



CSCI-3753: Operating Systems

Fall 2019

Abigail Fernandes

Department of Computer Science
University of Colorado Boulder

Logistics

- Office Hours on Monday, 10:15 – 12:15 (**ECAE 133**)
- Discussion forum - **Moodle**
- First Recitation Quiz!
- Special accommodations
- Problem Set 1 Questions



Week 2: PA1 & LKM



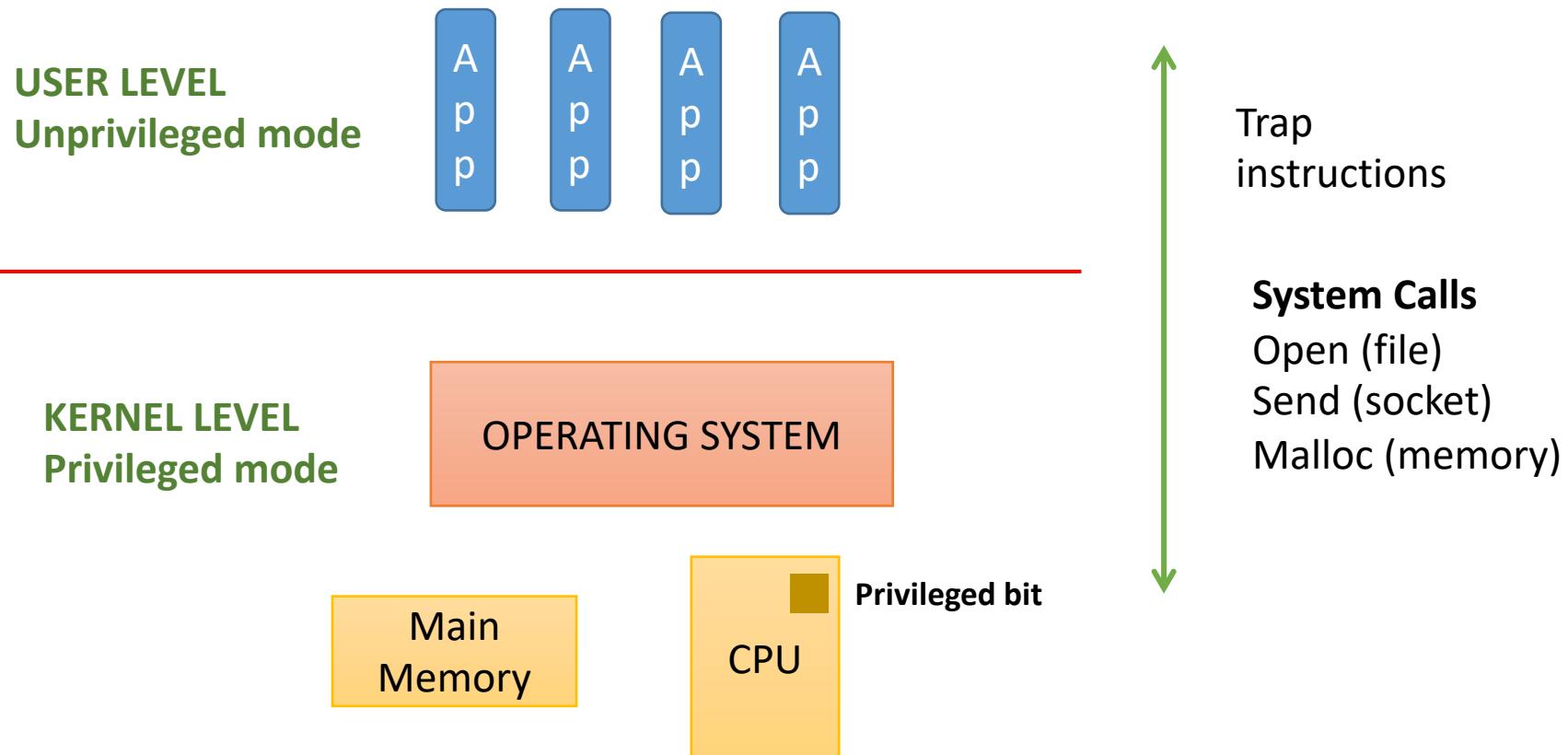
