

Memory Debugging

Using

Valgrind, Electric Fence, and gdb

William Jones

wjones@parl.clemson.edu

Parallel Architecture Research Laboratory
Clemson University, Clemson, South Carolina

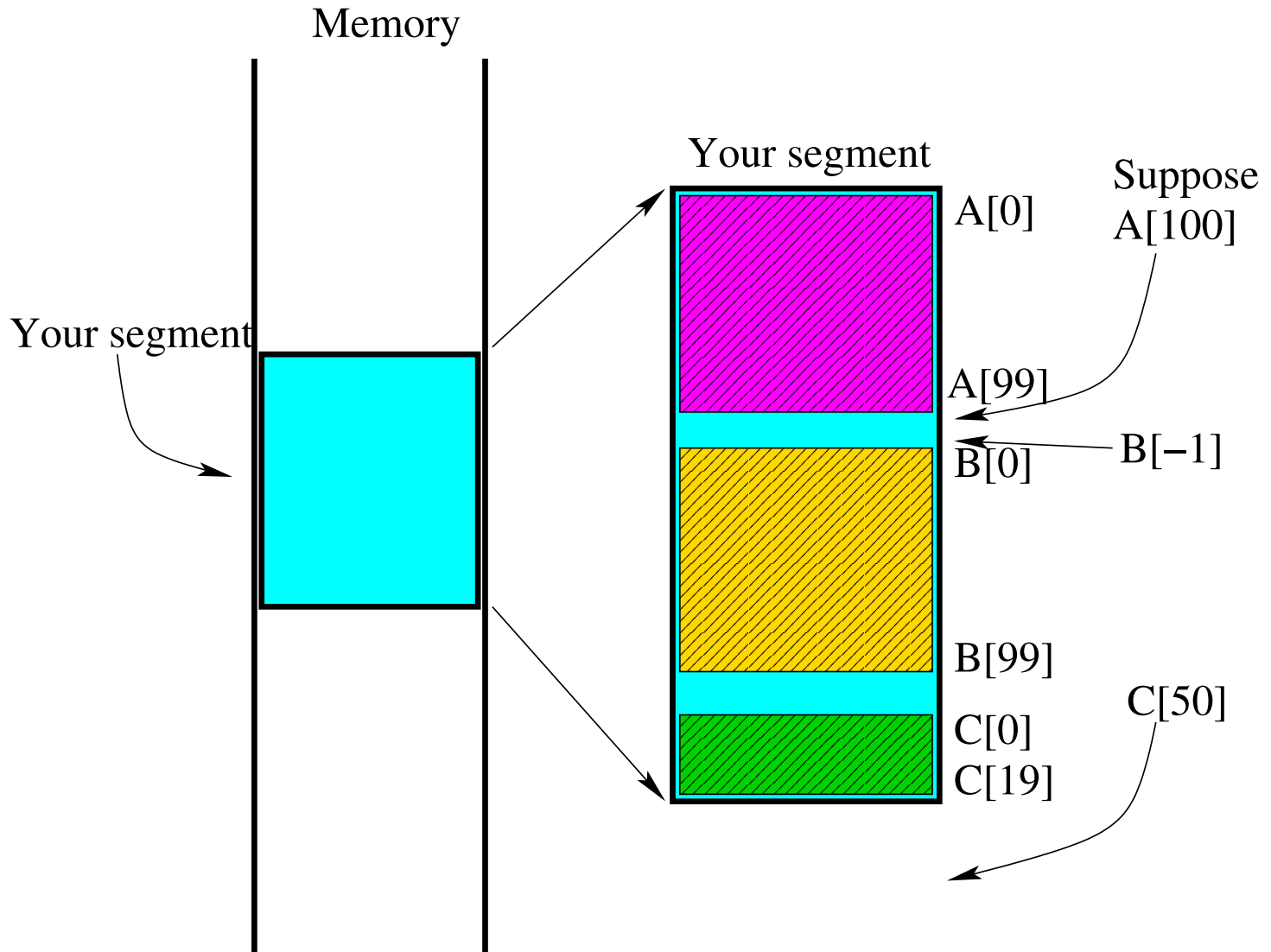
<http://www.parl.clemson.edu>

The Symptoms

- ⇒ Abnormal behavior
- ⇒ Nondeterministic behav.
- ⇒ or more commonly ...

```
[wjones@hades]$ gcc -Wall prog.c  
[wjones@hades]$ a.out  
...  
-- Normal program output  
...  
Segmentation fault  
[wjones@hades]$
```





