Pointers

Héctor Menéndez[1]

AIDA Research Group
Computer Science Department
Universidad Autónoma de Madrid

November 28, 2013

[1] based on the original slides of the subject

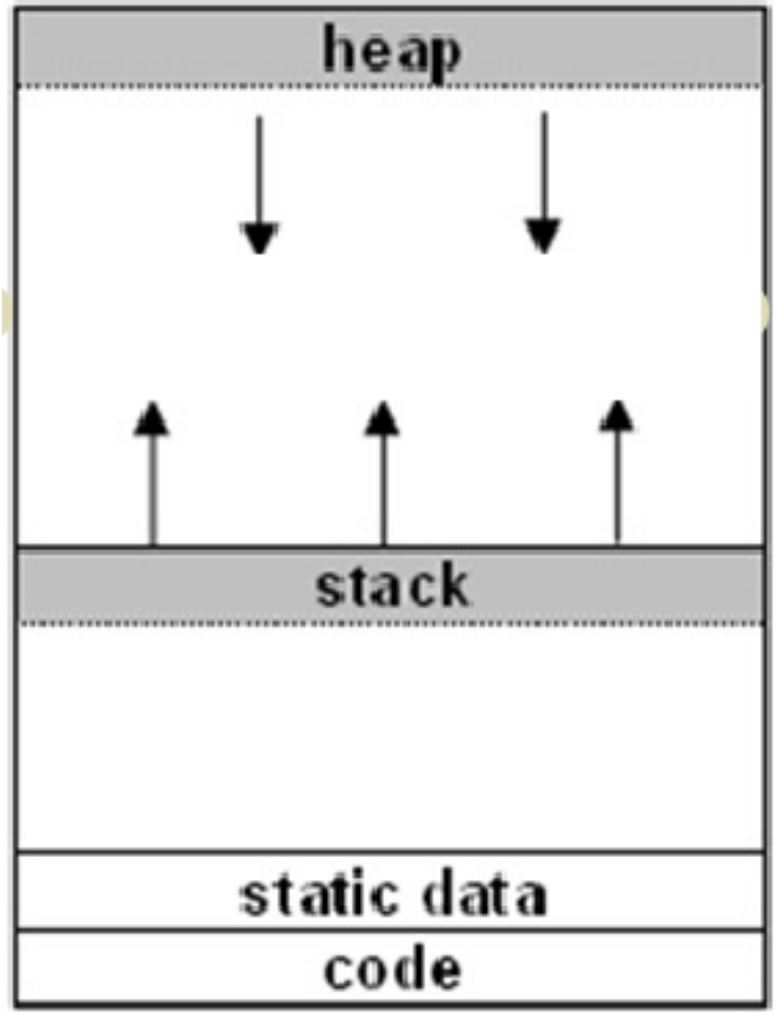
Index

Index

Dynamic Memory

- When a program is run, 4 memory segments are created:
  - **Code**: Contains the instructions.
  - **Static Data**: Variables and constants.
  - **Stack**: Function variables and function call.
  - **Heap**: Dynamic Memory segment.

Dynamic Memory



Dynamic Memory

- Existen varias funciones para trabajar con memoria dinámica:

```
void *malloc (size_t num_bytes);
void *calloc (size_t num_ítems, size_t size);
void *realloc (void *ptr, size_t size);
void free(void *ptr);
```

- **malloc (Memory ALLOCation)**: creates a block of num_bytes bytes and returns a pointer to that block.
- **calloc**: it is similar to malloc, initialize the block to 0.
- **realloc**: change the block size.
- **malloc**, **calloc** y calloc return a pointer or NULL (error).
- **free**: frees the memory.

Example

```
int n=20;
float *v;
// Dynamic Vector creation
v = malloc(n*sizeof(float));
...
...
free(v);
```

Index

Arrays and Pointers

- The array is used as follows:

```
int v[3];
v[0] = 6;
```

- Pointers are able to used matrices and arrays.

```
int v[3];
int * ptr;
...
ptr = &v[0];
*ptr = 6; // it is the same as: V[0] = 6;
```

- When a pointer is moved it is moving inside the memory.

