[06] VIRTUAL ADDRESSING

OUTLINE

- Memory Management
  - Concepts
  - Relocation, Allocation, Protection, Sharing, Logical vs Physical Organisation
- The Address Binding Problem
  - Relocation
  - Logical vs Physical Addresses
- Allocation
  - Scheduling
  - Fragmentation
  - Compaction

MEMORY MANAGEMENT

- **Memory Management**
  - **Concepts**
  - **Relocation, Allocation, Protection, Sharing, Logical vs Physical Organisation**
- The Address Binding Problem
- Allocation

CONCEPTS

In a multiprogramming system, have many processes in memory simultaneously

- Every process needs memory for:
  - Instructions ("code" or "text")
  - Static data (in program)
  - Dynamic data (heap and stack)
- In addition, operating system itself needs memory for instructions and data
  - Must share memory between OS and k processes

1. RELOCATION

- Memory typically shared among processes, so programmer cannot know address that process will occupy at runtime
- May want to swap processes into and out of memory to maximise CPU utilisation
- Silly to require a swapped-in process to always go to the same place in memory
- Processes incorporate addressing info (branches, pointers, etc.)
- OS must manage these references to make sure they are sane
- Thus need to translate logical to physical addresses

2. ALLOCATION

- This is similar to sharing below, but also related to relocation
- I.e. OS may need to choose addresses where things are placed to make linking or relocation easier

3. PROTECTION

- Protect one process from others
- May also want sophisticated `RWX` protection on small memory units
- A process should not modify its own code (...yuck...)
- Dynamically computed addresses (array subscripts) should be checked for sanity

4. SHARING

- Multiple processes executing same binary: keep only one copy
- Shipping data around between processes by passing shared data segment references
- Operating on same data means sharing locks with other processes

5. LOGICAL ORGANISATION

- Most physical memory systems are linear address spaces from 0 to `max`
- Doesn't correspond with modular structure of programs: want segments
- Modules can contain (modifiable) data, or just code
- Useful if OS can deal with modules: can be written, compiled independently
- Can give different modules diff protection, and can be shared thus easy for user to specify sharing model

6. PHYSICAL ORGANISATION

- Main memory: single linear address space, volatile, more expensive
- Secondary storage: cheap, non-volatile, can be arbitrarily structured
- One key OS function is to organise flow between main memory and the secondary store (cache?)
- Programmer may not know beforehand how much space will be available

THE ADDRESS BINDING PROBLEM

- Memory Management
- **The Address Binding Problem**
  - **Relocation**
  - **Logical vs Physical Addresses**
- Allocation

THE ADDRESS BINDING PROBLEM

Consider the following simple program:

```
int x, y;
x = 5;
y = x + 3;
```

We can imagine that this would result in some assembly code which looks something like:

```
str #5, [Rx]   ; store 5 into x
ldr R1, [Rx]   ; load value of x from memory
add R2, R1, #3 ; and add 3 to it
str R2, [Ry]   ; and store result in y
```

where the expression `[addr]` means *the contents of the memory at address addr*. Then the address binding problem is: what values do we give `Rx` and `Ry`?

Arises because we don't know where in memory our program will be loaded when we run it: e.g. if loaded at `0x1000`, then `x` and `y` might be stored at `0x2000`, `0x2004`, but if loaded at `0x5000`, then `x` and `y` might be at `0x6000`, `0x6004`

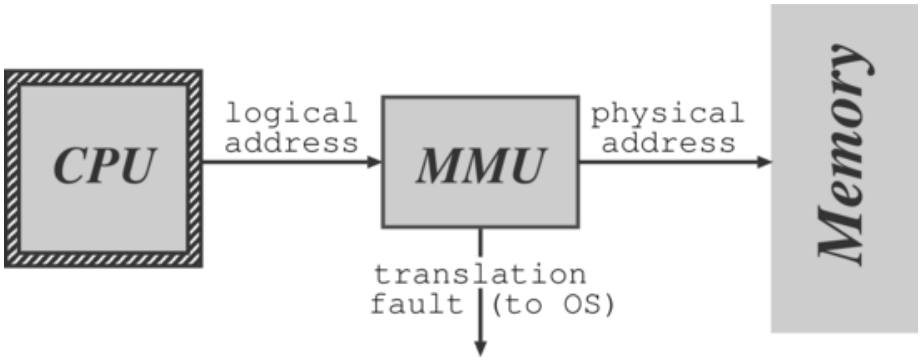
ADDRESS BINDING AND RELOCATION

Solution requires translation between *program* addresses and *real* addresses which can be done:

- At **compile time**:
  - Requires knowledge of absolute addresses, e.g. DOS `.exe` files
- At **load time**:
  - Find position in memory after loading, update code with correct addresses
  - Must be done every time program is loaded
  - Ok for embedded systems / boot-loaders
- At **run-time**:
  - Get hardware to automatically translate between program and real addresses
  - No changes at all required to program itself
  - The most popular and flexible scheme, providing we have the requisite hardware (MMU)

LOGICAL VS PHYSICAL ADDRESSES

Mapping of logical to physical addresses is done at run-time by **Memory Management Unit** (MMU)



1. Relocation register holds the value of the base address owned by the process
2. Relocation register contents are added to each memory address before it is sent to memory
3. e.g. DOS on 80x86 — 4 relocation registers, logical address is a tuple (s, o)
    - NB. Process never sees physical address — simply manipulates logical addresses
4. OS has privilege to update relocation register

ALLOCATION

- Memory Management
- The Address Binding Problem
- **Allocation**
  - **Scheduling**
  - **Fragmentation**
  - **Compaction**

CONTIGUOUS ALLOCATION

How do we support multiple virtual processors in a single address space? Where do we put processes in memory?

