

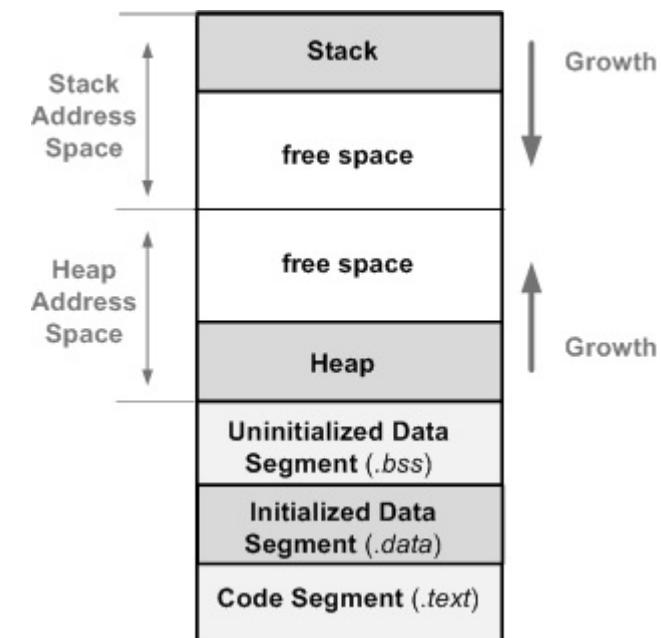
Memory Management in C

Yizhou Wang

<wangyzh2024>

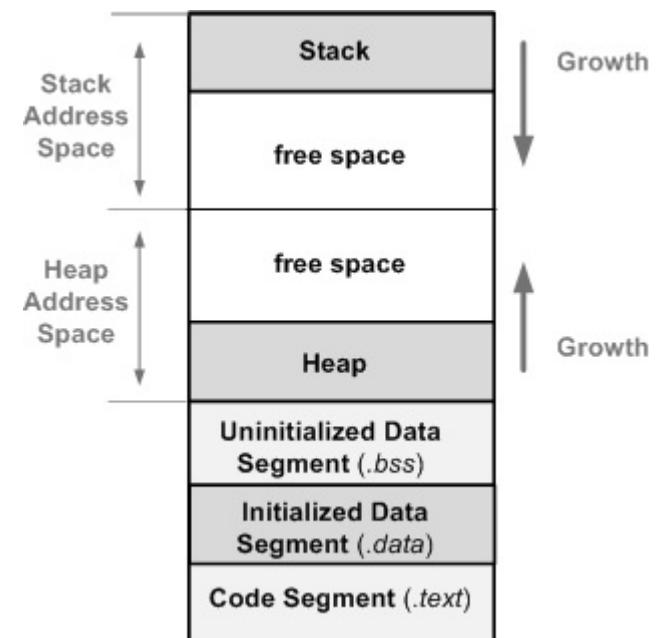
Recall: Memory Layout

- Stack growing downward
 - Function arguments
 - Return addresses
 - Local variables
- Heap growing upward
 - For dynamic demands
- .bss for global/static initially being zero
- .data for global/static with initial values
- .text for machine code



Why Bother?

- Different lifespans of variable
 - global/static: entire execution period
 - local/args/RA: within the function
 - dynamic: undeterminable
- Smaller program size
 - Many global/static start as zero, wasteful if stored explicitly
- Portable to non-von Neumann
 - e.g., separate devices for code, var, const



An Example: .bss vs .data

```
yitro@RM-server ~/tmp/disc3 07:58
[526] $ cat layout.c
int a = 1, b = 0, c;
int main() {
    int d;
    static int e = 1, f = 0, g;
    return 0;
}
yitro@RM-server ~/tmp/disc3 07:58
[527] $ objdump --syms a.out | grep '\.bss\|\.data'
000000000004024 l    0 .bss  0000000000000004          g.2
000000000004028 l    0 .bss  0000000000000004          f.1
000000000004014 l    0 .data 0000000000000004          e.0
000000000004000    w    .data 0000000000000000          data_start
00000000000401c g    0 .bss  0000000000000004          b
000000000004018 g    .data 0000000000000000          _edata
000000000004000 g    .data 0000000000000000          _data_start
000000000004008 g    0 .data 0000000000000000          .hidden __dso_handle
000000000004030 g    .bss  0000000000000000          _end
000000000004020 g    0 .bss  0000000000000004          c
000000000004010 g    0 .data 0000000000000004          a
000000000004018 g    .bss  0000000000000000          _bss_start
000000000004018 g    0 .data 0000000000000000          .hidden __TMC_END__
```

Out-of-range!

