Generating core files

Ubuntu include service names apport which intercepts crashes upon occurring and gathers potentially useful information about the crash and the OS environment e.g core dumps.

Due to it's potentially sensitive data gathering, it is disable by defult such as core dumps, stack traces, and log files.

in order to create a Core File which includes a snapshot of the current stack of threads of the process we need apport to be active and generate core files from dumps.

start apport service and verify it is active

```
systemctl status apport
● apport.service - LSB: automatic crash report generation
     Loaded: loaded (/etc/init.d/apport; generated)
     Active: active (exited) since Mon 2023-12-18 08:54:59 IST; 1 week 3 days
ago
       Docs: man:systemd-sysv-generator(8)
        CPU: 26ms

Dec 18 08:54:59 [HOSTNAME]systemd[1]: Starting LSB: automatic crash report
generation...
Dec 18 08:54:59 [HOSTNAME]apport[632]:  * Starting automatic crash report
generation: apport
Dec 18 08:54:59 [HOSTNAME]apport[632]:    ...done.
Dec 18 08:54:59 [HOSTNAME]systemd[1]: Started LSB: automatic crash report
generation.
```

In general core files are large in size and are control using
`ulimit -c`
where '-c' is the maximum size of core files created.

the mentioned command displays the limit of the core file and it defaults to 0.
in order to allow large core dumps to be generated it is recommended to set to some large number in bytes or just use
`ulimit -c unlimited`

Now we are ready to generate a core file from the system level.

assume we have the following c code.

```
void a()
{
    while (1)
    {
    }
}
void b()
```

```
{
    a();
}

void c()
{
    b();
}

void d()
{
    c();
}

void e()
{
    d();
}

void f()
{
    e();
}

void g()
{
    f();
}

void h()
{
    g();
}

void main()
{
    h();
}
```

and we compile it with -g3 which generates dwarf symbols.
and just in case use ‘-O0’ for disabling optimization.

```
gcc -g3 -O0 a.c -o sample_program
```

in case program is attachable using gdb than core dump is less likely to be used
but there are cases where program hangs and it is impossible to attach using gdb
we can force it to crash, allow apport to collect the crash data and create core file

Sending SIGABRT signal to the desired process generally initiate creating a core dump.

```
./sample_program.out &
 kill -SIGABRT $(ps -fe | awk '/[s]ample_program/ {print $2}')
ls -l /var/lib/apport/coredump/
```

```
-r-------- 1 root root    307200 Dec 27 18:48
core._tmp_sample_program.0.7606b628-0f80-47fd-b0b1-
0838ba9c51ea.286880.81296197
```

Now we can use gdb to inspect core file

```
gdb ./sample_program
/var/lib/apport/coredump/core._tmp_sample_program.0.7606b628-0f80-47fd-b0b1-
0838ba9c51ea.286880.81296197
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./sample_program...
[New LWP 286880]
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Core was generated by `./sample_program'.
Program terminated with signal SIGABRT, Aborted.
#0  a () at a.c:3
3           while (1)
(gdb) bt
#0  a () at a.c:3
#1  0x000056092047c145 in b () at a.c:9
#2  0x000056092047c15a in c () at a.c:14
#3  0x000056092047c16f in d () at a.c:19
#4  0x000056092047c184 in e () at a.c:24
#5  0x000056092047c199 in f () at a.c:29
#6  0x000056092047c1ae in g () at a.c:34
#7  0x000056092047c1c3 in h () at a.c:39
#8  0x000056092047c1d8 in main () at a.c:44
```

In this simple example not much can be deduce,
but in cases where program is heavy and unresponsive to gdb -p ${pid}
we can identify busy loops, mutex waits etc..