Procedures II
CSE 351 Winter 2024

Instructor:
Justin Hsia

Teaching Assistants:
Adithi Raghavan
Aman Mohammed
Connie Chen
Eyoel Gebre
Jiawei Huang
Malak Zaki
Naama Amiel
Nathan Khuat
Nikolas McNamee
Pedro Amarante
Will Robertson

http://xkcd.com/1270/
Lab 1b grades released later this week
- Regrade requests open ~24 hours after grade release (rounded to 12:00 am), close ~72 hours after grade release (rounded to 11:59 pm)

Lab 2 due Friday (2/2)
- Since you are submitting a text file (`defuser.txt`), there won’t be any Gradescope autograder output about compilation this time – check the Code tab after submission to make sure that everything looks right
- Extra credit (bonus) needs to be submitted to the extra credit assignment

Midterm (take home, 2/8–10)
- Make notes and use the [midterm reference sheet](#)
- Form study groups and look at past exams!
Procedures II
Lesson Summary (1/3)

❖ Each stack frame organized in the same way:
1) Return address pushed by call
   • The address of the instruction after call
2) Callee-saved registers
   • Only if procedure modifies/uses them
3) Local variables
   • Unavoidable if variable is too big for a register (e.g., array)
   • Unavoidable if variable needs an address (i.e., uses &var)
4) Caller-saved registers
   • Only if values are needed across a procedure call
5) Argument build
   • Only if procedure calls a procedure with more than six arguments
Lesson Summary (2/3)

❖ Important Points

- Procedures are a combination of instructions and conventions
  - Conventions prevent functions from disrupting each other
- Stack is the right data structure
  - “Last in, first out” matches lifetime of procedures
- Recursion handled by normal calling conventions

❖ Generally want to minimize the use of the stack

- Lean heavily on registers, which are faster to access

![Stack Frame Diagram]
## Lesson Summary (3/3)

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Caller Saved</th>
<th>Callee Saved</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rax</td>
<td>Return value</td>
<td>Caller saved</td>
<td></td>
</tr>
<tr>
<td>%rbx</td>
<td>冷静 saved</td>
<td>Caller saved</td>
<td></td>
</tr>
<tr>
<td>%rcx</td>
<td>Argument #4</td>
<td>Caller saved</td>
<td></td>
</tr>
<tr>
<td>%rdx</td>
<td>Argument #3</td>
<td>Caller saved</td>
<td></td>
</tr>
<tr>
<td>%rsi</td>
<td>Argument #2</td>
<td>Caller saved</td>
<td></td>
</tr>
<tr>
<td>%rdi</td>
<td>Argument #1</td>
<td>Caller saved</td>
<td></td>
</tr>
<tr>
<td>%rsp</td>
<td>Stack pointer</td>
<td></td>
<td>Caller saved</td>
</tr>
<tr>
<td>%rbp</td>
<td>冷静 saved</td>
<td>Caller saved</td>
<td></td>
</tr>
<tr>
<td>%r8</td>
<td>Argument #5</td>
<td>Caller saved</td>
<td></td>
</tr>
<tr>
<td>%r9</td>
<td>Argument #6</td>
<td>Caller saved</td>
<td></td>
</tr>
<tr>
<td>%r10</td>
<td>Callee saved</td>
<td>Caller saved</td>
<td></td>
</tr>
<tr>
<td>%r11</td>
<td>Callee saved</td>
<td>Caller saved</td>
<td></td>
</tr>
<tr>
<td>%r12</td>
<td>Callee saved</td>
<td>Caller saved</td>
<td></td>
</tr>
<tr>
<td>%r13</td>
<td>Callee saved</td>
<td>Caller saved</td>
<td></td>
</tr>
<tr>
<td>%r14</td>
<td>Callee saved</td>
<td>Caller saved</td>
<td></td>
</tr>
<tr>
<td>%r15</td>
<td>Callee saved</td>
<td>Caller saved</td>
<td></td>
</tr>
</tbody>
</table>
Silly Register Convention Analogy

1) Parents *(caller)* leave for the weekend and give the keys to the house to their child *(callee)*
   - Being suspicious, they put away/hid the valuables *(caller-saved)* before leaving
   - Warn child to leave the bedrooms untouched: “These rooms better look the same when we return!”

2) Child throws a wild party *(computation)*, spanning the entire house
   - To avoid being disowned, child moves all of the stuff from the bedrooms to the backyard shed *(callee-saved)* before the guests trash the house
   - Child cleans up house after the party and moves stuff back to bedrooms

3) Parents return home and are satisfied with the state of the house
   - Move valuables back and continue with their lives
Lesson Q&A

❖ Learning Objectives:

▪ Trace stack frame contents through the execution of x86-64 assembly instructions for both recursive and non-recursive programs.

▪ Identify how x86-64 register-saving conventions allow procedures to execute without destroying each other’s data.

❖ What lingering questions do you have from the lesson?

▪ Chat with your neighbors about the lesson for a few minutes to come up with questions
Procedures II – Practice
Polling Questions

- In the following function, how big is the stack frame? Which instruction(s) pertain to the **local variables** and **saved registers** portions of its stack frame?

```plaintext
call_mem_add2:
1  pushq  %rbx
2  subq  $16, %rsp
3  movq  %rdi, %rbx
4  movq  $351, 8(%rsp)
5  movl  $100, %esi
6  leaq  8(%rsp), %rdi
7  call  mem_add
8  addq  %rbx, %rax
9  addq  $16, %rsp
10 popq  %rbx
11 ret
```
Homework Setup

❖ **Caller**-saved register example:

- Saving is done just before calling the **callee** and restoring is done right after the call

```plaintext
Caller

...<use %rax>
pushq %rax # save old val
callq

popq %rax # restore old value
<use %rax> # same value as before
...
```

❖ **Callee**-saved register example:

- Saving is done early in procedure (before use) and restoring is done just before returning to **caller**

```plaintext
Caller

...<use %rbx>
callq

<use %rbx> # same value as before
...
```

```plaintext
Callee

pushq %rbx # save old val
<change %rbx value>
popq %rbx # restore old value
retq
```

Procedures II – Context
Recursive Example: Popcount

```c
/* Recursive popcount */
long pcount_r(unsigned long x) {
    if (x == 0)
        return 0;
    else
        return (x & 1) + pcount_r(x >> 1);
}
```

- Counts the 1’s in the binary representation of x
  - [https://godbolt.org/z/P8Mened14](https://godbolt.org/z/P8Mened14)
  - Compiled with -O1 instead of -0g for more natural instruction ordering

- Register usage:
  - Need x (in %rdi) after procedure call
  - Chooses to save %rdi by copying into %rbx
  - Chooses to save %rbx by pushing to stack (only in recursive case)
GDB Demo #2

❖ Let’s examine the `pcount_r` stack frames on a real machine!
  ▪ Using `pcount.c` from the course website

❖ You will need to use GDB to get through the Midterm
  ▪ Useful debugger in this class and beyond!

❖ Pay attention to:
  ▪ Checking the current stack frames (backtrace)
  ▪ Getting stack frame information (`info frame <#>`)  
  ▪ Examining memory (`x`)
Group Work Time

❖ During this time, you are encouraged to work on the following:
   1) If desired, continue your discussion
   2) Work on the homework problems
   3) Work on the lab (if applicable)

❖ Resources:
   ▪ You can revisit the lesson material
   ▪ Work together in groups and help each other out
   ▪ Course staff will circle around to provide support