Valgrind Cheatsheet

A comprehensive cheat sheet for Valgrind, a powerful memory debugging and profiling tool for Linux. This guide covers essential Valgrind tools, common options, and practical examples to help you identify and fix memory-related issues in your C/C++ programs.

Valgrind Fundamentals

Core Tools Overview

Memcheck: Detects memory management problems like memory leaks, invalid reads/writes, and use of uninitialized values.
Cachegrind: A cache and branch-prediction profiler.
Callgrind: A call graph generating cache and branch prediction profiler. Extends Cachegrind functionality.
Helgrind: Detects threading errors.
DRD (Data Race Detector): Another tool for detecting data races in multithreaded programs.
Massif: Heap profiler, measures how much heap memory your program uses.
DHAT (Dynamic Heap Analysis Tool): A different kind of heap profiler, useful for understanding memory usage over time.

Basic Memcheck Usage

Command	Description
`valgrind --leak-check=full ./myprogram`	Run `myprogram` under Memcheck with full leak checking enabled.
`valgrind --leak-check=summary ./myprogram`	Run `myprogram` under Memcheck, but only provide a summary of leaks.
`valgrind --leak-check=yes ./myprogram`	Enables basic leak checking (same as `--leak-check=summary`).
`valgrind --show-reachable=yes ./myprogram`	Shows reachable memory blocks at program exit (useful for debugging).
`valgrind --track-origins=yes ./myprogram`	Tracks the origin of uninitialized values (can help find the source of errors).

Understanding Memcheck Output

Memcheck reports different kinds of errors:

- **Invalid read/write:** Accessing memory that hasn't been allocated or is outside the bounds of an allocated block.
- **Use of uninitialised value:** Using a variable before it has been assigned a value.
- **Invalid free:** Attempting to free memory that was not allocated with `malloc` or that has already been freed.
- **Memory leak:** Memory that was allocated but never freed before the program exited.

Advanced Memcheck Options

Suppressing Errors

Sometimes Valgrind reports errors that are known and acceptable (e.g., from third-party libraries). You can suppress these errors using a suppression file.

1. Create a suppression file (e.g., `suppressions.txt`) with error descriptions.
2. Use the `--suppressions=suppressions.txt` option to load the file.

Example Suppression File Entry:

```
{
   <insert_a_suppression_name_here>
   Memcheck:Param
   fun:malloc
   ...other matching criteria...
}
```

Controlling Verbosity

Option	Description
`--verbose` or `-v`	Increases verbosity level. Can be specified multiple times for more detail.
`--quiet`	Suppresses most output. Useful for automated testing.

Error Kinds

Valgrind categorizes errors. Key error types include:

- `InvalidRead`
- `InvalidWrite`
- `InvalidFree`
- `Leak_DefinitelyLost`
- `Leak_PossiblyLost`
- `Leak_Reachable`
- `Leak_StillReachable`

Profiling with Cachegrind & Callgrind

Cachegrind Basics

Cachegrind simulates the cache of your CPU, providing insights into cache misses, branch prediction, and instruction counts.

```
valgrind --tool=cachegrind ./myprogram
```

This command generates a `cachegrind.out.pid` file containing profiling data.

Use `cg_annotate` to analyze the Cachegrind output:

```
cg_annotate cachegrind.out.pid
```

This command displays annotated source code with cache statistics.

Callgrind for Function-Level Profiling

Command	Description
`valgrind --tool=callgrind ./myprogram`	Runs `myprogram` under Callgrind, generating `callgrind.out.pid`.
`callgrind_annotate callgrind.out.pid`	Analyzes Callgrind output, showing function-level performance data.
`kcachegrind`	Graphical tool to visualize Callgrind profiling data.

Key Metrics in Cachegrind/Callgrind

- **Ir:** Instructions read
- **I1mr:** Level 1 instruction cache misses
- **Ilmr:** Last level instruction cache misses
- **Dr:** Data reads
- **Dw:** Data writes
- **D1mr:** Level 1 data cache misses
- **D1mw:** Level 1 data cache write misses
- **Dlmr:** Last level data cache reads misses
- **Dlmw:** Last level data cache write misses

Threading Errors with Helgrind & DRD

Helgrind - Threading Error Detection

Helgrind detects potential threading errors, primarily focusing on data races.

```
valgrind --tool=helgrind ./myprogram
```

It identifies locations where multiple threads access the same memory without proper synchronization (e.g., locks).

DRD (Data Race Detector)

DRD is another tool for detecting data races, and often complements Helgrind.

```
valgrind --tool=drd ./myprogram
```

It uses a different algorithm and may find data races that Helgrind misses (and vice versa).

Interpreting Helgrind/DRD Output

Helgrind and DRD reports highlight the lines of code where potential data races occur. Examine these locations carefully to ensure proper synchronization.

Look for missing locks, incorrect lock usage, or other synchronization issues.