

# Linux 操作系统测试软件分析与研发

## 实验报告

- 实验课题作业 1: Linux操作系统测试软件分析与研发
- 姓名: 梁嘉嘉
- 学号: 23125240
- 课程: 高级操作系统
- 提交日期: 2024/10/09

## 1. 实验目的

本实验旨在通过使用 `stress-ng` 测试软件对 Linux 系统的各个子系统（如 CPU、内存、I/O 等）进行压力测试，分析系统在高负载条件下的性能表现。通过实验过程，掌握系统性能测试的基本原理和方法，熟悉如何构建、编译、运行开源测试工具，并对测试结果进行分析和总结。

## 2. 实验环境

### 2.1 开发环境

- 操作系统: Ubuntu 24.04.1 LTS (noble)
- 虚拟机平台: VMware Workstation Pro 17
- 编译工具:
  - GCC 版本: 13.2.0 (Ubuntu)
  - Make 版本: 4.3 (GNU Make)
  - Git 版本: 2.43.0

```
1 $ sudo apt install gcc -y
2 $ sudo apt install make -y
```

### 2.2 运行环境

- 处理器: AMD Ryzen 7 8845H with Radeon 780M Graphics

- 架构: x86\_64
- CPU 核心数: 2 个核心
- 每个核心的线程数: 1
- L1 缓存: 64 KiB (2 实例)
- L2 缓存: 2 MiB (2 实例)
- L3 缓存: 32 MiB (2 实例)
- 内存: 3.8 GiB
  - 可用内存: 2.6 GiB
  - 交换分区: 3.7 GiB
- 存储:
  - 硬盘大小: 20 GB
  - 分区: 单一分区 / 挂载在 sda2

## 2.3 测试环境

测试工具: stress-ng, version 0.18.05 (gcc 13.2.0, x86\_64 Linux 6.8.0-45-generic) 🖥️🔥

测试类型:

- CPU 性能测试
- 内存性能测试
- I/O 性能测试

## 3. 实验步骤

### 3.1 依赖下载

```
1 $ sudo apt-get install -y gcc g++ libacl1-dev libaio-dev libapparmor-dev
  libatomic1 libattr1-dev libbsd-dev \ libcap-dev libeigen3-dev libgbm-dev
  libcrypt-dev libglvnd-dev libipsec-mb-dev libjpeg-dev libjudy-dev \
  libkeyutils-dev libkmod-dev libmd-dev libmpfr-dev libsctp-dev libxxhash-dev
  zlib1g-dev
```

Library	Version	Description
gcc	4:13.2.0-7ubuntu1	GNU Compiler Collection for C

g++	4:13.2.0-7ubuntu1	GNU Compiler Collection for C++
libacl1-dev	2.3.2-1build1	Access Control List support
libaio-dev	0.3.113-6build1	Asynchronous I/O support
libapparmor-dev	4.0.1really4.0.1-0ubuntu0.24.04.3	AppArmor security library
libatomic1	14-20240412-0ubuntu1	Atomic operations library
libattr1-dev	1:2.5.2-1build1	Extended attributes library
libbsd-dev	0.12.1-1build1	BSD specific development utilities
libcap-dev	1:2.66-5ubuntu2	Capabilities support
libeigen3-dev	3.4.0-4	Eigen matrix library
libgbm-dev	24.0.9-0ubuntu0.1	Generic Buffer Management for graphics
libcrypt-dev	1:4.4.36-4build1	Cryptographic libraries
libglvnd-dev	1.7.0-1build1	OpenGL vendor-neutral dispatch
libipsec-mb-dev	1.5-1build1	Intel IPsec multi-buffer library
libjpeg-dev	8c-2ubuntu11	JPEG image handling library
libjudy-dev	1.0.5-5.1build1	Judy arrays library
libkeyutils-dev	1.6.3-3build1	Key management utilities
libkmod-dev	31+20240202-2ubuntu7	Kernel module support
libmd-dev	1.1.0-2build1	Message Digest algorithm library
libmpfr-dev	4.2.1-1build1	Multiple Precision Floating-Point library
libsctp-dev	1.0.19+dfsg-2build1	Stream Control Transmission Protocol
libxxhash-dev	0.8.2-2build1	Fast hash algorithm
zlib1g-dev	1:1.3.dfsg-3.1ubuntu2.1	Compression library (zlib)