Typical Memory Bugs

- ⇒ Using uninitialized vars
- ⇒ Using undeclared memory
- ⇒ Using free'd memory
- ⇒ Freeing free'd memory
- ⇒ Malloc'ing size 0 buffers
- ⇒ Overflowing array bounds
- ⇒ Creating memory leaks



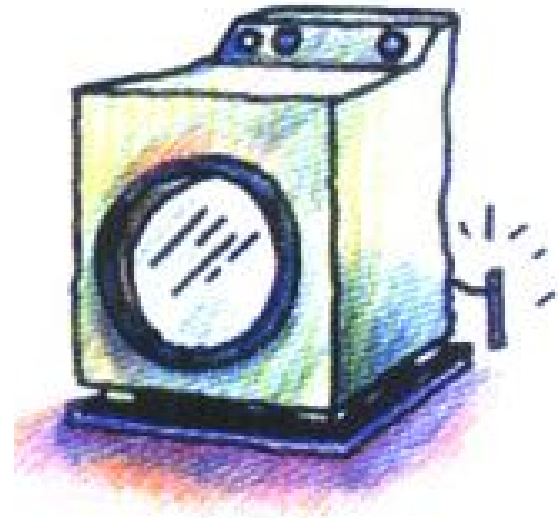
Memory Debugging Tools

- ⇒ Valgrind
 - ❑ Powerful
 - ❑ Easy to use
 - ❑ Needs to be installed
- ⇒ Electric Fence and gdb
 - ❑ Also powerful
 - ❑ More learning curve
 - ❑ Already installed



Valgrind – Cleanse your code!

- <http://valgrind.kde.org>
- cp to /tmp
- *.tar.bz2 → bunzip2, tar xvf
- ./configure, make, make install
(possibly as root)
- valgrind a.out
- Useful flags
- valgrind -help



Electric Fence and gdb

```
[wjones@hades ~]$ gcc -g -Wall -lefence overFlowMallocedArray.c
[wjones@hades ~]$ ./a.out
  Electric Fence 2.2.0 Copyright (C) 1987-1999
Segmentation fault
[wjones@hades ~]$ gdb a.out
(gdb) run
Starting program: /parl/wjones/memoryDebugging/examples/a.out
[New Thread 1024 (LWP 8138)]
  Electric Fence 2.2.0 Copyright (C) 1987-1999
Program received signal SIGSEGV, Segmentation fault.
[Switching to Thread 1024 (LWP 8138)]
0x0804863c in main () at overFlowMallocedArray.c:24
24             intArray[i+1] = i+1;
(gdb) where
```

Let's Look at Some Examples



➤ <http://www.parl.clemson.edu/~wjones/summerStudents>

➤ Download examples

```
[wjones@hades ~]$ tar xvfz examples.tar.gz
```

➤ Reference the line numbers cited by valgrind

Uninitialized Variables

```
[wjones@hades examples]$ gcc -g -Wall usingUninitializedVariables.  
[wjones@hades examples]$ valgrind a.out  
...  
Conditional jump or move depends on uninitialised value(s)  
  at 0x8048470: main (usingUninitializedVariables.c:12)  
  by 0x402471F5: start_main (../sysdeps/generic/libc-start.c:129)  
  by 0x8048380: (in /parl/wjones/memoryDebugging/examples/a.out)  
... program done  
ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)  
malloc/free: in use at exit: 0 bytes in 0 blocks.  
malloc/free: 0 allocs, 0 frees, 0 bytes allocated.  
For a detailed leak analysis, rerun with: --leak-check=yes
```

```
int main(void) {  
  
    int y;  
    /* 12 */    y+=5;  
    if(y>2) {  
        y = 0;  
    }  
    printf(" ... program done \n");  
    return 0;  
}
```