Arrays and Pointers: Example

```
int vector [ ] = {1, 2, 3, 4};
int index, resultAdd, elements =4;
int * ptr, * ptrE;
resultAdd = 0;

for(index =0; index<elements; index++)
resultAdd += vector[index];

ptr = &vector[0];
for(index =0; index<elements; index++)
resultAdd += *(ptr + index);

ptr = vector;
ptrE = ptr + elements;
for(ptr; ptr<ptrE; ptr++)
```

Index

Matrices and Pointers

- Matrices can be declared as follows:

```
int numRows = 2;
int numCols= 4;
int matrix[numRows][numCols];
```

- And with pointers as follows:

```
int numRows = 2;
int numCols= 4;
int * ptrMatrix = malloc( numRows*numCols*sizeof(int) );
```

Matrices and Pointers: Classic Example

```
int rows = 2;
int cols = 4;
int m [numRows][numCols];
int i, j;
...
for(i=0;i<rows;i++){
  for(j=0;j<cols;j++){
    printf("%d\n", m[i][j]);
  }
}
```

Matrices and Pointers: Pointer Example

```
int rows = 2;
int cols = 4;
int m [numRows][numCols];
...
int * ptrM = *m;
int i, j;

for(i=0;i<rows; i++)
    for(j=0;j<cols;j++)
        printf("%d\n", *(ptrM+i*cols+j) );

for(i=0;i<(rows*cols);i++){
    printf("%d\n", *(ptrM+i) );
}
```

Index

Valgrind

- Valgrind is a set of tools for debugging and profiling.
- Tools:
  - Cachegrind: cache profiler.
  - Callgrind: system call profiler.
  - Massif: heap profiler.
  - Helgrind/DRD: threads debugger.
  - **Memcheck**.
- What are Valgrind goals?
  - It does not need to recompile the code.
  - Useful for code refactoring.

Memcheck

- It analysethe code and detect memory errors, i.e., memory leaks or illegal read/write.
- Disadvantages:
  - The execution runs slowly.
  - Increments memory usage.
- To generate the debug information you need to compile with -g flag.

```
gcc -g myProgram.c -o myExecutable
```

- After, Valgrind is executed as follows:

```
valgrind --leak-check=yes myExecutable arg1 arg2
```

- `-leak-check` is used to detect memory errors.
- This tool is executed by default.

Memcheck: Example

```c
#include <stdlib.h>
#include <stdio.h>

int* f(int size){
    int* x;
    int* y;

    y = malloc(size * sizeof(int));
    x = malloc(size * sizeof(int));
    x[size] = 0; // Error 1: access to an invalid position

    return x;
}  // Error 2: memory assigned to y is not released

int main(){
    int* y = f(10);

    printf("First value: %d\n", y[0]); // Error 3: y is not initialized

    free(y);
    free(y); // Error 4: it is not possible to release the same block of memory more than once

    return 0;
}
```

Memcheck: Example

```
Memcheck, a memory error detector.
Copyright (C) 2002-2006, and GNU GPL'd, by Julian Seward et al.
Using LibVEX rev 1658, a library for dynamic binary translation.
Copyright (C) 2004-2006, and GNU GPL'd, by OpenWorks LLP.
Using valgrind-3.2.1, a dynamic binary instrumentation framework.
Copyright (C) 2000-2006, and GNU GPL'd, by Julian Seward et al.
For more details, rerun with: -v

Invalid write of size 4
    at 0x8048415: f (test_valgrind.c:11)
    by 0x804843C: main (test_valgrind.c:18)
  Address 0x40290A8 is 0 bytes after a block of size 40 alloc'd
    at 0x40053C0: malloc (vg_replace_malloc.c:149)
    by 0x8048408: f (test_valgrind.c:10)
    by 0x804843C: main (test_valgrind.c:18)

 Use of uninitialised value of size 4
    at 0xC72BFB: _itoa_word (in /lib/libc-2.5.so)
    by 0xC76390: vfprintf (in /lib/libc-2.5.so)
    by 0xC7DE42: printf (in /lib/libc-2.5.so)
    by 0x8048454: main (test_valgrind.c:20)

…


 Invalid free() / delete / delete[]
    at 0x4004FDA: free (vg_replace_malloc.c:233)
    by 0x804846A: main (test_valgrind.c:23)
  Address 0x4029080 is 0 bytes inside a block of size 40 free'd
    at 0x4004FDA: free (vg_replace_malloc.c:233)
    by 0x804845F: main (test_valgrind.c:22)

 ERROR SUMMARY: 7 errors from 7 contexts (suppressed: 12 from 1)
 malloc/free: in use at exit: 40 bytes in 1 blocks.
 malloc/free: 2 allocs, 2 frees, 80 bytes allocated.
 For counts of detected errors, rerun with: -v
 searching for pointers to 1 not-freed blocks.
 checked 46,956 bytes.


 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
    at 0x40053C0: malloc (vg_replace_malloc.c:149)
    by 0x80483F7: f (test_valgrind.c:9)
    by 0x804843C: main (test_valgrind.c:18)

 LEAK SUMMARY:
    definitely lost: 40 bytes in 1 blocks.
      possibly lost: 0 bytes in 0 blocks.
    still reachable: 0 bytes in 0 blocks.
         suppressed: 0 bytes in 0 blocks.
 Reachable blocks (those to which a pointer was found) are not shown.
 To see them, rerun with: --show-reachable=yes
```