- Bad in global/static
 - Secretly corrupt nearby variables
- Worse in heap
 - Secretly corrupt who-knows-where vars
 - May corrupt heap management
 - If allocation records are kept in heap
- Worst in stack!
 - Secretly corrupt nearby variables, arguments, callers' arguments, and even **return addresses**

```
ytro@RM-server ~/tmp/dic3 08:54
[558] $ cat global-overrun.c
#include <stdio.h>
int a[1] = {5}, b = 6;
int main() {
    a[1] = 0;
    printf("%d\n", b);
    return 0;
}
ytro@RM-server ~/tmp/dic3 08:54
[559] $ gcc global-overrun.c && ./a.out
0
```

```
ytro@RM-server ~/tmp/dic3 08:49
[553] $ cat stack-overrun.c
#include <stdio.h>
#include <stdlib.h>
void g() { puts("Surprise!"); exit(1); }
void f() { void(*a[1])(); a[3] = g; }
int main() { f(); return 0; }
ytro@RM-server ~/tmp/dic3 08:49
[554] $ gcc stack-overrun.c && ./a.out
Surprise!
```

Side Note: Undefined Behaviors

- “Out-of-range array subscripting is an undefined behavior.”
- What is UB?
- **Myth:** Programs with UB always die ugly.
- Consequence of undefined behaviors is undefined
 - Might just work as if there are no UB.
 - Might start/stop working because of one added/removed `printf()`
 - Might work on Windows, fail on Linux and fail differently on Mac OS
 - Windows is much more tolerant to memory-related UB than Linux

Management: .bss and .data

- Allocated and associated at compile-time
- Die with the process
- Not freed/reclaimed at all

Management: Stack

- Managed implicitly by compiler
 - Allocated on demand and reclaimed on leaving scope
 - Allocations are calculated at compile-time
- Generally, in a consistent manner, except:
 - Variadic functions, e.g., `scanf()` and `printf()`
 - Indefinite number of arguments
 - Callers explicitly provide hints, e.g. format strings
 - Goes wrong on incorrect hints
 - Variable Length Array (VLA) “`int a[n];`”
 - Hard to use safely! Starting from C11, only **optionally** supported.

Related Issues: Stack

- Local variables are uninitialized by default
 - Their values depend on the trashes left in stack
 - May from other processes or even OS
 - Some OS intentionally fills stack with special values to aid debugging
 - Mitigation: Always initialize at definition
- Local variables are reclaimed on leaving scope
 - Using them outside (e.g., returned as pointer) is UB
 - Mitigation: Don't do it. Enforce with warning options.

Management: Heap

- Managed explicitly by programmer
 - Allocate using `malloc()`, `calloc()`
 - Reclaim using `free()`
 - Reallocate using `realloc()` to change size
- `calloc()` fills the allocated space with zero and catches multiplication overflow in C11

Related Issues: Heap

- Use after freed
 - Access the piece of memory already freed
 - If allocated to others, data will be corrupted
 - Random crimes are hard to investigate
 - Mitigation: Set a freed pointer to NULL
- Double free
 - Free an already freed allocation
 - Mitigation: Set a freed pointer to NULL
 - **Fact:** Freeing NULL is well defined as doing nothing.
 - Or just let it die?

Related Issues: Heap

- Memory leak
 - Inaccessible allocated pieces not freed
 - Mainly due to address lost
 - Carelessly overwritten in an assignment
 - The variable holding the address reached end-of-life
 - Mitigation:
 - Think twice when assigning to a pointer
 - Do cleanup at every exits of the scope
 - Reclaim the associated resources before freeing a heap variable

