CS 206

Lecture 21 – Run time Systems

Runtime Systems

- Most HLL implementations require “libraries”
- Some are simple, others much more complex
  - Simple: Copy data from one location to another
  - Complex: Heap management; memory allocation
- Run-Time System/Runtime
  - set of “libraries” on which the language implementation depends for correct operation
- How do runtimes get the requisite information for correct functioning?
  - obtain information from subroutine arguments (can be easily replaced by alternative implementations) e.g. heap management
  - Maybe based on conventions (subroutine calling)
  - Compiler may provide program specific metadata e.g. garbage collection

Various Jobs of Runtime Systems

- *Garbage Collection*
  - find all the “root pointers” in the program, to identify the type of every reference and every allocated block
  - *Compacting collector*
- *Variable Numbers of Arguments*
  - declare functions that take an arbitrary number of arguments
  - In C: va_arg(my_args, arg_type)
  - Runtime needs an understanding of : which arguments are passed in which registers, and which arguments are passed on the stack
- *Exception Handling*
  - “unwind” the stack whenever control escapes the current subroutine
  - Support for allocation/deallocation of frames
- *Event Handling*
  - *“Callbacks”, handlers*
- Coroutine and Thread Implementation
- Remote Procedure Call
- Dynamic Linking

Garbage Collection

- Explicit reclamation of heap objects
  - Can be done by programmer : adds overhead, bugs
  - Alternative: have the language implementation notice when objects are no longer useful and reclaim them automatically -- *garbage collection*
    - E.g. via “delete”

- How does one find “useless” objects?
  - When no references exist
  - Reference counts : Set to 1 when creation, keep track of counts over time; destroy when reference count reaches 0

Mark and Sweep

- Collector walks through the heap, marks every block as “useless”
- With all pointers outside the heap, the collector
  - Recursively explores all linked data structures in the program
  - Marks each newly discovered block as “useful.”
- Collector again walks through the heap, moving every block that is still marked “useless” to the free list.

Garbage Collection : Other Techniques

- Stop and Copy
  - divide the heap into two regions of equal size.
    - All allocation happens in the first half
  - When this half is (nearly) full, the collector begins its exploration of reachable data structures
  - Each reachable block is copied into the second half of the heap
    - Reduces which type of fragmentation?
  - old version of the block, in the first half of the heap, is overwritten with a “useful” flag and a pointer to the new location
    - Any other pointer that refers to the same block (and is found later in the exploration) is set to point to the new location
  - Move all useful objects to second half
  - Swap notions of first and second half; carry as before

Garbage Collection : Other Techniques

- Generational Collection
  - Main observation : objects are short lived
  - Divide heap into multiple regions
  - When space runs low the collector first examines the youngest region (the “nursery”)
    - likely to have the highest proportion of garbage
  - Next older region examined only if not able to reclaim enough space.

Runtime Systems

The length and complexity of the list generally means that the compiler and the run-time system must be developed together.

- Some languages have very small run-time systems: most user level code required to execute a given source program is either generated directly by the compiler or contained in language-independent libraries.
- Other languages have extensive run-time systems. e.g. C# is heavily dependent on a run-time system defined by the Common Language Infrastructure standard which depends on data generated by the compiler

Virtual Machines

A virtual machine (VM) provides a complete programming environment

- Its application programming interface (API) provides all requirements for execution of programs that run above it
- Is term used for environments whose level of abstraction is comparable to that of a computer implemented in hardware

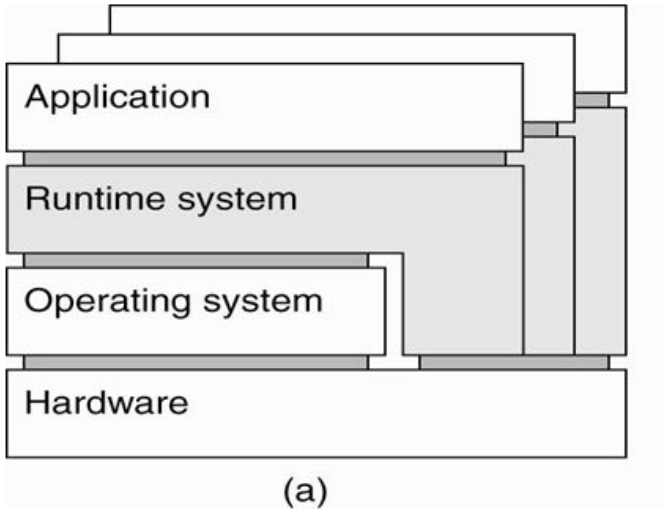
Virtual Machines

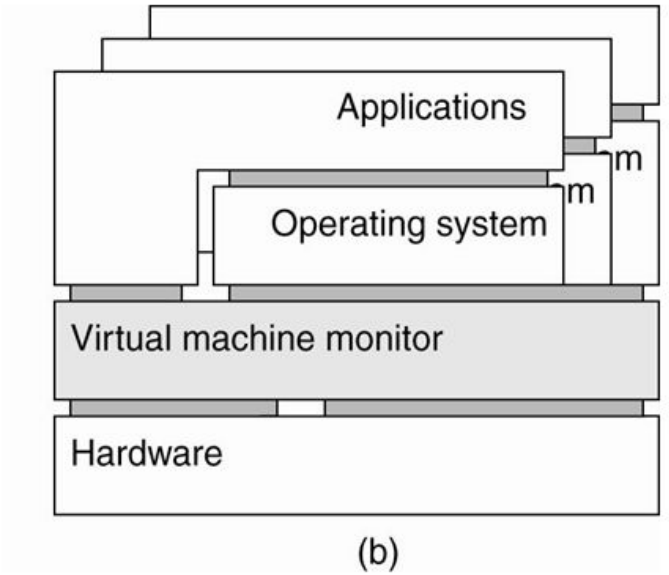
- System Virtual Machines
  - Faithfully emulates all the hardware facilities needed to run a standard OS, including both privileged and unprivileged instructions, memory-mapped I/O, virtual memory, and interrupt facilities. (sometimes referred to as virtual machine monitors (VMMs))

- Process Virtual Machines
  - Provides the environment needed by a single user-level process: the unprivileged subset of the instruction set and a library-level interface to I/O and other services.

Virtual Machines

Two Ways to Virtualize





Process Virtual Machine:
program is compiled to intermediate code, executed by a runtime system

Virtual Machine Monitor:
software layer mimics the instruction set; supports an OS and its applications

10

https://slideplayer.com/slide/4346200/