ISSN No. 0976-5697

Volume 6, No. 2, March-April 2015

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

# Detecting Memory Related Errors using a Valgrind Tool Suite and Detecting Data Races using Thread Sanitizer Clang

Saroj. A.Shambharkar
Kavikulguru Institute of Technology and Science
Nagpur, India

*Abstract:* It is important for a programmer to detect data races when a programming is done under concurrent systems or using shared memory or parallel programming. Many researchers are implementing their applications using the parallel programming and even for simple c programs ,they are not much concentrating on memory issues,accuracy of a result obtained, speed. It is important and one must take care it. There may found data races in their application code which has to be resolve and for that some data race detectors are required. There is also a need of detecting the memory related problems that may occur with shared memory. As a result of such memory related bugs and data races, the system may slow down the speed of the execution of a program .To improve the speed,the valgrind tool suite is used for memory related problems and ThreadSanitizer is used for detecting data races and some authors used Intel Thread Checker,ADAT,on-the-fly techniques for detecting them. Some tools are reducing the false positives but removing all false positives. Valgrind is providing a number of debugging and profiling tools that helps in improving the accuracy and to enhance the performance of system. This paper is demonstrating and showing the different results obtained by using the mentioned tool suite. The available tools in valgrind tool suite helps a programmer by displaying appropriate messages if any memory related errors and based on these messages the appropriate action can be taken by the programmer. And next time when user start writing similar type of application code, tries to avoid such errors and this makes our programs faster and avoids memory leaks. The memory leaks may happen when the programmer tries to write or read or access any location of an array beyond its bound. There are different categories of memory leaks. From the existing categories the two mostly used are definitely lost and probably lost and the result is showing both the memory leaks. The result through the stack trace tells where the memory leak is allocated. The output can also be store in a log file or simply we can display through appropriate error messages. This paper is showing some of the error messages and demonstrated through some programming example and related error messages using valgrind tool suite.

***Keywords:*** Memory leak; valgrind tool suite; profiling tools; stack trace; ThreadSanitizer; heap memory; Intel Thread Checker; false positive;

## I. INTRODUCTION

When the programmer is working under multithreading environment ,which is mostly used in many applications,there may be chance of data races. Data races are difficult to detect in concurrent systems and detected when working in shared memory. The data races must be handle by the programmer either by using some tools or before writing the application code taking care of it. The tools ThreadSanitizer,Address Sanitizer,Memory Sanitizer same as Memcheck tool of valgrind used by c and c++ programmer is used to detect the data races,and it important for the implementing any application .In concurrent system ,data races are mostly found type of debug and also they are harder to detect. In multhreading application more than one thread sharing the memory and tries to access the same variable at the same time,because of this data race occurs in the program. As result ,the system may crash and it may corrupt the useful data store in the memory .

Sometimes ,due to mistake done by the programmer there may be a need of memory management. And,the programmer should always try to keep track of memory errors. The tool is used for detecting memory errors, for measuring heap memory used by the programmer. In this paper some examples shown by using Memcheck tool for reporting memory leaks in figure 4.

## II. LITERATURE SURVEY

There are some applications where multiprocesing or multi Threading or shared memory concepts were used. The different languages available such as C,C++,Java,OpenGL,openMp,OpenACC,POSIX-thread API's and so on available. Earlier mostly preferred for parallel processing are openMP based on C and POSIX-thread but now other languages like CUDA programming is also used to enhance the processor speed and its performance. When writing the programs using multithreading concept,it may found that data races occurs,one variable reading a data at the same time another thread tries to write to same memory location. The programmer having loop carried dependencies in the code and there will problem in debugging. The author of the reference[1] made efforts to detect data races. The mentioned in the paper, the Intel thread Checker can also be used for detecting data races but it is not giving efficient performance and the ADAT data race detection mechanism is used .They have implemented an openMp parser and they have stated in their paper ADAT tool used by them having better functionality and giving better performance. They worked on more challenging target program models and achieved faster results than Intel thread Checker[1].

