LINUX PROGRAMMING

Linux /proc File System

www.cce.hcmut.edu.vn

08/11/2022

Nguyen Quang Hung

Outline – The /proc FS

- Introduction
- Linux file system
- The /proc FS
  - Process entries
  - Hardware Information
  - Kernel Information
  - File System
  - System statistic

The Linux Proc File System

- The **/proc file system** does not store data, rather, its contents are computed on demand according to user file I/O requests
- The **/proc** must implement a directory structure, and the file contents within; it must then define a unique and persistent inode number for each directory and files it contains
  - It uses this inode number to identify just what operation is required when a user tries to read from a particular file inode or perform a lookup in a particular directory inode
  - When data is read from one of these files, **proc** collects the appropriate information, formats it into text form and places it into the requesting process's read buffer

The Linux Proc File System (cont'd)

- **/proc** is a virtual file system that allow communication between kernel and use space
- It doesn't contain 'real' files but runtime system information system memory, devices mounted, hardware configuration
- Widely used for many reporting information
- E.g., `/proc/modules, /proc/meminfo, /proc/cpuinfo,...`

- **`http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html`**

Example: /proc, /proc/cpuinfo

```
hung@TonyStark2022:~$ ls /proc
1   16  bus      cmdline  filesystems  loadavg  mounts  self  sys  uptime   version_signature
15  83  cgroups  cpuinfo  interrupts   meminfo  net     stat  tty  version
hung@TonyStark2022:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 158
model name      : Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
stepping        : 10
microcode       : 0xffffffff
cpu MHz         : 2208.000
cache size      : 256 KB
physical id     : 0
siblings        : 12
core id         : 0
cpu cores       : 6
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse
sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2
x2apic movbe popcnt aes xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase tsc_adjust bmi1 avx2
 smep bmi2 erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 4416.00
clflush size    : 64
cache_alignment : 64
```

Example: /proc/meminfo

$ cat /proc/meminfo

```
hung@TonyStark2022:~$ cat /proc/meminfo
MemTotal:        33343772 kB
MemFree:         16995068 kB
Buffers:            34032 kB
Cached:            188576 kB
SwapCached:             0 kB
Active:            167556 kB
Inactive:          157876 kB
Active(anon):      103104 kB
Inactive(anon):     17440 kB
Active(file):       64452 kB
Inactive(file):    140436 kB
Unevictable:            0 kB
Mlocked:                0 kB
SwapTotal:       29257504 kB
SwapFree:        28632988 kB
Dirty:                  0 kB
Writeback:              0 kB
AnonPages:         102824 kB
Mapped:             71404 kB
Shmem:              17720 kB
Slab:               13868 kB
SReclaimable:        6744 kB
SUnreclaim:          7124 kB
KernelStack:         2848 kB
PageTables:          2524 kB
NFS_Unstable:           0 kB
Bounce:                 0 kB
WritebackTmp:           0 kB
CommitLimit:       515524 kB
```

proc manual

$ man 5 proc

```
PROC(5)                         Linux Programmer's Manual                        PROC(5)

NAME
       proc - process information pseudo-filesystem

DESCRIPTION
       The  proc filesystem is a pseudo-filesystem which provides an interface to kernel data structures.  It is com-
       monly mounted at /proc.  Typically, it is mounted automatically by the system, but it can also be mounted man-
       ually using a command such as:

           mount -t proc proc /proc

       Most of the files in the proc filesystem are read-only, but some files are writable, allowing kernel variables
       to be changed.

   Mount options
       The proc filesystem supports the following mount options:

       hidepid=n (since Linux 3.3)
              This option controls who can access the information in /proc/[pid] directories.  The  argument,  n,  is
              one of the following values:

              0   Everybody  may  access  all /proc/[pid] directories.  This is the traditional behavior, and the de-
                  fault if this mount option is not specified.

              1   Users may not access files and subdirectories inside any /proc/[pid] directories but their own (the
                  /proc/[pid]  directories  themselves  remain visible).  Sensitive files such as /proc/[pid]/cmdline
                  and /proc/[pid]/status are now protected against other users.  This makes it  impossible  to  learn
                  whether any user is running a specific program (so long as the program doesn't otherwise reveal it-
                  self by its behavior).

              2   As for mode 1, but in addition the /proc/[pid] directories belonging to other users become  invisi-
                  ble.  This means that /proc/[pid] entries can no longer be used to discover the PIDs on the system.
                  This doesn't hide the fact that a process with a specific PID value exists (it can  be  learned  by
                  other  means,  for  example,  by "kill -0 $PID"), but it hides a process's UID and GID, which could
                  otherwise be learned by employing stat(2) on a /proc/[pid] directory.  This greatly complicates  an
                  attacker's  task  of  gathering information about running processes (e.g., discovering whether some
                  daemon is running with elevated privileges, whether another user is running some sensitive program,
                  whether other users are running any program at all, and so on).

       gid=gid (since Linux 3.3)
              Specifies the ID of a group whose members are authorized to learn process information otherwise prohib-
              ited by hidepid (i.e., users in this group behave as though /proc was mounted  with  hidepid=0).   This
              group should be used instead of approaches such as putting nonroot users into the sudoers(5) file.

   Overview
       Underneath /proc, there are the following general groups of files and subdirectories:

       /proc/[pid] subdirectories
```