## 3.2 克隆源码

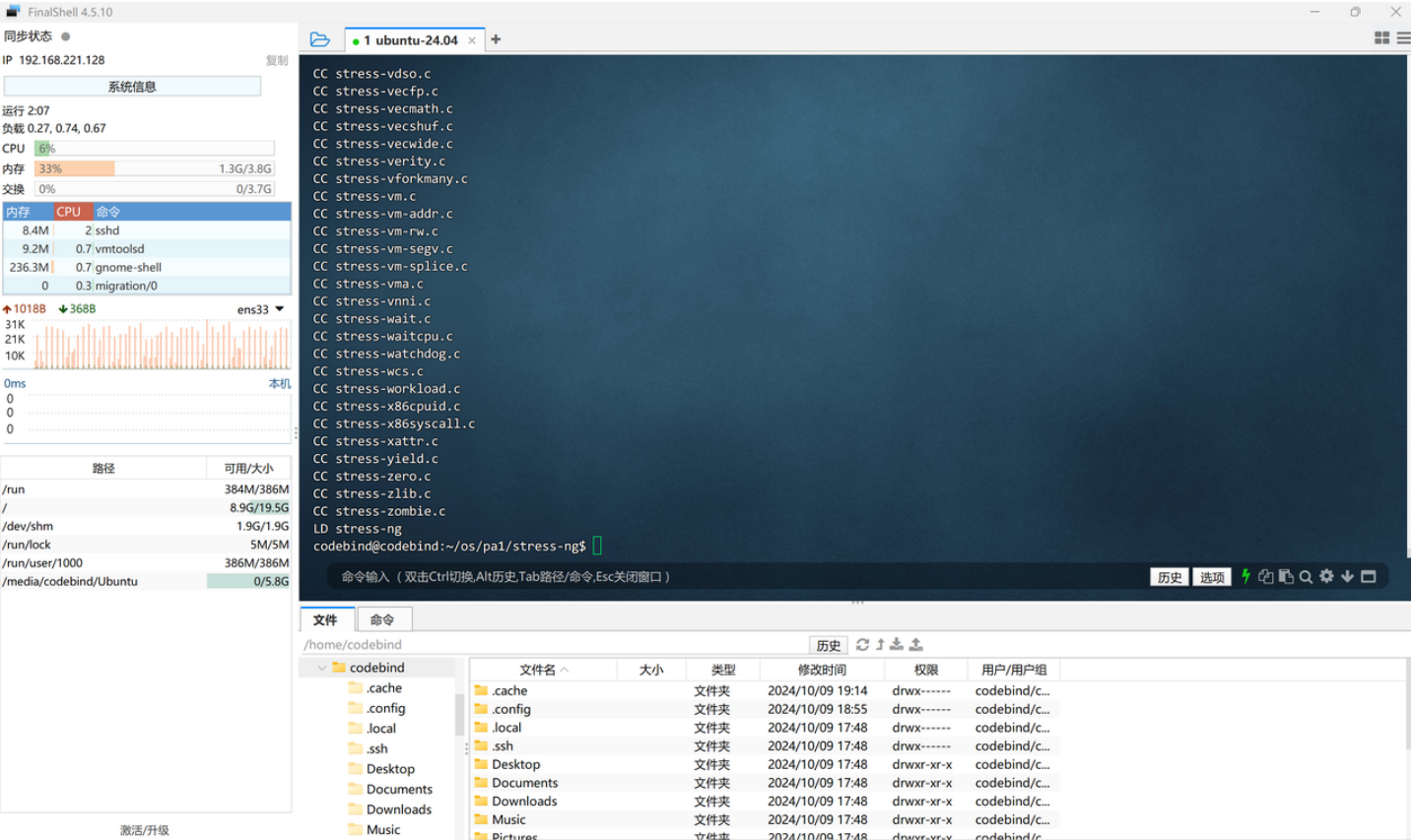
```

1 $ git clone https://github.com/ColinIanKing/stress-ng.git
2 $ cd stress-ng

```

### 3.3 编译构建

```
1 $ make
```



编译成功后，可以运行以下命令查看帮助文档，验证是否编译成功：

```
1 $ ./stress-ng --help
```

FinalShell 4.5.10

同步状态

IP 192.168.221.128

系统信息

运行 2:10

负载 0.21, 0.53, 0.60

CPU 6%

内存 34% 1.3G/3.8G

交换 0% 0/3.7G

内存	CPU	命令
8.4M	2.3	sshd
6.4M	1	top
236.3M	0.7	gnome-shell
0	0.3	rcu_preempt

25K

2K

ens33

32K

22K

11K

0ms

本机

路径	可用/大小
/run	384M/386M
/	8.9G/19.5G
/dev/shm	1.9G/1.9G
/run/lock	5M/5M
/run/user/1000	386M/386M
/media/codebind/Ubuntu	0/5.8G

