1B11 Operating Systems - 3

Memory Management and Protection

Prof. Steve R Wilbur
s.wilbur@cs.ucl.ac.uk

Lecture Objectives

1. What does the OS have to do to manage memory?

2. How does it provide required sharing when needed and yet inhibit inappropriate access?
**Process Life Cycle**

*Recap*

To start a program, the OS must:
- Find the binary/executable on disk
- Allocate storage for the code and data
- Allocate swap space on disk
- Map the pages in memory and swap space
- Copy the code into the physical pages
- Start execution by saving current context (see later) and jumping (loading the PC) to the start address of the code!

**Memory Management Problems**

![Memory diagram](image)
Memory Management Problems - 2

1. Programs may be placed in different locations each time they are loaded - references to absolute addresses will be invalid
2. When memory full and need to load program need replacement policy to decide which (part of) process to swap out
3. Need to protect against processes damaging others - containment or protection
4. May be enough room to load program, but no “hole” is big enough - fragmentation

Memory Management Solutions

1. Relative addressing deals with (1), e.g.:
   - Jump ●+50 instead of Jump 456
   - Load ●-90 instead of Load 20
   - where ● = address of current instruction
2. Base and Limit registers deal with (1, 3):
   - B and L registers set to address and size of process memory when it becomes current process
   - Program compiled as though starts at address 0
   - Processor adds B to addresses and checks result not less than zero or >L for every memory fetch
Memory Management Solutions - 2

1. Paged memory deals with (4):
   - Expensive to “shuffle up” processes
   - Page Table (multiple base registers with fixed limit) makes all holes same size
   - Page table allows fragments to be physically separate but have logically contiguous addresses

Memory Management Solutions - 3

1. **Replacement Policy** with some help from hardware deals with (2)
   - “dirty bit” in page table indicates which pages have been written to and are thus more expensive to swap out
   - “use bit” in page table set when page accessed and cleared every few milliseconds indicates those pages which were recently used - *Principle of Locality* suggests they are a bad choice to swap out
Memory Management Problems - 3

1. (5) Program too large to fit in real memory

Solution:
- Use paging scheme where address space greater than memory space
- Some part of process will always be on disk
- Will be swapped in when needed (after delay)

Virtual Memory

1. Result is Virtual Memory
2. Programs written by human programmers are translated (compiled or interpreted) into machine code + data
3. Machine code is executed in a virtual address space which may provide user with a virtual memory much larger than real machine, albeit somewhat slower than main memory
4. Courtesy of the OS and hardware combined
**Mapping Process**

1. For a 4 GB virtual address space and a 1 GB physical address space

   ![Diagram of mapping process](image)

**Virtual Memory - 2**

1. The CPU accesses memory via a memory management unit - this uses a page table to translate addresses the CPU it gives (virtual addresses) into physical addresses.

2. The operating system ("kernel") has its own page table that lets it get at everything.

3. User processes have page tables managed by the operating system.

4. These page tables map virtual addresses to physical.
Virtual Memory - 3

1. So each program is run apparently in the address space 0-max address
2. This is mapped in chunks of a page - usually the same size as a disk block
3. Mapped to real memory - or flagged if swapped out to disk
4. Context switches change the page tables….

Virtual Memory - 4

1. Q. If you run more programs than will fit in real memory (easy to do) what happens?
2. A. Paging. The most recently run programs are in memory - the oldest one has the blocks (pages) of physical memory written out to disk, and the page table (in memory) changed to say that the pages are on disk - in the swap area
3. This of course takes time!
4. The page table (kept by the OS) keeps a map of the swap area too
**Virtual Address Mapping**

<table>
<thead>
<tr>
<th>Virtual Address Space</th>
<th>Prog A Virtual Addresses</th>
<th>Physical Memory Addresses</th>
<th>Prog B Virtual Addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>Unused</td>
<td>Unused</td>
<td>Unused</td>
</tr>
<tr>
<td>0x1000</td>
<td></td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>0x2000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x4000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x5000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x6000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x7000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xD000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xE000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xF000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Disk Addresses**

---

**Thrashing**

1. If there are a lot of processes, and they do a lot of I/O, and they are constantly being de-scheduled and re-scheduled, then there *may* be a lot of paging (copying of blocks to and from disk/memory).