Create a directory under /proc

```
extern struct proc_dir_entry *proc_mkdir(const char *, struct proc_dir_entry *);
```

Create a directory under /proc

proc_mkdir()

```
static inline struct proc_dir_entry *proc_create(
        const char *name, umode_t mode, struct proc_dir_entry *parent,
        const struct file_operations *proc_fops)
{
        return proc_create_data(name, mode, parent, proc_fops, NULL);
}

```

Create a file under /proc

proc_create()

```c
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
        ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
        ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
        ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
        int (*iterate) (struct file *, struct dir_context *);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
        long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *, fl_owner_t id);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, loff_t, loff_t, int datasync);
        int (*aio_fsync) (struct kiocb *, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
        unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned long, unsig
        int (*check_flags)(int);
        int (*flock) (struct file *, int, struct file_lock *);
        ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *, size_t, unsigned int);
        ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_t, unsigned int);
        int (*setlease)(struct file *, long, struct file_lock **, void **);
        long (*fallocate)(struct file *file, int mode, loff_t offset,
                          loff_t len);
        int (*show_fdinfo)(struct seq_file *m, struct file *f);
};
```

Sample code: Using /proc

Sample code:

```
#define FILENAME "status"
#define DIRECTORY "mp1"
static struct proc_dir_entry *proc_dir;
static struct proc_dir_entry *proc_entry;
static ssize_t mp1_read (struct file *file, char __user *buffer, size_t count, loff_t *data){
    // implementation goes here...
}
static ssize_t mp1_write (struct file *file, const char __user *buffer, size_t count, loff_t
*data){
    // implementation goes here...
}
static const struct file_operations mp1_file = {
    .owner = THIS_MODULE,
    .read  = mp1_read,
    .write = mp1_write,
};
int __init mp1_init(void){
    proc_dir = proc_mkdir(DIRECTORY, NULL);
    proc_entry = proc_create(FILENAME, 0666, proc_dir, & mp1_file);
}
```

- Within MP1_read/mp1_write, you may need to move data between kernel/user space
  - copy_from_user()
  - copy_to_user()

Sample code (There are other ways of implementing it):

```
static ssize_t mp1_read (struct file *file, char __user *buffer, size_t count, loff_t *data){
    // implementation goes here...
    int copied;
    char * buf;
    buf = (char *) kmalloc(count,GFP_KERNEL);
    copied = 0;
    //… put something into the buf, updated copied
    copy_to_user(buffer, buf, copied);
    kfree(buf);
    return copied ;
}
```