激活/升级

1 ubuntu-24.04

```
--xattr N          start N workers stressing file extended attributes
--xattr-ops N       stop after N bogo xattr operations
-y N, --yield N     start N workers doing sched_yield() calls
--yield-ops N       stop after N bogo yield operations
--yield-procs N     specify number of yield processes per stressor
--yield-sched P     select scheduler policy [idle, fifo, rr, other, batch, deadline]
--zero N            start N workers exercising /dev/zero with read, mmap, ioctl, lseek
--zero-read         just exercise /dev/zero with reading
--zero-ops N        stop after N /dev/zero bogo read operations
--zlib N            start N workers compressing data with zlib
--zlib-level L      specify zlib compression level 0=fast, 9=best
--zlib-mem-level L  specify zlib compression state memory usage 1=minimum, 9=maximum
--zlib-method M     specify zlib random data generation method M
--zlib-ops N        stop after N zlib bogo compression operations
--zlib-strategy S   specify zlib strategy 0=default, 1=filtered, 2=huffman only, 3=rle, 4=fixed
--zlib-stream-bytes S specify the number of bytes to deflate until the current stream will be closed
--zlib-window-bits W specify zlib window bits -8-(-15) | 8-15 | 24-31 | 40-47
--zombie N          start N workers that rapidly create and reap zombies
--zombie-max N      set upper limit of N zombies per worker
--zombie-ops N      stop after N bogo zombie fork operations

Example: stress-ng --cpu 8 --iomix 4 --vm 2 --vm-bytes 128M --fork 4 --timeout 10s

Note: sizes can be suffixed with B, K, M, G and times with s, m, h, d, y
codebind@codebind:~/os/pa1/stress-ng$
```

命令输入 (双击Ctrl切换Alt历史Tab路径/命令, Esc关闭窗口)

