بسمه تعالی

گزارش تمرین عملی سوم – سوال سوم – درس سیستم‌های عامل دکتر اسدی – نیمسال اول 405-404

نویسندگان:

1. سیداحمد موسوی‌اول – 402106648
2. عرفان تیموری – 402105813

گزارشکار.

قسمت دوم

2.1. برای پیاده‌سازی تایمر مشابه قسمت‌های قبل یک سیسکال جدید تعریف می‌کنیم که در آن کافیست تایم را در کرنل لینوکس دریافت کنیم و آن را خروجی دهیم:

```
1   SYSCALL_DEFINE0(get_time_ns)
2   {
3       return ktime_get_real_ns();
4   }
```

مانند قسمت‌های قبل تعریف این سیسکال را در فایل‌های مربوطه وارد می‌کنیم تا بتوانیم از دستور get_time_ns() استفاده کنیم. توجه داریم از آنجایی که این تابع زمان را با دقت نانوثانیه خروجی می‌دهد از دقت بسیار بالایی برخوردار است.

2.2. به منظور اندازه‌گیری دقیق زمان ورود و خروج از فراخوانی‌های سیستمی و تعیین سربار ناشی از انتقال وضعیتی که (Context Switch) بین فضای کاربر و هسته وجود دارد، یک متدولوژی مشخص اتخاذ می‌گردد. این فرآیند بر مبنای ثبت چهار نقطه زمانی کلیدی (AA، BB، CC و DD) در نمودار انتقال U -> K -> K -> U استوار است. نقاط AA (آخرین لحظه در فضای کاربر، قبل از صدور دستورالعمل syscall) و DD (اولین لحظه پس از بازگشت کامل به فضای کاربر) مستقیما در فضای کاربر با استفاده از همان تابعی که در بخش 2.1 پیاده‌سازی کردیم اندازه‌گیری می‌شوند. برای ثبت نقاط BB (لحظه ورود به بدنه فراخوانی سیستمی در هسته) و CC (لحظه خروج از بدنه همان فراخوانی)، ساختار داده‌ای task_struct در هسته لینوکس اصلاح خواهد شد؛ بدین ترتیب که دو فیلد ۶۴ بیتی (به عنوان مثال t2 و t3) به آن افزوده می‌شود. این فیلدها در ابتدا و انتهای بدنه فراخوانی سیستمی مورد نظر، با استفاده از توابع زمان‌سنجی داخلی هسته مقداردهی می‌شوند. در نهایت، جهت بازیابی این دو برچسب زمانی از فضای هسته، یک فراخوانی سیستمی کمکی مجزا پیاده‌سازی خواهد شد که وظیفه آن صرفا خواندن این دو فیلد از task_struct فرآیند جاری و بازگرداندن آن‌ها به فضای کاربر جهت تحلیل نهایی است. اکیدا توصیه می‌شود که برای جلوگیری از ایجاد تداخل در نتایج، بروز خطاهای بازگشتی یا اختلال در پایداری سیستم (به‌ویژه هنگام آزمایش syscall های I/O)، از هیچ‌گونه عملیات چاپی مانند printk در داخل بدنه اصلی فراخوانی سیستمی که تحت اندازه‌گیری است، استفاده نگردد.

```c
SYSCALL_DEFINE0(get_last_syscall_times)
{
    u64 t2 = current->last_syscall_t2_ns;
    u64 t3 = current->last_syscall_t3_ns;
    const char *name = current->last_syscall_name;
    pid_t pid = task_tgid_vnr(current);

    if (!name)
        name = "N/A";

    printk(KERN_INFO "[syscall_report] PID=%d, Syscall=%s, T2=%llu, T3=%llu, Exec_Time=%llu ns\n",
           pid, name, t2, t3, (t3 - t2));

    return 0;
}
```

```c
SYSCALL_DEFINE3(read, unsigned int, fd, char __user *, buf, size_t, count)
{
    u64 t2_ns, t3_ns;
    ssize_t ret;

    t2_ns = ktime_get_real_ns();
    ret = ksys_read(fd, buf, count);
    t3_ns = ktime_get_real_ns();

    current->last_syscall_t2_ns = t2_ns;
    current->last_syscall_t3_ns = t3_ns;
    current->last_syscall_name = "read";

    return ret;
}

SYSCALL_DEFINE3(write, unsigned int, fd, const char __user *, buf,
        size_t, count)
{
    u64 t2_ns, t3_ns;
    ssize_t ret;

    t2_ns = ktime_get_real_ns();
    ret = ksys_write(fd, buf, count);
    t3_ns = ktime_get_real_ns();

    current->last_syscall_t2_ns = t2_ns;
    current->last_syscall_t3_ns = t3_ns;
    current->last_syscall_name = "write";

    return ret;
```

```c
long my_custom_field;

u64 last_syscall_t2_ns;
u64 last_syscall_t3_ns;

const char *last_syscall_name;
```