Using Undeclared Memory

```
[wjones@hades examples]$ gcc -g -Wall usingUndeclaredMemory.c
[wjones@hades examples]$ valgrind a.out
...
Use of uninitialised value of size 4
  at 0x8048504: main (usingUndeclaredMemory.c:29)
  by 0x80483B0: (in /parl/wjones/memoryDebugging/examples/a.out)
... program done
ERROR SUMMARY: 10 errors from 1 contexts (suppressed: 0 from 0)
malloc/free: in use at exit: 40 bytes in 1 blocks.
malloc/free: 1 allocs, 0 frees, 40 bytes allocated.
For a detailed leak analysis, rerun with: --leak-check=yes
```

```
/* dimension for toy integer array */
int intArraySize = 10;
/* loop index counter */
int i;
/* pointer to our array */
int* intArray, *intArray2;
/* dynamically declare some memory for our toy integer array */
intArray = (int *) malloc(intArraySize*sizeof(int));
/* loop over the array, and do some computations */
for(i=0; i<intArraySize; i++) {
    intArray[i] = i;
}
/* loop over the array, and do some computations */
for(i=0; i<intArraySize; i++) {
/* 29 */     intArray2[i] = i;
}
printf(" ... program done \n");
```

Using Free'd Memory

```
[wjones@hades examples]$ gcc -g -Wall usingUndeclaredMemory.c
[wjones@hades examples]$ valgrind a.out
...
Invalid write of size 4
  at 0x8048554: main (usingFreedMemory.c:32)
  by 0x80483F0: (in /parl/wjones/memoryDebugging/examples/a.out)
Address 0x40F6C024 is 0 bytes inside a block of size 40 free'd
  at 0x40168B6E: free (vg_clientfuncs.c:185)
  by 0x804852E: main (usingFreedMemory.c:28)
  by 0x80483F0: (in /parl/wjones/memoryDebugging/examples/a.out)
... program done
ERROR SUMMARY: 10 errors from 1 contexts (suppressed: 0 from 0)
malloc/free: in use at exit: 0 bytes in 0 blocks.
malloc/free: 1 allocs, 1 frees, 40 bytes allocated.
For a detailed leak analysis, rerun with: --leak-check=yes
```

```
/* dimation for toy integer array */
int intArraySize = 10;
/* loop index counter */
int i;
/* pointer to our array */
int* intArray;
/* dynamically declare some memory for our toy integer array */
intArray = (int *) malloc(intArraySize*sizeof(int));
/* loop over the array, and do some computations */
for(i=0; i<intArraySize; i++) {
    intArray[i] = i;
}
/* free malloc'd memory and end program */
/* 28 */ free(intArray);
/* loop over the array, and do some computations */
for(i=0; i<intArraySize; i++) {
/* 32 */     intArray[i] = i;
}
printf(" ... program done \n");
```

Freeing Free'd Memory

```
[wjones@hades examples]$ gcc -g -Wall freeingFreedMemory.c
[wjones@hades examples]$ valgrind a.out
...
Invalid free() / delete / delete[]
  at 0x40168B6E: free (vg_clientfuncs.c:185)
  by 0x804853C: main (freeingFreedMemory.c:30)
  by 0x80483F0: (in /parl/wjones/memoryDebugging/examples/a.out)
Address 0x40F6C024 is 0 bytes inside a block of size 40 free'd
  at 0x40168B6E: free (vg_clientfuncs.c:185)
  by 0x804852E: main (freeingFreedMemory.c:28)
  by 0x80483F0: (in /parl/wjones/memoryDebugging/examples/a.out)
ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
malloc/free: in use at exit: 0 bytes in 0 blocks.
malloc/free: 1 allocs, 2 frees, 40 bytes allocated.
For a detailed leak analysis, rerun with: --leak-check=yes
```

```
int main(void) {
    /* dimation for toy integer array */
    int intArraySize = 10;
    /* loop index counter */
    int i;
    /* pointer to our array */
    int* intArray;
    /* dynamically declare some memory for our toy integer array */
    intArray = (int *) malloc(intArraySize*sizeof(int));
    /* loop over the array, and do some computations */
    for(i=0; i<intArraySize; i++) {
        intArray[i] = i;
    }
    /* free malloc'd memory and end program */
    /* 28 */ free(intArray);
    /* 30 */ free(intArray);
    printf(" ... program done \n");
    return 0;
}
```

Size 0 Mallocs

```
[wjones@hades examples]$ gcc -g -Wall zeroSizeMallocs.c
[wjones@hades examples]$ valgrind a.out
```

```
...
```

```
ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
malloc/free: in use at exit: 0 bytes in 1 blocks.
malloc/free: 1 allocs, 0 frees, 0 bytes allocated.
For a detailed leak analysis, rerun with: --leak-check=yes
```

```
-----
```

```
[wjones@hades examples]$ gcc -g -Wall -lefence zeroSizeMallocs.c
[wjones@hades examples]$ a.out
```

```
Electric Fence 2.2.0 Copyright (C) 1987-1999
```

```
ElectricFence Aborting: Allocating 0 bytes, probably a bug.
```

```
Illegal instruction
```

```
[wjones@hades examples]$
```

```
int main(void) {  
  
    int* buffer;  
    int size = 0;  
    buffer = (int* ) malloc(size*sizeof(int));  
  
    return 0;  
}
```