历史 选项

文件 命令

/home/codebind

codebind

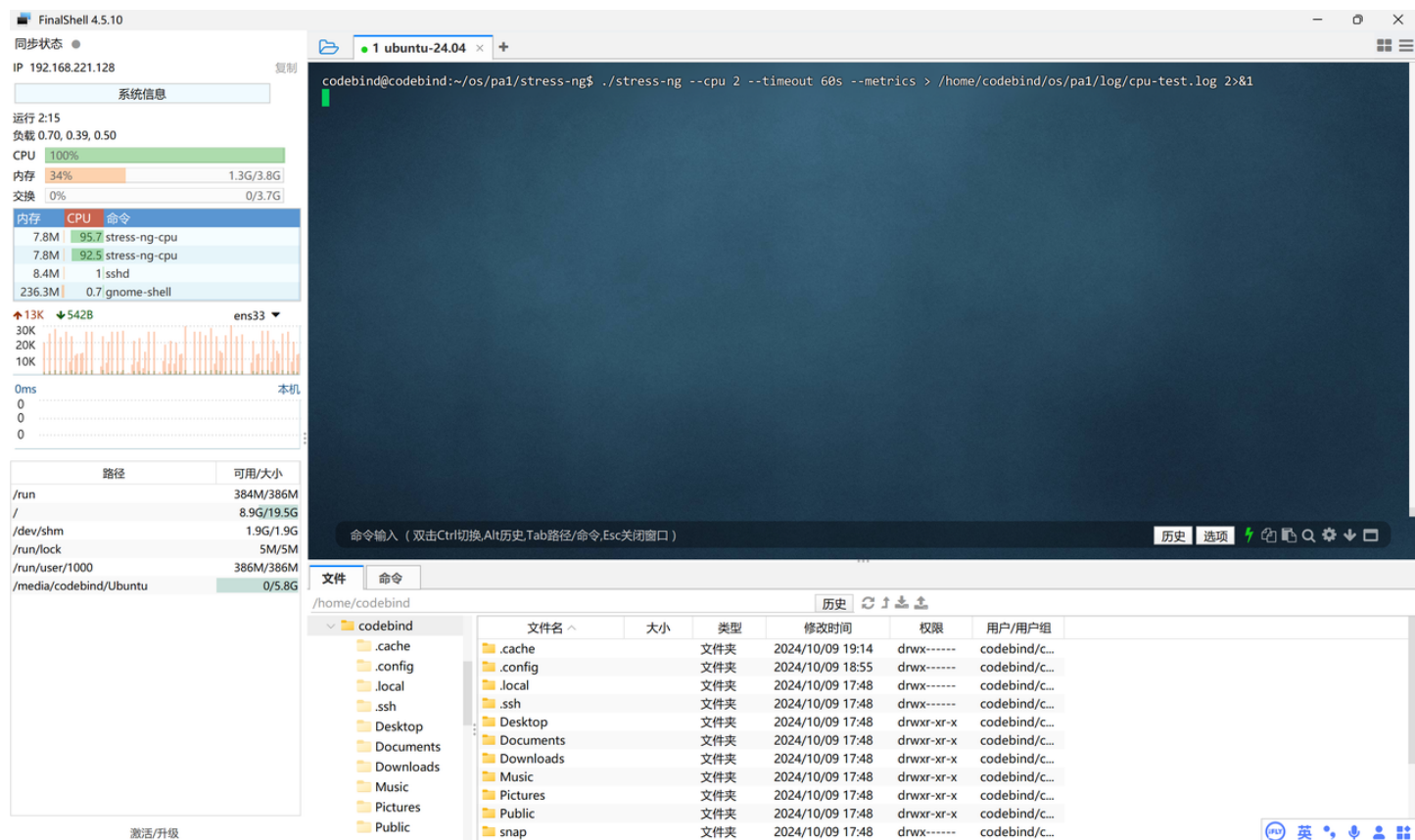
文件名	大小	类型	修改时间	权限	用户/用户组
.cache		文件夹	2024/10/09 19:14	drwx-----	codebind/c...
.config		文件夹	2024/10/09 18:55	drwx-----	codebind/c...
.local		文件夹	2024/10/09 17:48	drwx-----	codebind/c...
.ssh		文件夹	2024/10/09 17:48	drwx-----	codebind/c...
Desktop		文件夹	2024/10/09 17:48	drwxr-xr-x	codebind/c...
Documents		文件夹	2024/10/09 17:48	drwxr-xr-x	codebind/c...
Downloads		文件夹	2024/10/09 17:48	drwxr-xr-x	codebind/c...
Music		文件夹	2024/10/09 17:48	drwxr-xr-x	codebind/c...
Pictures		文件夹	2024/10/09 17:48	drwxr-xr-x	codebind/c...
Public		文件夹	2024/10/09 17:48	drwxr-xr-x	codebind/c...
Public		文件夹	2024/10/09 17:48	drwxr-xr-x	codebind/c...
snap		文件夹	2024/10/09 17:48	drwx-----	codebind/c...

## 3.4 测试结果

### 3.4.1 CPU 测试

模拟两个 CPU 核心的高负载，运行 60 秒：

```
1 $ ./stress-ng --cpu 2 --timeout 60s --metrics > /home/codebind/os/pa1/log/cpu-test.log 2>&1
```



```
1 $ cat /home/codebind/os/pa1/log/cpu-test.log
2 stress-ng: info: [105442] setting to a 1 min run per stressor
3 stress-ng: info: [105442] dispatching hogs: 2 cpu
4 stress-ng: info: [105442] note: /proc/sys/kernel/sched_autogroup_enabled is 1
  and this can impact scheduling throughput for processes not attached to a tty.
  Setting this to 0 may improve performance metrics
5 stress-ng: metrc: [105442] stressor      bogo ops real time  usr time  sys
  time    bogo ops/s      bogo ops/s CPU used per      RSS Max
6 stress-ng: metrc: [105442]                                (secs)      (secs)
  (secs)  (real time) (usr+sys time) instance (%)          (KB)
7 stress-ng: metrc: [105442] cpu              101801      60.00      112.74
  0.44      1696.55          899.41      94.31      8112
8 stress-ng: info: [105442] skipped: 0
9 stress-ng: info: [105442] passed: 2: cpu (2)
10 stress-ng: info: [105442] failed: 0
11 stress-ng: info: [105442] metrics untrustworthy: 0
12 stress-ng: info: [105442] successful run completed in 1 min
```

