

Interprocess Communication

Jonathan Windle

University of East Anglia

J.Windle@uea.ac.uk

June 10, 2017

Overview I

- 1 Terminology
- 2 Requirements
- 3 Memory Hierarchy
 - Memory Usage
 - Memory Binding
- 4 Memory Management
 - Uniprogramming Memory Management
 - Multiprogramming Memory Addressing
- 5 Static Translation
 - Problems
- 6 Dynamic Translation
 - Alternate...
 - Diagram
- 7 Memory Segmentation
 - Fixed and Equal-Sized Memory Segments
 - Fixed and Unequal-sized Memory Segments

Overview II

- Limitations

8 Memory Protection

- **Memory:** Numerically addressable data storage
- **Address:** Number used to identify a memory location
- **Segment:** A contiguous block of memory
- **Address space:** Range of addresses used by a process
- **Memory image:** Contents of an address space
- **Primary memory:** Main memory
- **Secondary memory:** Disk storage

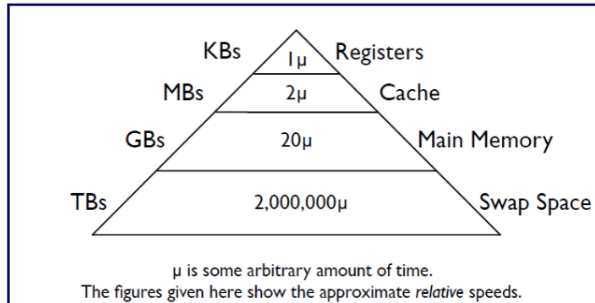
Requirements

- OS Memory manager must provide:
 - Ability to relocate a process in memory
 - Protection of memory belonging to a process
 - Ability to share memory between processes
 - Logical and efficient organisation of memory
 - Physical organisation of memory
- Programmers ideally like:
 - Unlimited amount of memory
 - Super-fast read-write access times
- In reality they get a range of memory systems, each having:
 - Different capacities
 - Different access time
 - Different cost per bit
- Although size of main memory is increasing, the faster programmers fill it.

Memory Hierarchy

- General Rule:

- Faster memory access times cost more
- Computers are designed to provide:
 - Large capacity of (cheap) slow memory
 - Smaller capacities of (expensive) fast memory
- OS must decide what data gets stored where
- Optimise overall performance



Memory Usage

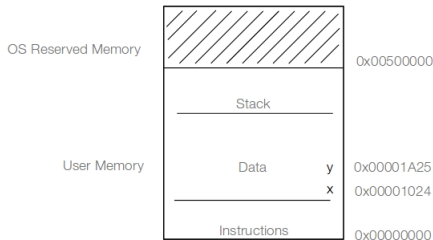
- Initially processes are loaded into secondary memory
- Some, or all of data is then copied into main memory
- Frequently used data can be copied into cache
- When processes run, variables are copied into CPU registers
- We potentially have up to four copies of the data
 - We COPY rather than MOVE data since only need to MOVE when updates occur.

Memory Binding

- Source code is written in the form of text
- The source code indirectly defines the instructions to execute
- The variables define placeholders for data to act upon
- Instructions are usually executed in sequential fashion, but there are often branches to different locations
- For example, sub-routine calls, `switch` statements, `for` loops etc.

Uniprogramming Memory Management

- Only one process can be run at a time
- Process is loaded into memory and runs
- No real need for memory management
 - Process has access to whole of main user memory
 - Virtual memory == physical memory
- Programs loaded to the first location in user memory
- Location referred to as the base address
- References to variables are offsets from base address.



Program code can use the physical address to reference variables.

Addressing is set at **compile** time.

Multiprogramming Memory Addressing

- Addressing is more tricky if there are many processes in memory
- Processes cannot share the same physical location
- Divide memory into number of segments:
 - Segment sizes could be fixed/variable or unequal/equal
- Processes must know where their variable are stored:
 - This is not known at compile time
 - Generally memory references are still relative to a base address
 - Proceses cannot share same physical locations
- Memory address used at compile time is virtual.

- All programs assume base location of 0.
- **Relocation loader** adjusts memory references to reflect actual base location when process is loaded:
 - Offset relative to (compile time) base address (i.e. 0).
 - Relocation is performed when program is loaded.
 - Linker (compilation stage) lists which references need relocation
 - References are corrected when the program is loaded.
 - References are corrected when program is loaded.
- Physical memory references set at LOAD time.

