Systems I

Machine-Level Programming VI: Procedures

Topics
- Register saving conventions
- Creating pointers to local variables
IA32/Linux Stack Frame

Current Stack Frame ("Top" to Bottom)
- Parameters for function about to call
  - “Argument build”
- 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

Frame Pointer (%ebp)
Saved Registers + Local Variables
Arguments
Return Address
Old %ebp
Argument Build
Stack Pointer (%esp)
Revisiting swap

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

int zip1 = 15213;
int zip2 = 91125;

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

Calling swap from call_swap

call_swap:
```
    pushl $zip2     # Global Var
    pushl $zip1     # Global Var
    call swap
    ...
```

Resulting Stack

```
    &zip2
    &zip1
    Rtn adr
    %esp
```
Revisiting \texttt{swap}

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

\texttt{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 %ebp,%esp
popl %ebp
ret
```

\texttt{Set Up}

\texttt{Body}

\texttt{Finish}
swap Setup #1

Entering Stack

Resulting Stack

\[\text{pushl } \%\text{ebp} \]
\[\text{movl } \%\text{esp},\%\text{ebp} \]
\[\text{pushl } \%\text{ebx} \]
swap Setup #2

Entering Stack

Resulting Stack

swap:
pushl %ebp
movl %esp,%ebp
pushl %ebx
swap Setup #3

Entering Stack

\[
\begin{array}{l}
\text{\textbullet} \\
\text{\textbullet} \\
\text{\textbullet} \\
\text{&zip2} \\
\text{&zipl} \\
\text{Rtn adr} \\
\end{array}
\]

\[
%ebp
\]

\[
%esp
\]

Resulting Stack

\[
\begin{array}{l}
\text{\textbullet} \\
\text{\textbullet} \\
\text{\textbullet} \\
\text{YP} \\
\text{xp} \\
\text{Rtn adr} \\
\text{Old %ebp} \\
\text{Old %ebx} \\
\end{array}
\]

\[
%ebp
\]

\[
%esp
\]

\[
%ebp
\]

\[
%esp
\]

\[
%esp
\]

swap:

\[
pushl \ %ebp
\]

\[
movl \ %esp,\ %ebp
\]

\[
pushl \ %ebx
\]
Effect of swap Setup

Entering Stack

Resulting Stack

Offset (relative to %ebp)

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

Body
swap Finish #1

Observation
- Saved & restored register %ebx

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

swap’s Stack

Offset
12 yp
8 xp
4 Rtn adr
0 Old %ebp
-4 Old %ebx

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

swap’s Stack

Offset
12 yp
8 xp
4 Rtn adr
0 Old %ebp

%ebp
%esp
swap Finish #3

swap’s Stack

Offset
12 YP
8 xp
4 Rtn adr
0 Old %ebp

%ebp
%esp

swap’s Stack

Offset
12 YP
8 xp
4 Rtn adr

%ebp
%esp

movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
ret
Observation

- Saved & restored register `%ebx`
- Didn’t do so for `%eax`, `%ecx`, or `%edx`
Register Saving Conventions

When procedure `yoo` calls `who`:
- `yoo` is the caller, `who` is the callee

Can Register be Used for Temporary Storage?

**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{you} calls \texttt{who}:
- \texttt{you} is the \textit{caller}, \texttt{who} is the \textit{callee}

Can Register be Used for Temporary Storage?

Conventions
- “Caller Save”
  - Caller saves temporary in its frame before calling
- “Callee Save”
  - Callee saves temporary in its frame before using
Integer Registers

- Two have special uses
  - %ebp, %esp
- Three managed as callee-save
  - %ebx, %esi, %edi
  - Old values saved on stack prior to using
- Three managed as caller-save
  - %eax, %edx, %ecx
  - Do what you please, but expect any callee to do so, as well
- Register %eax also stores returned value
Recursive Factorial

```c
int rfact(int x) {
    int rval;
    if (x <= 1)
        return 1;
    rval = rfact(x-1);
    return rval * x;
}
```

Registers

- `%eax` used without first saving
- `%ebx` used, but save at beginning & restore at end