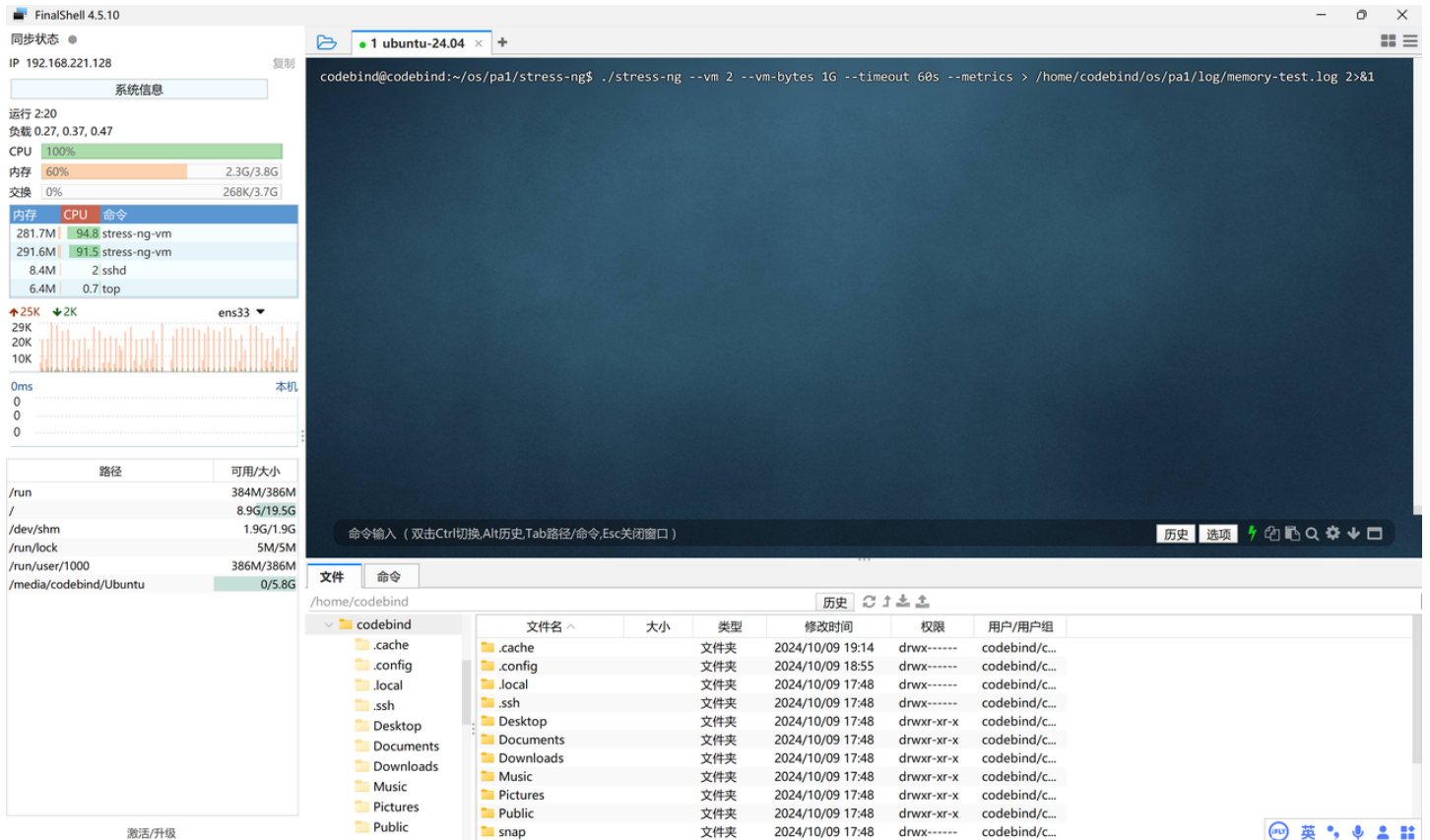
结论：在这次 CPU 压力测试中，`stress-ng` 成功运行了 2 个 CPU 任务，每个任务在 1 分钟内完成。CPU 使用率达到 94.31%，每秒执行了约 1696 次虚拟操作，总共执行了 101801 次虚拟操作。测试过程中没有出现任何错误，系统表现稳定，最大内存占用为 8112 KB。

### 3.4.2 内存测试

模拟两个虚拟内存进程，分配 1GB 内存，持续 60 秒：



```
1 $ ./stress-ng --vm 2 --vm-bytes 1G --timeout 60s --metrics > /home/codebind/os/pa1/log/memory-test.log 2>&1
```



