

Where the block at address 8100 is to be freed. The resulting free list is:

Beginning Address	Memory Block Size	Status
7560	20	Free
7600	800	Free
*		(null entry)
10000	50	Free

We add a null entry to prevent the shifting all the entries in the free list at the expense of memory.

1.6.4 Isolated Free Block

The block to be freed is isolated from other free blocks, i.e. it is not adjacent to any other free block. In this case the block is added to the free list at the appropriate location i.e. below the block with the next lowest beginning address. For example this free list and busy list:

Table 1.1: Free List

Beginning Address	Memory Block Size	Status
1000	100	Free
*		(null entry)
2000	200	Free

Table 1.2: Busy List

Beginning Address	Memory Block Size	Status
1100	300	Busy
1400	250	Busy
1650	300	Busy

Where the block at address 8400 is to be freed. The resulting free and busy lists are:

Table 1.3: Free List

Beginning Address	Memory Block Size	Status
1000	100	Free
1400	250	Free
2000	200	Free

Table 1.4: Busy List

Beginning Address	Memory Block Size	Status
1100	300	Busy
*		(null entry)
1650	300	Busy

Chapter 2

Memory Management Includes Virtual Memory

2.1 Paged Memory Allocation

- Incoming jobs are divided into pages of equal size
- Pages are loaded into page frames in main memory
- In the best case pages, sectors and page frames are the same size, with sizes determined by a disk's sector size
- The memory manager prior to program execution:
  - Determines the number of pages in a program
  - Locates enough empty page frames in main memory
  - Loads all program pages into page frames
- Programs can be stored in non-contiguous page frames
- Internal fragmentation can occur if a page is not completely filled and only happens on the job's last page

Definition 2.1.1: Sector

A fixed-length contiguous block of data on a disk

Definition 2.1.2: Page

An equal sized division of a job.

Definition 2.1.3: Page Frame / Frame

A fixed sized division of main memory that holds a page.

Paged memory allocation, is based on the idea that jobs are divided into units of equal size called pages. These pages are loaded into memory occupying page frames. The size of a page frame is determined by the size of a disk's sectors, as pages are often read from disk into memory. Pages can be stored non-contiguously in main memory. The memory manager prior to program execution performs the following steps:

- Determines the number of pages in a job
- Locates enough empty page frames in main memory
- Loading all of the job's pages into page frames

Example 2.1.1

A job of size 350 bytes is to be loaded into memory using paged memory allocation, where the page size is 100 bytes. The job is divided into 4 pages, each 100 bytes except the last page which is 50 bytes. There is internal fragmentation of 50 bytes in the last page of the job when loaded into memory.

There are three tables used to track pages:

- Job Table (JT) - Stores information for each active job
  - Job Size
  - Memory location of the job's PMT
- Page Map Table (PMT) - Stores information for each page in a job, every active job has a PMT
  - Page number starting from 0
  - Memory address of the page frame where the page is loaded
- Memory Map Table (MMT) - Stores information for each page frame in main memory
  - The locations of the page of which this frame is holding
  - Free/Busy status of each frame

2.1.1 Page Displacement

Definition 2.1.4: Line / Byte / Word

The smallest unit of data that can be transferred between main memory and the CPU. A page frame is made up of multiple lines. Also called a word or byte depending on the architecture.

Definition 2.1.5: Page Displacement / Offset

The distance of a line from the beginning of a page. It is a relative factor used to locate a certain line within its page frame.

To determine the page number and displacement of a line we:

1. Divide the job space address by the page size
2. The page number is the integer quotient
3. The displacement is the remainder

I.e.:

$$\text{Page Number} = \left\lfloor \frac{\text{Job Space Address}}{\text{Page Size}} \right\rfloor$$
$$\text{Displacement} = \text{Job Space Address} \mod \text{Page Size}$$

Example 2.1.2

Question 1

With a page size of 4096 bytes find the page and displacement of line 7149

Solution:

$$\begin{aligned}\text{Page Number} &= 7149 \div 4096 \\ &= 1 \\ \text{Displacement} &= 7149 \bmod 4096 \\ &= 3053\end{aligned}$$

To determine the exact location of an instruction or data item in main memory we:

1. Determine the page number/displacement of the line
2. Refer to the job's PMT to determine the page frame containing the required page
3. Obtain the beginning address of the page frame
4. Multiply the page frame number by the page size
5. Add the displacement to the starting address of the page frame

This is also called address resolution / translation converting a logical address (job space address) to a physical address (main memory address).