University of Colorado
Boulder

PA 1 – Questions & Answers

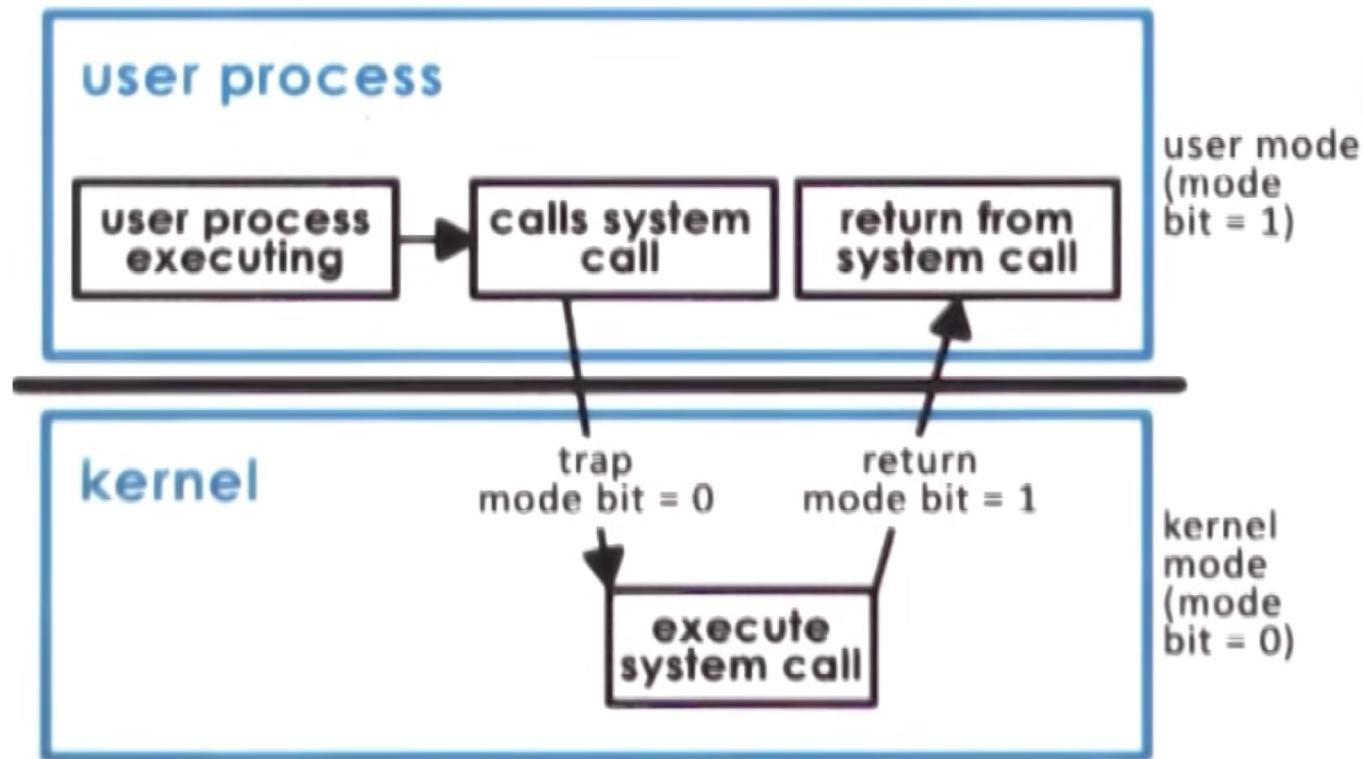


University of Colorado
Boulder

User/Kernel Protection Boundary



System call Flowchart



System call

To make a system call, an application must

- Write arguments
- Save relevant data to a well defined location
- Make system call (using the specific system call number)



