King Abdulaziz University – Faculty of Engineering - EE-463

The C Programming Environment in Linux gcc, make, makefiles and gdb

A Tutorial by Example

Operating Systems Lab#4

Name	ID
Abdulrahman Ayman Mekwar	**1937268**

Before applying the fixes

Before applying any changes, when we compile the code provided in the worksheet and run it we get Segmentation fault, following the lab sheets we will be working on fixing this bug and two other bugs mentioned in the paper, we start by running the original version provided in the lab sheet, here is the

results:

```
Creating Node, 1 are in existence right now
Creating Node, 2 are in existence right now
Creating Node, 3 are in existence right now
Creating Node, 4 are in existence right now
The fully created list is:
4
3
2
1

Now removing elements:
Creating Node, 5 are in existence right now
Destroying Node, 4 are in existence right now
4
3
2
1

Segmentation fault
```

```
GNU gdb (Debian 8.2.1-2+b3) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file main
Reading symbols from main...done.
(gdb) run
Starting program: /home/muhannad/main
Creating Node, 1 are in existence right now
Creating Node, 2 are in existence right now
Creating Node, 3 are in existence right now
Creating Node, 4 are in existence right now
The fully created list is:
4
3
2
1

Now removing elements:
Creating Node, 5 are in existence right now
Destroying Node, 4 are in existence right now
4
3
2
1


Program received signal SIGSEGV, Segmentation fault.
0x00005555555553ea in Node::next (this=0x0) at main.c:23
23      Node* next() const { return next_; }
```

By removing the line 68 we can solve this issue as it is mentioned in the lab work sheet, the results after removing the line 68 shown below:

```
Creating Node, 1 are in existence right now
Creating Node, 2 are in existence right now
Creating Node, 3 are in existence right now
Creating Node, 4 are in existence right now
The fully created list is:
4
3
2
1

Now removing elements:
Creating Node, 5 are in existence right now
Destroying Node, 4 are in existence right now
4
3
2
1

Destroying Node, 3 are in existence right now
4
3
-129514752
2

Destroying Node, 2 are in existence right now
4
3
-129514816
3

Destroying Node, 1 are in existence right now
-129514816
4
Destroying Node, 0 are in existence right now
Destroying Node, -1 are in existence right now
Destroying Node, -2 are in existence right now
Destroying Node, -3 are in existence right now
Destroying Node, -4 are in existence right now
Destroying Node, -5 are in existence right now
Destroying Node, -6 are in existence right now
Destroying Node, -7 are in existence right now
Destroying Node, -8 are in existence right now
Destroying Node, -9 are in existence right now
free(): invalid pointer
Aborted
```

1) fix the memory leak issue

initially the code after applying the fixes in the worksheets, shows that we have a memory leak as follows:

```
==1551== 16 bytes in 1 blocks are definitely lost in loss record 1 of 1
==1551==    at 0x4835DEF: operator new(unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==1551==    by 0x109468: LinkedList::insert(int const&) (main.c:39)
==1551==    by 0x1091C7: main (main.c:103)
```

the issue is noted when removing, this we need to re-write the method for removing to fix the memory leak issue. And then check if there is a memory leak or not. Thus, by re-writing the remove as follows:

```
 // returns 0 on success,-1 on failure
int remove (const int &item_to_remove) {
    Node *marker = head_;
    Node *temp = 0; // temp points to one behind as we iterate

    if(marker->value() == item_to_remove){
    temp = marker;
    if(marker->next() == 0){
      head_ = 0;
            delete marker; // marker is the only element in the list
            marker = 0;
  }else{

      marker = marker->next();
      head_ = marker;
      delete temp;
    }
    return 0;
}


while(marker !=0 ){
  if(marker->value() == item_to_remove){
   temp->next( marker->next() );
   //marker->next(0);
   delete marker;
   return 0;
  }

  temp = marker;
  marker = marker->next();
}

    return -1; // failure
}
```

Now after applying the changes we run the code and we see the results as follows :

```
Creating Node, 1 are in existence right now
Creating Node, 2 are in existence right now
Creating Node, 3 are in existence right now
Creating Node, 4 are in existence right now
The fully created list is:
4
3
2
1

Now removing elements:
Destroying Node, 3 are in existence right now
3
2
1

Destroying Node, 2 are in existence right now
3
2

Destroying Node, 1 are in existence right now
3

Destroying Node, 0 are in existence right now
```

Check and validiate our work as in the image below :

```
==1604==
==1604== HEAP SUMMARY:
==1604==     in use at exit: 0 bytes in 0 blocks
==1604==   total heap usage: 7 allocs, 7 frees, 73,800 bytes allocated
==1604==
==1604== All heap blocks were freed -- no leaks are possible
```

Thus successfully the memory leak issue was solved.

2) Fixing the second bug

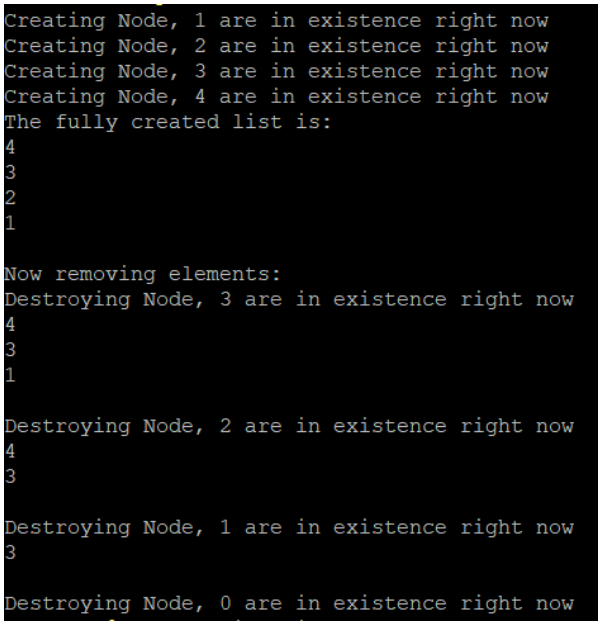
The bug occurred when inserting the numbers in the following sequence 1,2,3,4 then removing the number 2 immediately. The bug need to re-write the remove method as follows :

```
 // returns 0 on success,-1 on failure
int remove (const int &item_to_remove) {
    Node *marker = head_;
    Node *temp = 0; // temp points to one behind as we iterate

    if(marker->value() == item_to_remove){
    temp = marker;
    if(marker->next() == 0){
      head_ = 0;
            delete marker; // marker is the only element in the list
            marker = 0;
  }else{

      marker = marker->next();
      head_ = marker;
      delete temp;
    }
    return 0;
}


while(marker !=0 ){
  if(marker->value() == item_to_remove){
   temp->next( marker->next() );
   //marker->next(0);
   delete marker;
   return 0;
  }

  temp = marker;
  marker = marker->next();
}

    return -1; // failure
}
```

Then we change the remove sequence to start by removing the number 2 first then the others, as follows:

```
int main (int argc, char **argv) {

 LinkedList *list = new LinkedList ();
 list->insert (1);
 list->insert (2);
 list->insert (3);
 list->insert (4);
 printf("%s\n", "The fully created list is:");
 list->print ();
 printf("\n%s\n", "Now removing elements:");
 list->remove (2);
 list->print ();
 printf("\n");
 list->remove (1);
 list->print ();
 printf("\n");
 list->remove (4);
 list->print ();
 printf("\n");
 list->remove (3);
 list->print ();
 delete list;
 return 0;
}
```

Note the above code removes the elements starting from the node with number 2 to illustrates the bug was fixed and the results are shown below:



Thus, successfully the second bug was fixed.