The openMP is used to achieve high performance and the compiler directives or pragma's are used to convert sequential program into parallel program. When writing the programs in OpenMP ,it is important to detect data races,otherwise on execution of such program, they may lead to unpredictable results. They have used on-the-fly technique. The existing tool Helgrind+ also helps in data race detection and reducing false positives. In [2],the authors concluded that the Helgrind+ is not giving precise results and it is not efficient technique of detecting data races. The on-the-fly technique is used for large openMP programs and there will not be any false positives[2].

For large High Performance Computing,the existing static or dynamic data detection techniques are not very useful,as creating high runtime overhead and not providing much accuracy[6].

The limitation of clang is given in the paper[8] ,that when clang used as c or c++ language,there is lack of support of openmp implementation .But, it was mentioned that Intel has contributed openMP implementation to clang[8].

In copyright material from Intel 2014,it is mentioned support in clang/llvm is under development and also it mentioned, there exist Clang omp-repo,based on clang or llvm includes used trunk based version[7] .

## III. DATA RACE DETECTION AND DEGUGGING MEMORY RELATED PROBLEMS

### *A.* Data Race Detection Using ThreadSanitizer and Thread Analyzer

During runtime, Thread Analyzer is used to detect the data races. Before running any application the user cannot predict the behavior of the written application code. The behavior of the some application programs using different number of threads and different data sets, there may be a chance of data races,in such situation Thread Analyzer is used to detect them[5]. There are some drawbacks of using Thread Analyzer , (1) When we have multiple threads,in parallel programming environment,they are competing for the processors and the threads are may be spawned between the processes,here it is difficult for the Thread Analyzer to report about the data races (2) Thread Analyzer also unable to about the name of the variable accessed by different thread , the programmer has to get this variable name by observing the lines of source code (3) In some cases Thread Analyzer is unable to detect false positives, example synchronization of threads or when memory is recycled between the threads, it is unable to recognize the synchronization points. In such situations, the programmer can prefer other tools mentioned in section II named as on-the-fly.

Another tool for detecting data races is ThreadSanitizer. In many respect ,the ThreadSanitizer is same as Helgrind data race detector. ThreadSanitizer is supported by many operating systems,like Ubuntu 14.04.The accuracy of ThreadSanitizer is depends on the number of bugs and false reports detected by it and it is not same for all,the speed many vary from one system to another system.

### *B.* Debugging Memory related Problems using valgrind

Valgrind is a memory debugger tool that provides a number of tools for debugging memory related errors and profiling tools. The tools available in its tool suite helps the programmer to run their program faster and obtain results correct. Thus ,it improves the accuracy of the output[3].

Many tools available in tool suite,the most popular is Memcheck and is the default tool. The Memcheck tool helps in performing various memory checking functions to detect errors related to memory ,due to which may give unexpected behavior after running the c and c++ programs. The tools of valgrind automatically detect many memory and thread related problems associated with shared memory. Some tools are also used to detect bad memory usages, used to check any misuse of memory allocated to the program, profile CPU cache usage,profile heap usage. For example reading uninitialized memory, writing past the buffer[3].

When loops are used in the program ,chunk of memory is allocated and it not release by the the program variables and there may happen memory leak. When the Memcheck tool is used ,at the time of compilation -g option is used for displaying debugging information,and using this option the error messages include exact line numbers. The other options used by Memcheck are -00,-01,-02 and above -02.But above -02 is not suggested for debugging or displaying memory related errors[4].Memcheck uses a option –leak-check to tun on memory leak detector. If the program having memory related errors then running process gets slow down .Memcheck helps in memory leak by issuing appropriate error messages[4].This tool for some programs produces false positives.