2. This makes the system *run like a dog* (purely technically speaking, you understand).

3. Its called *thrashing* - very little progress is made under such circumstances....
Protection

1. Operating System provides various degrees of *protection* for the user and programs...
   - **Safety** - Ps should not be able to damage others accidentally or deliberately
   - **Fairness** - Ps should get fair treatment based on policies in force
   - **Integrity** - basic assumptions should not be violated
   - **Authenticity** - ensure objects are what they purport to be

1. OS enhances hardware support, to give a smarter virtual service

---

MMU Protection

1. Memory *maps* (page tables) provide basic protection against damage, because P can *only* “see” real memory that has been mapped to it
2. Each page can be marked as:
   - read
   - write
   - execute
   - or a combination
MMU Protection - 2

1. For example:
   - code would normally be execute-only to prevent copies being taken or being overwritten accidentally
   - constants, including strings, would be read-only so they cannot be changed
     1. Consider \( x := x + 1 \) if “constant” 1 was changed to 4!!
   - working data and the stack would be read+write

MMU Protection - 3

1. Two or more processes can share memory pages (not necessarily at the same address in each P)
   - one might have write access, the other read access - producer-consumer processes e.g., print server
   - both might have read-write access - application-defined sharing - need to carefully control use (lost updates?)
   - both might have execute access - shared libraries

1. Only OS can change the page maps so it controls access
## Sharing Pages

**Diagram:**

<table>
<thead>
<tr>
<th>Virtual Address Space</th>
<th>Physical Memory</th>
<th>Process A</th>
<th>Process B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>Unused</td>
<td></td>
<td>Unused</td>
</tr>
<tr>
<td>0x0100</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0200</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0300</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0400</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0500</td>
<td>RW</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>0x0600</td>
<td>RW</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>0x0700</td>
<td>-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Yellow pages in real memory are mapped to different addresses in Processes A and B. Even though pages are disjoint in memory they appear consecutive in both processes.

## Kernel and User Mode

1. **Q.** How does the OS stop user processes from getting at these data structures (e.g. Page Tables, Process tables etc)?

2. **A.** The processor can be set to **kernel mode** or **user mode**.

   (When the machine is **booted** it is set to **kernel mode** and memory mapping is set to **Virtual = Real**)

3. **The processor mode is changed as a side effect when the processor executes a trap or interrupt.**
Kernel and User Mode - 2

- In kernel mode any instruction can be executed (OS runs in this mode)
- In user mode, (normal processes,) I/O, halt and some other instructions cause a mode change to kernel mode and for a routine at a fixed location to be invoked
- This mechanism is known as a trap

Traps

There are at least 3 types of trap:

- System calls - Special instructions which can be used to invoke OS functions e.g. create a new process
- Exceptions - these occur when the processor executes an instruction which makes no sense, e.g.
  - Divide by 0
  - illegal instruction - undefined or not allowed in user mode
  - illegal memory access
- Interrupts - generated by I/O devices when they are done
**Cold Start**

- Machine starts in kernel mode
- Memory management off - i.e. mapped Virtual = Real
- ROM bootstrap executed to load OS
- Page table set for operating system to “see” whole memory
  - OS R, X, W as appropriate
  - free space R, W
- Memory management switched on

**Cold Start - 2**

- Initial user process loaded
  - command processor
  - login process, etc.
- PCB created
- User P’s initial PC, SP, page table pre-loaded into PCB
- PCB added to head of Ready to Run Q
- Dispatcher invoked
Cold Start - 3

- Set up user mode page table for P1 from “faked” PCB
- “Return” to P1
  - causes PC to be set
  - causes mode change to user mode

- P1 executes

Cold Start - 4

- P1 needs to do I/O - e.g. get user’s login name
  - System call for synchronous read
- Change to kernel mode
- Volatile registers saved in PCB by OS
- OS sets up I/O operation
- P1 status = Blocked
- Idle process (internal in OS) run until I/O complete

1 Try extending this for multiple user processes
Summary

- Memory management needed to ensure M used efficiently
- Page tables provide safe mechanism
- Pages can be flagged as read, write or execute or a mixture
- Only OS allowed to change tables
- User mode ensures normal user cannot access I/O devices or act as OS
- Traps provide means of changing mode and communication between user P and OS