Profiling

Profiling measures the performance of a program and can be used to find CPU or memory bottlenecks.

- `time` A stopwatch
- `gprof` The GNU (CPU) Profiler
- `callgrind` Valgrind's CPU profiling tool
- `massif` Valgrind's memory profiling tool

Timing programs with `time`

- Just run `time your_program`!
- Reading `time`'s output:
  - **Real**: The wall-clock or total time the program took to run.
  - **User**: The time the program (and libraries) spent executing CPU instructions.
  - **System**: The time the program spent waiting on system calls (usually I/O).

Profiling with `gprof`

- You must compile with `g++ -pg program.cpp -o program`.
- Then, run your program like normal. It will create a file named `gmon.out`.
- Finally, `gprof program gmon.out` will display profiling statistics!

Understanding `gprof` Output

- Flat profile: Overview of function usage.
- Time measures are based on sampling 100 times/second.
- Function call counts are exact.

Understanding `gprof` Output

- Flat profile: Overview of function usage.
- Time measures are based on sampling 100 times/second.
- Function call counts are exact.
- Call graph: A listing of which functions called each other.
- The line with the index entry is the function under consideration.
- Lines above that are functions that called this function.
- Lines below that are functions that this function called.

Profiling with `callgrind`

- As with Memcheck, compile with `g++ -g program.cpp -o program`
- Run `valgrind --tool=callgrind ./program`. It will create a file named `callgrind.out.NNNN`.
- `callgrind_annotate --auto=yes callgrind.out.NNNN` will print some statistics on your program.
- You can also view the output file directly, although the results are not easy to read.

Understanding `callgrind` Output

- Callgrind counts instructions executed, not time spent.
- The annotated source shows the number of instruction executions a specific line caused.
- Function calls are annotated on the right with the number of times they are called.

Recursion and `callgrind`

- Recursion can confuse both `gprof` and `callgrind`.
- The `--separate-recs=N` option to Valgrind separates function calls up to N deep.
- The `--separate-callers=N` option to Valgrind separates functions depending on which function called them.
- In general, when you have recursion, the call graph and call counts may be wrong, but the instruction count will be correct.

Profiling with `massif`

- Compile with `g++ -g program.cpp -o program`
- Run `valgrind --tool=massif --time-unit=B ./program`. It will create a file named `massif.out.NNNN`.
- To get information on stack memory usage as well, include `--stacks=yes` after `--time-unit=B`.
- `ms_print massif.out.NNNN` will print statistics for you.

Understanding `massif` Output

- Snapshots: `massif` takes a snapshot of the heap on every allocation and deallocation.
  - Most snapshots are **plain**. They record only how much heap was allocated.
  - Every 10th snapshot is **detailed**. These record where memory was allocated in the program.
  - A detailed snapshot is also taken at peak memory usage.

Understanding `massif` Output

- Snapshots: `massif` takes a snapshot of the heap on every allocation and deallocation.
  - Most snapshots are **plain**. They record only how much heap was allocated.
  - Every 10th snapshot is **detailed**. These record where memory was allocated in the program.
  - A detailed snapshot is also taken at peak memory usage.
- The graph: Memory allocated vs. time. Time can be measured in milliseconds, instructions, or bytes allocated.
- Colons (:) indicate plain snapshots, 'at' signs (@) indicate detailed snapshots, and pounds (#) indicate the peak snapshot.

Understanding `massif` Output

- Snapshots: `massif` takes a snapshot of the heap on every allocation and deallocation.
    - Most snapshots are **plain**. They record only how much heap was allocated.
    - Every 10th snapshot is **detailed**. These record where memory was allocated in the program.
    - A detailed snapshot is also taken at peak memory usage.
- The graph: Memory allocated vs. time. Time can be measured in milliseconds, instructions, or bytes allocated.
- Colons (:) indicate plain snapshots, 'at' signs (@) indicate detailed snapshots, and pounds (#) indicate the peak snapshot.
- The chart shows the snapshot number, time, total memory allocated, currently-allocated memory, and extra allocated memory.
- The chart also shows the allocation tree from each detailed snapshot.