Topic : Valgrind - tool for memory debugging, memory leak detection, and profiling

Prepared By :

1. Sayantan Pandit (Exam Roll-111205022)
2. Suman Banerjee (Exam Roll-111205040)

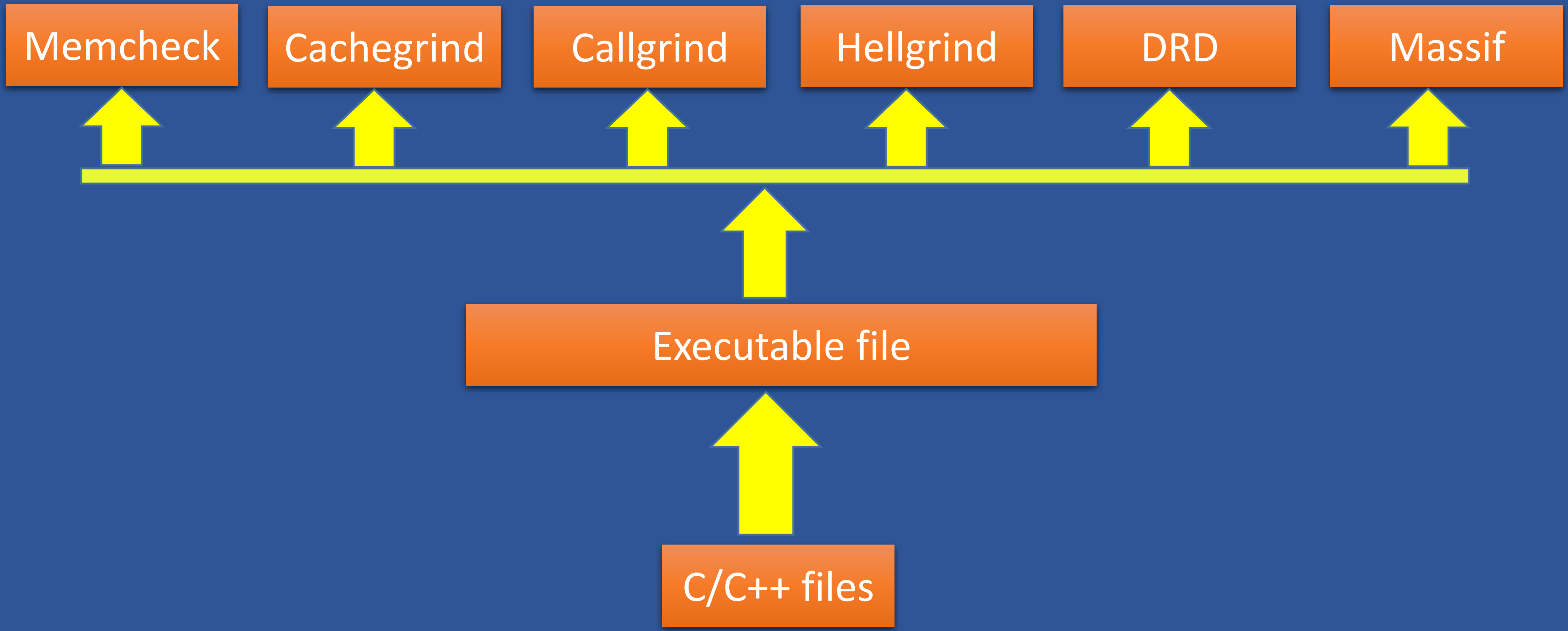
INTRODUCTION

A. What is Valgrind ?

1. Open-Source tool
2. Detects memory leaks/corruption
3. Provides a number of debugging and profiling tools.

INTRODUCTION

B. How does Valgrind work ?



Memcheck

Cachegrind

Callgrind

Hellgrind

DRD

Massif

Executable file

C/C++ files

Installation of Software

A. Directly from Repository

1. Open the terminal window
2. Type the command 'sudo apt-get install valgrind'
3. Enter your root password when prompted

B. Download the Software

1. tar xvfz valgrind–1.0.0.tar.gz
2. cd valgrind–1.0.0
3. ./configure
4. make
5. make install

