System Call Analysis Report: `__sys_killall`

Overview

This report provides a detailed explanation and line-by-line commentary of the C function `__sys_killall`, which is responsible for terminating all processes with a specific name from the running and ready queues of a given process.

Function Purpose

The function `__sys_killall` scans both the running and ready queues of the calling process. It reads a target process name from a memory region and terminates any process matching that name.

Source Code with Comments

```c
int __sys_killall(struct pcb_t *caller, struct sc_regs* regs) {
    char proc_name[100];              // Buffer to store the process name
    uint32_t data;                    // Temp variable for memory reading

    uint32_t memrg = regs->a1;        // Get memory region ID from register

    memset(proc_name, 0, sizeof(proc_name));  // Clear the proc_name buffer

    int index = 0;
    int result = 0;

    // Read the process name from memory
    while (index < 99) {
        result = libread(caller, memrg, index, &data);
        if (result != 0) break; // Stop if read fails

        if (data == -1 || data == 0) {
            proc_name[index] = '\0'; // End of string
            break;
        }

        if (data > 0 && data < 128) {
            proc_name[index++] = (char)data;
        } else {
            proc_name[index] = '\0'; // Invalid char, terminate string
            break;
        }
    }

    proc_name[index] = '\0'; // Ensure null termination

    if (index == 0) {
        strcpy(proc_name, "P0"); // Default name if input invalid
    }

    printf("The procname retrieved from memregionid %d is \"%s\"\n", memrg, proc_name);

    int terminated_count = 0;

    // Handle running queue
    if (caller->running_list != NULL) {
        struct queue_t temp_queue;
        temp_queue.size = 0;

        while (!empty(caller->running_list)) {
            struct pcb_t *proc = dequeue(caller->running_list);
            if (proc == NULL) break;

            if (proc->path != NULL && strcmp(proc->path, proc_name) == 0) {
                printf("Terminating running process pid=%d, name=%s\n", proc->pid, proc->path);
                terminated_count++;
            } else {
                enqueue(&temp_queue, proc);
            }
        }

        while (!empty(&temp_queue)) {
            enqueue(caller->running_list, dequeue(&temp_queue));
        }
    }

    // Handle ready queue
    if (caller->ready_queue != NULL) {
        struct queue_t temp_queue;
        temp_queue.size = 0;

        while (!empty(caller->ready_queue)) {
```

```c
            struct pcb_t *proc = dequeue(caller->ready_queue);
            if (proc == NULL) break;

            if (proc->path != NULL && strcmp(proc->path, proc_name) == 0) {
                printf("Terminating ready process pid=%d, name=%s\n", proc->pid, proc->path);
                terminated_count++;
            } else {
                enqueue(&temp_queue, proc);
            }
        }

        while (!empty(&temp_queue)) {
            enqueue(caller->ready_queue, dequeue(&temp_queue));
        }
    }

    printf("Total %d processes named \"%s\" terminated\n", terminated_count, proc_name);
    return terminated_count;
}
```

Summary

- The function reads a process name from a memory region.
- It loops through both the running and ready queues of the caller.
- For every process that matches the given name, it simulates termination (printing only).
- The number of terminated processes is returned.

Future Improvements

- Re-enable and test memory cleanup via `libfree()` for real termination.
- Add error handling for malformed memory reads.
- Consider supporting wildcard or regex matching.

Test Case Output

Below is the console output for a test case using the `__sys_killall` system call. The process writes a name into a memory region and attempts to kill processes matching that name.

```
Time slot   0
ld_routine
Time slot   1
Time slot   2
Time slot   3
Time slot   4
Time slot   5
Time slot   6
Time slot   7
Time slot   8
Time slot   9
        Loaded a process at input/proc/sc2, PID: 1 PRIO: 15
Time slot  10
        CPU 0: Dispatched process  1
===== PHYSICAL MEMORY AFTER ALLOCATION =====
PID=1 - Region=1 - Address=00000000 - Size=100 byte
print_pgtbl: 0 - 512
00000000: 80000000
00000004: 00000000
Time slot  11
write region=1 offset=0 value=80
print_pgtbl: 0 - 512
00000000: 80000000
00000004: 00000000
===== PHYSICAL MEMORY DUMP =====
===== PHYSICAL MEMORY END-DUMP =====
Time slot  12
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
write region=1 offset=1 value=48
print_pgtbl: 0 - 512
00000000: 90000000
00000004: 00000000
===== PHYSICAL MEMORY DUMP =====
BYTE 00000000: 80
===== PHYSICAL MEMORY END-DUMP =====
Time slot  13
write region=1 offset=2 value=-1
print_pgtbl: 0 - 512
00000000: 90000000
00000004: 00000000
===== PHYSICAL MEMORY DUMP =====
BYTE 00000000: 80
BYTE 00000001: 48
===== PHYSICAL MEMORY END-DUMP =====
Time slot  14
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
```

```
read region=1 offset=0 value=80
print_pgtbl: 0 - 512
00000000: 90000000
00000004: 00000000
===== PHYSICAL MEMORY DUMP =====
BYTE 00000000: 80
BYTE 00000001: 48
BYTE 00000002: -1
===== PHYSICAL MEMORY END-DUMP =====
read region=1 offset=1 value=48
print_pgtbl: 0 - 512
00000000: 90000000
00000004: 00000000
===== PHYSICAL MEMORY DUMP =====
BYTE 00000000: 80
BYTE 00000001: 48
BYTE 00000002: -1
===== PHYSICAL MEMORY END-DUMP =====
read region=1 offset=2 value=-1
print_pgtbl: 0 - 512
00000000: 90000000
00000004: 00000000
===== PHYSICAL MEMORY DUMP =====
BYTE 00000000: 80
BYTE 00000001: 48
BYTE 00000002: -1
===== PHYSICAL MEMORY END-DUMP =====
The procname retrieved from memregionid 1 is "P0"
Total 0 processes named "P0" terminated
        CPU 0: Processed  1 has finished
Time slot  15
        CPU 0 stopped
```

Interpretation

- A process wrote the ASCII values ‘80‘, ‘48‘, and ‘-1‘ to memory.
- These values decode to ‘"P0"‘ and mark the end of string with ‘-1‘.
- The function searched for processes named ‘"P0" ("default name")‘ in the queues.
- No such process existed, so zero processes were terminated.