Today

- OS Memory Management
- Virtual Memory
- Paging and Swapping

Process Memory Layout

<table>
<thead>
<tr>
<th>High Address</th>
<th>Low Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Args, environment</td>
<td>Program Text</td>
</tr>
<tr>
<td>Stack</td>
<td>Data Segment</td>
</tr>
<tr>
<td></td>
<td>bss</td>
</tr>
<tr>
<td>Heap</td>
<td>Initialized Static Data</td>
</tr>
<tr>
<td>Dynamic Memory</td>
<td>Uninitialized Static Data</td>
</tr>
<tr>
<td>Activation Records (function params, local vars, saved registers)</td>
<td>Binary Code</td>
</tr>
</tbody>
</table>

Where does the memory come from?

- Registers, cache, main memory (RAM)
- Process has to reside in the main memory
  - The instructions being executed
  - Data segment
  - Heap and stack
- How does the whole process fit in the main memory?
  - E.g.: 1 GB memory map of process in 512 MB physical RAM
Multiprogramming

- OS executes multiple processes in quick succession
- Multiple processes need to reside in the memory
- What if there are 20 processes where:
  - Each process has a memory map of 1GB
  - Amount of RAM in your machine is 512 MB
- Where does the memory come from?
- What happens to the memory of a process when it is not running?

OS Memory Management

- OS manages the available physical memory to meet demands of multiple processes
- Needs to “fit” individual processes into the memory when running
- Needs to multiplex the available memory among multiple processes
- Goals:
  - Maximum memory utilization
  - Highest level of performance

Memory Management Techniques

- Virtual Memory
  - Hiding the physical memory details and constraints from the processes and the programmer
- Paging
  - Breaking a process memory layout to “fit” it into available memory
- Swapping
  - Using secondary storage to multiplex memory among multiple processes

Virtual Memory

- The process is given a “logical” view of the memory
  - Contiguous
  - Starting from low address (typically just above the OS image)
  - Ending at high address (based on the architecture)
- The process references “virtual” memory addresses
  - Not physical memory addresses
  - Process has no knowledge of where an address would actually lie in physical memory
**Virtual Memory Mapping**

- Virtual addresses are mapped to physical addresses
  - The instructions and data have to be in physical memory while being used
- The mapping is controlled by the OS and the hardware
  - Transparent to the process
- Multiple processes can have same virtual addresses mapped to different physical addresses

**Virtual Memory: Benefits**

- Process does not have to be aware of physical memory layout
  - Virtual addresses can be generated at link/load time
- Multiple processes can use the same (virtual) address space
  - Generally the whole address space
- Multiple processes can share same physical memory region
  - Map to different virtual memory regions
  - E.g.: shared libraries

**Virtual Memory: Benefits (contd.)**

- Memory Protection
  - A process cannot access another process’s memory directly
- Process memory is not constrained by available physical memory
  - Virtual address space can be of any size
  - Typically constrained by architecture
How to do Virtual Memory Mapping?

- Approach 1: Map whole virtual address space to a contiguous physical memory region
  - Virtual space may be bigger than physical space
  - May not need whole virtual space at all times
- Approach 2: Map each virtual address independently to a physical address
  - Too much overhead
  - Neighboring addresses typically related

Paging

- Page: Contiguous chunk of memory addresses
  - Process virtual memory is divided up into equal-size pages
- Frame: Physical memory is also divided up into same-sized chunks
- Virtual memory mapping: Maps virtual pages to physical frames

Paging: Benefits

- Virtual memory mapping is more efficient
- Process does not have to be placed contiguously
  - OS can assign any available frame
- Locality: Bunches together memory chunks
  - Nearby addresses are used together
  - E.g.: instructions in a loop, array elements
**Swapping**

- How does the OS “fit” multiple processes into a small physical memory?
- What happens to the memory of a process when it is not running on the CPU?
- Solution: Use secondary storage (disk) as a backup

**Swapping**

- Move inactive process to secondary storage
- Bring in active process when scheduled to run
- Swap space: Portion of hard disk devoted to swapping

**Demand Paging**

- Combines swapping and paging
- Process uses only part of its whole memory at a given time
  - Current set of instructions
  - Portion of data, heap, and stack currently being used
- Move inactive pages to disk
- Bring in pages from secondary memory when accessed

**Page Faults**

- What happens when a process accesses an address, but the page is on the disk?
- OS generates an exception called page fault
  - Goes to the swap space and finds the page
  - Allocates a physical memory page
  - Copies the contents of the page from the swap space into the physical page
- Performance penalty: Takes a long time compared to a direct memory access
Page Fault Example

- How many page faults would happen?
  - Assume page size=4K
  - Be careful in how you access memory
  - Can have substantial effect on performance

OS Memory Management Summary

- Virtual Memory
- Paging and Swapping
- Demand Paging and Page Faults