Different Valgrind tools

1. Memcheck
2. Cachegrind
3. Callgrind
4. Helgrind
5. DRD
6. Massif
7. DHAT

Memcheck

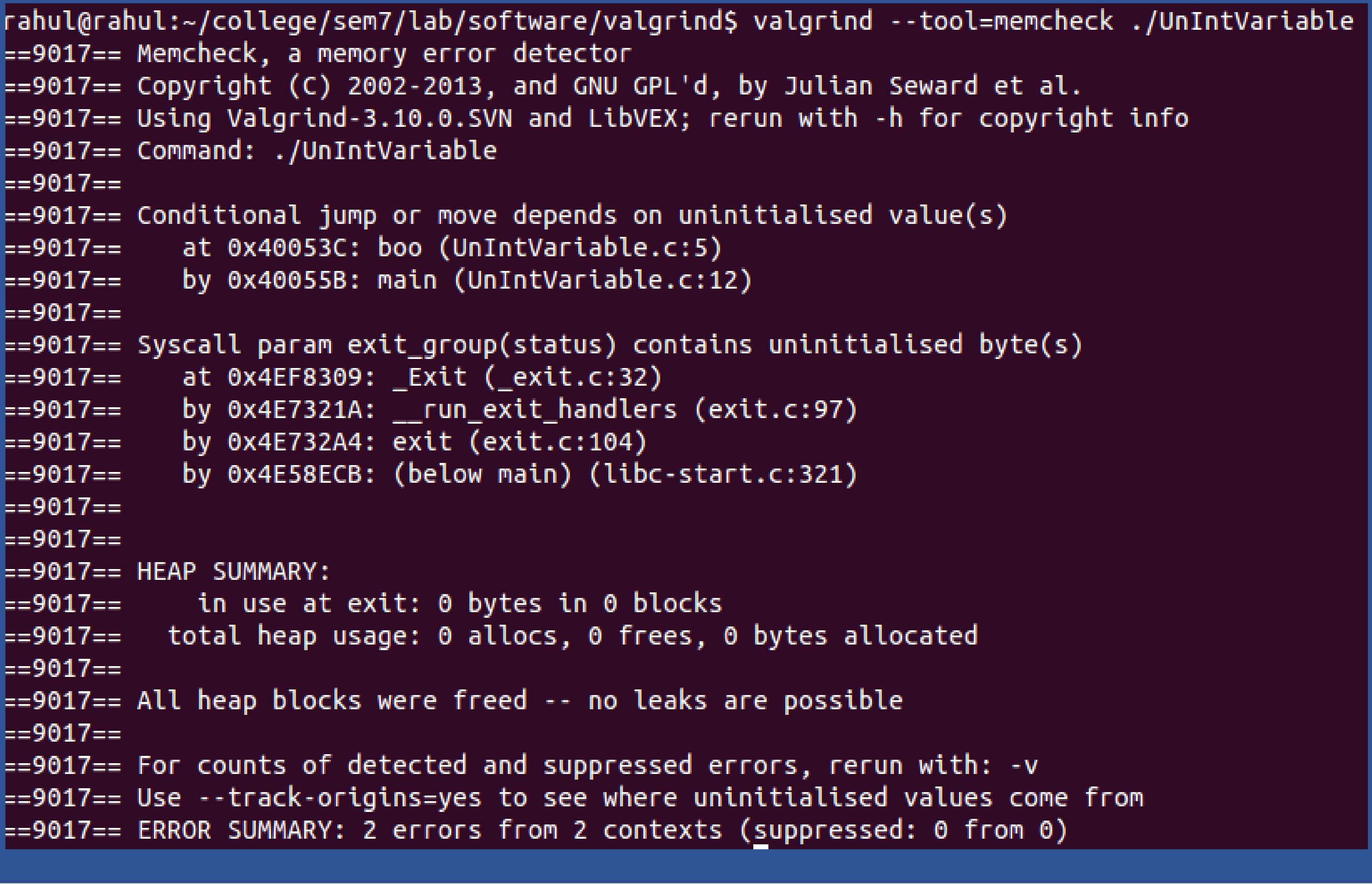
Memory detected error

1. Use of Uninitialized values
2. Memory leaks detection
3. Invalid Pointer use
4. Accessing unavailable memory

Memcheck

A. Use of uninitialized values

```
int boo(int y)
{
   if(y == 2)
     printf("Correct\n");
}
int main()
{
   int x;
   boo(x);
}
```

