Memory III

Dmitri Loguinov
Texas A&M University

November 17, 2020
Why are lookup tables useful?
- Allows verification of set membership in 1 cache access

How to initialize?
- E.g., need to set up LUT to verify that character belongs to set {+, -, =, /, *}

When bool maps to 1 byte, can use it instead of char
- Keep in mind though that BOOL is 4 bytes

Make sure to test code on various input and buf size
- Debugging: elimination of crashes/incorrect output
- Testing: discovery of input configurations that expose previously unseen problems

Homework #4

```c
char LUT[256];
memset(LUT, false, 256);
const char special[] = "+-=/*";
for (int i = 0; i < strlen(special); i++)
    LUT[special[i]] = true;
```
Chapter 7: Roadmap

7.1 Requirements
7.2 Partitioning
7.3 Paging
7.4 Segmentation
7.5 Security

8.1 Hardware virtual memory
8.2 OS software
Memory Dumps

• Process crash is usually good news
  - Attach debugger, examine location of crash…
• Except when product has shipped to customers
  - Users do stuff with code that makes it crash
  - Developer is unable to replicate bug locally, what next?
• Idea: catch faults thrown by SEH (Structured Exception Handling)
  - Create a crash dump, send it to main server, then probably restart
Memory Dumps

• Instead of dumping entire RAM contents, Windows allows much smaller files called MiniDumps
  – Can be customized during exception handling to vary in size from a few KB to a few MB

• MiniDumps can be loaded into Visual Studio
  – Shows the exact location of crash, call stack, certain variables (even if crashed in release mode)

• Example:
  – Important application that must work 24/7, years in a row
  – When it crashes, saves internal data and dump, restarts
  – Debugging is done offline from a collection of minidumps

• See MiniDumpWriteDump on MSDN
Chapter 7: Roadmap

7.1 Requirements
7.2 Partitioning
7.3 Paging
7.4 Segmentation
7.5 Security

8.1 Hardware virtual memory
8.2 OS software
Buffer Overflow Attacks

Example 1:

```c
int CheckPassword (HANDLE user) {
    char correctHash [16];
    char userHash [16];

    GetPassHash (user, correctHash);
    // remote desktop hashes password
    // and sends the hash to server
    Network.Read (userHash);
    if (!strcmp (correctHash, userHash))
        return MATCH;
    else
        return BOGUS;
}
```

Example 2:

```c
void HandleServerRequest (void) {
    char request [256];
    Network.Read (request);
    ...
}
```

- Stack grows backwards
- Overflow of `userHash` rewrites `correctHash`
- Long overflow, contains virus code
- Execution continues from PC, virus runs
**Buffer Overflow Attacks**

- Modern OS usually puts a guard page between data, code, and stack

- **Example 3:**

  ```c
  void HandleServerRequest (void) {
      char request [256];
      Network.Read (request);
      ...
  }
  ```

  The return address is rewritten to jump back to execute virus code.

- Modern OS marks data and stack pages as non-executable (DEP)
Buffer Overflow Attacks

- **Example 4:**

  ```c
  void HandleServerRequest (void) {
      char request[256];
      Network.Read(request);
      ...
  }
  ```

  - Request
  - Garbage
  - Ret addr
  - Ret Ntdll.A

  rewrites return address to jump to specific kernel function that gives elevated privileges

  - Ntdll.A: admin user logged in
  - Ntdll.B: change admin password
  - Ntdll.C: wipe C:

- **Example 5:**

  ```c
  void HandleServerRequest (void) {
      char *ptr = new char[50];
      char request[256];
      Network.Read(request);
      strcpy(ptr, "hello world");
  }
  ```

  - Request
  - Garbage
  - Hijacked ptr
  - Ret addr

  admin password in RAM

  - Request
  - Garbage
  - Ret addr
  - Ret addr

  kernel space

  more in CSCE 465
OpenSSL is a library that encrypts/decrypts traffic
- Commonly used in HTTPS, SSH, secure IMAP/SMTP

Heartbeat extension introduced in 2011
- OpenSSL periodically sends a request that is echoed back to verify the connection is alive

Request message format:

```
header  len  buffer
```

Response is supposed to echo the buffer
- Implementation →

```c
size = Network.GetNextPacketSize();
char *packet = new char [size];
Network.Read (packet);
len = ExtractLenField (packet);
Network.Send (packet, len+sizeof(header)+sizeof(short));
```
Chapter 7: Roadmap

7.1 Requirements
7.2 Partitioning
7.3 Paging
7.4 Segmentation
7.5 Security

8.1 Hardware virtual memory
8.2 OS software
Managing Virtual Memory

• The OS has to make two main decisions when managing virtual memory and swapping
  – Which page to bring back to RAM (fetch policy)
  – Which page to offload to disk (replacement policy)
• Similar concepts may be useful in user-mode programs (e.g., object caching, browser prefetch)
• Fetch policy
  – Demand paging: bring page only on access (Windows)
  – Prepaging: OS attempts to guess future demand, bring those pages in memory ahead of the request
• Replacement policy
  – FIFO: treats all pages as circular buffer, evicts the next one
Managing Virtual Memory

- **Replacement policy (cont’d)**
  - **LRU**: evicts the page that has not been used the longest
  - **Optimal**: evicts the page that won’t be used the longest (only used in simulations for comparison purposes)

- **How to implement LRU?**
  - Can’t tag each page with an access timestamp (updating timestamps incurs huge overhead)
  - Can’t organize all pages into a linked list either (moving items to the front of the list on access is expensive)

- **Idea**: replace LRU with an approximation algorithm
  - Assume a set of pages 0, ..., N-1 that the OS manages
  - Associate a bit B (e.g., in the TLB) with each page
  - CPU sets the bit to 1 upon each read/write access
Managing Virtual Memory

- Upon page fault that needs more space:
  - OS scans from current position CP in [0, N-1] forward
  - If next page has B = 1, flag is reset to 0 and scan continues
  - If next page has B = 0, OS stops and evicts that page

- This policy is called CLOCK
  - Next page evicted?

- Quality of algorithm measured by number of hard page faults (PF)
  - FIFO 2x worse than optimal in PF
  - CLOCK better than FIFO, but not as good as LRU
Managing Virtual Memory

• Should pages that were read be replaced at the same rate as those that have been written to?
  – Probably more expensive to evict a modified page
• **Idea:** set up an extra bit $W$ for each page
  – CPU modifies them on access, CLOCK first evicts eligible pages with $W = 0$; if none left, then those with $W = 1$
• CLOCK is quicker than LRU even in user mode
• Examples where CLOCK might be useful:
  – Web crawler keeps a list of recently seen URLs
  – Search engine caches answers to popular queries
  – Homework #4: 50% of all hash table lookups refer to 1,270 words (20% to just 36 words), possible ways to speed up?