CSC 453 Operating Systems

Lecture 8 : Memory Management

Memory Is A Resource

- Main memory is an array of words or bytes that can be accessed directly by their own unique addresses.
- Executing an instruction will involve accessing memory several times to:
  - fetch the instruction
  - fetch operand(s)
  - store the result
- Memory allocated to one process cannot be used by another process (unless the process allows it).
The Same Program May Be Loaded Into Different Memory Locations

Preparing A Source Program For Execution
Address Binding

Address binding can occur at:

- **Compile time**
  - when the program is translated, absolute addresses are generated.
  - This assumes that the compiler knows where in memory the program will be loaded.

- **Load time**
  - The compiler generates relocatable addresses.
  - The binding occurs when the program is loaded

- **Execution time**
  - If the process can moved during execution, address binding is delayed until run time.

Dynamic Loading

- The main program is loaded into memory and it is executed. All other routines are stored on disk with relocatable addresses.
- When a routine is called, it is read into memory and the program’s address table is updated to show the change.
- This allows large programs to run using only a fraction of the memory that they would need if the entire program is stored in memory.
- Users design their programs to use this approach, so no special operating system support is needed.
Dynamic Linking

- Most programs are statically linked, i.e. the libraries are combined with other modules to form the executable module.
- Some systems support dynamic linking where a routine called a stub is linked for every library routine instead of the actual code.
- When the procedure is called, the stub is executed.
  - If the procedure is not in memory, it loads it into memory.
  - The procedure replaces itself with the routine’s address.
- Dynamic linking requires operating system support because the procedures may be in an area of memory belonging to another process.

Overlays

```
Operating System
Common procedures
Overlay driver
Common data
```

```
Overlay 1
Operating System
Common procedures
Overlay driver
Common data
```

```
Overlay 2
Operating System
Common procedures
Overlay driver
Common data
```
Logical Vs. Physical Address

- **Logical Address** – The address used by the instructions to load and store data, jump to other instructions, etc.
- **Physical Address** – Used by the hardware to load and store data, fetch instructions, etc.

Relocatable Addressing

![Diagram showing logical and physical addresses with an example calculation: Jump to 497 + 25000 = 25497]
What Is Swapping?

Operating System
User Space Occupied by Process P₁

Backing Store

Swapping Out A Process

Operating System
Process P₁ Is Written on Backing Store

Backing Store
Swapping In A Process

Operating System

Process $P_2$
Is Read From Backing Store

Backing Store

Bare Machine

No Restrictions On Memory Use
No Operating System Services
Organizing Memory

- Main memory is comprised of two partitions:
  - Operating system
  - User processes
- The operating system can be at the high end or low end of memory – the location of the interrupt vector is largely responsible for where the operating system will reside.

Single-Partition Allocation

000000

Operating System

User Process

FFFFFF
Single-Partition Allocation With Transient Operating-System Code

Operating System

User Process

Rarely-used device driver

Hardware Support For Limit and Relocation Registers

Limit register

Relocation register

CPU

logical address

physical address

Memory
Multiple-Partition Allocation

- More than one user process can reside in memory at a time in contiguous memory allocations. There are two possible approaches:
  - There are a fixed maximum number of processes than can be resident at a time.
  - The number of processes that can reside in memory can vary.

MFT

- MFT (Multiprogramming with Fixed number of Tasks) assumes that there will be up to N processes resident in memory.
- Memory is divided into N partitions of fixed sizes (that are not necessarily equal).
- MFT was used in IBM’s OS/360 and is no longer used.
MVT

- MVT (Multiprogramming with Variable number of Tasks) assumes that there is no fixed maximum number of resident processes.
- The operating system keeps track of what parts of memory are in use and when a process needs a memory allocation, it looks for a partition in memory in which it will fit.

MVT and Allocating Memory

000000 019000
Operating system
Available memory
FFFFFF

000000 019000
Operating system
P1
270000
allocate P1
FFFFFF

000000 019000
Operating system
P1
270000
allocate P1
670000
FFFFFF

000000 019000
Operating system
P2
FFFFFF
MVT and Terminating Processes

MVT and Fragmentation
The Dynamic Storage-Allocation Problem

• There are three different algorithms that we can use to select an available block of memory into which the operating system will load the program:
  • First fit
    – Best fit
    – Worst fit

External & Internal Fragmentation
Compaction

Before compaction

After compaction

Other Compaction Strategies
What Is Paging?

- In a paged memory management system, physical memory is divided into fixed-sized block called frames into which are loaded blocks of logical memory.
- All the addresses appear to be contiguous but they are not necessarily stored in contiguous areas of memory.
- This eliminates the problem of external fragmentation.

<table>
<thead>
<tr>
<th>Logical Address</th>
<th>Page Number</th>
<th>Displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Address</td>
<td>Frame Number</td>
<td>Displacement</td>
</tr>
</tbody>
</table>
Hardware Support For Paging

CPU

logical address

Page Table

physical address

Physical Memory

Pages and Frames

Page Table

Page 0
Page 1
Page 2
Page 3
Page 4

Logical memory

Page 1
Page 2
Page 0
Page 4
Page 6

Physical memory
Loading a New Process

![Diagram showing a new process being loaded into memory]

Using Registers For the Page Table

![Diagram showing page table with page number and displacement]

Page Table

<table>
<thead>
<tr>
<th>p</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Associative Registers and Paging

CPU

p | d

TLB Hit

p

f

Main Memory

TLB Miss

Paging And Memory Protection

PTLR

f | v/i

f | r/w

f | r/w/x

(a) (b) (c)
2-Level 32-bit Paging

logical address
10 bits 10 bits 12 bits

Paging on the VAX

logical address
2 bits 21 bits 9 bits
Inverted Page Tables

Logical address

<table>
<thead>
<tr>
<th>pid</th>
<th>p</th>
<th>d</th>
</tr>
</thead>
</table>

Physical address

| i | d |

Search

Page Table

Shared Pages

Process 1

| pgm 1 | 2 |
| pgm 2 | 1 |
| pgm 3 | 6 |
| pgm 4 | 4 |
| data 1 | 0 |
| data 2 | 3 |

Process 1 Page Table

| pgm 1 | data 1 |
| pgm 2 | pgm 1 |
| pgm 3 | data 2 |
| pgm 4 | data 1 |

Physical memory

Process 2

| pgm 1 | 2 |
| pgm 2 | 1 |
| pgm 3 | 6 |
| pgm 4 | 4 |
| data 1 | 8 |
| data 2 | 9 |

Process 2 Page Table
A user’s program can be thought of as a collection of separate segments.
Segmentation and Memory Protection

- Segments represent different components of a program, such as code and data.
  - Code segments can be marked as read-only or execute-only.
  - Segments containing arrays can check array indices automatically.
  - Code and data segments can be shared by different processes.

Segments and Fragmentation

- Memory allocation works in a manner similar to paging, except that segments are of variable size.
  - There will external fragmentation, which can be alleviated by compaction.
  - The problem can be minimized by working with smaller segments.
Segmentation With Paging

• The two approaches could be used together to improve memory management
  – MULTICS paged the memory segments (which could be as large as 64K) to minimize external fragmentation.
  – The 32-bit version of OS/2 took a similar approach.