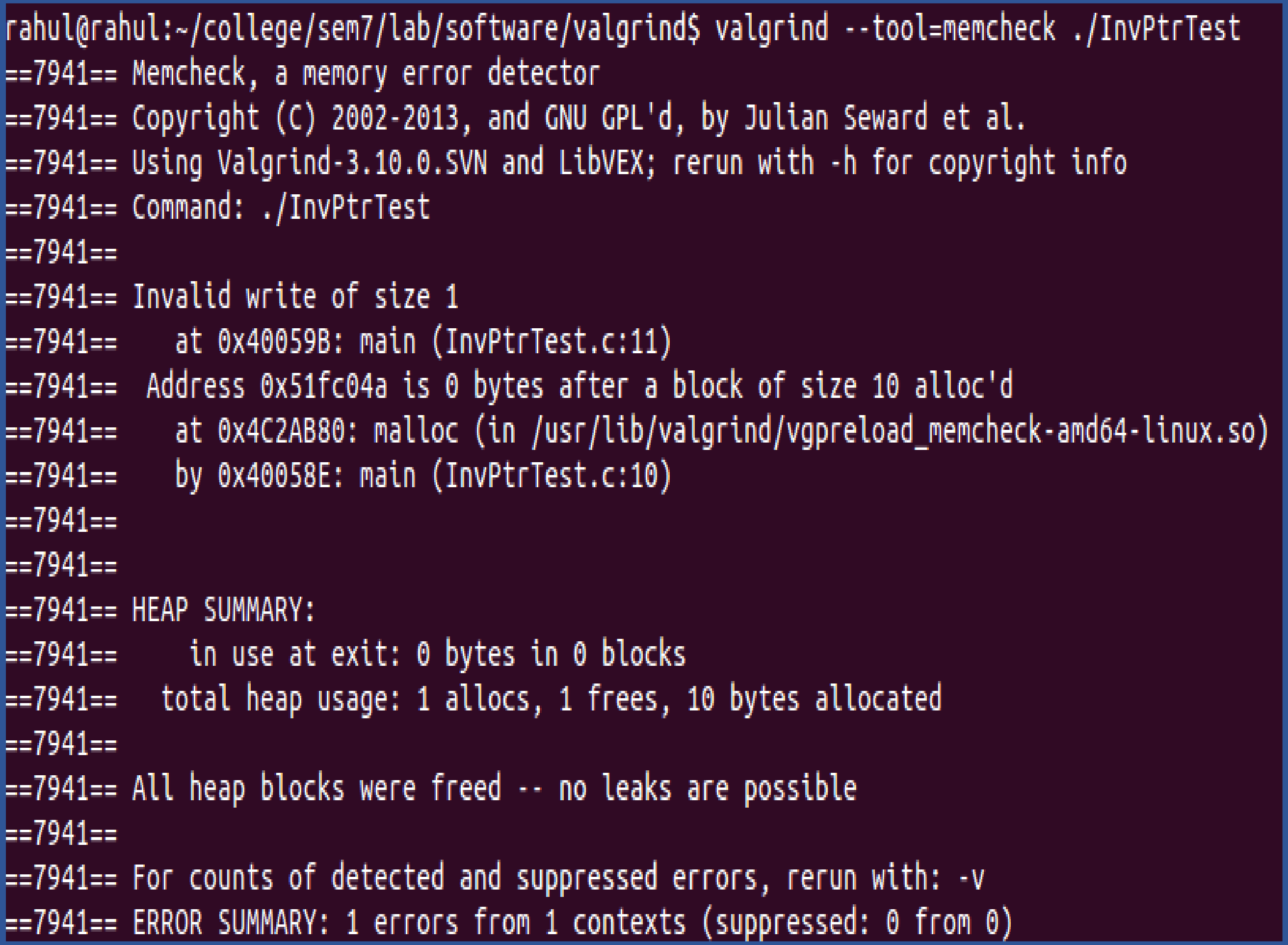


```
rahul@rahul:~/college/sem7/lab/software/valgrind$ valgrind --tool=memcheck ./UnIntVariable
==9017== Memcheck, a memory error detector
==9017== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==9017== Using Valgrind-3.10.0.SVN and LibVEX; rerun with -h for copyright info
==9017== Command: ./UnIntVariable
==9017==
==9017== Conditional jump or move depends on uninitialised value(s)
==9017==    at 0x40053C: boo (UnIntVariable.c:5)
==9017==    by 0x40055B: main (UnIntVariable.c:12)
==9017==
==9017== Syscall param exit_group(status) contains uninitialised byte(s)
==9017==    at 0x4EF8309: _Exit (_exit.c:32)
==9017==    by 0x4E7321A: __run_exit_handlers (exit.c:97)
==9017==    by 0x4E732A4: exit (exit.c:104)
==9017==    by 0x4E58ECB: (below main) (libc-start.c:321)
==9017==
==9017==
==9017== HEAP SUMMARY:
==9017==     in use at exit: 0 bytes in 0 blocks
==9017==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==9017==
==9017== All heap blocks were freed -- no leaks are possible
==9017==
==9017== For counts of detected and suppressed errors, rerun with: -v
==9017== Use --track-origins=yes to see where uninitialised values come from
==9017== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Memcheck

B. Invalid Pointer Use

