Chapter 2 ICE

1. Use the command **systemctl status nginx**

For example, for the sshd service:

bob@raspberrypi:~ $ **systemctl status ssh**d

● ssh.service - OpenBSD Secure Shell server

Loaded: loaded (/lib/systemd/system/ssh.service; enabled; preset: enabled)

Active: active (running) since Sat 2025-09-06 05:54:12 PDT; 3 days ago

Docs: man:sshd(8)

man:sshd\_config(5)

Process: 821 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)

Main PID: 847 (sshd)

Tasks: 1 (limit: 4761)

CPU: 1.934s

CGroup: /system.slice/ssh.service

└─847 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"…

2. No Answer Required.

3. No Answer Required.

4. No Answer Required.

5. systemctl is-active --quiet ssh && echo "running" || (systemctl is-enabled --quiet ssh && echo "installed but not running" || echo "not installed")

6.The main difference lies in their focus and the type of hierarchy they present.

**systemd-cgls**:

systemd-cgls is specific to systems using systemd. It’s used to display the hierarchy of control groups (cgroups) managed by systemd. It shows the systemd unit hierarchy and control group relationships, allowing you to visualize how systemd organizes and manages processes in cgroups.

**pstree**:

pstree is a more general-purpose command that displays the process hierarchy of the entire system.

It provides a hierarchical view of processes, showing their parent-child relationships, starting from the init process (usually PID 1). It displays a tree-like structure of processes, making it easier to understand the parent-child relationships and the overall process tree.

7.Short answer first:  
  
PIDs vs threads: each process has a unique PID; each thread inside that process has its own TID (thread-id) but shares the process’s TGID (the PID of the thread-group leader). pstree shows processes by default; add -T to also list threads with their TIDs.  
  
cgroups: on modern systems (cgroup v2), all threads of a process share the same cgroup (same cgroup path under /proc/<pid>/cgroup). systemd-cgls lists processes (thread-group leaders) in their cgroup; individual threads aren’t usually shown separately unless you intentionally use cgroup “threaded” mode. Children created with fork() inherit the parent’s cgroup unless you or a manager (e.g., systemd) moves them.  
  
Below is a compact C program that:  
  
forks multiple processes (your “persistent threads”),  
  
also creates POSIX threads inside each child (so you can compare process vs thread behavior),  
  
prints PID, TID, TGID, PPID, and cgroup path for each entity,  
  
then sleeps so you can inspect with pstree and systemd-cgls.  
  
Compile & run:

gcc -O2 -pthread -Wall -Wextra -o cg\_demo cg\_demo.c

./cg\_demo 3 2 # 3 children; each child spawns 2 pthreads

# In another terminal:

pstree -Tp $(pgrep -n cg\_demo) # show process + thread tree (TIDs)

systemd-cgls | grep -E 'cg\_demo|PID' -n

#define \_GNU\_SOURCE

#include <errno.h>

#include <pthread.h>

#include <sched.h>

#include <signal.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/syscall.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

static long get\_tid(void) { return syscall(SYS\_gettid); }

static void read\_cgroup\_path(char \*buf, size\_t bufsz) {

// Works for cgroup v2; prints the path after the last ':'

FILE \*f = fopen("/proc/self/cgroup", "r");

if (!f) { snprintf(buf, bufsz, "ERR: fopen: %s", strerror(errno)); return; }

char line[4096], last[4096] = {0};

while (fgets(line, sizeof(line), f)) { strncpy(last, line, sizeof(last)-1); }

fclose(f);

// Format examples:

// v2: "0::/user.slice/user-1000.slice/session-2.scope\n"

// v1: "5:cpuacct,cpu:/user.slice/..."; we keep the whole line for clarity

char \*nl = strchr(last, '\n'); if (nl) \*nl = '\0';

char \*after = strrchr(last, ':'); after = after ? after + 1 : last;

snprintf(buf, bufsz, "%s", after);

}

struct thread\_arg { int child\_idx; int thread\_idx; };

static void \*thread\_fn(void \*argp) {

struct thread\_arg \*a = (struct thread\_arg \*)argp;

char cg[4096]; read\_cgroup\_path(cg, sizeof(cg));

printf("[pthread] child=%d thr=%d PID=%d TGID=%d TID=%ld PPID=%d cgroup=%s\n",

a->child\_idx, a->thread\_idx, getpid(), getpid(), get\_tid(), getppid(), cg);

fflush(stdout);

// stay alive so you can inspect with pstree/systemd-cgls

sleep(30);

return NULL;

}

static void child\_work(int child\_idx, int n\_threads) {

// Report from the process (thread-group leader)

char cg[4096]; read\_cgroup\_path(cg, sizeof(cg));

printf("[process] child=%d PID=%d TGID=%d TID=%ld PPID=%d cgroup=%s\n",

child\_idx, getpid(), getpid(), get\_tid(), getppid(), cg);

fflush(stdout);

pthread\_t \*ths = calloc((size\_t)n\_threads, sizeof(pthread\_t));

struct thread\_arg \*args = calloc((size\_t)n\_threads, sizeof(struct thread\_arg));

for (int i = 0; i < n\_threads; i++) {

args[i].child\_idx = child\_idx; args[i].thread\_idx = i;

int rc = pthread\_create(&ths[i], NULL, thread\_fn, &args[i]);

if (rc) fprintf(stderr, "pthread\_create: %s\n", strerror(rc));