در نهایت برای هر 5 سیسکال گفته شده در سوال یک کد سمت کاربر می‌زنیم که این زمان‌ها را محاسبه کرده و خروجی می‌دهد. نمونه‌ای از این قسمت در عکس زیر قابل مشاهده است و فایل کامل آن به گزارش ضمیمه شده است.

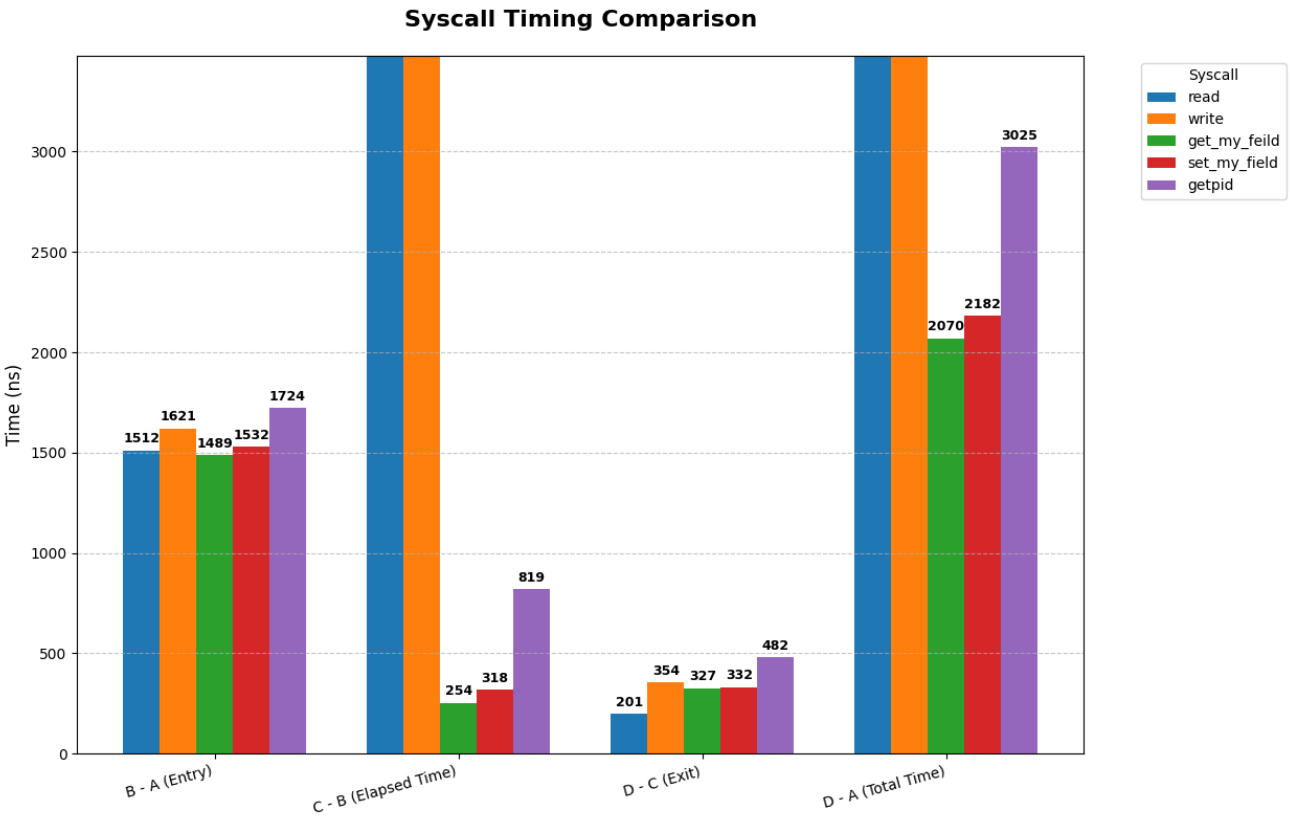
```c
// --- Test1: GETPID ---
t1 = syscall(SYS_GET_TIME_NS);
ret_val = syscall(SYS_GETPID);
t4 = syscall(SYS_GET_TIME_NS);
my_pid = ret_val;
syscall(SYS_GET_LAST_SYSCALL_TIMES);
printf("--- getpid Test (PID=%ld) ---\n", my_pid);
printf("Total User-Perceived Time (T4 - T1): %llu ns\n", (t4 - t1));

// --- Test2: SET_MY_FIELD ---
t1 = syscall(SYS_GET_TIME_NS);
ret_val = syscall(SYS_SET_MY_FIELD, 999);
t4 = syscall(SYS_GET_TIME_NS);
syscall(SYS_GET_LAST_SYSCALL_TIMES);
printf("\n--- set_my_field Test ---\n");
printf("Total User-Perceived Time (T4 - T1): %llu ns\n", (t4 - t1));

// --- Test3: GET_MY_FIELD ---
t1 = syscall(SYS_GET_TIME_NS);
ret_val = syscall(SYS_GET_MY_FIELD);
t4 = syscall(SYS_GET_TIME_NS);
syscall(SYS_GET_LAST_SYSCALL_TIMES);
printf("\n--- get_my_field Test (Val=%ld) ---\n", ret_val);
printf("Total User-Perceived Time (T4 - T1): %llu ns\n", (t4 - t1));
```