/proc/cpuinfo

$ cat /proc/cpuinfo

```
hung@TonyStark2022:~$ cat /proc/cpuinfo | less
hung@TonyStark2022:~$ cat /proc/cpuinfo | more
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 158
model name      : Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
stepping        : 10
microcode       : 0xffffffff
cpu MHz         : 2208.000
cache size      : 256 KB
physical id     : 0
siblings        : 12
core id         : 0
cpu cores       : 6
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse
sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2
x2apic movbe popcnt aes xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase tsc_adjust bmi1 avx2
 smep bmi2 erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 4416.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

Extract CPU Clock Speed from /proc/cpuinfo

- A simple way to extract a value from this output is to read the file into a buffer and parse it in memory using `sscanf`. Listing 7.1 shows an example of this. The program includes the function `get_cpu_clock_speed` that reads from `/proc/cpuinfo` into memory and extracts the first CPU's clock speed.
- Code: Listing 7.1 (clock-speed.c): Extract CPU Clock Speed from /proc/cpuinfo (page 132-133, ALP).
  - http://www.advancedlinuxprogramming.com/listings/chapter-7/clock-speed.c

Extract CPU Clock Speed from /proc/cpuinfo

- $ cat /proc/cpuinfo | grep "cpu MHz"
- Code: Listing 7.1 (clock-speed.c): Extract CPU Clock Speed from /proc/cpuinfo (page 132-133, ALP).
  - http://www.advancedlinuxprogramming.com/listings/chapter-7/clock-speed.c

Process Entries under /proc

- The **/proc** file system contains a directory entry for each process running on the GNU/Linux system. The name of each directory is the process ID of the corresponding process.
  - On some UNIX systems, the process IDs are padded with zeros. On GNU/Linux, they are not.
- These directories appear and disappear dynamically as processes start and terminate on the system. Each directory contains several entries providing access to information about the running process. From these process directories the **/proc** file system gets its name.

Process Entries under /proc (cont'd)

- Each process directory contains these entries:
  - **cmdline** contains the argument list for the process. The cmdline entry is described in Section 7.2.2, "Process Argument List."
  - **cwd** is a symbolic link that points to the current working directory of the process (as set, for instance, with the chdir call).
  - **environ** contains the process's environment. The environ entry is described in Section 7.2.3, "Process Environment."
  - **exe** is a symbolic link that points to the executable image running in the process. The exe entry is described in Section 7.2.4, "Process Executable."
  - **fd** is a subdirectory that contains entries for the file descriptors opened by the process. These are described in Section 7.2.5, "Process File Descriptors."

Process Entries under /proc (cont'd)

- Each process directory contains these entries (cont'd)
  - maps
  - root
  - stat
  - statm
  - status
  - cpu

/proc/self

- The entry **/proc/self** is a symbolic link to the **/proc** directory corresponding to the current process. The destination of the **/proc/self** link depends on which process looks at it: Each process sees its own process directory as the target of the link.
- For example, the program in Listing 7.2 reads the target of the /proc/self link to determine its process ID.

Listing 7.2 (get-pid.c) Obtain the Process ID from /proc/self

Process File Descriptors

- The **fd** entry is a subdirectory that contains entries for the file descriptors opened by a process. Each entry is a symbolic link to the file or device opened on that file descriptor.
- You can write to or read from these symbolic links; this writes to or reads from the corresponding file or device opened in the target process

```
$ ls -l /proc/16/fd
total 0
lrwx------ 1 hung hung 0 Nov  7 22:53 0 -> /dev/tty1
lrwx------ 1 hung hung 0 Nov  7 22:53 1 -> /dev/tty1
lrwx------ 1 hung hung 0 Nov  7 22:53 2 -> /dev/tty1
lrwx------ 1 hung hung 0 Nov  7 22:53 255 -> /dev/tty1
```

Listing 7.6 (open-and-spin.c) Open a File for Reading

Listing 7.6 (*open-and-spin.c*) Open a File for Reading

```
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main (int argc, char* argv[])
{
  const char* const filename = argv[1];
  int fd = open (filename, O_RDONLY);
  printf ("in process %d, file descriptor %d is open to %s\n",
          (int) getpid (), (int) fd, filename);
  while (1);
  return 0;
}
```

Try running it in one window:

```
% ./open-and-spin /etc/fstab
```

Process Memory Statistics

- The **statm** entry contains a list of seven numbers, separated by spaces. Each number is a count of the number of pages of memory used by the process in a particular category.
- The categories, in the order the numbers appear, are listed here:
  - The total process size
  - The size of the process resident in physical memory
  - The memory shared with other processes—that is, memory mapped both by this process and at least one other (such as shared libraries or untouched copy-on-write pages)
  - The text size of the process—that is, the size of loaded executable code
  - The size of shared libraries mapped into this process
  - The memory used by this process for its stack
  - The number of dirty pages—that is, pages of memory that have been modified by the program

Process Statistics

- The **status** entry contains a variety of information about the process, formatted for comprehension by humans. Among this information is the process ID and parent process ID, the real and effective user and group IDs, memory usage, and bit masks specifying which signals are caught, ignored, and blocked.

Hardware Information

- CPU Information
- Device Information
- PCI Bus Information
- Serial Port Information

Kernel Information

- Version Information
  - cat /proc/version
  - cat /proc/sys/kernel/ostype
  - cat /proc/sys/kernel/osrelease
  - cat /proc/sys/kernel/version
- Hostname and Domain Name
  - The /proc/sys/kernel/hostname and /proc/sys/kernel/domainname entries contain the computer's hostname and domain name, respectively

Memory Usage

- The /proc/meminfo entry contains information about the system's memory usage. Information is presented both for physical memory and for swap space. The first three lines present memory totals, in bytes; subsequent lines summarize this information in kilobytes – e.g:
  - cat /proc/meminfo

```
hung@TonyStark2022:~$ cat /proc/meminfo
MemTotal:        33343772 kB
MemFree:         20350092 kB
Buffers:            34032 kB
Cached:            188576 kB
SwapCached:             0 kB
Active:            167556 kB
Inactive:          157876 kB
Active(anon):      103104 kB
Inactive(anon):     17440 kB
Active(file):       64452 kB
Inactive(file):    140436 kB
Unevictable:            0 kB
Mlocked:                0 kB
SwapTotal:       29257504 kB
SwapFree:        29209052 kB
Dirty:                  0 kB
Writeback:              0 kB
AnonPages:         102824 kB
Mapped:             71404 kB
Shmem:              17720 kB
Slab:               13868 kB
SReclaimable:        6744 kB
SUnreclaim:          7124 kB
KernelStack:         2848 kB
PageTables:          2524 kB
NFS_Unstable:           0 kB
Bounce:                 0 kB
WritebackTmp:           0 kB
```

Drives, Mounts, and File Systems

- **/proc/filesystems**
  - The contents of **/proc/filesystems** list only file system types that either are statically linked into the kernel or are currently loaded.
  - Other file system types may be available on the system as modules but might not be loaded yet.

Drives and Partitions

- The /proc file system includes information about devices connected to both IDE controllers and SCSI controllers (if the system includes them).
  - **/proc/ide** subdirectory may contain either or both of two subdirectories, ide0 and ide1, corresponding to the primary and secondary IDE controllers on the system

Table 7.1 **Full Paths Corresponding to the Four Possible IDE Devices**

Controller	Device	Subdirectory
Primary	Master	`/proc/ide/ide0/hda/`
Primary	Slave	`/proc/ide/ide0/hdb/`
Secondary	Master	`/proc/ide/ide1/hdc/`
Secondary	Slave	`/proc/ide/ide1/hdd/`

/proc/scsi/scsi

If SCSI devices are present in the system, `/proc/scsi/scsi` contains a summary of their identification values. For example, the contents might look like this:

```
% cat /proc/scsi/scsi
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00
  Vendor: QUANTUM  Model: ATLAS_V__9_WLS   Rev: 0230
  Type:   Direct-Access                    ANSI SCSI revision: 03
Host: scsi0 Channel: 00 Id: 04 Lun: 00
  Vendor: QUANTUM  Model: QM39100TD-SW     Rev: N491
  Type:   Direct-Access                    ANSI SCSI revision: 02
```

/proc/partitions, /proc/sys/dev/cdrom/

- The **/proc/partitions** entry displays the partitions of recognized disk devices. For each partition, the output includes the major and minor device number, the number of 1024-byte blocks, and the device name corresponding to that partition.
- The **/proc/sys/dev/cdrom/** info entry displays miscellaneous information about the capabilities of CD-ROM drives

Mounts

- The **/proc/mounts** file provides a summary of mounted file systems. Each line corresponds to a single mount descriptor and lists the mounted device, the mount point, and other information. Note that **/proc/mounts** contains the same information as the ordinary file **/etc/mtab**, which is automatically updated by the mount command.

Locks

- The **/proc/locks** entry describes all the file locks currently outstanding in the system. Each row in the output corresponds to one lock.

% **touch /tmp/test-file**

% **./lock-file /tmp/test-file**

file /tmp/test-file

opening /tmp/test-file

locking

locked; hit enter to unlock...

In another window, look at the contents of /proc/locks.

% cat /proc/locks

1: POSIX ADVISORY WRITE 5467 08:05:181288 0 2147483647

d1b5f740 00000000

dfea7d40 00000000 00000000

% ps 5467

PID TTY STAT TIME COMMAND

5467 pts/28 S 0:00 ./lock-file /tmp/test-file

System statistic

- **/proc/stat**
  - Various statistics about the system, such as the number of page faults since the system was booted.
- **/proc/uptime**
  - The time the system has been up.
- **/proc/version**
  - The kernel version.
- **/proc/loadavg**
  - The `load average' of the system; three meaningless indicators of how much work the system has to do at the moment.

Summary the /proc

- **/proc/PID**
  - A directory with information about process number PID. Each process has a directory below /proc with the name being its process identification number.
- **/proc/cpuinfo**
  - Information about the processor, such as its type, make, model, and performance.
- **/proc/devices**
  - List of device drivers configured into the currently running kernel.
- **/proc/dma**
  - Shows which DMA channels are being used at the moment.

Summary the /proc (cont'd)

- **/proc/filesystems**
  - Filesystems configured into the kernel.
- **/proc/interrupts**
  - Shows which interrupts are in use, and how many of each there have been.
- **/proc/ioports**
  - Which I/O ports are in use at the moment.
- **/proc/kcore**
  - An image of the physical memory of the system. This is exactly the same size as your physical memory, but does not really take up that much memory; it is generated on the fly as programs access it. (Remember: unless you copy it elsewhere, nothing under /proc takes up any disk space at all.)
- **/proc/kmsg**
  - Messages output by the kernel. These are also routed to syslog.

Summary the /proc (cont'd)

- **/proc/ksyms**
  - Symbol table for the kernel.
- **/proc/meminfo**
  - Information about memory usage, both physical and swap.
- **/proc/modules**
  - Which kernel modules are loaded at the moment.
- **/proc/net**
  - Status information about network protocols.
- **/proc/self**
  - A symbolic link to the process directory of the program that is looking at /proc. When two processes look at /proc, they get different links. This is mainly a convenience to make it easier for programs to get at their process directory.

Linux Virtual File Systems

- To the user, Linux's file system appears as a hierarchical directory tree obeying UNIX semantics
- Internally, the kernel hides implementation details and manages the multiple different file systems via an abstraction layer, that is, the **Virtual File System (VFS)**
- The **Linux VFS** is designed around object-oriented principles and is composed of four components:
  - A set of definitions that define what a file object is allowed to look like
    - The **inode object** structure represent an individual file
    - The **file object** represents an open file
    - The **superblock object** represents an entire file system
    - A **dentry object** represents an individual directory entry

Linux Virtual File Systems (Cont.)

- To the user, Linux's file system appears as a hierarchical directory tree obeying UNIX semantics
- Internally, the kernel hides implementation details and manages the multiple different file systems via an abstraction layer, that is, the virtual file system (VFS)
- The Linux VFS is designed around object-oriented principles and layer of software to manipulate those objects with a set of operations on the objects
  - E.g. for the file object operations include (from struct file_operations in /usr/include/linux/fs.h

    int **open**(. . .) — Open a file

    ssize t **read**(. . .) — Read from a file

    ssize t **write**(. . .) — Write to a file

    int **mmap**(. . .) — Memory-map a file

Linux kernel sources:

- If you're interested in exactly how `/proc` works, take a look at the source code in the Linux kernel sources, under:

  `/usr/src/linux/fs/proc/`

Linux Kernel Lists

- You will use Linux list to store all registered user processes
- Linux kernel list is a widely used data structure in Linux kernel
  - Defined in <linux/linux.h>
  - You MUST get familiar of how to use it
  - Can be used as follows

```
struct list_head{
    struct list_head *next;
    struct list_head *prev;
};
```