2.1.2 Advantages

- Pages don't have to be loaded contiguously
- Efficient use of memory, as jobs are loaded into any available page frame
- Compaction and relocation are not required

2.1.3 Disadvantages

- Internal fragmentation can occur on the last page of a job
- Additional overhead is required for address translation
- Requires entire job to be loaded into memory before execution can begin

2.2 Demand Paging Memory Allocation

- Loads only a part of the program into memory
- Exploits programming techniques where only a small part of the program is needed at any one time
- Simulates a larger amount of memory than is physically available, i.e. Virtual Memory
- Modifies PMT to include:
  - If the page is already in memory
  - Are the page contents modified
  - Has the page been referenced recently
  - Page Frame Number
- Swapping / Paging is used to move pages between main memory and secondary memory
  - When a page is needed that is not in memory a page fault occurs
  - A resident memory page is freed based on a policy
  - If the resident page has been modified it is copied to secondary memory

  - The new page is copied into the freed page frame

Definition 2.2.1: Page Fault

The event that occurs when a program tries to access a page that is not currently in main memory, causing a page interrupt

Definition 2.2.2: Page Interrupt

An interrupt generated when a page fault occurs, causing the operating system to fetch the required page from secondary memory into main memory

Definition 2.2.3: Swapping / Paging

The process of moving pages between main memory and secondary memory

Definition 2.2.4: Thrashing

Excessive swapping of pages between main memory and secondary memory, leading to a significant decrease in system performance. Mainly occurs when:
- There is insufficient memory to hold the working set of a process, i.e. large number of jobs and limited free pages
- The operations of pages cross page boundaries frequently, i.e. a loop that spans multiple pages

Demand paging

2.2.1 Page Replacement Policies

- First-In-First-Out (FIFO)
  - The oldest page in memory is replaced
- Least Recently Used (LRU)
  - The page that has not been used for the longest time is replaced

2.2.1.1 First-In First-Out (FIFO)

- Removes the page that has been in memory the longest
- Failure rate is determined by the ratio of page interrupts to page requests
- More memory does not guarantee better performance (Belady's Anomaly)

2.2.1.2 Least Recently Used (LRU)

- Removes the page that has not been used for the longest time
- Takes advantage of the principle of locality, i.e. if a page has not been used for a long time it is unlikely to be used in the near future
- More memory guarantees better performance
- Has various implementations
  - Clock Replacement - A pointer steps through active pages' reference bits and replaces the first page with a reference bit of 0
  - Bit-Shifting - Each page has an 8-bit register, every time a page is referenced its register is shifted right by one bit and a 1 is placed in the leftmost bit. The page with the smallest value is replaced

2.2.2 Working Set

- Loads a set of related pages into memory allowing direct access without incurring a page fault
- Takes advantage of the principle of locality, i.e. programs tend to use a small set of pages intensively for a period of time
- Requires the system define the number of pages that makes up a working set and the maximum number of pages allowed in a working set.

2.2.3 Advantages

- Reduces the amount of memory required by a job
- Reduces the time required to load a job into memory
- Allows larger jobs to be run in memory

2.2.4 Disadvantages

- Requires high-speed page access

2.3 Segmented Memory Allocation

Definition 2.3.1: Segment

A logical unit of a program containing code the performs related functions, e.g. main program, subroutine, data table, etc.

- A program is divided into segments of variable length
- Each segment is loaded into a memory partition large enough to hold it
- Segments are loaded non-contiguously
- A segment table (ST) is used to track segments

2.4 Segmented/Demand Paged Memory Allocation

Each segment is divided into pages, and only the required pages of a segment are loaded into memory.