Valgrind How-To

Nikolaus Correll

This is a mini tutorial on fighting segmentation faults and memory leaks using valgrind. Valgrind is a flexible program for debugging and profiling Linux executables. It consists of a core, which provides a synthetic CPU in software, and a series of "tools", each of which is a debugging or profiling tool.

This mini tutorial uses the examples from the seg-fault How-To.

Consider the following example (based on example 2 from the seg-fault How-To).

Example

```
#include <stdlib.h>

struct Matrix
{
   int rows, cols;
   int** data;
};

struct Matrix* rotation3D=0;

int main(){

   //rotation3D = (struct Matrix*) malloc(sizeof(struct Matrix));
   rotation3D->rows=4;
   rotation3D->cols=4;
   rotation3D->data=(int**) malloc(sizeof(int*)*rotation3D->rows);

   for(i=0;i<rotation3D->rows;i++)
     rotation3D->data[i]=(int*) malloc(sizeof(int)*rotation3D->cols);

   free(text);
   free(string.text);

   //for(i=0;i<rotation3D->rows;i++)
   //     free(rotation3D->data[i]);
   free(rotation3D->data);
   free(rotation3D);

   return 0;
 }
```

Notice that no memory is allocated for the “rotation3D” structure and that the for-loop releasing the matrix data has been commented out. This program will lead to a segmentation fault, and after this has been fixed, to a memory leak.

We will now see, how valgrind can be used to find these errors for us. First, compile your code with the –g option enabled. This will generate debug information so that valgrind can tell you where in your code the error happened.

```
gcc –g –o example example.c
```

Now, run your program using valgrind

```
valgrind ./example
```

Within the output, valgrind will report an invalid write and indicate at which line it occurred. In our example, valgrind will detect that the first access to “rotate3D” is invalid. You know now, that you have been writing to non-allocated memory. Allocate the memory now properly (by un-commenting the appropriate line in the example). Recompile (with the –g flag), and run valgrind again.

Hunt for the “Leak summary” at the end. Valgrind correctly finds 4 blocks that have not been released. It also tells you to run it with the –leak-check=full option.

```
valgrind --leak-check=full ./example
```

will now include a reference to the line in your C code where the memory that has not been released was initially allocated.

Check out the Valgrind user manual for what else valgrind can do for you to make your life easier http://valgrind.org/docs/manual/manual.html