```
int main()
{
  char *x;
  x = (char *)malloc(10*sizeof(char));
  x[10] = 'a';
  free(x);
  return 0;
}
```

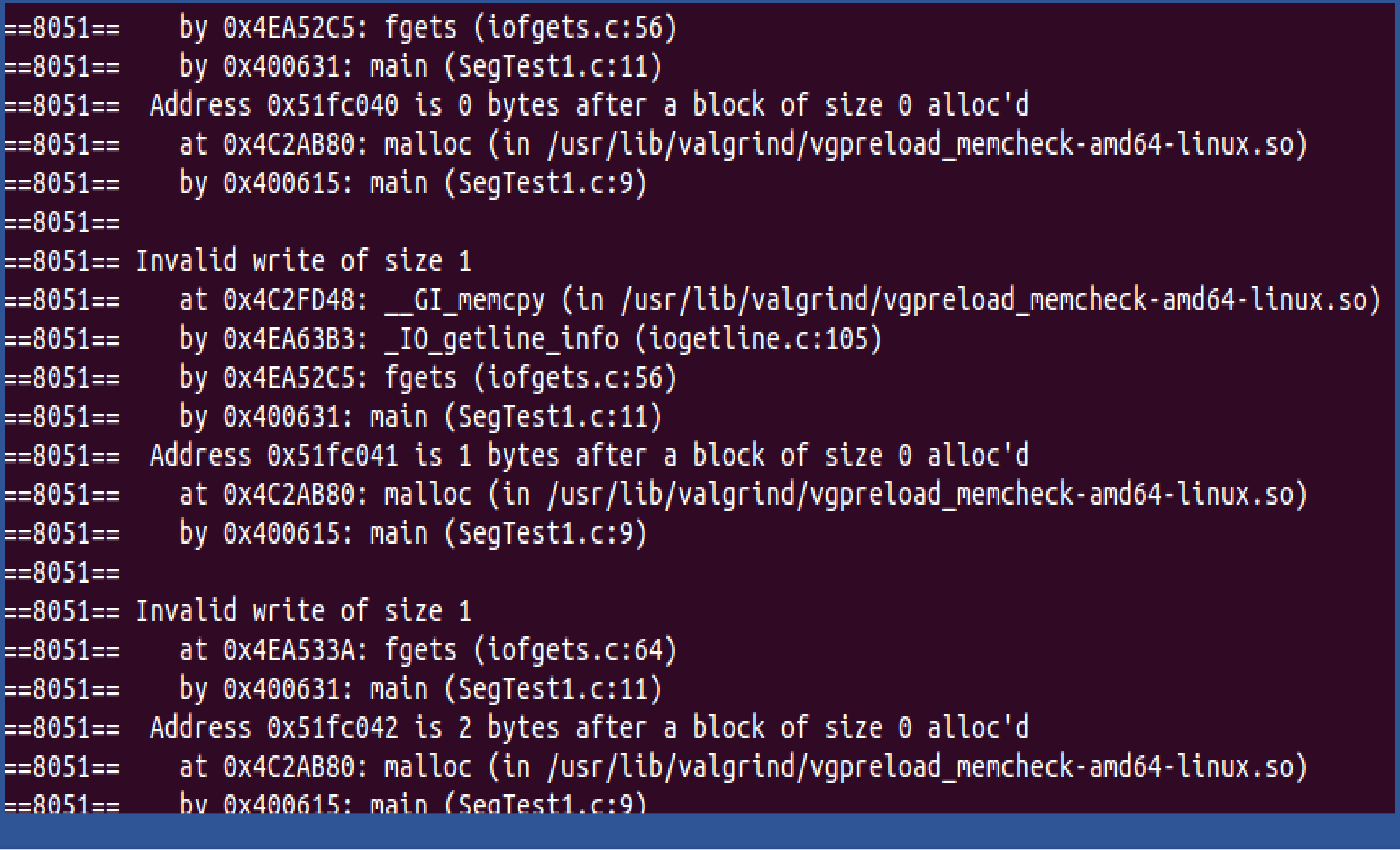


```
rahul@rahul:~/college/sem7/lab/software/valgrind$ valgrind --tool=memcheck ./InvPtrTest
==7941== Memcheck, a memory error detector
==7941== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==7941== Using Valgrind-3.10.0.SVN and LibVEX; rerun with -h for copyright info
==7941== Command: ./InvPtrTest
==7941==
==7941== Invalid write of size 1
==7941==    at 0x40059B: main (InvPtrTest.c:11)
==7941==  Address 0x51fc04a is 0 bytes after a block of size 10 alloc'd
==7941==    at 0x4C2AB80: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==7941==    by 0x40058E: main (InvPtrTest.c:10)
==7941==
==7941==
==7941== HEAP SUMMARY:
==7941==     in use at exit: 0 bytes in 0 blocks
==7941==   total heap usage: 1 allocs, 1 frees, 10 bytes allocated
==7941==
==7941== All heap blocks were freed -- no leaks are possible
==7941==
==7941== For counts of detected and suppressed errors, rerun with: -v
==7941== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Memcheck

C. Accessing unavailable memory

```
int main()
{
   char *buf;
   buf = malloc(1<<32);
   fgets(buf, 1024, stdin);
   printf("s\n", buf);
   return 0;
}
```