Overflowing Array Bounds – Static

```
[wjones@hades examples]$ gcc -g -Wall overFlowStaticArray.c
[wjones@hades examples]$ valgrind a.out
```

```
...
```

```
... program done
```

```
ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
malloc/free: in use at exit: 0 bytes in 0 blocks.
malloc/free: 0 allocs, 0 frees, 0 bytes allocated.
For a detailed leak analysis, rerun with: --leak-check=yes
For counts of detected errors, rerun with: -v
```

```
-----
[wjones@hades examples]$ gcc -g -Wall -lefence overFlowStaticArray
[wjones@hades examples]$ a.out
... program done
[wjones@hades examples]$
```

```
int main(void) {
    /* declare and initialize a toy array of size 10 */
    int intArray[10] = {0,1,2,3,4,5,6,7,8,9};
    /* loop index counter */
    int i;
    /* loop over the array, and do some computations */
    for(i=0; i<10; i++) {
        intArray[i+1] = i+1;
    }
    printf(" ... program done \n");
    return 0;
}
```

Overflowing Array Bounds – Dynamic

```
[wjones@hades examples]$ gcc -g -Wall overFlowMallocedArray.c  
[wjones@hades examples]$ valgrind a.out
```

```
...
```

```
Invalid write of size 4
```

```
at 0x804851C: main (overFlowMallocedArray.c:24)
```

```
by 0x80483F0: (within /parl/wjones/memoryDebugging/examples/a.out
```

```
Address 0x40F6C04C is 0 bytes after a block of size 40 alloc'd
```

```
at 0x40168890: malloc (vg_clientfuncs.c:103)
```

```
by 0x80484ED: main (overFlowMallocedArray.c:20)
```

```
by 0x80483F0: (within /parl/wjones/memoryDebugging/examples/a.out
```

```
... program done
```

```
ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

```
malloc/free: in use at exit: 0 bytes in 0 blocks.
```

```
malloc/free: 1 allocs, 1 frees, 40 bytes allocated.
```

```
For a detailed leak analysis, rerun with: --leak-check=yes
```

```
int main(void) {
    /* dimation for toy integer array */
    int intArraySize = 10;
    /* loop index counter */
    int i;
    /* pointer to our array */
    int* intArray;
    /* dynamically declare some memory for our toy integer array */
/* 20 */    intArray = (int *) malloc(intArraySize*sizeof(int));
    /* loop over the array, and do some computations */
    for(i=0; i<intArraySize; i++) {
/* 24 */        intArray[i+1] = i+1;
    }
    /* free malloc'd memory and end program */
    free(intArray);
    printf(" ... program done \n");
    return 0;
}
```

Creating Memory Leaks

```
[wjones@hades examples]$ gcc -g -Wall memoryLeak.c
[wjones@hades examples]$ valgrind --leak-check=yes a.out
...
40 bytes in 1 blocks are definitely lost in loss record 1 of 1
  at 0x40168890: malloc (vg_clientfuncs.c:103)
  by 0x80484ED: main (memoryLeak.c:21)
  by 0x80483F0: (within /parl/wjones/memoryDebugging/examples/a.out)
LEAK SUMMARY:
  definitely lost: 40 bytes in 1 blocks.
```

```
/* dimation for toy integer array */
int intArraySize = 10;
/* loop index counter */
int i;
/* pointer to our array */
int* intArray;
/* dynamically declare some memory for our toy integer array */
intArray = (int *) malloc(intArraySize*sizeof(int));
/* loop over the array, and do some computations */
for(i=0; i<intArraySize; i++) {
    intArray[i] = i;
}
/* dynamically declare some memory for our toy integer array */
/* 21 */ intArray = (int *) malloc(intArraySize*sizeof(int));
/* loop over the array, and do some computations */
for(i=0; i<intArraySize; i++) {
    intArray[i] = 2*i;
}
```

Questions?

Thank you for your attention!



<http://www.parl.clemson.edu/~wjones/summerStudents>