System call implementation of solus operating system

Since Solus is an operating system based on Linux, it implements system calls in a manner that is quite similar to that of other Linux distributions. Nevertheless, Solus is a stand-alone distribution that uses eopkg and its own kernel build procedure instead of common package managers like apt or yum.

Here's a step-by-step guide tailored for Solus, assuming you're working with its kernel source:

1. Get the Solus Kernel SourceYou'll need to build the Solus kernel with your custom syscall:

1. Install build dependencies:

```
sudo eopkg it -c system.devel

sudo eopkg it git bc flex bison openssl-devel elfutils-libelf-devel ncurses-devel
```

2. Clone the kernel repo: Solus uses its own packaging system, so get the kernel source via the ypkg or from their GitHub:

```
git clone https://github.com/getsolus/kernel.git

cd kernel
```

2. Add Your System Call

Let's say we're adding a syscall called say_hello.

1. Add implementation in a C file: In kernel/ or a new file, say sayhello.c:

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

#include <linux/syscalls.h>

SYSCALL_DEFINE0(say_hello)

{

   printk(KERN_INFO "Hello from Solus syscall!\n");

   return 0;
}
```

2. Modify kernel Makefile to compile your syscall (optional if new file).

Example in kernel/Makefile:

```
obj-y += sayhello.o
```

3. Register the syscall in:

arch/x86/entry/syscalls/syscall_64.tbl

Add this line (pick an unused syscall number):

```
548   common   say_hello   __x64_sys_say_hello
```

4. Declare it in:

include/linux/syscalls.h

```
asmlinkage long sys_say_hello(void);
```

3. Build the Kernel

1. Configure it:

```
make defconfig
```

2. Compile:

```
make -j$(nproc)

sudo make modules_install

sudo make install
```

3. Reboot into the new kernel.

4. Test from User Space

```
#include <stdio.h>

#include <sys/syscall.h>

#include <unistd.h>

#define __NR_say_hello 548
```

```
int main() {
    long ret = syscall(__NR_say_hello);
    printf("System call returned: %ld\n", ret);
    return 0;
}
```

Compile and run:

```
gcc test_hello.c -o test_hello
./test_hello
```

Here's a brief description of the system-related commands listed in your document for the Solus operating system:

1. sudo eopkg update-repo

Updates the local list of available software packages.

2. sudo eopkg upgrade -y

Upgrades all installed packages to their latest versions.

3. sudo eopkg install -c system.devel

Installs development tools like GCC, make, and other essential build utilities.

4. sudo eopkg install linux-current-headers

Installs the current Linux kernel headers, required for compiling kernel modules.

5. sudo eopkg install [tools...]

Installs various tools and text editors such as GParted, Neofetch, Vim, Geany, etc.

6. sudo eopkg install [media packages...]

Installs media codecs and plugins for better audio/video support.

7. sudo eopkg install firefox filezilla transmission

Installs commonly used internet applications.

8. sudo eopkg install flatpak

Installs Flatpak support for managing universal Linux apps.

9. flatpak remote-add ... flathub

Adds the Flathub repository to access Flatpak applications.

10. sudo eopkg check

Verifies package integrity and checks for missing files.

11. sudo eopkg rmo

Cleans up old and orphaned packages to save space.