}

// Keep child process alive while threads run

sleep(35);

free(ths); free(args);

\_exit(0);

}

int main(int argc, char \*\*argv) {

int n\_children = (argc > 1) ? atoi(argv[1]) : 2;

int n\_threads = (argc > 2) ? atoi(argv[2]) : 2;

if (n\_children < 1) n\_children = 1;

if (n\_threads < 0) n\_threads = 0;

char cg[4096]; read\_cgroup\_path(cg, sizeof(cg));

printf("[parent ] PID=%d TGID=%d TID=%ld PPID=%d cgroup=%s\n",

getpid(), getpid(), get\_tid(), getppid(), cg);

fflush(stdout);

for (int i = 0; i < n\_children; i++) {

pid\_t pid = fork();

if (pid == 0) {

child\_work(i, n\_threads);

} else if (pid < 0) {

perror("fork");

}

}

// Parent waits so you can inspect

int status;

while (wait(&status) > 0) { }

return 0;

}

What you should observe in the program’s output:  
  
Threads show different TIDs but the same PID/TGID as their owning process.  
  
All threads inside a process print the same cgroup path (e.g., /user.slice/.../session-2.scope), confirming shared cgroup membership.  
  
Forked children usually inherit the same cgroup as the parent, so they’ll print the same path unless your system/service manager moves them.  
  
With pstree -T you’ll see the process tree with threads listed under each cg\_demo child; TIDs will match the TID= numbers printed by the threads.  
  
With systemd-cgls you’ll see the processes (thread-group leaders) placed under a slice/scope. Individual threads normally won’t appear separately; they live in the same cgroup as their leader.  
  
If you want to take it further, you can manually move one child into a different cgroup (e.g., with systemd-run --scope -p ... or by writing that PID into another cgroup’s cgroup.procs) and rerun—its cgroup path will differ while its threads still match that new path.

8. What systemd units are  
  
In systemd’s model, there are three common types of unit you’ll see when you look at systemd-cgls:  
  
Slice units (\*.slice):  
Hierarchical buckets that group resources for services/scopes. For example, user.slice or system.slice. They don’t directly contain processes — they contain other slices, scopes, or services.  
  
Service units (\*.service):  
These correspond to long-running daemons or programs started and managed by systemd (like nginx.service, sshd.service). systemd knows how to start, stop, restart them.  
  
Scope units (\*.scope):  
These are transient cgroups created when processes are started externally (by a user session, systemd-run --scope, or a login manager). They’re not managed as services but systemd still tracks them.  
  
Where your test program fits  
  
When you launch your cg\_demo binary manually from your login shell, systemd doesn’t create a new service unit for it.  
  
Instead, the process is part of your user session scope unit (e.g. session-2.scope) which itself sits under user-1000.slice (if you’re UID 1000).  
  
That means:  
  
All forked children and all pthread threads of your program belong to that same scope unit.  
  
They’re not slices (slices are higher-level containers).  
  
They’re not services (because you didn’t ask systemd to manage cg\_demo as a service).  
  
They’re plain processes inside a scope.  
  
 Why threads don’t get their own units  
  
systemd’s unit model works at the process (PID / thread-group) level, not at the individual thread level. All threads of a process live in the same cgroup as the process itself. systemd doesn’t subdivide them further because:  
  
cgroup v2 accounts resources per cgroup, not per pthread.  
  
The kernel itself exposes threads separately (via TIDs in /proc), but systemd chooses process-level granularity.  
  
So the answer really is:

Your demo program’s threads are just threads inside processes; those processes live inside a scope unit (your login session). They are not slices or services, because:  
  
slices are only containers for units, services are managed daemons, scopes are the default bucket for externally started processes (like anything you run in your shell.)

9. The first one showed when the system was first booted, the second one the current boot time(if you scroll down far enough.)

10. Shows journal entries since the last boot.

11. This is a repeat of In-Chapter Exercise 5. **journalctl app\_name**, where **app\_name** is the misbehaving app. **journalctl -r**

12. No answer required.

13. Create the [Timer} section in the service unit file to do that.

14. a. **sudo systemctl stop** the myscript service unit file.

b. **sudo systemctl disable myscript.timer**

15. Runs the script file based on a calendar event, like Monday through Friday, not a clock-based interval.

16. a. No answer required.

b. Either customize ufw filters to do this on the server, or the computer you’re running simp.service on, or on the modem/router(if that’s possible on your LAN) to safely allow access to your system.

17. a) After **<Ctrl> + C**, use the **kill -9** command on its PID number.

b) A web page not found eror is displayed in the browser window.

18. No answer required.