Side Note: Warning Options

- Controlling for what the compiler should warn you
- E.g., `-Wall -Wpedantic -Werror -Wno-unused-result`

```
Warning Options
  -fsyntax-only  -fmax-errors=n  -pedantic  -pedantic-errors  -w  -extra  -Wall  -abi=n  -address  -Wno-address-of-packed-member  -Waggregate-return  -Walloc-size-larger-than-byte-size
  -Walloc-zero  -Walloca  -Walloca-larger-than-byte-size  -Wno-aggressive-loop-optimizations  -Warith-conversion  -Warray-bounds  -Warray-bounds=n  -Warray-compare  -Wno-attributes  -Wattribute-alias=n
  -Wno-attribute-alias  -Wno-attribute-warning  -Wbidi-chars=[none|unpaired|any|ucn]  -Wbool-compare  -Wbool-operation  -Wno-built-in-declaration-mismatch  -Wno-built-in-macro-redefined  -Wc90-c99-compat
  -Wc99-c11-compat  -Wc11-c2x-compat  -Wc++-compat  -Wc++11-compat  -Wc++14-compat  -Wc++17-compat  -Wc++20-compat  -Wno-c++11-extensions  -Wno-c++14-extensions  -Wno-c++17-extensions
  -Wno-c++20-extensions  -Wno-c++23-extensions  -Wcast-align  -Wcast-align=strict  -Wcast-function-type  -Wcast-qual  -Wchar-subscripts  -Wclobbered  -Wcomment  -Wno-complain-wrong-lang  -Wconversion
  -Wno-coverage-mismatch  -Wno-cpp  -Wdangling-else  -Wdangling-pointer  -Wdangling-pointer=n  -Wdate-time  -Wno-deprecated  -Wno-deprecated-declarations  -Wno-designated-init  -Wdisabled-optimization
  -Wno-discarded-array-qualifiers  -Wno-discarded-qualifiers  -Wno-div-by-zero  -Wdouble-promotion  -Wduplicated-branches  -Wduplicated-cond  -Wempty-body  -Wno-endif-labels  -Wenum-compare
  -Wenum-conversion  -Wenum-int-mismatch  -Werror  -Werror=*  -Wexpansion-to-defined  -Wfatal-errors  -Wfloat-conversion  -Wfloat-equal  -Wformat  -Wformat=2  -Wno-format-contains-nul
  -Wno-format-extra-args  -Wformat-nonliteral  -Wformat-overflow=n  -Wformat-security  -Wformat-signedness  -Wformat-truncation=n  -Wformat-y2k  -Wframe-address  -Wframe-larger-than-byte-size
  -Wno-free-nonheap-object  -Wno-if-not-aligned  -Wno-ignored-attributes  -Wignored-qualifiers  -Wno-incompatible-pointer-types  -Wimplicit  -Wimplicit-fallthrough  -Wimplicit-fallthrough=n
  -Wno-implicit-function-declaration  -Wno-implicit-int  -Winfinite-recursion  -Wint-self  -Winline  -Wno-int-conversion  -Wint-in-bool-context  -Wno-int-to-pointer-cast  -Wno-invalid-memory-model
  -Winvalid-pch  -Winvalid-utf8  -Wno-unicode  -Wjump-misses-init  -Wlarger-than-byte-size  -Wlogical-not-parentheses  -Wlogical-op  -Wlong-long  -Wno-lto-type-mismatch  -Wmain  -Wmaybe-uninitialized
  -Wmemset-elt-size  -Wmemset-transposed-args  -Wmisleading-indentation  -Wmissing-attributes  -Wmissing-braces  -Wmissing-field-initializers  -Wmissing-format-attribute  -Wmissing-include-dirs
  -Wmissing-noreturn  -Wno-missing-profile  -Wno-multichar  -Wmultistatement-macros  -Wnonnull  -Wnonnull-compare  -Wnonnull-[none|id|nfcl|nfkc]  -Wnull-dereference  -Wno-odr  -Wopenacc-parallelism
  -Wopenmp-simd  -Wno-overflow  -Woverlength-strings  -Wno-override-init-side-effects  -Wpacked  -Wno-packed-bitfield-compat  -Wpacked-not-aligned  -Wpadded  -Wparentheses  -Wno-pedantic-ms-format
  -Wpointer-arith  -Wno-pointer-compare  -Wno-pointer-to-int-cast  -Wno-pragmas  -Wno-prio-ctor-dtor  -Wredundant-decls  -Wrestrict  -Wno-return-local-addr  -Wreturn-type  -Wno-scalar-storage-order
  -Wsequence-point  -Wshadow  -Wshadow-global  -Wshadow-local  -Wshadow-compatible-local  -Wno-shadow-tvar  -Wno-shift-count-negative  -Wno-shift-count-overflow  -Wshift-negative-value
