CSCI-460-Operating Systems
Lecture 02
Memory management—early systems

adiesha

August 2024

1 History about the OS development

1. Early Computing (1940s - 1950s)

   No Operating System: Users had the full control over the machine.

2. First Generation OS (1950s - 1960s): Mainframes and Batch Systems: Multi-programming (ability to run multiple programs)

3. Second Generation OS (1960s - 1970s): Multi-programming and time sharing. Allowing multiple users to interact simultaneously.

   UNIX (1969) powerful, portable, multi-user, multitasking OS. It influenced many later operating systems, including Linux.

4. Third Generation OS (1970s - 1980s) Personal Computing: The rise of personal computers (PCs) led to the development of OSs like CP/M (1974) and MS-DOS (1981).

   Development of GUI in operating systems.

5. Fourth Generation OS (1980s - 2000s) Growth of networked and distributed operating systems.

   Microkernels: A modular approach to operating system design, where the kernel is divided into smaller, more manageable components.

6. Modern OS (2000s - Present) Mobile Operating Systems: The rise of mobile computing saw the development of OSs

   Cloud and Virtualization: Cloud-based operating systems, such as Google Chrome OS and Microsoft Azure, offer on-demand computing resources and services.

   Distributed operating systems: work seamlessly across different devices.

   Real-Time and Embedded Systems: Operating systems tailored for specific devices and applications, such as automotive systems, industrial control, and medical devices.

2 Objectives and Functions

1. Convenience: An OS makes a computer more convenient to use.
2. Efficiency: An OS allows the computer system resources to be used in an efficient manner.
3. Ability to evolve: An OS should be constructed in such a way as to permit the effective development, testing, and introduction of new system functions without interfering with service.

3 Types of Operating Systems

1. Serial processing: the user interacted directly with the computer hardware; there was no OS. These computers were run from a console consisting of display lights, toggle switches, some form of input device, and a printer. Programs in machine code were loaded via the input device (e.g., a card reader). If an error halted the program, the error condition was indicated by the lights. If the program proceeded to a normal completion, the output appeared on the printer.

   problems: Scheduling and set-up time.

   Over time, various system software tools were developed to attempt to make serial processing more efficient. These include libraries of common functions, linkers, loaders, debuggers, and I/O driver routines that were available as common software for all users.

2. Batch Systems:

   (a) User no longer had direct access to the processor.

   (b) Use of a software known as a **Monitor**.

   (c) the user submits the job on cards or tape to a computer operator, who batches the jobs together sequentially and places the entire batch on an input device, for use by the monitor.

   (d) Each program is constructed to branch back to the monitor when it completes processing, at which point the monitor automatically begins loading the next program.

3. Interactive OS: real time user interaction, GUI, CLI. Ex: MS DOS
4. Real-time operating systems: Specialized operating system designed to manage hardware resources, run applications, and process data in real-time. Used in aircrafts, military applications.
5. OS for phones
6. Embedded operating systems

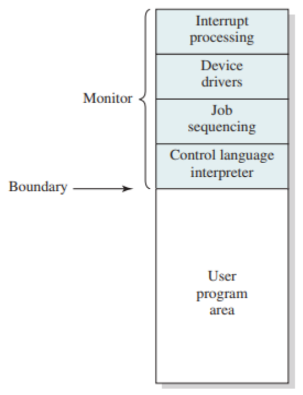

Figure 2.3 **Memory Layout for a Resident Monitor**

4 Early Memory management systems

1. In earlier systems, computers only allowed one user and one program to run at a time.
2. To run a program, program should be loaded entirely and contiguously loaded into the memory.

4.1 A simple algorithm

1. Assume that the job resides in the storage device contiguously.
2. Store the first memory location M which is free.
3. Check whether the job size is greater than the available memory. If yes stop loading.
4. Load the job into the memory.
5. set the program counter to M.

5 Fixed partitions

1. Assume the OS occupies some fixed portion of main memory, and the rest of main memory is available for use by multiple processes.
2. Idea: we partition the memory into set of fixed partitions.
3. Two choices:

   Equal size partitions.

   Unequal size partitions.

4. Any process whose size is less than or equal to the partition size can be loaded into available partition.

5. If partitions are full, and no process is ready or running state, the OS can swap a process out of any partition and load in another process.

6. Problems:

   If the program is too big for the available partition, the programmer should develop the program with the use of overlays so that only a portion of the program is in the memory.

   Main memory utilization is inefficient. (program uses the entire partition regardless of the size) – internal fragmentation

7. The number of partitions specified at system generation time limits the number of active (not suspended) processes in the system.

8. Advantages:

   Simple to implement/ less overhead

9. We can use the unequal size partitions to lessen these problems.

5.1 Placement algorithms

1. For equal sized partitions placement algorithm is trivial.

2. Algorithms: Find an empty partition that can fit the job. Then load the job to that partition. If the job is larger than the partition size, programmer must use the overlay logic or loading is rejected. all partitions are occupied with processes that are not ready to run, then one of these processes must be swapped out to make room for a new process.

3. For unequal size partitions, you have couple of choices:

   Find the smallest partition within which it will fit the job we are trying to assign. (requires queue for each partition)

   Disadvantage: We might not know the memory requirement for each process in advance.

   Advantage: process are assigned in way that minimizes the memory wastage.

   Keep one queue for all processes, find the smallest partition that fits the job and assign it. If all partitions are filled, swapping decision must be made. (Preference can be given to swapping the smallest partition that fits the incoming job.)

6 Dynamic Partitioning

1. The partitions are of variable length and number.
2. When a process is brought into main memory, it is allocated exactly as much memory as it requires and no more.
3. Leads to external fragmentation.
4. solution: compaction; OS shifts the processes so that they are contiguous and all of the free memory is together in one block.

6.1 Placement algorithms

1. Best-Fit: OS chooses the empty block that is the closest in size.
2. First-Fit: OS scans the memory from the beginning and find the first available block that is large enough for the job.
3. Next-Fit: Begins to scan the memory from the location of the last placement and chooses the next available block that is large enough.

7 Memory deallocation

1. When the partition is not used anymore, it should be released.
2. Fixed-partition: reset the free/busy status variable.
3. Dynamic partition: Need to combine free areas.

8 Relocatable Dynamic Partition

1. Fixed and Dynamic partition approaches, both introduce internal/external fragmentation.
2. For dynamic partitions, compaction is the solution.
3. How?

   Bounds register and base register is used to find the memory block of the proccess.