```
struct my_cool_list{
    struct list_head list; /* kernel's list structure */
    int my_cool_data;
    void* my_cool_void;
};
```

Linux Kernel Module

- LKM are pieces of code that can be loaded and unloaded into the kernel upon demand
  - No need to modify the kernel source code
- Separate compilation
- Runtime linkage
- Entry and Exit functions

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

static int __init myinit(void){
    printk(KERN_ALERT "Hello, world\n");
    return 0;
}

static void __exit myexit(void){
    printk(KERN_ALERT "Goodbye, World\n");
}

module_init(myinit);
module_exit(myexit);
MODULE_LICENSE("GPL");
```

Ref:

1. Mitchell, M., Oldham, J., & Samuel, A. (2001). *Advanced linux programming* (pp. 95-129). Berkeley, CA: New riders.
2. **Linux System Administrators Guide:** https://tldp.org/LDP/sag/html/proc-fs.html

Discussions

LKM Example: Hello world

1. Edit
2. Makefile

```c
#include <linux/module.h>
#include <linux/kernel.h>
static int __init myinit(void)
{
        printk(KERN_ALERT "Hello, world\n");
        return 0;
}
static void __exit myexit(void)
{
        printk(KERN_ALERT "Goodbye, World\n");
}
module_init(myinit);
module_exit(myexit);
MODULE_LICENSE("GPL");
```

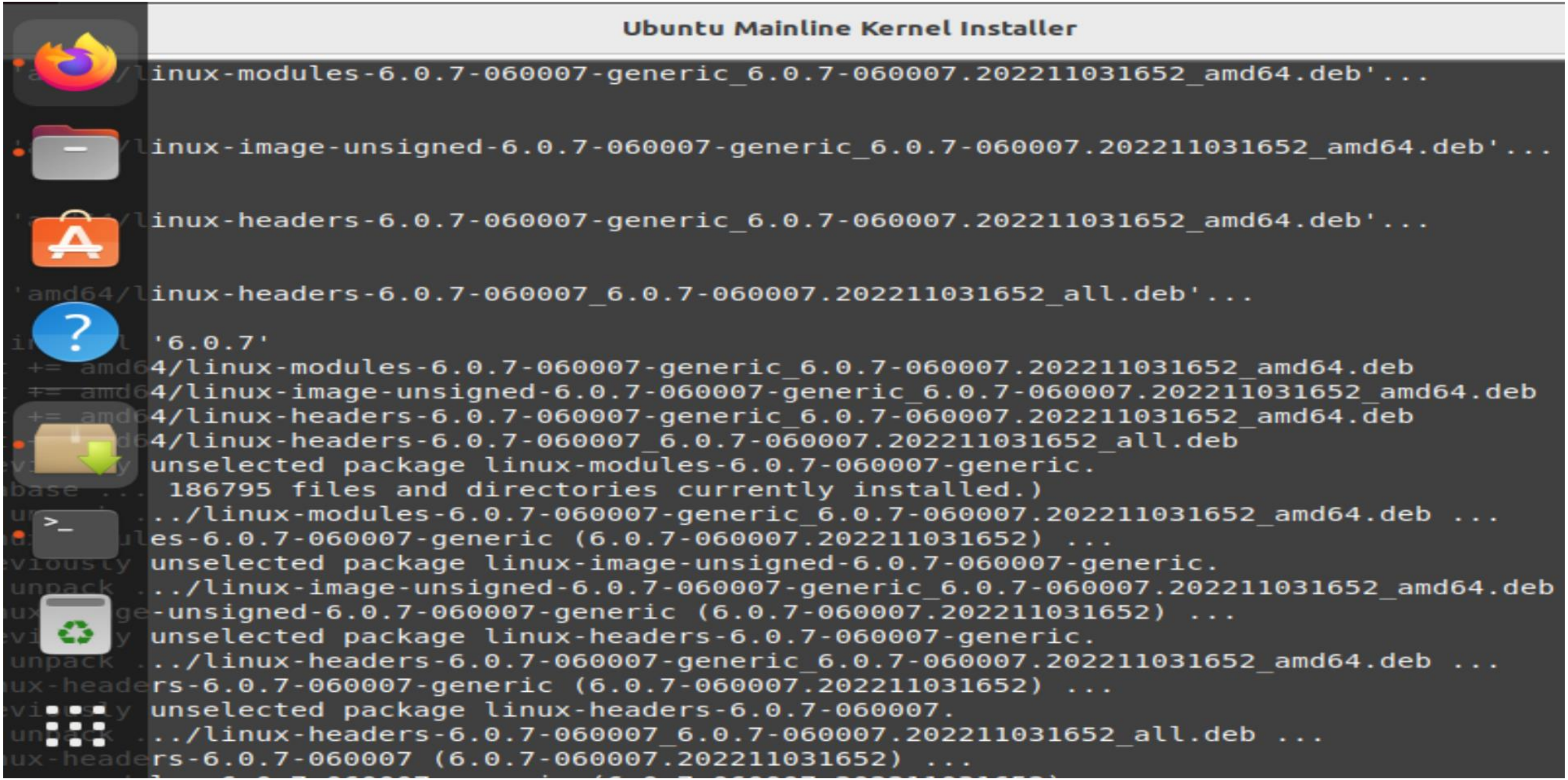
```makefile
obj-m += hello.o
all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

LKM Example: Hello world (2)

- Make
  - (Compiles the module)
- ls
  - Show module has been compiled to hello.ko