  -Wno-shift-overflow  -Wshift-overflow=n  -Wsign-compare  -Wsign-conversion  -Wno-sizeof-array-argument  -Wsizeof-array-div  -Wsizeof-pointer-div  -Wsizeof-pointer-memaccess  -Wstack-protector
  -Wstack-usage-byte-size  -Wstrict-aliasing  -Wstrict-aliasing=n  -Wstrict-overflow  -Wstrict-overflow=n  -Wstring-compare  -Wno-stringop-overflow  -Wno-stringop-overread  -Wno-stringop-truncation
  -Wstrict-flex-arrays  -Wsuggest-attribute=[pure|const|noreturn|format|malloc]  -Wswitch  -Wno-switch-bool  -Wswitch-default  -Wswitch-enum  -Wno-switch-outside-range  -Wno-switch-unreachable
  -Wsync-nand  -Wsystem-headers  -Wtautological-compare  -Wtrampolines  -Wtrigraphs  -Wtrivial-auto-var-init  -Wtsan  -Wtype-limits  -Wundef  -Wuninitialized  -Wunknown-pragmas
  -Wunsuffixified-float-constants  -Wunused  -Wunused-but-set-parameter  -Wunused-but-set-variable  -Wunused-const-variable  -Wunused-const-variable  -Wunused-function  -Wunused-label
  -Wunused-local-typedefs  -Wunused-macros  -Wunused-parameter  -Wno-unused-result  -Wunused-value  -Wunused-variable  -Wno-varargs  -Wvariadic-macros  -Wvector-operation-performance  -Wvla
  -Wvla-larger-than-byte-size  -Wno-vla-larger-than  -Wvolatile-register-var  -Wwrite-strings  -Wxor-used-as-pow  -Wzero-length-bounds

Static Analyzer Options
  -fanalyzer  -fanalyzer-call-summaries  -fanalyzer-checker=name  -fno-analyzer-feasibility  -fanalyzer-fine-grained  -fno-analyzer-state-merge  -fno-analyzer-state-purge
  -fno-analyzer-suppress-followups  -fanalyzer-transitivity  -fno-analyzer-undo-inlining  -fanalyzer-verbose-edges  -fanalyzer-verbose-state-changes  -fanalyzer-verbosity-level  -fdump-analyzer
  -fdump-analyzer-callgraph  -fdump-analyzer-explored-graph  -fdump-analyzer-explored-nodes  -fdump-analyzer-explored-nodes-2  -fdump-analyzer-explored-nodes-3  -fdump-analyzer-explored-paths
  -fdump-analyzer-feasibility  -fdump-analyzer-json  -fdump-analyzer-state-purge  -fdump-analyzer-stder  -fdump-analyzer-supergraph  -fdump-analyzer-untracked  -Wno-analyzer-double-fclose
  -Wno-analyzer-double-free  -Wno-analyzer-exposure-through-output-file  -Wno-analyzer-exposure-through-unit-copy  -Wno-analyzer-fd-type-mismatch  -Wno-analyzer-fd-access-mode-mismatch  -Wno-analyzer-fd-double-close
  -Wno-analyzer-fd-leak  -Wno-analyzer-fd-phase-mismatch  -Wno-analyzer-fd-type-mismatch  -Wno-analyzer-fd-use-after-close  -Wno-analyzer-fd-use-without-check  -Wno-analyzer-file-leak
  -Wno-analyzer-free-of-non-heap  -Wno-analyzer-imprecise-fp-arithmetic  -Wno-analyzer-infinite-recursion  -Wno-analyzer-jump-through-null  -Wno-analyzer-malloc-leak
  -Wno-analyzer-mismatching-deallocation  -Wno-analyzer-null-argument  -Wno-analyzer-null-dereference  -Wno-analyzer-out-of-bounds  -Wno-analyzer-possible-null-argument
  -Wno-analyzer-possible-null-dereference  -Wno-analyzer-putenv-of-auto-var  -Wno-analyzer-shift-count-negative  -Wno-analyzer-shift-count-overflow  -Wno-analyzer-stale-setjmp-buffer
  -Wno-analyzer-tainted-allocation-size  -Wno-analyzer-tainted-assertion  -Wno-analyzer-tainted-array-index  -Wno-analyzer-tainted-divisor  -Wno-analyzer-tainted-offset  -Wno-analyzer-tainted-size
  -Wno-analyzer-too-complex  -Wno-analyzer-unsafe-call-within-signal-handler  -Wno-analyzer-use-after-free  -Wno-analyzer-use-of-pointer-in-stale-stack-frame  -Wno-analyzer-use-of-uninitialized-value
  -Wno-analyzer-va-arg-type-mismatch  -Wno-analyzer-va-list-exhausted  -Wno-analyzer-va-list-leak  -Wno-analyzer-va-list-use-after-va-end  -Wno-analyzer-write-to-const
  -Wno-analyzer-write-to-string-literal
```