Executing a System call (Flowchart explained)

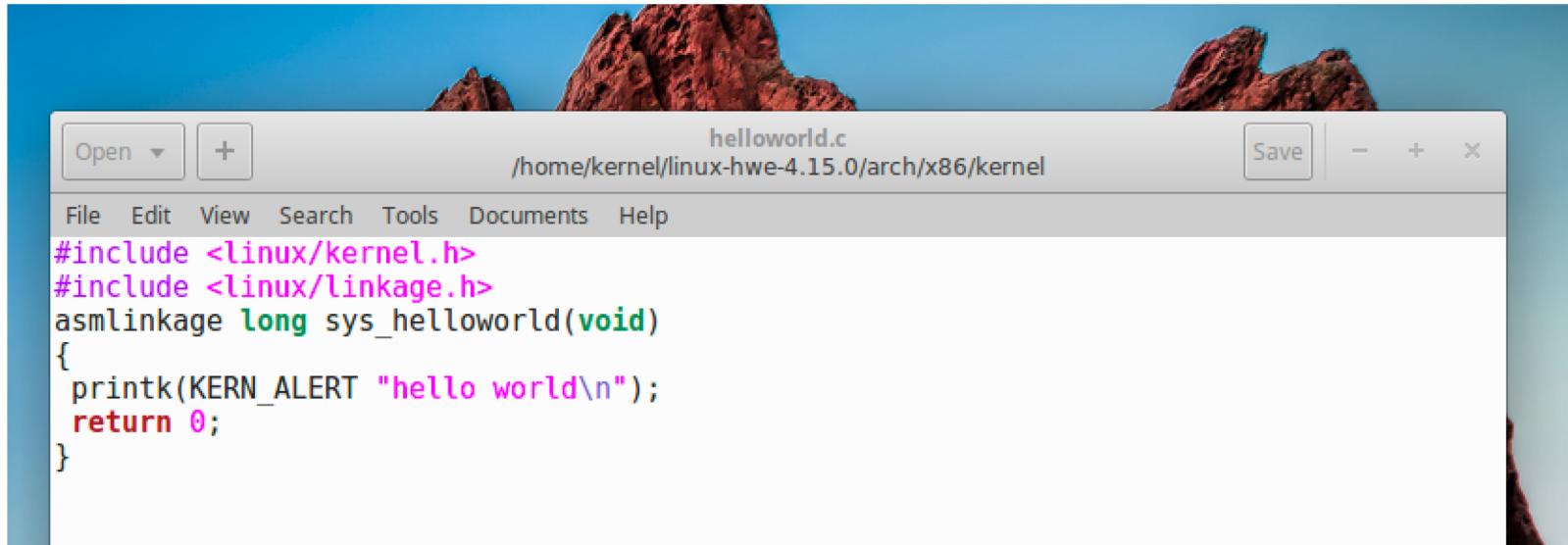
1. A user space program invokes the syscall
2. A (typically) software interrupt called a trap is triggered (INT)
3. Mode bit is flipped from user to kernel (1 to 0)
4. The interrupt tells the kernel which syscall was called
 1. Requisite data may be passed in
 2. The kernel verifies if all parameters are legal before executing the system call
5. After execution, mode bit flips and user program resumes



Adding a System Call

1. Write the system call source code

```
sudo gedit arch/x86/kernel/helloworld.c
```

A screenshot of a Linux desktop environment. In the foreground, a terminal window titled 'helloworld.c' is open, showing the path '/home/kernel/linux-hwe-4.15.0/arch/x86/kernel'. The window has standard Linux window controls (minimize, maximize, close) at the top right. The terminal itself has a dark background and light-colored text. The menu bar includes 'File', 'Edit', 'View', 'Search', 'Tools', 'Documents', and 'Help'. The code in the terminal is:

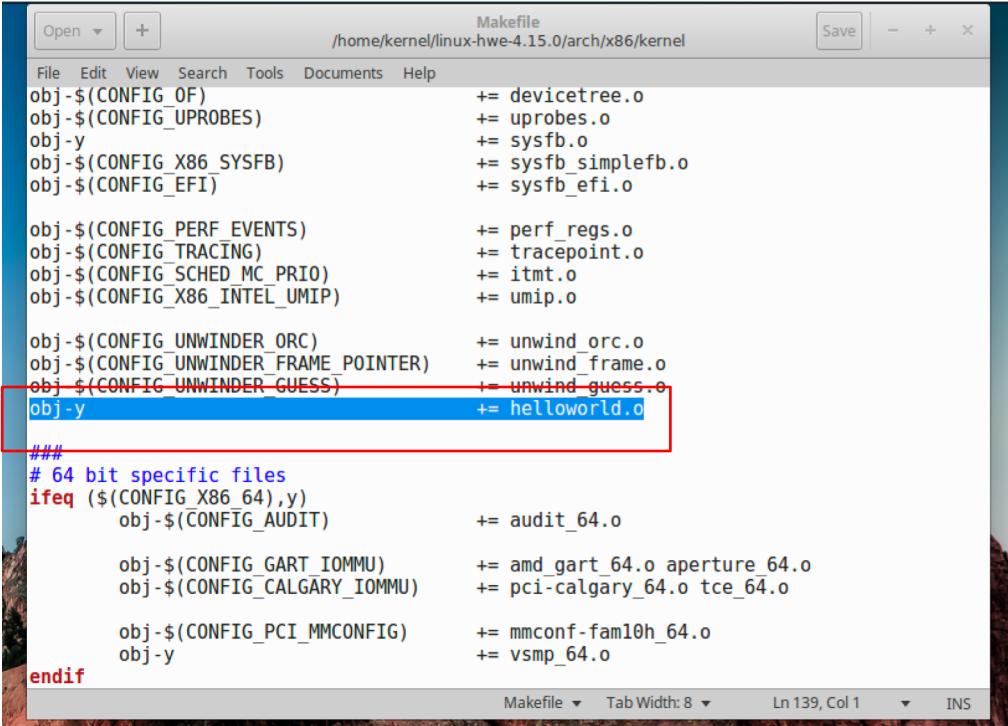
```
#include <linux/kernel.h>
#include <linux/linkage.h>
asmlinkage long sys_helloworld(void)
{
    printk(KERN_ALERT "hello world\n");
    return 0;
}
```



Adding a System Call

2. Add the new syscall to the Makefile

arch/x86/kernel/Makefile



```
Makefile
/home/kernel/linux-hwe-4.15.0/arch/x86/kernel
File Edit View Search Tools Documents Help
obj-$(CONFIG_OF)          += devicetree.o
obj-$(CONFIG_UPROBES)      += uprobes.o
obj-y                     += sysfb.o
obj-$(CONFIG_X86_SYSFB)    += sysfb_simplefb.o
obj-$(CONFIG_EFI)          += sysfb_efi.o

obj-$(CONFIG_PERF_EVENTS)  += perf_regs.o
obj-$(CONFIG_TRACING)      += tracepoint.o
obj-$(CONFIG_SCHED_MC_PRIO) += itmt.o
obj-$(CONFIG_X86_INTEL_UMIP) += umip.o

obj-$(CONFIG_UNWINDER_ORC)  += unwind_orc.o
obj-$(CONFIG_UNWINDER_FRAME_POINTER) += unwind_frame.o
obj-$(CONFIG_UNWINDER_GUESS)  += unwind_guess.o
obj-y                      += helloworld.o

###
# 64 bit specific files
ifeq ($(CONFIG_X86_64),y)
    obj-$(CONFIG_AUDIT)      += audit_64.o
    obj-$(CONFIG_GART_IOMMU)   += amd_gart_64.o aperture_64.o
    obj-$(CONFIG_CALGARY_IOMMU) += pci-calgary_64.o tce_64.o
    obj-$(CONFIG_PCI_MMCONFIG) += mmconf-fam10h_64.o
    obj-y                    += vsmp_64.o
endif

