• http://blog.natekohl.net/abomination-and-astonishment/
• Non-reentrant code
  – Kernel vs. interrupt
  – Vs. thread-safe

• Setting/using %esp & %ebp during procedure calls
IA32 Stack

• Region of memory managed with stack discipline
• Grows toward lower addresses

Register `%esp` contains lowest stack address = address of “top” element
Procedure Control Flow

• Use stack to support procedure call and return
  • Procedure call: `call label`
    – Push return address on stack
    – Jump to `label`
  • Return address:
    – Address of instruction beyond `call`
    – Example from disassembly
      804854e: e8 3d 06 00 00  call   8048b90 <main>
      8048553: 50              pushl  %eax
    – Return address = 0x8048553
  • Procedure return: `ret`
    – Pop address from stack
    – Jump to address
Stack Frames

• Contents
  – Local variables
  – Return information
  – Temporary space

• Management
  – Space allocated when enter procedure
    • “Set-up” code
  – DEALLOCATED when return
    • “Finish” code
```c
yoo(...) {
    .
    .
    who();
    .
    .
}
```
Example

```
void amI(...) {
    ...
    amI();
    ...
}
```

Stack:
- yoo
- who
- amI
- amI
- amI
- amI
- %ebp
- %esp
IA32/Linux Stack Frame

• Current Stack Frame (“Top” to Bottom)
  – “Argument build:” Parameters for function about to call
  – Local variables
    If can’t keep in registers
  – Saved register context
  – Old frame pointer

• Caller Stack Frame
  – Return address
  – Pushed by `call` instruction
  – Arguments for this call
Revisiting `swap`

```c
int zip1 = 15213;
int zip2 = 91125;

void call_swap()
{
    swap(&zip1, &zip2);
}

void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

**Calling `swap` from `call_swap`**

```c
void call_swap()
{
    // Global Var
    pushl $zip2
    // Global Var
    pushl $zip1
    call swap
}
```

**Resulting Stack**

```
    %esp

    Rtn adr

    &zip1

    &zip2
```

Calling `swap` from `call_swap`
Revisiting `swap`

```c
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

```assembly
swap:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
    movl 12(%ebp),%ecx
    movl 8(%ebp),%edx
    movl (%ecx),%eax
    movl (%edx),%ebx
    movl %eax,(%edx)
    movl %ebx,(%ecx)
    movl -4(%ebp),%ebx
    movl %esp,%ebp
    popl %ebp
    ret
```

- **Set Up**
- **Body**
- **Finish**
swap Setup #1

Entering Stack

Resulting Stack

swap:
    pushl %ebp
    movl %esp, %ebp
    pushl %ebx
swap Setup #1

Entering Stack

\[
\begin{array}{c}
\text{%ebp} \\
\text{%esp} \\
&\text{\&zip2} \\
&\text{\&zip1} \\
\text{Rtn adr}
\end{array}
\]

\[
\begin{array}{c}
\text{%ebp} \\
\text{%esp} \\
\text{yp} \\
\text{xp} \\
\text{Rtn adr} \\
\text{Old %ebp}
\end{array}
\]

\[
\text{swap:}
\begin{align*}
\text{pushl %ebp} \\
\text{movl %esp, %ebp} \\
\text{pushl %ebx}
\end{align*}
\]
**swap Setup #1**

**Entering Stack**

- `%ebp`
- `&zip2`
- `&zip1`
- `Rtn adr`

**Resulting Stack**

- `%ebp`
- `yp`
- `xp`
- `Rtn adr`
- `Old %ebp`

swap:

```plaintext
pushl %ebp
movl %esp, %ebp
pushl %ebx
```
swap Setup #1

Entering Stack

```
swap:
pushl %ebp
movl %esp,%ebp
pushl %ebx
```

```
 &zip2
 &zip1
 Rtn adr
```

```
 %ebp
 %esp
```

```
 &zip2
 &zip1
 Rtn adr
Old %ebp
```

```
 %ebp
 %esp
```

```
 %ebp
 %esp
```
swap Setup #1