## IV. DISCUSSION ON RESULT OBTAINED USING VALGRIND

It has been observed that the programmer sometimes are not that much concern about any data races or memory related errors. But,its shows some output and not displaying any error messages related to memory errors or any other bugs. When the application based on parallel processing,it is very much important to detect the data races,existing loop carried dependencies, a memory location is accessing the value of if the variable var is uninitialized for example a[i] is dependent on its value using a[i-1] and other things which may degrade the performance of an application .

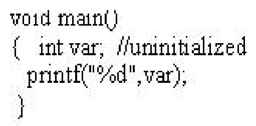
To obtain ThreadSanitizer ,the required is clang and gcc.The tool is used is after installing clang.In this paper after installing clang learned about threadsanitizer and how to test a c program using clang.The option used is -fsanitize=thread -g -01 -FPIE -pie while executing the code.To turn warning the option -warn is used.The additional flags can be set through environment variables.To the environment variable TSAN_ARGS ,the additional flags are provided.Already mention the threadsanitizer is used to detect data races present in a program.While doing parallel programming using c code it is very important to detect the m.

The below lines after execution using clang tells that the code or program executed is found to have data races when using clang followed by -fsanitize=thread -g -O1 option and followed by program name is run on terminal.When the simple program is executed by the programmer it cannot analyze and cannot able to the warnings obtained by using the Threaddsanitizer data race detector tool.Also the option -warn can be used to see the warning.It is suggested that the programmers should not ignore the warning and if they understand theresult of their code on execution may affects the performanceof the appication program for which they have written the code and also some sometimes warning may affect the execution and performance of the system.

```
WARNING: ThreadSanitizer: data race (pid=2896)

Write of size 4 at 0x7fd6ae56cc3c by thread T1:

 #0 Thr1 /home/saroj/testThread.c:5 (exe+0x0000000a02bf)

 Previous write of size 4 at 0x7fd6ae56cc3c by main thread:

 #0 main /home/saroj/testThread.c:12
(exe+0x0000000a0313)

 Thread T1 (tid=2898, running) created by main thread at:
```

```
#0 pthread_create ??:0 (exe+0x0000000458db)

#1 main /home/saroj/testThread.c:11 (exe+0x0000000a0304)

SUMMARY: ThreadSanitizer: data race /home/saroj/testThread.c:5 Thr1

ThreadSanitizer: reported 1 warnings
```

Consider another a simple program to demonstrate the use of valgrind tool a simple program shown in figure 1. In figure 1 it is clearly shown that the program is having uninitialized variable var. The program containing a variable var declared as integer ,but not initialized any value,and after execution and compilation not displayed any error messages. And,it displayed the output as 0,which is not an accurate result of this program.



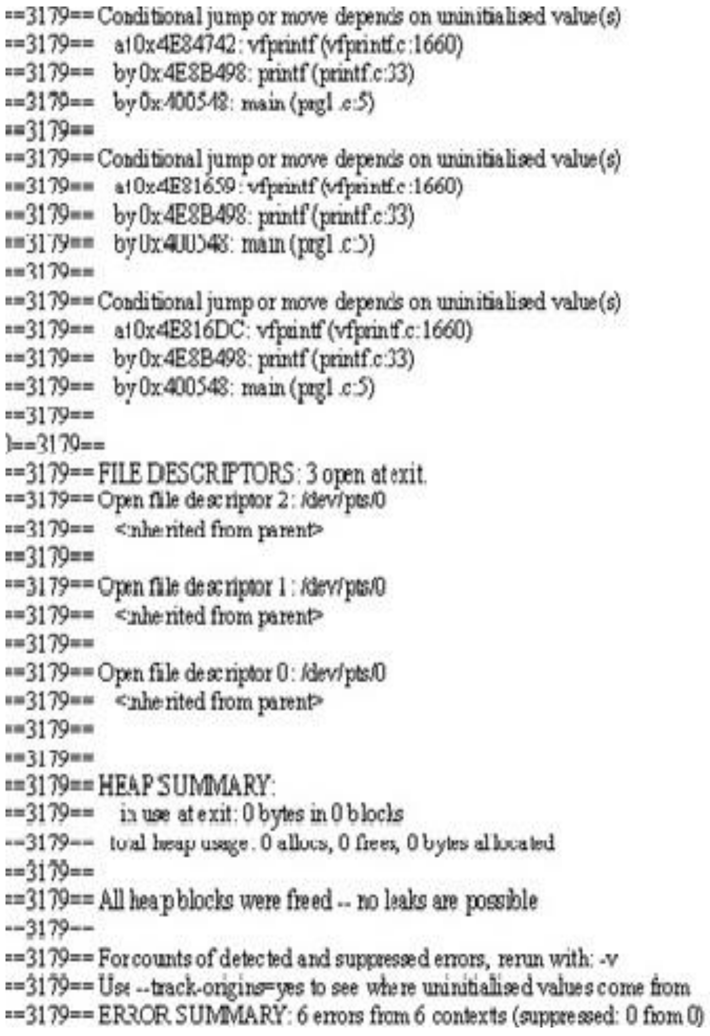
```
void main()
{   int var;  //uninitialized
  printf("%d",var);
 }
```