```asm
.globl rfact
.type
rfact,@function
rfact:
pushl %ebp
movl %esp,%ebp
pushl %ebx
movl 8(%ebp),%ebx
cmpl $1,%ebx
jle .L78
leal -1(%ebx),%eax
pushl %eax
call rfact
imull %ebx,%eax
jmp .L79
.align 4
.L78:
movl $1,%eax
.L79:
movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
ret
```
Rfact Stack Setup

Entering Stack

rfact:
  pushl %ebp
  movl %esp,%ebp
  pushl %ebx
int rfact(int x) {
   int rval;
   if (x <= 1)
      return 1;
   rval = rfact(x-1);
   return rval * x;
}

movl 8(%ebp),%ebx  # ebx = x
cmp $1,%ebx       # Compare x : 1
jle .L78          # If <= goto Term
leal -1(%ebx),%eax # eax = x-1
pushl %eax        # Push x-1
call rfact        # rfact(x-1)
imull %ebx,%eax   # rval * x
jmp .L79          # Goto done

.L78:               # Term:
   movl $1,%eax    # return val = 1
.L79:               # Done:

%ebx  Stored value of x
%eax

- Temporary value of x-1
- Returned value from rfact(x-1)
- Returned value from this call
Rfact Recursion

leal -1(%ebx),%eax

pushl %eax

call rfact
Return from Call

Assume that \texttt{rfact}(x-1) returns \((x-1)!\) in register \%eax
Rfact Completion

pre %ebp
pre %ebx

Rtn adr

x

Old %ebp

%ebp

%esp

Old %ebx

x-1

%eax  x!
%ebx  Old %ebx

%eax  x!
%ebx  Old %ebx

movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
ret
Pointer Code

Recursive Procedure

```c
void s_helper
    (int x, int *accum)
{
    if (x <= 1)
        return;
    else {
        int z = *accum * x;
        *accum = z;
        s_helper (x-1, accum);
    }
}
```

Top-Level Call

```c
int sfact(int x)
{
    int val = 1;
    s_helper(x, &val);
    return val;
}
```

- Pass pointer to update location
Creating & Initializing Pointer

Initial part of \texttt{sfact}

\begin{verbatim}
_sfact:
pushl %ebp     # Save %ebp
movl %esp, %ebp # Set %ebp
subl $16, %esp  # Add 16 bytes
movl 8(%ebp), %edx # edx = x
movl $1, -4(%ebp) # val = 1
\end{verbatim}

Using Stack for Local Variable

\begin{itemize}
    \item Variable \texttt{val} must be stored on stack
        \begin{itemize}
            \item Need to create pointer to it
        \end{itemize}
    \item Compute pointer as $-4(\%ebp)$
    \item Push on stack as second argument
\end{itemize}

\begin{verbatim}
int sfact(int x)
{
    int val = 1;
    s_helper(x, &val);
    return val;
}
\end{verbatim}
# Passing Pointer

## Calling `s_helper` from `sfact`

### Code Snippet

```c
int sfact(int x)
{
    int val = 1;
    s_helper(x, &val);
    return val;
}
```

### Assembly Code

```assembly
leal -4(%ebp),%eax # Compute &val
pushl %eax        # Push on stack
pushl %edx        # Push x
call s_helper     # call
movl -4(%ebp),%eax # Return val
...               # Finish
```

### Stack Diagram

- **8**: `x`
- **4**: `Rtn adr`
- **0**: `Old %ebp`
- **-4**: `val = x!`
- **-8**: `Unused`
- **-12**: `&val`
- **-16**: `x`

---

24
Using Pointer

void s_helper (int x, int *accum) {
    ...
    int z = *accum * x;
    *accum = z;
    ...
}

- Register %ecx holds x
- Register %edx holds pointer to accum
  - Use access (%edx) to reference memory
Summary

The Stack Makes Recursion Work

- Private storage for each *instance* of procedure call
  - Instantiations don’t clobber each other
  - Addressing of locals + arguments can be relative to stack positions
- Can be managed by stack discipline
  - Procedures return in inverse order of calls

IA32 Procedures Combination of Instructions + Conventions

- Call / Ret instructions
- Register usage conventions
  - Caller / Callee save
  - `%ebp` and `%esp`
- Stack frame organization conventions