- Simple relocation strategy is Static:
 - Difficult to recover fragmented memory
- Processes cannot be moved into physical memory
- No mechanism for sharing memory
- There is no memory protection between processes
- Assumes memory requirements are known and fixed.

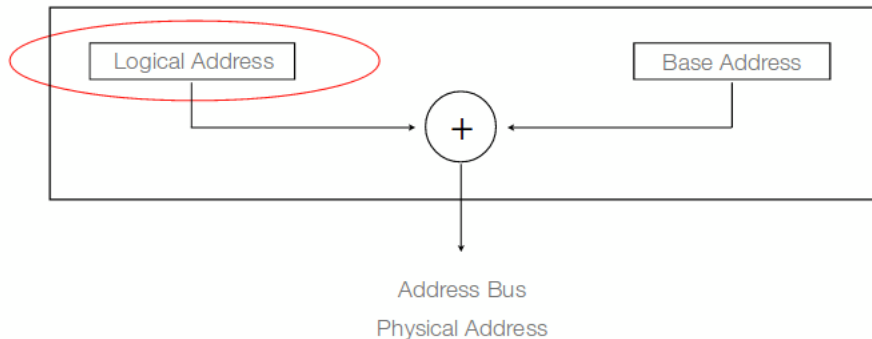
Dynamic Translation

- Add base register to store segment base address
- Use the contents of this register to offset address references within the program
- In this way, addresses used by the CPU are virtual.
- Base register translates memory addresses.
- Translation is transparent to the process/CPU
- Physical addresses are set at run time
- $\text{Base} + \text{offset}$ is computed for every read/write operation
- Offset computed by hardware
- Need a record of base address for each process.
- Cannot have a dedicated (Base) register for each process
- Instead the address is stores in the Process Control Block (PCB).

Alternate...

- To allow processes to be relocated in memory while running, the **base register** can store the base address of the current segment for the process.
- The contents of this register are then often used to offset memory references within the program
- The CPU is utilising a **virtual address**:
 - It does not use the address of physical location in memory
 - Thus, the base register translates virtual addresses to physical addresses
- The translation is transparent to process/CPU
- Addressing is set at run time.
- Memory references must be offset for every read/write operation, this must be quick and so it is done in hardware
- Every process has a different base address, but there cannot be a dedicated base register for each
- Instead the base address is stored in the Process Control Block (PCB).

Diagram

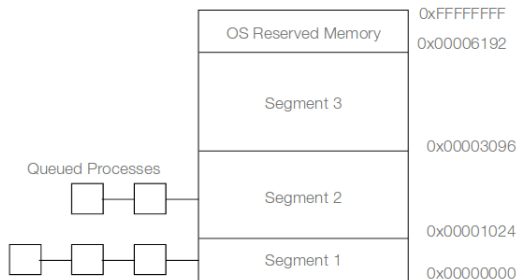


Fixed and Equal-Sized Memory Segments

- Equal segment sizes are wasteful
- Small processes are likely to fill a segment
- If segment size is large, lots of memory unused
- Only one process may occupy a segment

Fixed and Unequal-sized Memory Segments

- Create fixed-sized segments of differing size
- Assign processes to ones of best fit
- As a process completes, load next in the queue.

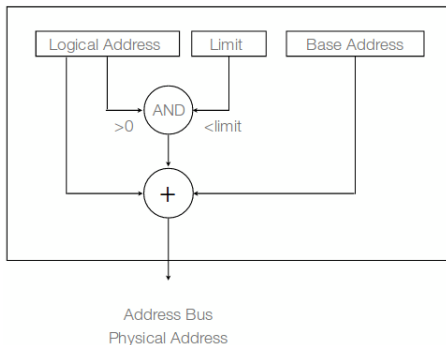


Limitations

- Fixed segment sizes leads to internal fragmentation, assumes fixed memory requirements for all processes
- Doesn't allow for more physical memory than is available
- Queued processes must wait for an appropriate sized segment
- No mechanism for protecting memory

Memory Protection

- Simple approach uses another hardware register
- Register is loaded with segment size
- Logical address is compared with contents of limit register
- If outside the allowed range, generate exception (Seg fault).



Allocating Processes

- Must select a process to fit a segment:
 - First-fit strategy
 - Best-fit strategy
 - Worst-fit strategy

- Memory comprises a range of technologies configured as a hierarchy
- Memory divided into segments, running processes
- CPU operates in virtual memory space which is mapped onto physical space
- Programs need to be bound to memory:
 - By the compiler
 - By the loader
- Later supports relocation: efficient use of main memory

The End