Morden cpp notes — profiling

Thursday, December 10, 2020 1:44 PM

Top
- 按M看内存占用(RES/MEM)
- 按P看CPU占用.
- 按 x b. 用<>手动选择排序的列.

看哪些操作消耗了CPU

pstack
打印进程的调用栈,

Strace
显示进程正在运行的系统调用

Perf
- 连续执行多此pstack
- perf top -k -p xxx

Google Performance Tools
gPerftools有CPUProfiler和HeapProfiler两种工具,
CPUProfiler只要在源码里添加3个函数:
- ProfilerStart() 开始性能分析,把数据存入指定文件
- ProfilerRegisterThread() 允许对线程做性能分析
- ProfilerStop() 停止性能分析.

```
auto make_cpu_profiler = //lambda表达式启动性能分析
  [] (const string& filename)  //传入性能分析的数据文件名.
{
   ProfilerStart(filename.c_str()); //启动性能分析
   ProfilerRegisterThread(); //对线程做性能分析.

   return std::shared_ptr<void>( //返回智能指针
      nullptr, //空指针,只用来占位.
      [](void*){          //用删除函数执行停止操作
         ProfilerStop();   //停止性能分析.
      }
   );
};
```

测试正则表达式处理文本性能:

```
auto cp = make_cpu_profiler("case1.perf"); //启动性能分析
auto str = "neir:automata";
for(int i=0; i<1000; i++) { //循环1000次
   auto reg = make_regex(R"(^(\w+)\:(\w+)$)"); //正则表达式对象.
   auto what = make_match();
   assert(regex_match(str, what, reg)); //正则匹配.
}
```

编译运行后得到case1.perf,是二进制的,要用pprof查看.

```
pprof --text ./a.out case1.perf > case1.txt.
```

也可生成图形报告:

```
pprof --svg ./a.out case1.perf > case1.svg
```

通过分析,正则表达式占用了大部分CPU时间,可以提到循环外.

```
auto reg = make_regex(R"(^(\w+)\:(\w+)$)");
auto what = make_match();
for(int i=0; i<1000; i++) {
   assert(regex_match(str, what, reg));
}
```

Tips:
- GCC/Clang内置了Google开发的Sanitizer工具,编译时用 -fSanitize=address 可以检查内存泄漏.