```
File  Edit  View  Search  Terminal  Help
cs423@cs423-vm:~/cs423/demo/mp1$ vim Makefile
cs423@cs423-vm:~/cs423/demo/mp1$ make
make -C /lib/modules/3.13.0-44-generic/build M=/home/cs423/cs423/demo/mp1 modules
make[1]: Entering directory `/usr/src/linux-headers-3.13.0-44-generic'
  CC [M]  /home/cs423/cs423/demo/mp1/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/cs423/cs423/demo/mp1/hello.mod.o
  LD [M]  /home/cs423/cs423/demo/mp1/hello.ko
make[1]: Leaving directory `/usr/src/linux-headers-3.13.0-44-generic'
cs423@cs423-vm:~/cs423/demo/mp1$ ls
hello.c  hello.ko  hello.mod.c  hello.mod.o  hello.o  Makefile  modules.order  Module.symvers
cs423@cs423-vm:~/cs423/demo/mp1$
```

```
File  Edit  View  Search  Terminal  Help
cs423@cs423-vm:~/cs423/demo/mp1$ vim Makefile
cs423@cs423-vm:~/cs423/demo/mp1$ make
make -C /lib/modules/3.13.0-44-generic/build M=/home/cs423/cs423/demo/mp1 modules
make[1]: Entering directory `/usr/src/linux-headers-3.13.0-44-generic'
  CC [M]  /home/cs423/cs423/demo/mp1/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/cs423/cs423/demo/mp1/hello.mod.o
  LD [M]  /home/cs423/cs423/demo/mp1/hello.ko
make[1]: Leaving directory `/usr/src/linux-headers-3.13.0-44-generic'
cs423@cs423-vm:~/cs423/demo/mp1$ ls
hello.c  hello.ko  hello.mod.c  hello.mod.o  hello.o  Makefile  modules.order  Module.symvers
cs423@cs423-vm:~/cs423/demo/mp1$
```


Ubuntu Mainline Kernel Installer
inux-modules-6.0.7-060007-generic_6.0.7-060007.202211031652_amd64.deb'...
inux-image-unsigned-6.0.7-060007-generic_6.0.7-060007.202211031652_amd64.deb'...
inux-headers-6.0.7-060007-generic_6.0.7-060007.202211031652_amd64.deb'...
'amd64/linux-headers-6.0.7-060007_6.0.7-060007.202211031652_all.deb'...
'6.0.7'
4/linux-modules-6.0.7-060007-generic_6.0.7-060007.202211031652_amd64.deb
4/linux-image-unsigned-6.0.7-060007-generic_6.0.7-060007.202211031652_amd64.deb
4/linux-headers-6.0.7-060007-generic_6.0.7-060007.202211031652_amd64.deb
4/linux-headers-6.0.7-060007_6.0.7-060007.202211031652_all.deb
unselected package linux-modules-6.0.7-060007-generic.
186795 files and directories currently installed.)
../linux-modules-6.0.7-060007-generic_6.0.7-060007.202211031652_amd64.deb ...
es-6.0.7-060007-generic (6.0.7-060007.202211031652) ...
unselected package linux-image-unsigned-6.0.7-060007-generic.
../linux-image-unsigned-6.0.7-060007-generic_6.0.7-060007.202211031652_amd64.deb