Memcheck: Example

```
Memcheck, a memory error detector.
Copyright (C) 2002-2006, and GNU GPL'd, by Julian Seward et al.
Using LibVEX rev 1658, a library for dynamic binary translation.
Copyright (C) 2004-2006, and GNU GPL'd, by OpenWorks LLP.
Using valgrind-3.2.1, a dynamic binary instrumentation framework.
Copyright (C) 2000-2006, and GNU GPL'd, by Julian Seward et al.
For more details, rerun with: -v
```

- Program presentation.

```
Invalid write of size 4
    at 0x8048415: f (test_valgrind.c:11)
    by 0x804843C: main (test_valgrind.c:18)
  Address 0x40290A8 is 0 bytes after a block of size 40 alloc'd
    at 0x40053C0: malloc (vg_replace_malloc.c:149)
    by 0x8048408: f (test_valgrind.c:10)
    by 0x804843C: main (test_valgrind.c:18)
```

- Error 1: Invalid write of an integer. It also shows the function where the invalid write is produce.

Memcheck: Example

```
Use of uninitialised value of size 4
   at 0xC72BFB: _itoa_word (in /lib/libc-2.5.so)
   by 0xC76390: vfprintf (in /lib/libc-2.5.so)
   by 0xC7DE42: printf (in /lib/libc-2.5.so)
   by 0x804B454: main (test_valgrind.c:20)
```

- **Error 3**: Invalid read to uninitialised variable.

```
Invalid free() / delete / delete[]
   at 0x4004FDA: free (vg_replace_malloc.c:233)
   by 0x804B46A: main (test_valgrind.c:23)
 Address 0x4029080 is 0 bytes inside a block of size 40 free'd
   at 0x4004FDA: free (vg_replace_malloc.c:233)
   by 0x804B45F: main (test_valgrind.c:22)
```

- **Error 4**: Frees unreserved memory block.

Memcheck: Example

```
40 bytes in 1 blocks are definitely lost in loss record 1 of 1
   at 0x4005300: malloc (vg_replace_malloc.c:149)
   by 0x804B3F7: f (test_valgrind.c:9)
   by 0x804B43C: main (test_valgrind.c:18)
```

- **Error 2**: Not reserved memory block.

```
ERROR SUMMARY: 7 errors from 7 contexts (suppressed: 12 from 1)
malloc/free: in use at exit: 40 bytes in 1 blocks.
malloc/free: 2 allocs, 2 frees, 80 bytes allocated.
For counts of detected errors, rerun with: -v
searching for pointers to 1 not-freed blocks.
checked 46,956 bytes.
```

- ERROR summary.

Memcheck: Example

```
LEAK SUMMARY:
   definitely lost: 40 bytes in 1 blocks.
     possibly lost: 0 bytes in 0 blocks.
   still reachable: 0 bytes in 0 blocks.
        suppressed: 0 bytes in 0 blocks.
Reachable blocks (those to which a pointer was found) are not shown.
To see them, rerun with: --show-reachable=yes
```

- Leaks summary.