Data Segment and Linking

Source

- IA OS
- IB C and C++
    - `static`, `auto`, `extern`
- IB Compiler Construction
    - Compilers Principles, Techniques and Tools(2013)
        - Ch 7 Runtime Environments
    - 2014p3q4
- IB Concurrent System
    - Threads
- Adapted from Wiki Data segment

Symbol table

```c
// Declare an external function
extern double bar(double x);

// Define a public function
double foo(int count)
{
    double sum = 0.0;

    // Sum all the values bar(1) to bar(count)
    for (int i = 1; i <= count; i++)
        sum += bar((double) i);
    return sum;
}
```

Symbol name	Type	Scope
bar	function, double	extern
x	double	function parameter
foo	function, double	global
count	int	function parameter
sum	double	block local
i	int	for-loop statement

Memory

From lower to higher parts of memory (address space)

1. Code / Text segment

Hold program code and constants (string literals).

- read-only and fixed size

```
char * message = "This is a string literal.";
```

Note: Code is shared by all threads, shared libraries, and dynamically loaded modules in a process.

2. Static Data

Global variables

- `static`
  - preventing variables or function from being called externally
- `extern`
  - symbol tables will `export` them
  - linker will resolve symbols (match `inputs` & `exports`)

Local `static` variables

- retain its value between function calls

Note:

- Global variables are shared by all threads, shared libraries, and dynamically loaded modules in a process.

2a. Initialized static data segment

```
(extern) int i = 3;
static int b = 2023;
void foo (void) {
    static int c = 2023;
}
```

2b. Uninitialized static data segment / Block Starting Symbol

Hold variables above and constants.

- that do not have explicit initialization in source code.
- will be initialized to zero in C by `exec`.

```
static int i;
static char a[12];
```

3. Heap

Hold dynamically allocated memory

- `malloc`, `calloc`, `realloc`, and `free`
    - which may use the `brk` and `sbrk` system calls to adjust its size (`mmap`/`munmap` to reserve/unreserve potentially non-contiguous regions of virtual memory into the process' virtual address space).

```
ptr = (int*)malloc(n * sizeof(int));
free (ptr);
```

Heap	Note
v2	Higher addresses
v1	
HT_CLOSURE	Lower addresses

- begins at the end of the BSS segment and grows to larger addresses from there.
- The heap segment is shared by all threads, shared libraries, and dynamically loaded modules in a process.
- Garbage Collection enables the run-time system to detect useless data elements and reuse their storage.

4. Stack

Hold `auto` variables are also allocated on the stack.

- function parameters, local variables
    - when the procedure is called, space for its local variables are pushed onto a stack
    - when the procedure terminates, that space is popped off the stack

```
void f(int k){
    k++;
}

void main() {
    (auto) int j = 3;
    f(j);
}
```

activation record / stack frame, various types of linkage information

- the set of values pushed for
    - lock entry and exit
    - procedure (function, method or subroutine) call and return
- **frame pointer** (fp) register serves as an *anchor point*
    - points to special location *within* an activation record
    - used as a fixed reference point to locate other elements of the frame
- **stack pointer** (sp) register
    - points to the *top* of the stack
    - when a value is pushed onto the stack, adjust the stack pointer each time
    - when a block is exited or function returns, adjust the stack pointer to the dynamic/control link.
- thread pointers

Stack Frame	**Note**
code pointer↓ (when caller apply)	return address
-- Previous Frame --	Higher addresses
	← frame pointer (fp)
dynamic/control link	return link to previous fp
saved machine status [**return addr**, registers]	before the call value of PC
access/static link	non-local free var at other scopes
return values	optional (void)
local data	optional (...)
temporaries	outside registers
parameters	optional (void)
	← stack pointer (sp)
-- Next Frame --	Lower addresses

static/access link

- point to the most recent stack frame of enclosing scope
    - for nested functions, access the non local free variables stored in previous activation records in the stack
    - If not lexically contained in any other function, its static link is null.
    - There may be multiple frames of the same function, so point to the most recent one.

```c
int i = 1;
void f(void){
    printf("%d",i);
}
```

static chain

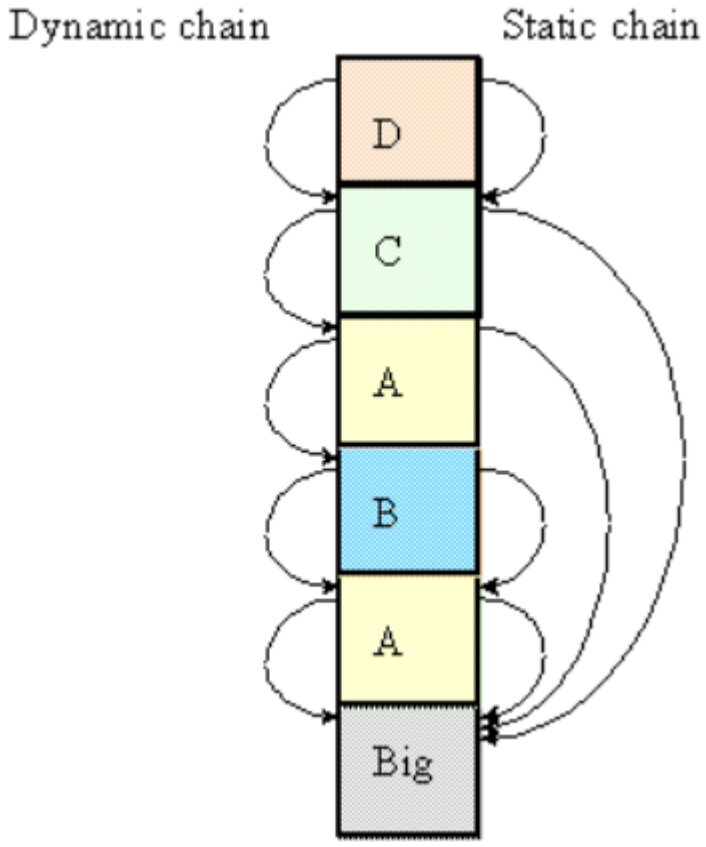
- frames linked together with static links
- chaining together activation records, from blocks deeply nested to blocks less deeply nested.

dynamic/control link

- points to the previous frame (the caller's frame)
- when we pop the callee from the stack
    - set the stack pointer to be the value of the dynamic link of the current frame

dynamic chain

- all frames are linked together in the stack using dynamic links.



Note:

A callee should not make any assumptions about who is the caller because there may be many different functions that call the same function.

The frame layout in the stack should reflect this. When we execute a function, its frame is located on top of the stack. The function does not have any knowledge about what the previous frames contain.

- LIFO structure, growing towards the heap
- when the stack pointer met the heap pointer, free memory was exhausted.
- per-thread stack pointers and program counters

Memory Layout

Addr	Data Segment		ELF	Note
0x 8000	`int argc, char *argv[]`	command-line		
	Stack ↓	auto and linkage		
	----Free ---			
	Heap ↑	dynamic		
	Uninitialized static .bss	variables const	.data .rodata	initialized to 0 by `exec`
	Initialized static data	variables	.data	
0x 0000	Code / Text	program, const	.text	