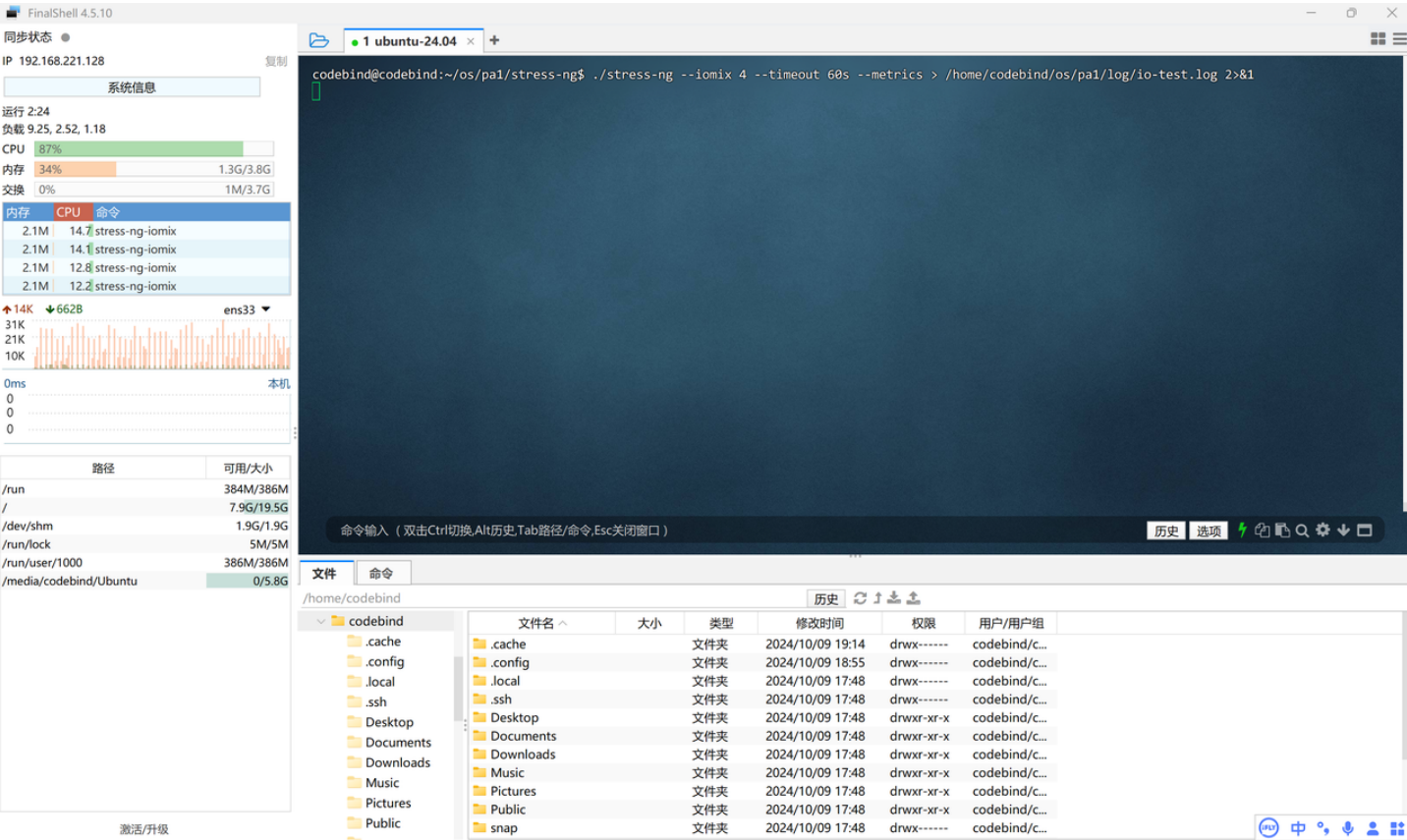
```
1 $ cat /home/codebind/os/pa1/log/memory-test.log
2 stress-ng: info: [109149] setting to a 1 min run per stressor
3 stress-ng: info: [109149] dispatching hogs: 2 vm
4 stress-ng: info: [109149] note: /proc/sys/kernel/sched_autogroup_enabled is 1
  and this can impact scheduling throughput for processes not attached to a tty.
  Setting this to 0 may improve performance metrics
5 stress-ng: metrc: [109149] stressor          bogo ops real time  usr time  sys
  time    bogo ops/s    bogo ops/s CPU used per      RSS Max
6 stress-ng: metrc: [109149]                                (secs)    (secs)
  (secs)  (real time) (usr+sys time) instance (%)          (KB)
7 stress-ng: metrc: [109149] vm                    5221167    61.03    74.67
  38.03    85547.16    46325.62    92.33    593016
8 stress-ng: info: [109149] skipped: 0
9 stress-ng: info: [109149] passed: 2: vm (2)
10 stress-ng: info: [109149] failed: 0
11 stress-ng: info: [109149] metrics untrustworthy: 0
12 stress-ng: info: [109149] successful run completed in 1 min, 1.51 sec
```

