Assignment No: 9

Title: System Call

AIM:

Implement a new system call, `swipe()`, in the Linux kernel that transfers the remaining time slice of each process in a specified set to a target process. Demonstrate various uses of the system call.

OBJECTIVE:

To create and integrate the `swipe()` system call into the Linux kernel, and illustrate its effects both beneficially and detrimentally.

THEORY:

Steps to Add a New System Call:

1. **Download the Kernel Source:**

wget https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.17.4.tar.xz

Steps to Add a New System Call:
Download the Kernel Source:
bash
Copy code

```
wget
https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.17.4.tar.xz
```

Extract the Kernel Source Code:
bash
Copy code

```
sudo tar -xvf linux-4.17.4.tar.xz -C /usr/src/
cd /usr/src/linux-4.17.4/
```

Create the System Call Implementation:

Create a directory for the new system call:
bash

Copy code

```
mkdir swipe
cd swipe
```

-

Create the `swipe.c` file:
bash
Copy code

```
gedit swipe.c
```

-

Implement the `swipe()` function:
c
Copy code

```
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/syscalls.h>

asmlinkage long sys_swipe(pid_t target_pid, pid_t *source_pids, int
num_sources) {
    struct task_struct *target_task;
    struct task_struct *source_task;
    long time_slice;

    target_task = pid_task(find_vpid(target_pid), PIDTYPE_PID);
    if (!target_task) return -ESRCH;  // Target process not found

    for (int i = 0; i < num_sources; i++) {
        source_task = pid_task(find_vpid(source_pids[i]),
PIDTYPE_PID);
        if (!source_task) return -ESRCH;  // Source process not
found

        // Transfer time slice (hypothetical implementation)
        time_slice = source_task->sched_info.time_slice;
        source_task->sched_info.time_slice = 0;  // Reset source
time slice
        target_task->sched_info.time_slice += time_slice;  // Add to
target time slice
    }
```

```
    return 0;  // Success
}
```

-

Create a Makefile:
bash
Copy code
```
gedit Makefile
```
Add the line:
makefile
Copy code
```
obj-y := swipe.o
```

Modify the Kernel Makefile:

Go to the parent directory:
bash
Copy code
```
cd ..
gedit Makefile
```

-

Find the line with `core-y` and add `swipe/` at the end:
makefile
Copy code
```
core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ swipe/
```

-

Add to System Call Table:

Open the syscall table for 64-bit systems:
bash
Copy code
```
cd arch/x86/entry/syscalls/
gedit syscall_64.tbl
```

-

Add the new system call at the end:
plaintext
Copy code
```
549
64
```

```
swipe
sys_swipe
```

- **Add to System Call Header:**

Open the system call header:
bash
Copy code
```
cd ../../../../include/linux/
gedit syscalls.h
```

-

Add the prototype for `swipe()`:

```
asmlinkage long sys_swipe(pid_t target_pid, pid_t *source_pids, int
num_sources);
```

- **Compile the Kernel:**

Install necessary packages:

```
sudo apt-get install gcc libncurses5-dev bison flex libssl-dev
libelf-dev
```

Configure the kernel:

```
sudo make menuconfig
```

Compile the kernel:

```
sudo make -j$(nproc)
```

Install/Update Kernel:

```
sudo make modules_install install
```

```
sudo shutdown -r now
```

Test the System Call:

- Create a user space program:

```
cd ~
gedit userspace.c
```

```
#include <stdio.h>
#include <unistd.h>
#include <sys/syscall.h>

#define SYS_swipe 549  // Update this number accordingly

int main() {
    pid_t target_pid = 1234;  // Example target PID
    pid_t source_pids[] = {5678, 91011};  // Example source PIDs
    int num_sources = 2;

    long result = syscall(SYS_swipe, target_pid, source_pids,
num_sources);
    printf("System call sys_swipe returned %ld\n", result);

    return 0;
}
```

Compile and Run the User Space Program:

```
gcc userspace.c -o userspace
./userspace
Dmesg
```

```
#include <linux/kernel.h>
asmlinkage long sys_hello(void)
{
    printk(KERN_INFO "Hello world\n");
    return 0;
}
```

```
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>
int main()
{
    long int r = syscall(358);
    printf("System call sys_hello returned %ld\n", r);
    return 0;
}
```

```
308     common  setns                   __x64_sys_setns
309     common  getcpu                  __x64_sys_getcpu
310     64      process_vm_readv        __x64_sys_process_vm_readv
311     64      process_vm_writev       __x64_sys_process_vm_writev
312     common  kcmp                    __x64_sys_kcmp
313     common  finit_module            __x64_sys_finit_module
314     common  sched_setattr           __x64_sys_sched_setattr
315     common  sched_getattr           __x64_sys_sched_getattr
316     common  renameat2               __x64_sys_renameat2
317     common  seccomp                 __x64_sys_seccomp
318     common  getrandom               __x64_sys_getrandom
319     common  memfd_create            __x64_sys_memfd_create
320     common  kexec_file_load         __x64_sys_kexec_file_load
321     common  bpf                     __x64_sys_bpf
322     64      execveat                __x64_sys_execveat/ptregs
```