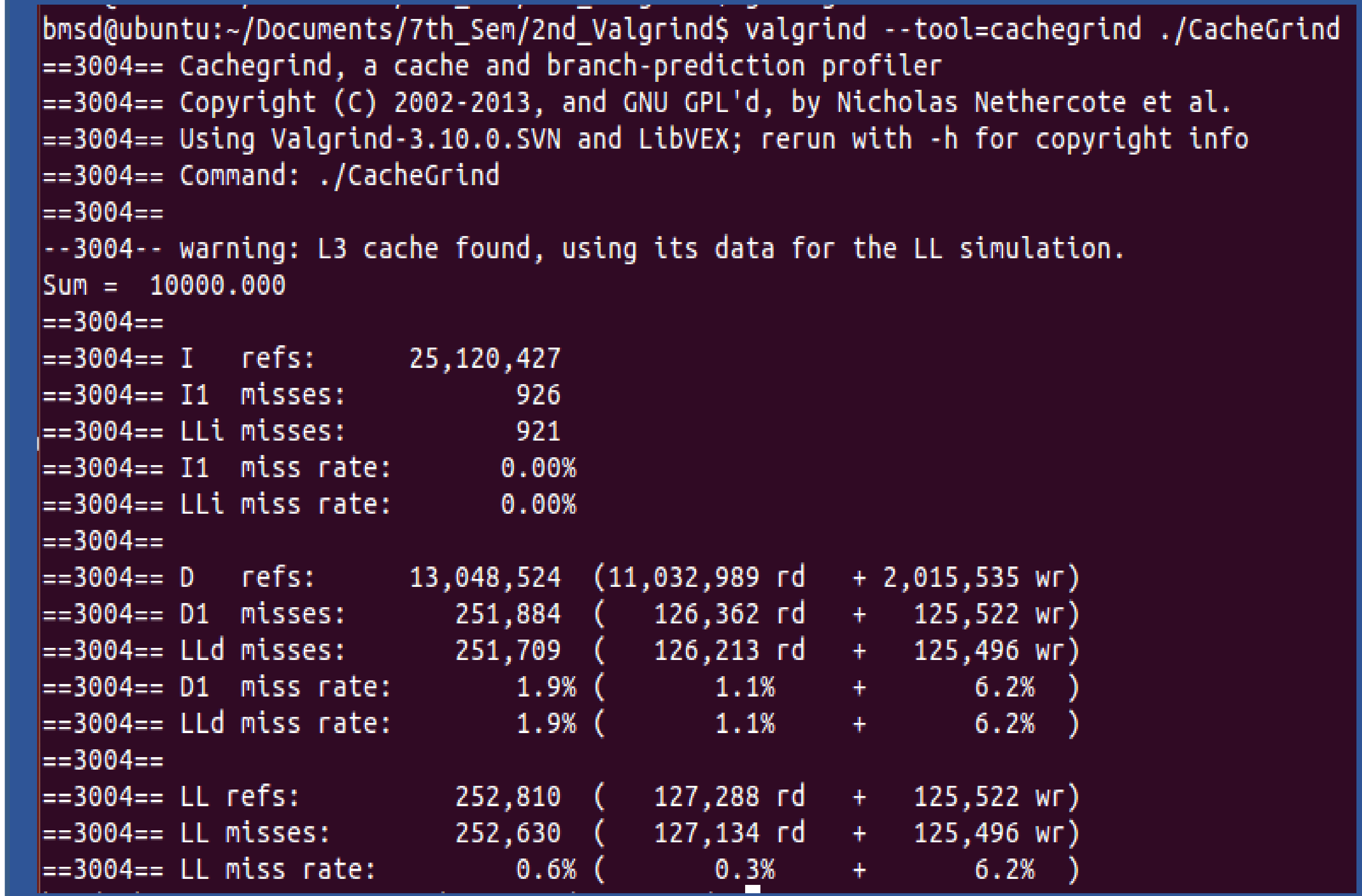
```
==8051==    by 0x4EA52C5: fgets (iofgets.c:56)
==8051==    by 0x400631: main (SegTest1.c:11)
==8051==  Address 0x51fc040 is 0 bytes after a block of size 0 alloc'd
==8051==    at 0x4C2AB80: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==8051==    by 0x400615: main (SegTest1.c:9)
==8051==
==8051== Invalid write of size 1
==8051==    at 0x4C2FD48: __GI_memcpy (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==8051==    by 0x4EA63B3: _IO_getline_info (iogetline.c:105)
==8051==    by 0x4EA52C5: fgets (iofgets.c:56)
==8051==    by 0x400631: main (SegTest1.c:11)
==8051==  Address 0x51fc041 is 1 bytes after a block of size 0 alloc'd
==8051==    at 0x4C2AB80: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==8051==    by 0x400615: main (SegTest1.c:9)
==8051==
==8051== Invalid write of size 1
==8051==    at 0x4EA533A: fgets (iofgets.c:64)
==8051==    by 0x400631: main (SegTest1.c:11)
==8051==  Address 0x51fc042 is 2 bytes after a block of size 0 alloc'd
==8051==    at 0x4C2AB80: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==8051==    by 0x400615: main (SegTest1.c:9)
```

Cachegrind

Cachegrind is a tool for doing cache simulations and annotating the source line-by-line with the number of cache misses and interacts with a machine's cache hierarchy and (optionally) branch predictor.

