Assembly: Procedures, function calls, stack discipline, recursion.

Yipeng Huang

Rutgers University

March 25, 2021
Announcements

Procedures and function calls: Transferring control
  Special state
  Stack instructions: push and pop
  Procedure call and return: call and ret
  Example in GDB

Procedures and function calls: Transferring data
Looking ahead

Class plan


2. Starting next week: Recitations will have specialized topics for remainder of semester. https://rutgers.instructure.com/courses/104725/pages/recitation-and-office-hour-information

Table of contents

Announcements

Procedures and function calls: Transferring control
  Special state
  Stack instructions: push and pop
  Procedure call and return: call and ret
  Example in GDB

Procedures and function calls: Transferring data
Procedures and function calls

To create the abstraction of functions, need to:

- Transfer control to function and back
- Transfer data to function (parameters)
- Transfer data from function (return type)

Figure: Steps of a C function call. Image credit CS:APP
CPU and memory state in support of procedures and functions

Assembly/Machine Code View

Programmer-Visible State
- **PC**: Program counter
  - Address of next instruction
  - Called “RIP” (x86-64)
- **Register file**
  - Heavily used program data
- **Condition codes**
  - Store status information about most recent arithmetic or logical operation
  - Used for conditional branching

Memory
- **Memory**
  - Byte addressable array
  - Code and user data
  - Stack to support procedures

Relevant state in CPU:
- `%rip` register / instruction pointer / program counter
- `%rsp` register / stack pointer

Relevant state in Memory:
- Stack
Stack instructions: **push** and **pop**

<table>
<thead>
<tr>
<th>OP SRC DEST</th>
<th>Initially</th>
<th>pushq %rax</th>
<th>popq %rdx</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSHQ SRC</td>
<td>%rax: 0x123</td>
<td>%rax: 0x123</td>
<td>%rax: 0x123</td>
</tr>
<tr>
<td>POPQ DEST</td>
<td>%rdx: 0</td>
<td>%rdx: 0</td>
<td>%rdx: 0</td>
</tr>
<tr>
<td></td>
<td>%rsp: 0x108</td>
<td>%rsp: 0x100</td>
<td>%rsp: 0x108</td>
</tr>
</tbody>
</table>

etically

<table>
<thead>
<tr>
<th>Initially</th>
<th>pushq %rax</th>
<th>popq %rdax</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rax: 0x123</td>
<td>%rax: 0x123</td>
<td>%rax: 0x123</td>
</tr>
<tr>
<td>%rdx: 0</td>
<td>%rdx: 0</td>
<td>%rdx: 0</td>
</tr>
<tr>
<td>%rsp: 0x108</td>
<td>%rsp: 0x100</td>
<td>%rsp: 0x108</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stack “top”</th>
<th>Stack “bottom”</th>
<th>Stack “bottom”</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x108</td>
<td>0x108</td>
<td>0x108</td>
</tr>
<tr>
<td>0x100</td>
<td>0x123</td>
<td>0x123</td>
</tr>
<tr>
<td></td>
<td>0x123</td>
<td>0x123</td>
</tr>
<tr>
<td></td>
<td>0x123</td>
<td>0x123</td>
</tr>
</tbody>
</table>

Figure: x86-64 offers dedicated instructions to work with stack in memory. In addition to moving data, the updating of %rsp is implied. Image credit: CS:APP.
Procedure call and return: `call` and `ret`

Figure: Effect of `call 0x400540` instruction and subsequent return. `call` and `ret` instructions update the instruction pointer, the stack pointer, and the stack to create the procedure / function call abstraction. Image credit: CS:APP.
Example in GDB

```c
#include <stdio.h>

int return_neg_one() {
    return -1;
}

int main() {
    int num = return_neg_one();
    printf("%d", num);
    return 0;
}
```

Compile, and then run it in GDB:
gdb return

In GDB, see evolution of %rip, %rsp, and stack:
- (gdb) layout split
- (gdb) break return_neg_one
- (gdb) print /a $rip
- (gdb) print /a $rsp
- (gdb) x /a $rsp

Step past return instruction, and inspect again:
- (gdb) stepi
Table of contents

Announcements

Procedures and function calls: Transferring control
  Special state
  Stack instructions: push and pop
  Procedure call and return: call and ret
  Example in GDB

Procedures and function calls: Transferring data
Procedures and function calls: Transferring data

For purposes of this class, the Bomb Lab, and the CS:APP textbook, we study the x86-64 Linux Application Binary Interface (ABI). Would be different on ARM or in Windows. So, don’t memorize this, but it is helpful for PA4 Bomb Lab.

Passing parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Register / stack</th>
<th>Subset registers</th>
<th>Mnemonic¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>%rdi</td>
<td>%edi, %di</td>
<td>Diane’s</td>
</tr>
<tr>
<td>2nd</td>
<td>%rsi</td>
<td>%esi, %si</td>
<td>silk</td>
</tr>
<tr>
<td>3rd</td>
<td>%rdx</td>
<td>%edx, %dx, %dl</td>
<td>dress</td>
</tr>
<tr>
<td>4th</td>
<td>%rcx</td>
<td>%ecx, %cx, %cl</td>
<td>cost</td>
</tr>
<tr>
<td>5th</td>
<td>%r8</td>
<td>%r8d</td>
<td>$8</td>
</tr>
<tr>
<td>6th</td>
<td>%r9</td>
<td>%r9d</td>
<td>9</td>
</tr>
<tr>
<td>7th and beyond</td>
<td>Stack</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

¹http://csappbook.blogspot.com/2015/08/dianes-silk-dress-costs-89.html
Procedures and function calls: Transferring data

Passing function return data

Function return data is passed via:

▶ the 64-bit %rax register
▶ the 32-bit subset %eax register