Bahir Dar University

Faculty of Computing

Department of Software Engineering

OSSP Individual Assignment

System Call Implementation

Full Name: Amanuel Fikadie Debeb

Student ID: BDU1600914

Section: A

System Call: nice()

Submitted To: Lecturer Wondimu B.

Submission Date:

System Call Implementation Example

nice() in User Space (glibc)

When you call nice(int inc) in C, you're using the glibc wrapper for the system call. This wrapper simplifies the interaction with the kernel. Here’s how it works:

```
#include <unistd.h>
#include <sys/resource.h>
#include <errno.h>
int nice(int inc) {
   int ret;
   ret = getpriority(PRIO_PROCESS, 0) + inc;
   if (ret < -20) ret = -20;
   if (ret > 19) ret = 19;
   if (setpriority(PRIO_PROCESS, 0, ret) == -1) {
      return -1;
   }
   return ret;}
```

Explanation:

The nice() function adjusts the priority of the calling process.

It retrieves the current nice value using getpriority(), adjusts it by inc, and enforces bounds (-20 to 19).

The new nice value is set using setpriority().

Kernel Space: The Actual System Call

In the Linux kernel source code, the nice() system call is implemented in kernel/sched/core.c. Here’s a simplified breakdown:

```
SYSCALL_DEFINE1(nice, int, inc) {
   long ret;
   ret = task_nice(current, inc);
   return ret;}
```

Key Steps:

SYSCALL_DEFINE1: Macro to define a system call with one argument (inc).

task_nice(): Kernel function that adjusts the nice value of the current process (current).

Bounds Check: Ensures the nice value stays within the valid range (-20 to 19).

Under the Hood: task_nice()

The task_nice() function delegates the priority adjustment to the scheduler. Here's a simplified view:

```
long task_nice(struct task_struct *p, int inc) {
   int new_nice = PRIO_TO_NICE(p->static_prio) + inc;
   if (new_nice < -20) new_nice = -20;
   if (new_nice > 19) new_nice = 19;
   set_user_nice(p, new_nice);
   return new_nice;}
```

Explanation:

PRIO_TO_NICE: Converts the process's static priority to a nice value.

set_user_nice(): Updates the process's priority in the scheduler's data structures.

Summary of Layers

Layer	Function	Role
User Space	nice(int inc)	Glibc wrapper that adjusts process priority and handles bounds.
Syscall	SYSCALL_DEFINE1(nice)	Kernel entry point for the nice system call.
Scheduler	task_nice()	Adjusts the process's priority and interacts with the scheduler.

Practical Example

Writing a C Program to Use nice()

Here's a complete example demonstrating the nice() system call:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/resource.h>
#include <errno.h>
int main() {
   int current_nice = getpriority(PRIO_PROCESS, 0);
   printf("Current nice value: %d\n", current_nice);

   int result = nice(10); // Lower priority
   if (result == -1 && errno != 0) {
      perror("nice failed");
      return 1;
   }

   int new_nice = getpriority(PRIO_PROCESS, 0);
   printf("New nice value: %d\n", new_nice);
```

```
    // Simulate work
    for (int i = 0; i < 5; i++) {
        printf("Working... %d\n", i + 1);
        sleep(1);
    }

    return 0;}
```

Steps to Compile and Run:
CTRL + O then ENTER and CTRL + X
Save the code to nice_example.c.
Compile: gcc nice_example.c -o nice_example.
Run: ./nice_example.

Expected Output:

```
Current nice value: 0
New nice value: 10
Working... 1
Working... 2
...
```

Key Takeaways

The nice() system call modifies process priority, influencing CPU scheduling.
Lower nice values give higher priority (more CPU time), while higher values reduce priority.
The kernel enforces bounds (-20 to 19) to prevent misuse.
Understanding nice() is essential for system programming and resource management.