در نهایت بنچمارک را 3 بار اجرا کرده و نتایجی مشابه زیر بدست می‌آوریم: (سایر نتایج در زیپ گزارش آماده است)

```
osvm@Teymouri-Mossaviawal:~/Desktop$ ./benchmark
--- Start Benchmark ---

--- getpid Test (PID=1484) ---
Total User-Perceived Time (T4 - T1): 3025 ns

--- set_my_field Test ---
Total User-Perceived Time (T4 - T1): 2182 ns

--- get_my_field Test (Val=999) ---
Total User-Perceived Time (T4 - T1): 2070 ns

--- read (0 bytes) Test ---
Total User-Perceived Time (T4 - T1): 6098 ns

test--- write (5 bytes) Test ---
Total User-Perceived Time (T4 - T1): 23510 ns
osvm@Teymouri-Mossaviawal:~/Desktop$ sudo dmesg
[  139.803150] [syscall_report] PID=1484, Syscall=getpid, T2=1762871717003285963, T3=1762871717003286782, Exec_Time=819 ns
[  139.803225] [syscall_report] PID=1484, Syscall=set_my_field, T2=1762871717003362383, T3=1762871717003362701, Exec_Time=318 ns
[  139.805509] [syscall_report] PID=1484, Syscall=get_my_field, T2=1762871717005644874, T3=1762871717005645128, Exec_Time=254 ns
[  139.805559] [syscall_report] PID=1484, Syscall=read, T2=1762871717005691806, T3=1762871717005696191, Exec_Time=4385 ns
[  139.805659] [syscall_report] PID=1484, Syscall=write, T2=1762871717005774052, T3=1762871717005795587, Exec_Time=21535 ns
```

حال به کمک یک اسکریپت نتایج بدست آماده را به نمودارهای زیر تبدیل می‌کنیم:

بر اساس داده‌ها، واضح است که بخش اصلی زمان در فراخوانی‌های سیستمی ساده و سریع (مانند get_my_field, set_my_field, getpid)، صرف سربار (overhead) ورود و خروج از هسته می‌شود، نه اجرای خود تابع. زمان "Entry" (B-A)، که هزینه ورود به هسته است، به تنهایی (حدود ۱۵۰۰ تا ۱۷۰۰ نانوثانیه) چندین برابر زمان واقعی اجرای تابع در هسته (C-B) و همچنین بسیار بیشتر از زمان خروج (D-C) است.



نکته قابل توجه دیگر، عدم تقارن بین ورود و خروج است؛ هزینه ورود (B-A) به طور مداوم ۳ تا ۵ برابر بیشتر از هزینه خروج (D-C) است. در مقابل، برای عملیات‌های پیچیده I/O مانند read و write، زمان اجرای واقعی (C-B) آنقدر طولانی است (خارج از مقیاس نمودار) که سربار ورود و خروج در برابر آن ناچیز است. در مجموع، این نمودار نشان می‌دهد که برای syscall های سبک، هزینه اصلی همان تعویض وضعیت (Context Switch) است.

در دو عکس دیگر نیز نتایجی مشابه استدلال بالا بدست می‌آید که این موضوع نشان‌دهنده طولانی بودن زمان اجرای دستورات read, write نسبت به 3 دستور دیگر است.