- OS typically must be in low memory due to location of interrupt vectors
- Easiest way is to statically divide memory into multiple fixed size partitions:
  - Bottom partition contains OS, remainder each contain exactly one process
- When a process terminates its partition becomes available to new processes.
  - e.g. OS/360 MFT
- Need to protect OS and user processes from malicious programs:
  - Use base and limit registers in MMU
  - Update values when a new process is scheduled
  - NB. Solving both relocation and protection problems at the same time!

STATIC MULTIPROGRAMMING

- Partition memory when installing OS, and allocate pieces to different job queues
- Associate jobs to a job queue according to size
- Swap job back to disk when:
  - Blocked on IO (assuming IO is slower than the backing store)
  - Time sliced: larger the job, larger the time slice
- Run job from another queue while swapping jobs
  - e.g. IBM OS/360 MVT, ICL System 4
- Problems: fragmentation, cannot grow partitions

DYNAMIC PARTITIONING

More flexibility if allow partition sizes to be dynamically chosen (e.g. OS/360 MVT):

- OS keeps track of which areas of memory are available and which are occupied
  - e.g. use one or more linked lists:
- For a new process, OS searches for a hole large enough to fit it:
  - **First fit**: stop searching list as soon as big enough hole is found
  - **Best fit**: search entire list to find "best" fitting hole
  - **Worst fit**: counterintuitively allocate largest hole (again, search entire list)

---

1. First and Best fit perform better statistically both in time and space utilisation — typically for N allocated blocks have another $0.5N$ in wasted space using first fit
2. Which is better depends on pattern of process swapping
3. Can use *buddy system* to make allocation faster
4. When process terminates its memory returns onto the free list, coalescing holes where appropriate

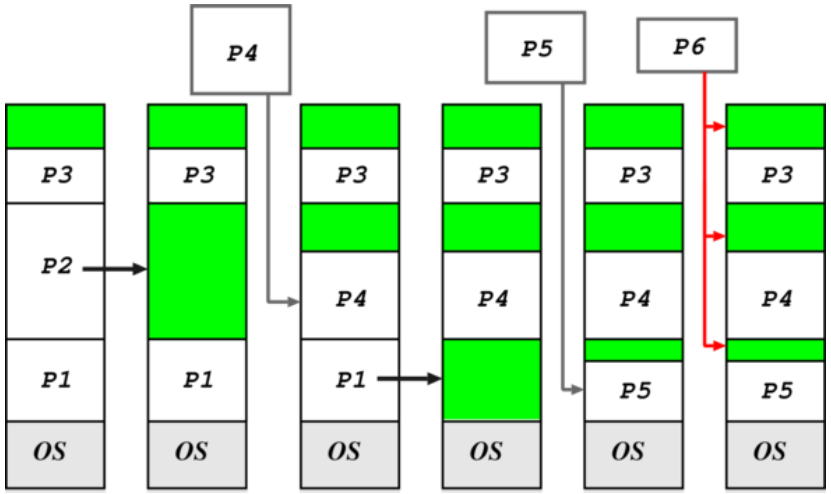
SCHEDULING EXAMPLE

Consider a machine with total of 2560kB memory, and an OS requiring 400kB

- The following jobs are in the queue:

Process	Memory	Time
P_1	600kB	10
P_2	1000kB	5
P_3	300kB	20
P_4	700kB	8
P_5	500kB	15

EXTERNAL FRAGMENTATION

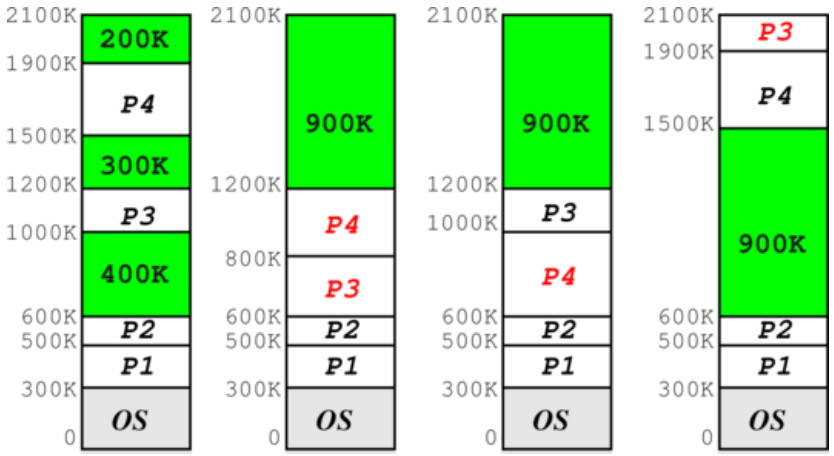


Dynamic partitioning algorithms suffer from external fragmentation: as processes are loaded they leave little fragments which may not be used. Can eventually block due to insufficient memory to swap in

External fragmentation exists when the total available memory is sufficient for a request, but is unusable as it is split into many holes

Can also have problems with tiny holes when keeping track of hole costs more memory than hole! Requires periodic compaction

COMPACTION



- Choosing optimal strategy quite tricky. Note that:
  - Require run-time relocation
  - Can be done more efficiently when process is moved into memory from a swap
  - Some machines used to have hardware support (e.g., CDC Cyber)
- Also get fragmentation in backing store, but in this case compaction not really viable

SUMMARY

- Memory Management
  - Concepts
  - Relocation, Allocation, Protection, Sharing, Logical vs Physical Organisation
- The Address Binding Problem
  - Relocation
  - Logical vs Physical Addresses
- Allocation
  - Scheduling
  - Fragmentation
  - Compaction