Cachegrind

```
#include <stdio.h>
#define N 1000
double array_sum(double a[][N]);
int main(int argc, char **argv) {
        double a[N][N];
        int i,j;
        for(i=0; i<N; i++) {
                for(j=0; j<N; j++)
                a[i][j] = 0.01;
         }
        printf("Sum = %10.3f\n", array_sum(a));
        return 0;
}
double array_sum(double a[][N]) {
        int i,j;
        double s;
        s=0;
        for(i=0; i<N; i++)
        for(j=0; j<N; j++)
        s += a[i][j];
        return s;
}
```



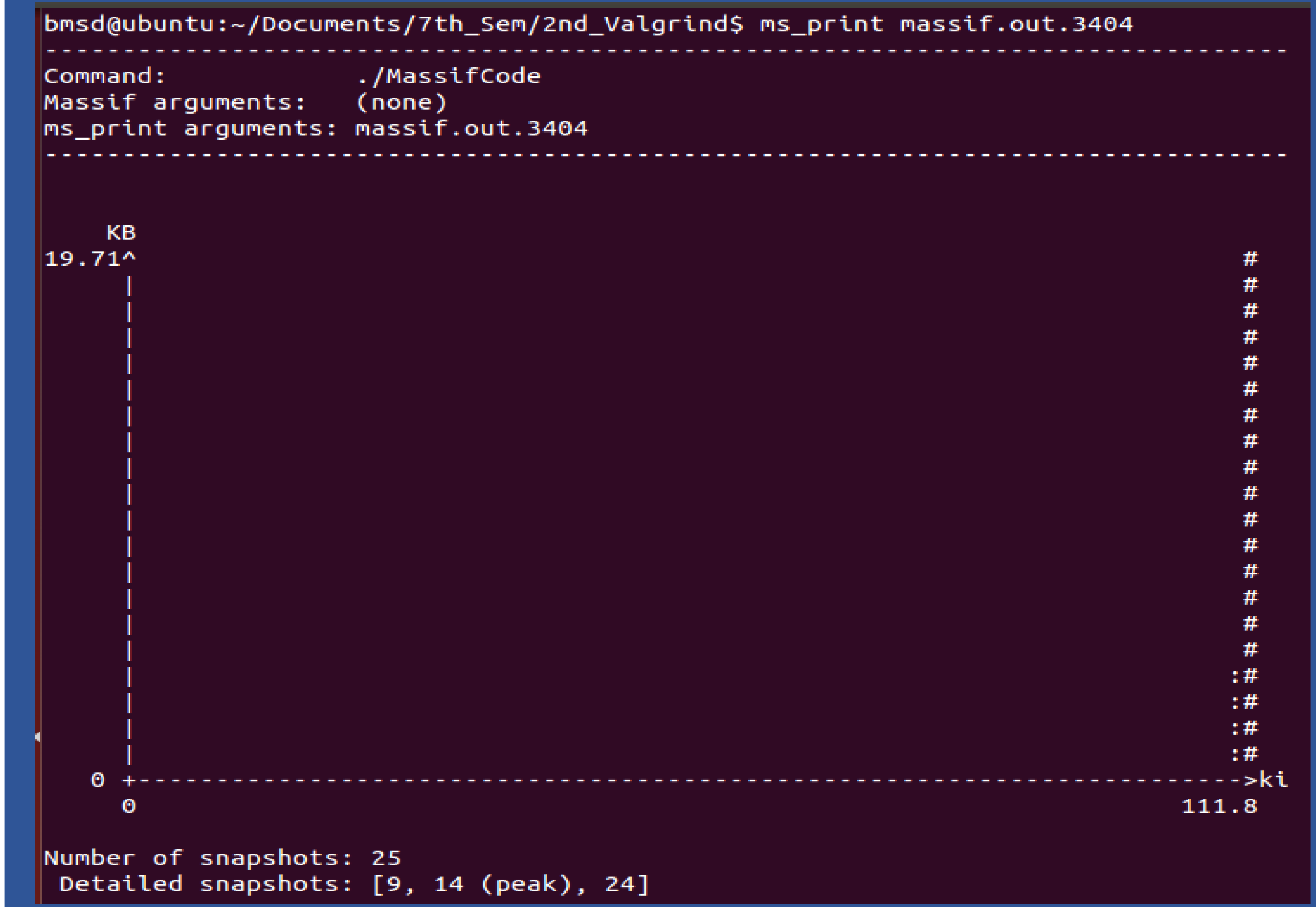
```
bmsd@ubuntu:~/Documents/7th_Sem/2nd_Valgrind$ valgrind --tool=cachegrind ./CacheGrind
==3004== Cachegrind, a cache and branch-prediction profiler
==3004== Copyright (C) 2002-2013, and GNU GPL'd, by Nicholas Nethercote et al.
==3004== Using Valgrind-3.10.0.SVN and LibVEX; rerun with -h for copyright info
==3004== Command: ./CacheGrind
==3004==
--3004-- warning: L3 cache found, using its data for the LL simulation.
Sum =  10000.000
==3004==
==3004== I   refs:      25,120,427
==3004== I1  misses:           926
==3004== LLi misses:           921
==3004== I1  miss rate:       0.00%
==3004== LLi miss rate:       0.00%
==3004==
==3004== D   refs:      13,048,524  (11,032,989 rd   + 2,015,535 wr)
==3004== D1  misses:       251,884  (   126,362 rd   +   125,522 wr)
==3004== LLd misses:       251,709  (   126,213 rd   +   125,496 wr)
==3004== D1  miss rate:        1.9% (       1.1%     +       6.2%  )
==3004== LLd miss rate:        1.9% (       1.1%     +       6.2%  )
==3004==
==3004== LL refs:          252,810  (   127,288 rd   +   125,522 wr)
==3004== LL misses:        252,630  (   127,134 rd   +   125,496 wr)
==3004== LL miss rate:         0.6% (       0.3%     +       6.2%  )
```

