Shell Lab + How to C
Agenda

• Problem 5 and 9
• C Stuff from Cachelab.
• Processes and Signals.
• Shellab hints and tips.
Problem 5

• Initial stack frame

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNK</td>
<td>0XFFFF1008</td>
</tr>
<tr>
<td>18</td>
<td>0XFFFF1004</td>
</tr>
<tr>
<td>213</td>
<td>0XFFFF1000</td>
</tr>
<tr>
<td>Return address</td>
<td>0XFFFF0FFC</td>
</tr>
</tbody>
</table>

• 1<sup>st</sup> call stack frame

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNK</td>
<td>0XFFFF0FF8</td>
</tr>
<tr>
<td>UNK</td>
<td>0XFFFF0FF4</td>
</tr>
<tr>
<td>UNK</td>
<td>0XFFFF0FF0</td>
</tr>
<tr>
<td>15</td>
<td>0XFFFF0FEC</td>
</tr>
<tr>
<td>18</td>
<td>0XFFFF0FE8</td>
</tr>
<tr>
<td>0X080483B7</td>
<td>0XFFFF0FE4</td>
</tr>
</tbody>
</table>
Problem 5

- 2\textsuperscript{nd} stack frame

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0XFFFF0FF8</td>
<td>0XFFFF0FE0</td>
</tr>
<tr>
<td>UNK</td>
<td>0XFFFF0FDC</td>
</tr>
<tr>
<td>UNK</td>
<td>0XFFFF0FD8</td>
</tr>
<tr>
<td>3</td>
<td>0XFFFF0FD4</td>
</tr>
<tr>
<td>15</td>
<td>0XFFFF0FD0</td>
</tr>
<tr>
<td>0X080483B7</td>
<td>0XFFFF0FCC</td>
</tr>
</tbody>
</table>

- 3\textsuperscript{rd} stack frame

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0XFFFF0FE0</td>
<td>0XFFFF0FC8</td>
</tr>
<tr>
<td>UNK</td>
<td>0XFFFF0FC4</td>
</tr>
<tr>
<td>UNK</td>
<td>0XFFFF0FC0</td>
</tr>
<tr>
<td>0</td>
<td>0XFFFF0FBC</td>
</tr>
<tr>
<td>3</td>
<td>0XFFFF0FB8</td>
</tr>
</tbody>
</table>
Problem 9

• Part A:
  – Accessing A row wise for 1 miss ... 15 hits.
  – Accessing B column wise for 1 miss ... 1 hit.
  – Total is roughly \( \frac{1}{2} \)
  – The idea here is to look at access patterns and not individual memory accesses.
  – Need to move to a higher level.
Problem 9

• Part 9
  – Access A row wise.
  – Access B row wise.
  – Both are 1 miss ... 15 hits.
  – Approx miss rate 1/16.
Agenda

• Problem 5 and 9
• **C Stuff from Cachelab.**
• Processes and Signals.
• Shellab hints and tips.
C Stuff from Cachelab

- Always check for NULL return from a malloc.
- If your code does not look like this:

```c
char *foo = malloc(200);
if (!foo) {
    error(“malloc failed”);
}
```

- Then you are doing something terribly wrong.
- This should be second nature.
C Stuff from Cachelab

• Sanity checking arguments.
• If you get a value from outside of your program, and you expect it to have a particular value, you must check to make sure it has that value.
C Stuff from Cachelab

• 80 column limit:
  – grep '.\{81,\}' *.c

• Indentation, convert all tabs to spaces.
  – :set expandtab and :retab in VIM
  – M-x untabify in EMACS
Agenda

- Problem 5 and 9
- C Stuff from Cachelab.
- **Processes and Signals.**
- Shellab hints and tips.
Processes

• Four basic process control functions
  – fork()
  – exec()
  – exit()
  – wait()
• Standard on all Unix systems
Processes

• fork()
  – Creates a process
  – Parent and child are exactly alike
  • Equal but separate
    – Execution (%eip)
    – Registers
    – Memory
    – File descriptors, the files themselves are shared.
Processes

• exec()
  – Replaces process context
  – How programs are run
    • Replace memory image with a new program
    • Set up stack with arguments
    • Start execution at the entry point (main)
  – A family of functions (man 3 exec)
Processes

• `exit()`
  – Terminates a process
  – OS frees resources used by the process
  – Tiny leftover data
    • Exit status for the parent
    • Must be freed
    • Which brings us to ...
Processes

• wait()
  – Waits for a child to change state
  – If a child terminates, the parent “reaps” the child, freeing all resources and getting the exit status
  – Lots of details (man 2 wait)
Signals

- Interprocess communication/notification
- **Asynchronous** with normal execution
- Come in many types (man 7 signal)
- Sent in various ways
  - \^C, \^Z, \^\`
  - kill (which we will demonstrate)
Signals

• Disposition
  – Ignore
  – Catch and run a signal handler
  – Terminate
  – man sigaction

• Blocking
  – man sigprocmask

• Waiting
  – man sigsuspend (don’t need this, but cool)
Agenda

• Problem 5 and 9
• C Stuff from Cachelab.
• Processes and Signals.
• **Shellab hints and tips.**
Shellab

• Read the code we’ve given you.
  – There is a lot of stuff you don’t need to write for yourself.
  – It’s a good example of the kind of code we expect from you.
Shellab

• If you find yourself using `sleep()` as a way of avoiding race conditions, you are doing it VERY wrong. We will dock performance points for this.

• You should only use it for performance to avoid your code having to execute useless instructions. Your code should still work if we remove calls to sleep.
Shellab

- Hazards
  - Race conditions
    - Hard to debug so start early.
  - Reaping zombies
    - Race conditions
    - Fiddle with signals
  - Waiting for foreground job
    - One of the only places where sleep is acceptable (though you don’t NEED it)