结论：在这次内存压力测试中， stress-ng 成功运行了 2 个虚拟内存任务，每个任务持续了约 1 分钟。测试过程中，每秒执行了约 85547 次虚拟内存操作，总共完成了 5221167 次操作。CPU 使用率为 92.33%，最大驻留集大小（内存占用）为 593016 KB。整个测试顺利完成，没有出现任何错误，系统表现稳定。

### 3.4.3 I/O 测试

运行 4 个 I/O 负载进程，持续 60 秒：

```
1 $ ./stress-ng --iomix 4 --timeout 60s --metrics > /home/codebind/os/pa1/log/io-test.log 2>&1
```




```
1 $ cat /home/codebind/os/pa1/log/io-test.log
2 stress-ng: info:  [111366] setting to a 1 min run per stressor
3 stress-ng: info:  [111366] dispatching hogs: 4 iomix
4 stress-ng: info:  [111366] note: /proc/sys/kernel/sched_autogroup_enabled is 1
   and this can impact scheduling throughput for processes not attached to a tty.
   Setting this to 0 may improve performance metrics
5 stress-ng: metrc: [111366] stressor      bogo ops real time  usr time  sys
   time  bogo ops/s      bogo ops/s CPU used per      RSS Max
6 stress-ng: metrc: [111366]
   (secs) (real time) (usr+sys time) instance (%)      (KB)
7 stress-ng: metrc: [111366] iomix          116028      60.25      4.04
   73.88      1925.86      1489.12      32.33      3064
```



```
8 stress-ng: info: [111366] skipped: 0
9 stress-ng: info: [111366] passed: 4: iomix (4)
10 stress-ng: info: [111366] failed: 0
11 stress-ng: info: [111366] metrics untrustworthy: 0
12 stress-ng: info: [111366] successful run completed in 1 min
13
```

结论：在这次 I/O 压力测试中，`stress-ng` 成功运行了 4 个 I/O 混合任务，测试持续了约 1 分钟。每秒执行了约 1926 次虚拟操作，总共完成了 116028 次操作。CPU 使用率为 32.33%，而最大驻留集大小（内存占用）为 3064 KB。整个测试顺利完成，没有出现任何错误，系统在 I/O 负载下表现稳定。

## 4. stress-ng 源码分析

<p>Stress-ng</p> 	<p><a href="https://github.com/ColinIanKing/stress-ng">https://github.com/ColinIanKing/stress-ng</a></p> <p>GitHub - ColinIanKing/stress-ng: This is the stress-ng upstream project git repository. stress-ng w</p> <p>This is the stress-ng upstream project git repository. stress-ng will stress test a...</p>
--	---

`stress-ng` 的入口在 `stress-ng.c` 文件中。负责初始化系统环境、解析命令行参数、设置压力测试环境、执行各类压力测试，并在测试结束后收集性能指标。

### 4.1. 程序初始化

主函数首先进行一些基本的初始化操作，包括获取进程 ID、启用栈溢出检查、设置临时路径以及初始化全局变量：

```
1 main_pid = getpid();
2 stress_set_stack_smash_check_flag(true);
3 stress_set_temp_path(".");
```

这些初始化步骤确保程序在运行过程中能够捕获栈溢出错误，并为后续测试任务准备了临时文件存储路径。

### 4.2. 命令行参数解析

`stress-ng` 使用命令行参数来决定需要运行的压力测试类型及其配置。通过 `stress_parse_opts()` 函数解析输入的命令行参数：

```
1 ret = stress_parse_opts(argc, argv, false);
```

这个函数会根据用户输入的参数，配置 CPU 测试、内存测试、I/O 测试等。如果参数解析失败，程序将会终止。

在参数解析之前，还检查了一些常见的命令行参数组合错误，如不能同时指定 `stderr` 和 `stdout`：

```
1 if ((g_opt_flags & (OPT_FLAGS_STDERR | OPT_FLAGS_STDOUT)) ==  
2     (OPT_FLAGS_STDERR | OPT_FLAGS_STDOUT)) {  
3     (void)fprintf(stderr, "stderr and stdout cannot "  
4         "be used together\n");  
5     ret = EXIT_FAILURE;  
6     goto exit_stressors_free;  
7 }
```