Figure 1. Program containing uninitialized variable var

It has been also observed that without using the tool Memcheck , of valgrind,the programmer is not able to get any information for the uninitialized variable. The command used using valgrind tool , memcheck is saroj@saroj:~$ valgrind --tool=memcheck --leak-check=yes --show-reachable=yes --num-callers=20 –track-fds=yes ./prg1out.,but this command is executed,using valgrind tool it shows the details about this uninitialized variable.

```
#include <stdio.h>

#include <malloc.h>

void main()

{

   int *ptr;

   ptr=(int *)malloc(10);

   free(ptr);

   ptr=(int *)malloc(12);

   ptr=(int *)malloc(14);

}
```

Figure 2. Second program for demonstrating memory leak



```
==3179== Conditional jump or move depends on uninitialised value(s)
==3179==    at 0x4E84742: vfprintf (vfprintf.c:1660)
==3179==    by 0x4E8B498: printf (printf.c:33)
==3179==    by 0x400548: main (prg1.c:5)
==3179==
==3179== Conditional jump or move depends on uninitialised value(s)
==3179==    at 0x4E81659: vfprintf (vfprintf.c:1660)
==3179==    by 0x4E8B498: printf (printf.c:33)
==3179==    by 0x400548: main (prg1.c:5)
==3179==
==3179== Conditional jump or move depends on uninitialised value(s)
==3179==    at 0x4E816DC: vfprintf (vfprintf.c:1660)
==3179==    by 0x4E8B498: printf (printf.c:33)
==3179==    by 0x400548: main (prg1.c:5)
==3179==
]==3179==
==3179== FILE DESCRIPTORS: 3 open at exit.
==3179== Open file descriptor 2: /dev/pts/0
==3179==    <inherited from parent>
==3179==
==3179== Open file descriptor 1: /dev/pts/0
==3179==    <inherited from parent>
==3179==
==3179== Open file descriptor 0: /dev/pts/0
==3179==    <inherited from parent>
==3179==
==3179==
==3179== HEAP SUMMARY:
==3179==     in use at exit: 0 bytes in 0 blocks
==3179==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==3179==
==3179== All heap blocks were freed -- no leaks are possible
==3179==
==3179== For counts of detected and suppressed errors, rerun with: -v
==3179== Use --track-origins=yes to see where uninitialised values come from
==3179== ERROR SUMMARY: 6 errors from 6 contexts (suppressed: 0 from 0)
```

Figure 3. Showing the error messages for an uninitialized variable var

The Memcheck tool is showing the error messages using available options of valgrind Memcheck tool helping to detect the uninitialized variable. When ,the c program code is executed using simple gcc compiler,it is not possible to get any information regarding whether any memory location that is whether the variable is initialized or not.