Entering Stack

Resulting Stack

Offset relative to %ebp

movl 12(%ebp),%ecx  # get yp
movl 8(%ebp),%edx   # get xp

. . .
```plaintext
Observation: Saved and restored register %ebx
```
swap Finish #2

swap's Stack

movl -4(%ebp), %ebx
movl %esp, %ebp
popl %ebp
ret
swap Finish #2

swap’s Stack

Resulting Stack

movl -4(%ebp),%ebx
movl %esp,%ebp
popl %ebp
ret
swap Finish #2

swap’s Stack

movl -4(%ebp),%ebx
movl %esp,%ebp
popl %ebp
ret
swap Finish #3

swap’s Stack

Resulting Stack

movl -4(%ebp),%ebx
movl %esp,%ebp
popl %ebp
ret
swap Finish #4

swap’s Stack

movl -4(%ebp),%ebx
movl %esp,%ebp
popl %ebp
ret
swap Finish #4

Observation
- Saved & restored register %ebx
- Didn’t do so for %eax, %ecx, or %edx
Disassembled swap

080483a4 <swap>:

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>80483a4:</td>
<td>55</td>
<td>push %ebp</td>
</tr>
<tr>
<td>80483a5:</td>
<td>89 e5</td>
<td>mov %esp,%ebp</td>
</tr>
<tr>
<td>80483a7:</td>
<td>53</td>
<td>push %ebx</td>
</tr>
<tr>
<td>80483a8:</td>
<td>8b 55 08</td>
<td>mov 0x8(%ebp),%edx</td>
</tr>
<tr>
<td>80483ab:</td>
<td>8b 4d 0c</td>
<td>mov 0xc(%ebp),%ecx</td>
</tr>
<tr>
<td>80483ae:</td>
<td>8b 1a</td>
<td>mov (%edx),%ebx</td>
</tr>
<tr>
<td>80483b0:</td>
<td>8b 01</td>
<td>mov (%ecx),%eax</td>
</tr>
<tr>
<td>80483b2:</td>
<td>89 02</td>
<td>mov %eax,(%edx)</td>
</tr>
<tr>
<td>80483b4:</td>
<td>89 19</td>
<td>mov %ebx,(%ecx)</td>
</tr>
<tr>
<td>80483b6:</td>
<td>5b</td>
<td>pop %ebx</td>
</tr>
<tr>
<td>80483b7:</td>
<td>c9</td>
<td>leave</td>
</tr>
<tr>
<td>80483b8:</td>
<td>c3</td>
<td>ret</td>
</tr>
</tbody>
</table>

Calling Code

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>8048409:</td>
<td>e8 96 ff ff ff ff</td>
<td>call 80483a4 &lt;swap&gt;</td>
</tr>
<tr>
<td>804840e:</td>
<td>8b 45 f8</td>
<td>mov 0xfffffffffffffff8(%ebp),%eax</td>
</tr>
</tbody>
</table>
Register Saving Conventions

• When procedure `yoo` calls `who`:
  – `yoo` is the `caller`
  – `who` is the `callee`

• Can Register be used for temporary storage?

```assembly
yoo:
  ...
  movl $15213, %edx
  call who
  addl %edx, %eax
  ...
  ret

who:
  ...
  movl 8(%ebp), %edx
  addl $91125, %edx
  ...
  ret
```

– Contents of register `%edx` overwritten by `who`
Register Saving Conventions

• When procedure \texttt{yoo} calls \texttt{who}:
  – \texttt{yoo} is the \textit{caller}
  – \texttt{who} is the \textit{callee}

• Can register be used for temporary storage?

• Conventions
  – “\textit{Caller Save}”
    • Caller saves temporary in its frame before calling
  – “\textit{Callee Save}”
    • Callee saves temporary in its frame before using
IA32/Linux Register Usage

- `%eax`, `%edx`, `%ecx`
  - Caller saves prior to call if values are used later

- `%eax`
  - also used to return integer value

- `%ebx`, `%esi`, `%edi`
  - Callee saves if wants to use them

- `%esp`, `%ebp`
  - special
• 3.32
• 3.33