### 4.3. 压力测试初始化

在解析命令行参数并验证成功后，程序开始初始化各类压力测试任务。这里的 `stressors_head` 和 `stressors_tail` 是用于管理所有压力测试任务的链表，初始化之后，会通过一系列函数启动特定的压力测试任务（如 CPU、I/O、内存等）：

```
1 stressor_set_defaults();  
2 stress_enable_all_stressors(0); // 启动所有支持的压力测试
```

这些函数会根据用户配置的命令行参数，决定启动哪些类型的压力测试，并且初始化相关的任务调度和资源分配。

### 4.4. 日志记录与性能数据收集

在进行压力测试时，`stress-ng` 会记录系统的信息、测试任务的配置，以及在运行时收集的各种性能指标：

```
1 stress_log_args(argc, argv); // 记录命令行参数  
2 stress_log_system_info();    // 记录系统信息  
3 stress_log_system_mem_info(); // 记录内存信息  
4 stress_runinfo();            // 记录运行信息  
5 stress_cpuidle_log_info();    // 记录 CPU 空闲信息
```

通过这些日志记录，程序可以对系统运行状态和压力测试任务的配置进行详细追踪，方便后续的性能分析。

## 4.5. 并发管理与任务执行

`stress-ng` 支持并行执行多个压力测试任务，通过 `stress_run_parallel()` 或 `stress_run_sequential()` 函数来管理这些测试任务：

```
1 if (g_opt_flags & OPT_FLAGS_SEQUENTIAL) {
2     stress_run_sequential(ticks_per_sec, &duration, &success,
3     &resource_success, &metrics_success);
4 } else {
5     stress_run_parallel(ticks_per_sec, &duration, &success, &resource_success,
6     &metrics_success);
7 }
```

这段代码根据用户的命令行参数，决定是以并行还是串行的方式执行压力测试任务，并将任务的运行时长、成功状态等信息记录到相应的变量中。

## 4.6. 压力测试结束与数据收集

在测试任务完成后，`stress-ng` 会通过一系列函数收集系统的性能指标数据，并将这些数据输出到日志或 YAML 文件中：

```
1 if (g_opt_flags & OPT_FLAGS_METRICS)
2     stress_metrics_dump(yaml);
3
4 if (g_opt_flags & OPT_FLAGS_INTERRUPTS)
5     stress_interrupts_dump(yaml, stressors_head);
```

除了记录常规的运行时间、操作次数 (bogo ops) 等数据外，还可以记录 CPU 的中断数据、性能统计（如使用 `perf` 统计的硬件计数器）以及热区温度等信息。

## 4.7. 资源清理与退出

当所有压力测试任务完成并记录完性能数据后，程序会释放分配的内存和其他资源。通过调用 `stress_shared_unmap()`、`stress_stressors_free()` 等函数，确保程序不会有资源泄漏：

```
1 stress_shared_heap_deinit();
2 stress_stressors_deinit();
3 stress_shared_unmap();
4
```

最后，程序根据测试任务的成功与否退出，并返回相应的退出代码。

## 4.8. 异常处理与退出机制

在整个函数中，`setjmp()` 和 `longjmp()` 被用于捕获全局错误和异常。通过这种机制，`stress-ng` 能够在程序运行过程中发生异常时，优雅地退出并释放所有已分配的资源。

## 5. 疑难解惑与经验教训



```
$ git clone https://github.com/ColinIanKing/stress-ng.git
```

```
Cloning into 'stress-ng'...
```

```
error: RPC failed; curl 7 Recv failure: Connection reset by peer
```

```
fatal: expected flush after ref listing
```

解决方案： `$ git config --global http.sslVerify "false"`



```
$ git clone https://github.com/ColinIanKing/stress-ng.git
```

```
Cloning into 'stress-ng'...
```

```
fatal: unable to access 'https://github.com/ColinIanKing/stress-ng.git/': Failed to connect to github.com port 443 after 21060 ms: Couldn't connect to server
```

解决方案：网络问题，尝试多试几次。

- **问题回顾：**在编译过程中遇到依赖安装问题，及时查找并解决。
- **经验教训：**通过实验加深了对 Linux 性能测试工具的理解，掌握了如何通过模拟不同的负载场景来评估系统的性能表现。

## 6. 结论与体会

通过本次实验，深入学习了 `stress-ng` 等 Linux 测试工具的使用及源码结构，对系统的性能测试有了更加深入的认识。实验过程中遇到的依赖安装和高负载响应问题也进一步锻炼了查找问题和解决问题的能力。此外，本次实验提升了对编译构建流程的理解，也增强了对 Linux 系统性能的直观感知。