Massif

Massif is a heap profiler. It measures how much heap memory the program uses. This includes both the useful space, and the extra bytes allocated for book-keeping and alignment purposes. It can also measure the size of the program’s stack(s).

Massif

```
#include <stdio.h>
void g(void) {
         malloc(4000);
}
void f(void) {
       malloc(2000);
       g();
}
int main(void) {
       int i;
       int *a[10];
       for(i=0; i<10; i++)
       a[i] = malloc(1000);
       f();
       g();
       for(i=0; i<10; i++)
       free(a[i]);
       return 0;
}
```

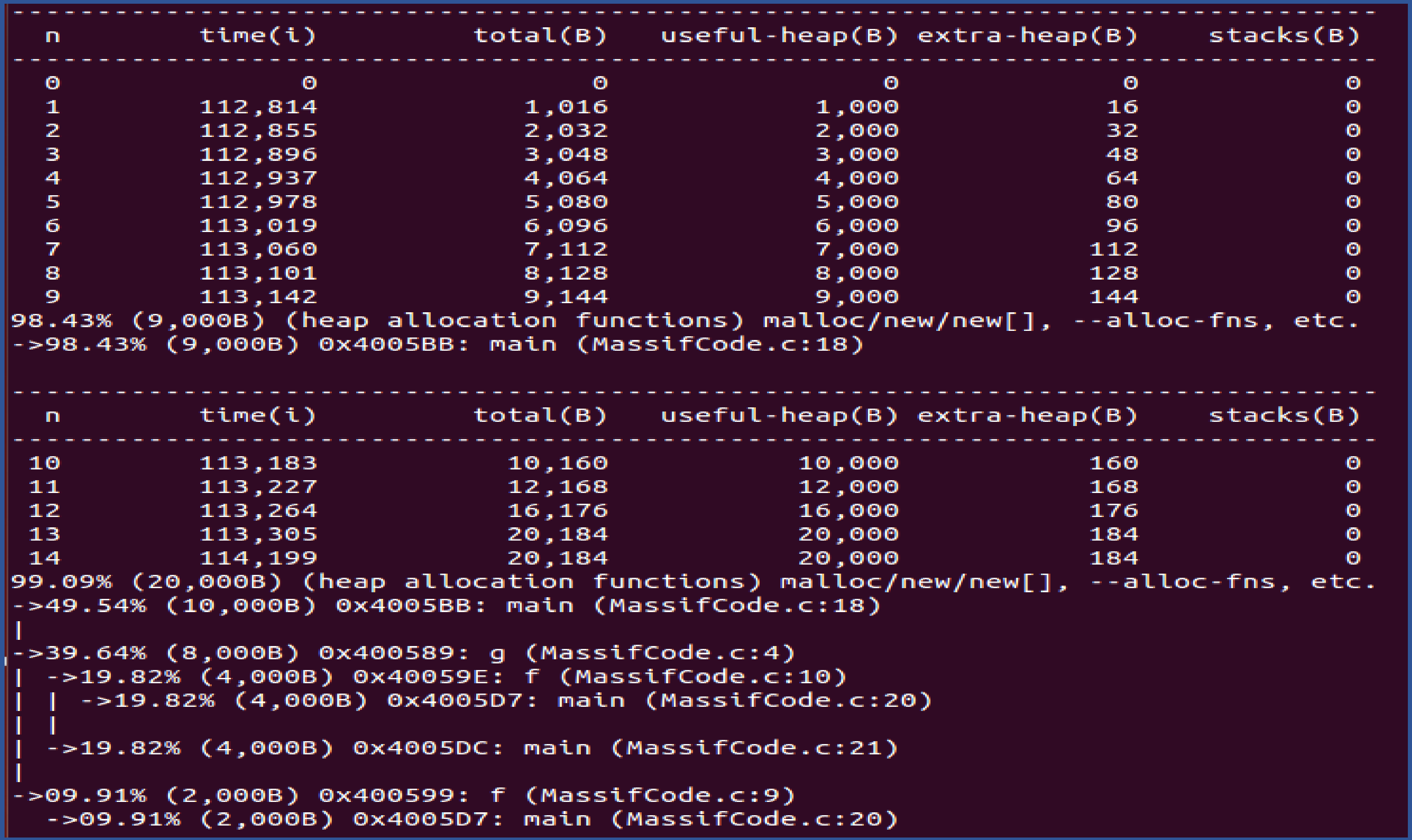


```
bmsd@ubuntu:~/Documents/7th_Sem/2nd_Valgrind$ ms_print massif.out.3404
--------------------------------------------------------------------------------
Command:            ./MassifCode
Massif arguments:   (none)
ms_print arguments: massif.out.3404
--------------------------------------------------------------------------------


    KB
19.71^                                                                        #
     |                                                                        #
     |                                                                        #
     |                                                                        #
     |                                                                        #
     |                                                                        #
     |                                                                        #
     |                                                                        #
     |                                                                        #
     |                                                                        #
     |                                                                        #
     |                                                                        #
     |                                                                        #
     |                                                                        #
     |                                                                        #
     |                                                                        #
     |                                                                       :#
     |                                                                       :#
     |                                                                       :#
     |                                                                       :#
   0 +----------------------------------------------------------------------->ki
     0                                                                    111.8

Number of snapshots: 25
 Detailed snapshots: [9, 14 (peak), 24]
```