```

Adding a System Call

3. Add the syscall to the syscalls table

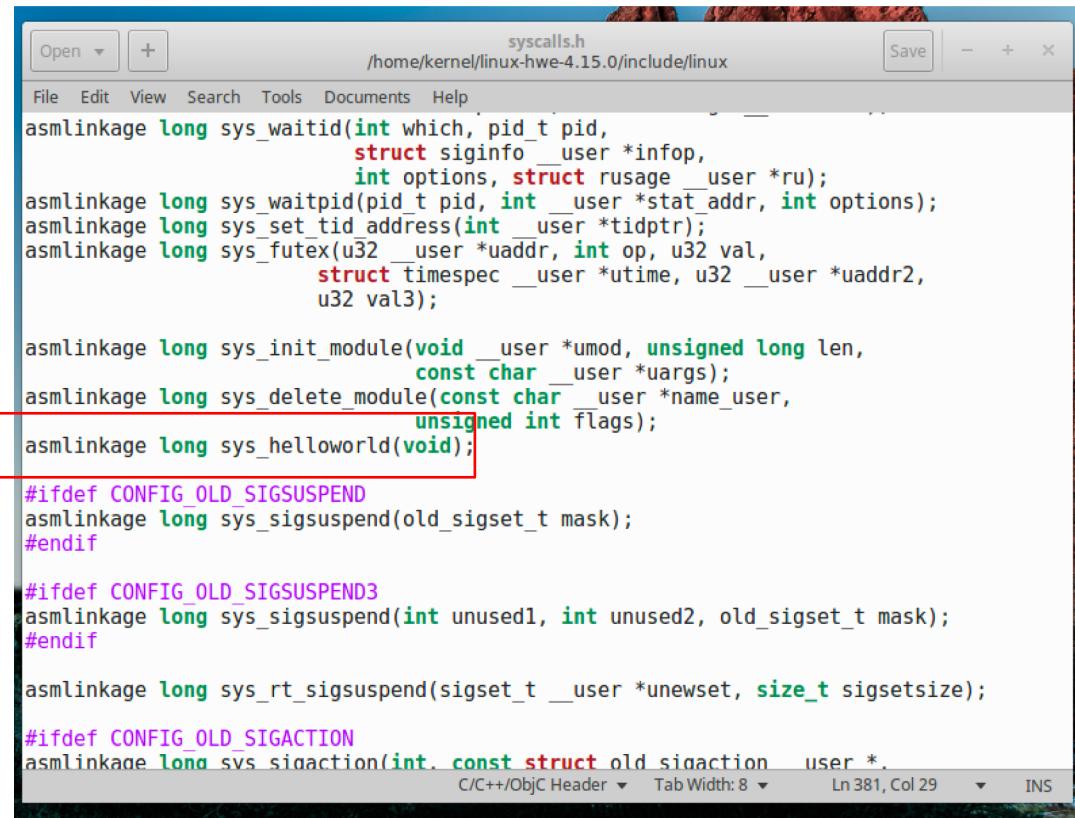
[arch/x86/entry/syscalls/syscall_64.tbl](#)

325	common	mlock2	sys_mlock2
326	common	copy_file_range	sys_copy_file_range
327	64	preadv2	sys_preadv2
328	64	pwritev2	sys_pwritev2
329	common	pkey_mprotect	sys_pkey_mprotect
330	common	pkey_alloc	sys_pkey_alloc
331	common	pkey_free	sys_pkey_free
332	common	statx	sys_statx
333	64	helloworld	sys_helloworld