Side Note: Warning Options

- Pros
 - Second chances to notice things stupid.
 - Timely reminders of things you might not fully know
 - Sincere hints for improvement in coding style
- Cons
 - Too wordy and anxious when overly enabled
 - Complaining even common daily practices
 - Comparing int and size_t
 - Unused function results, including printf()'s

Valgrind --tool=memcheck

Detecting

- Illegal memory access
 - Overrunning, underrunning, use-after-freed
- Use of uninitialized values
- Double free of heap blocks
- Fishy size parameter to allocation function
- Memory leaks
- ...

Illegal Access

```
int *f(void) { int a; return &a; }
int main() { *f() = 1; return 0; }
```

```
==78779== Invalid write of size 4
```

```
==78779==     at 0x10918C: main (in /home/yitro/tmp/dsc3/valgrind/a.out)
```

```
==78779== Address 0x0 is not stack'd, malloc'd or (recently) free'd
```

Use of Uninitialized

```
int y = 0;  
int main() {  
    int x;  
    if (x == 42) y += 8;  
    return y; }
```

```
==11956== Conditional jump or move depends on uninitialized value(s)  
==11956==      at 0x109135: main (in /home/yitro/tmp/disc3/valgrind/a.out)
```

Illegal Free

```
#include <stdlib.h>

int main() { void *buf = malloc(0); free(buf), free(buf); }

==16354== Invalid free() / delete / delete[] / realloc()
==16354==       at 0x484495F: free (vg_replace_malloc.c:989)
==16354==       by 0x10919A: main (in /home/yitro/tmp/disc3/valgrind/a.out)
==16354==   Address 0x4a31040 is 0 bytes after a block of size 0 free'd
==16354==       at 0x484495F: free (vg_replace_malloc.c:989)
==16354==       by 0x10918E: main (in /home/yitro/tmp/disc3/valgrind/a.out)
==16354==   Block was alloc'd at
==16354==       at 0x4841878: malloc (vg_replace_malloc.c:446)
==16354==       by 0x10917E: main (in /home/yitro/tmp/disc3/valgrind/a.out)
```

Fishy Size

```
#include <stdlib.h>
void main() { malloc(-1); }

==28552== Argument 'size' of function malloc has a fishy (possibly
negative) value: -1
==28552==      at 0x4841878: malloc (vg_replace_malloc.c:446)
==28552==      by 0x10915C: main (in /home/yitro/tmp/disc3/valgrind/a.out)
```

Memory Leaks --leak-check

```
#include <stdlib.h>                                char **a;
void f(void) {                                         int main() {
    char *c = malloc(10);                            f();
    char **b = malloc(9);                            a = malloc(8);
    *b = c;                                         *a = g();
}                                                       return 0;
char *g(void) {                                         }
    char *d = malloc(11);
    return d + 1;
}
```

Memory Leaks --leak-check

```
==34098== LEAK SUMMARY:
```

```
==34098==     definitely lost: 9 bytes in 1 blocks
```

```
==34098==     indirectly lost: 10 bytes in 1 blocks
```

```
==34098==     possibly lost: 11 bytes in 1 blocks
```

```
==34098==     still reachable: 8 bytes in 1 blocks
```

```
==34098==           suppressed: 0 bytes in 0 blocks
```

Memory Leaks --leak-check=full

```
==34098== 11 bytes in 1 blocks are possibly lost in loss record 3 of 4
==34098==       at 0x4841878: malloc (vg_replace_malloc.c:446)
==34098==       by 0x109194: g (in /home/yitro/tmp/dsc3/valgrind/a.out)
==34098==       by 0x1091D1: main (in /home/yitro/tmp/dsc3/valgrind/a.out)
==34098==

==34098== 19 (9 direct, 10 indirect) bytes in 1 blocks are definitely
lost in loss record 4 of 4
==34098==       at 0x4841878: malloc (vg_replace_malloc.c:446)
==34098==       by 0x10916C: f (in /home/yitro/tmp/dsc3/valgrind/a.out)
==34098==       by 0x1091B4: main (in /home/yitro/tmp/dsc3/valgrind/a.out)
```