Parameter Passing

COMP360
“It should be noted that no ethically-trained software engineer would ever consent to write a DestroyBaghdad procedure. Basic professional ethics would instead require him to write a DestroyCity procedure, to which Baghdad could be given as a parameter.”

Nathaniel S. Borenstein
Stacks

• Many programming languages use stacks to pass parameters
• Many computer architectures have stack instructions to help implement these programming languages
• Most architectures have stack pointer register. The stack pointer always points to the top item on the stack.
Program Memory Organization

- Heap
- Stack
- Global data
- Program instructions
Program Memory Organization

- Heap
- Stack
- Global data
- Program instructions

Intel method
Function Call Hardware

• All computers have machine language instructions to support function calls
• The level of hardware support varies with modern computers providing more support
Intel Call instruction

• The **CALL** instruction basically pushes the program counter on the stack and branches to a new location.

• There are many versions of the Intel **CALL** instruction to support different addressing modes and changes in privileges.
Intel RET instruction

• The RET or return instruction pops a value from the stack and places it in the program counter register
• Since the program counter contains the address of the next instruction to execute, this has the effect of branching back to the calling program
Basic Steps to Call a Method

• Compute any equations used in the parameters, such as $x = \text{func}(a + b)$;

• Push the parameter values on the stack

• Execute a call instruction to push the return address on the stack and start execution at the first address of the function
Upon function entry

• Save the contents of the registers
  • Many systems have the convention that a method should return with the registers just the way they were when called
• Increase the stack pointer to reserve memory for the local variable
• Start executing the function code
Upon function exit

- Reduce the stack by the size of the local variable
- Pop the register values
- Execute the return instruction to pop the address from the stack into the program counter
Example Function Call

• Consider the function

```cpp
void thefunc(Widget b, int a ){
    int r = a;
}
```

• that is called by the main program

```cpp
int x = 5;
Widget y = new Widget();
thefunc( y, x );
```

• The Widget y is passed by reference. The int x is passed by value.
Stack for Call Parameters

• push \( x \)
Stack for Call Parameters

- push $x$
- push address of $y$

<table>
<thead>
<tr>
<th>5 (value of $x$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>address of Widget $y$</td>
</tr>
</tbody>
</table>
Stack for Call

- push x
- push address of y
- call thefunc

Stack diagram:

5 (value of x)
address of Widget y
return address
Stack with Activation Records

- push \( x \)
- push address of \( y \)
- call thefunc
- Link to previous activation record

<table>
<thead>
<tr>
<th>5 (value of x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>address of Widget ( y )</td>
</tr>
<tr>
<td>return address</td>
</tr>
<tr>
<td>addr of prev act rec</td>
</tr>
</tbody>
</table>
Stack Use by Function

• push x
• push address of y
• call thefunc
• Link to previous activation record
• increment stack

<table>
<thead>
<tr>
<th>5 (value of x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>address of Widget y</td>
</tr>
<tr>
<td>return address</td>
</tr>
<tr>
<td>addr of prev act rec</td>
</tr>
<tr>
<td>local variable r</td>
</tr>
</tbody>
</table>
Stack for Return

- push \( x \)
- push address of \( y \)
- call \texttt{thefunc}
- Link to previous activation record
- increment stack
- decrement stack
Stack for Return

• push x
• push address of y
• call thefunc
• Link to previous activation record
• increment stack
• decrement stack
• return
Cleanup Stack

- push x
- push address of y
- call thefunc
- increment stack
- decrement stack
- return
- decrement stack by 2
Linked Stacks

• Some systems use a doubly linked list to simulate a stack
• Upon entry to a method, a block of memory is acquired which is linked to the previous block
• This block of memory contains the register save area
• Upon exit, the registers are restored and the block released
Activation Records

• An activation record or frame contains the stack information for a method call
• The activation records are linked together
## Activation Record Format

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>locals</td>
<td>The local variables of the method. This can vary in size.</td>
</tr>
<tr>
<td>Frame pointer</td>
<td>The address of the previous activation frame.</td>
</tr>
<tr>
<td>Return address</td>
<td>The address of the instruction after the method call in the calling program.</td>
</tr>
<tr>
<td>parameter 1</td>
<td>The first parameter to the method</td>
</tr>
<tr>
<td>parameter 2</td>
<td>The second parameter to the method</td>
</tr>
</tbody>
</table>
Finding the Activation Record

• In the Windows / Intel world, the EBP register points to the activation record
• Local variables are located on the stack and accessed using the EBP register as an index
Reading

• In the next class we will discuss more about parameter passing

Read:

• textbook section 9.3
• https://en.wikipedia.org/wiki/Parameter_(computer_programming)  (if you have not already done so.)