The memory leak detection is also very important for the programmer,compiling a program using gcc compiler and running it does not show any memory leaks,Figure 2 containing the C program is written to demonstrate the memory allocation .And without using a tool, the proper allocation and wastage of memory is not detected and one cannot analyze memory related problems present in the program .

When using the valgrind tool , the details about the error messages related to memory is shown in figure 4 it is showing the error messages related to memory on executing a command using Memcheck tool .

```
==4324== HEAP SUMMARY:
==4324==     in use at exit: 26 bytes in 2 blocks
==4324==   total heap usage: 3 allocs, 1 frees, 36 bytes allocated
==4324==
==4324== 12 bytes in 1 blocks are definitely lost in loss record 1 of 2
```

```
==4324==    at 0x4C2AB80: malloc (in
/usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==4324==    by 0x4005A8: main (prg2meml.c:8)
==4324==
==4324== 14 bytes in 1 blocks are definitely lost in loss
record 2 of 2
==4324==    at 0x4C2AB80: malloc (in
/usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==4324==    by 0x4005B6: main (prg2meml.c:9)
==4324==
==4324== LEAK SUMMARY:
==4324==    definitely lost: 26 bytes in 2 blocks
==4324==    indirectly lost: 0 bytes in 0 blocks
==4324==      possibly lost: 0 bytes in 0 blocks
==4324==    still reachable: 0 bytes in 0 blocks
==4324==         suppressed: 0 bytes in 0 blocks
==4324==
==4324== For counts of detected and suppressed errors,
rerun with: -v
==4324== ERROR SUMMARY: 2 errors from 2 contexts
(suppressed: 0 from 0)
```

Figure 4. Showing the memory leak related error messages and leak summary

## V. CONCLUSION AND FUTURE WORK

The valgrind tool is useful as shown from the figure 4, it is clearly shown,from the figure 4,of Leak Summary,the memory lost of 26 bytes, which may be sometimes not taken care by the programmer. From figure 3 related to uninitialized variable not showing any information related to memory leak,the error messages are displayed according to the problems found in the program. There are some drawback of using Memcheck tool ,it cannot detect static allocation,out-of-range reads or writes to arrays .There are other tools available with valgrind are addresses check which is similar to Memcheck, cache grind,lackey,massif not worked out. These tools of valgrind can also be tested to analyze the actual errors that should be known to the programmer and it should be known other types of errors before writing application using C and C++ languages.

## VI. REFERENCES

[1] Young-Joo Kim, Sejun Song,Young-Kee Jun, "ADAT :An Adaptable Dynamic Analysis Tool for Race Detection in OpenMp Programs", Parallel and Distributed Processing with Applications (ISPA) ,2011 IEEE 9th International Symposium ,26-28 may 2011 conference.

[2] OK-Kyoon Ha, In-Bon Kuh, Guy Martin Tchamgous, Yong-Kee Jun, "On-the-fly detection of data races in OpenMP programs," PADTAD 2012 Proceedings of the 2012 workshop on parallel and Distributed Systems : Testing, Analysis and Debugging.

[3] HTTP://Valgrind.org .

[4] HTTP:// Valgrind.org/docs/manual/quick-start.html.

[5] HTTP: //Docs.oracle.com/cd E18659-01/html/821-2124/gkgov.html.

[6] Joachim. Protze,Simone Atzani, Dong H.Ahn, Martin Schulz, Ganesh GopaloKrishnan,Matthias S. Mul ler, Ignacia Laguna, Zvonimir Rakamuric, greg L. Lee," Towards Providing Low-Overhead Data Race Detection for Large OpenMP Applications ", This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-660004).

[7] Alexey Bataev, Zinovy Nis Intel ,"OpenMP* Support in Clang/LLVM:Status Update and Future Directions", LLVM Developer's Meeting,2014.

[8] David Chisnall ,"LLVM in the FreeBSD Toolchain*",*AsiaBSDCon 2014, March 15, 2014.