Adding a System Call

4. Add the syscall prototype to the syscalls header file [include/linux/syscalls.h](#)



```
syscalls.h
/home/kernel/linux-hwe-4.15.0/include/linux

asmlinkage long sys_waitid(int which, pid_t pid,
                           struct siginfo __user *infop,
                           int options, struct rusage __user *ru);
asmlinkage long sys_waitpid(pid_t pid, int __user *stat_addr, int options);
asmlinkage long sys_set_tid_address(int __user *tidptr);
asmlinkage long sys_futex(u32 __user *uaddr, int op, u32 val,
                        struct timespec __user *utime, u32 __user *uaddr2,
                        u32 val3);

asmlinkage long sys_init_module(void __user *umod, unsigned long len,
                               const char __user *uargs);
asmlinkage long sys_delete_module(const char __user *name_user,
                                 unsigned int flags);
asmlinkage long sys_helloworld(void);

#ifndef CONFIG_OLD_SIGSUSPEND
asmlinkage long sys_sigsuspend(old_sigset_t mask);
#endif

#ifndef CONFIG_OLD_SIGSUSPEND3
asmlinkage long sys_sigsuspend(int unused1, int unused2, old_sigset_t mask);
#endif

asmlinkage long sys_rt_sigsuspend(sigset_t __user *unewset, size_t sigsetsize);

#ifndef CONFIG_OLD_SIGACTION
asmlinkage long sys_sigaction(int __user *, const struct old_sigaction __user *,
```

Compiling the Kernel

1. Create a config file and setup a name to append to the release

```
sudo cp /boot/config-$(uname -r) .config
```

2. Make

```
sudo make -j2 CC="ccache gcc"
```

```
sudo make -j2 modules_install
```

```
sudo make -j2 install
```

3. Reboot



Passing values safely between user and kernel spaces

USER SPACE MEMORY ACCESS

```
unsigned long copy_from_user(void *to,  
    const void __user *from,  
    unsigned long n);  
unsigned long copy_to_user(void __user *to,  
    const void *from,  
    unsigned long n);
```

get_user()

put_user()

printk

cat /var/log/syslog

dmesg



University of Colorado
Boulder

You have to write a new system call.
This system call will be given two
numbers and an address of where to
store the results



Loadable Kernel Module (LKM)

How to add a code to a Linux Kernel

- Method 1:
 - Step 1: Add source files to the Linux kernel source tree
 - Step 2: Recompile the kernel

→ PA1

→ It takes a long time to compile the kernel source code (2-3 hours) !!!

- Method 2: Add code to the Linux kernel while it is running

Loadable Kernel Module (LKM)

Loadable Kernel Module (LKM)

- LKM is a *chunk of code* that we add to the Linux kernel while it is running.
- Typically, one of the following modules:
 1. Device drivers
 2. Filesystem drivers
 3. System calls



Loadable Kernel Module (LKM)

- LKMs (when loaded) are very much part of the kernel.
→ Part of the kernel that is bound into the image that you boot is “*base kernel*”.
- LKMs communicate with the base kernel.



Why LKM?

- DON'T have to rebuild the kernel
- Help diagnose system problems
 - A bug in a device driver which is bound into the kernel can stop the system from booting at all.
- Save memory
 - Have them loaded only when we are actually using them
- Much faster to maintain and debug



LKM Utilities

- ***insmod***: Insert an LKM into the kernel.
- ***rmmod***: Remove an LKM from the kernel.
- ***lsmod***: List currently loaded LKMs.
- ***kerneld***: Kernel daemon program
 - allows kernel modules to be loaded automatically rather than manually with insmod/modprobe
- ***modprobe***: Insert/remove an LKM or set of LKMs intelligently.
 - e.g., if you must load A before loading B, modprobe will automatically load A when you tell it to load B.



LKM Utilities

- ***depmod***: Determine interdependencies between LKMs.
- ***ksyms***: Display symbols that are exported by the kernel for use by new LKMs.
- ***modinfo***: Display contents of .modinfo section in an LKM object file.



Week 2 – Checklist

- Submit PS1 by 6PM today !!!
- Start and complete PA1
- Read more about LKM