Massif



```
--------------------------------------------------------------------------------
  n        time(i)         total(B)   useful-heap(B) extra-heap(B)    stacks(B)
--------------------------------------------------------------------------------
  0              0                0                0             0            0
  1        112,814            1,016            1,000            16            0
  2        112,855            2,032            2,000            32            0
  3        112,896            3,048            3,000            48            0
  4        112,937            4,064            4,000            64            0
  5        112,978            5,080            5,000            80            0
  6        113,019            6,096            6,000            96            0
  7        113,060            7,112            7,000           112            0
  8        113,101            8,128            8,000           128            0
  9        113,142            9,144            9,000           144            0
98.43% (9,000B) (heap allocation functions) malloc/new/new[], --alloc-fns, etc.
->98.43% (9,000B) 0x4005BB: main (MassifCode.c:18)

--------------------------------------------------------------------------------
  n        time(i)         total(B)   useful-heap(B) extra-heap(B)    stacks(B)
--------------------------------------------------------------------------------
 10        113,183           10,160           10,000           160            0
 11        113,227           12,168           12,000           168            0
 12        113,264           16,176           16,000           176            0
 13        113,305           20,184           20,000           184            0
 14        114,199           20,184           20,000           184            0
99.09% (20,000B) (heap allocation functions) malloc/new/new[], --alloc-fns, etc.
->49.54% (10,000B) 0x4005BB: main (MassifCode.c:18)
|
->39.64% (8,000B) 0x400589: g (MassifCode.c:4)
| ->19.82% (4,000B) 0x40059E: f (MassifCode.c:10)
| | ->19.82% (4,000B) 0x4005D7: main (MassifCode.c:20)
| |
| ->19.82% (4,000B) 0x4005DC: main (MassifCode.c:21)
|
->09.91% (2,000B) 0x400599: f (MassifCode.c:9)
  ->09.91% (2,000B) 0x4005D7: main (MassifCode.c:20)
```

Helgrind

Helgrind is a thread error detector.

A Simple Data Race :

```
#include <stdio.h>
#include <pthread.h>
int var = 0;
void* child_fn ( void* arg ) {
        var++;      /*  Unprotected  relative to  parent */  /* this is line 6 */
        return NULL;
}
int main ( void ) {
        pthread_t child;
        pthread_create(&child, NULL, child_fn, NULL);
        var++;  /* Unprotected  relative  to  child */   /*this is line 13 */
        pthread_join(child, NULL);
        return 0;
}
```

Limitation

1. Program runs 20 to 50 times slower
2. Increased memory consumption
3. Highly optimized code may cheat Valgrind

THANK YOU