-unsigned-6.0.7-060007-generic (6.0.7-060007.202211031652) ...
unselected package linux-headers-6.0.7-060007-generic.
../linux-headers-6.0.7-060007-generic_6.0.7-060007.202211031652_amd64.deb ...
rs-6.0.7-060007-generic (6.0.7-060007.202211031652) ...
unselected package linux-headers-6.0.7-060007.
../linux-headers-6.0.7-060007_6.0.7-060007.202211031652_all.deb ...
rs-6.0.7-060007 (6.0.7-060007.202211031652) ...

Review

- Solving CPU Scheduling exercises?
- Answer questions

Thank you!

Q/A?

Other File Systems

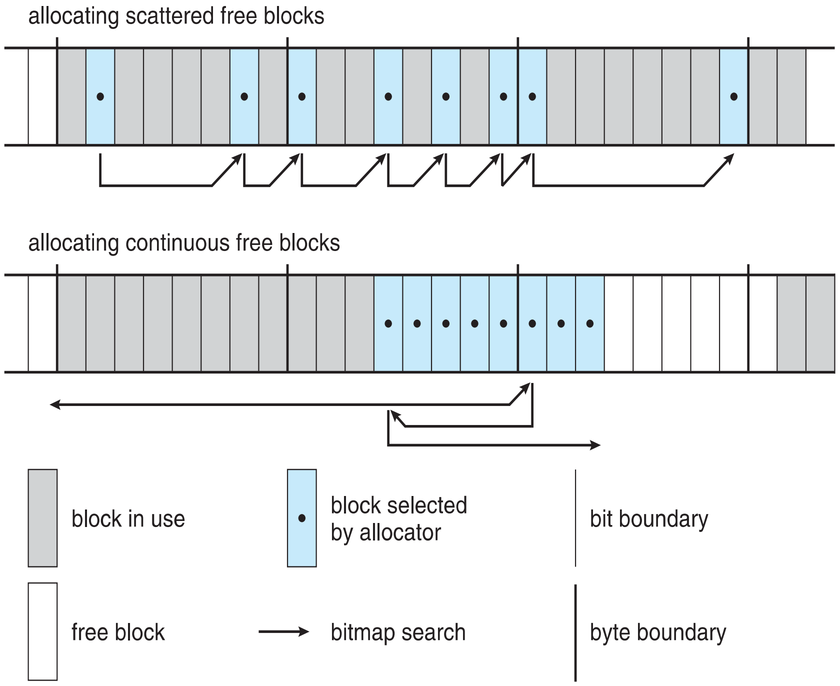
The Linux ext3 File System

- **ext3** is standard on disk file system for Linux
  - Uses a mechanism similar to that of BSD Fast File System (FFS) for locating data blocks belonging to a specific file
  - Supersedes older **extfs**, **ext2** file systems
  - Work underway on ext4 adding features like extents
  - Of course, many other file system choices with Linux distros

The Linux ext3 File System (Cont.)

- The main differences between ext2fs and FFS concern their disk allocation policies
  - In ffs, the disk is allocated to files in blocks of 8Kb, with blocks being subdivided into fragments of 1Kb to store small files or partially filled blocks at the end of a file
  - ext3 does not use fragments; it performs its allocations in smaller units
    - The default block size on ext3 varies as a function of total size of file system with support for 1, 2, 4 and 8 KB blocks
  - ext3 uses cluster allocation policies designed to place logically adjacent blocks of a file into physically adjacent blocks on disk, so that it can submit an I/O request for several disk blocks as a single operation on a **block group**
  - Maintains bit map of free blocks in a block group, searches for free byte to allocate at least 8 blocks at a time

Ext2fs Block-Allocation Policies



Journaling

- ext3 implements **journaling**, with file system updates first written to a log file in the form of **transactions**
  - Once in log file, considered committed
  - Over time, log file transactions replayed over file system to put changes in place
- On system crash, some transactions might be in journal but not yet placed into file system
  - Must be completed once system recovers
  - No other consistency checking is needed after a crash (much faster than older methods)
- Improves write performance on hard disks by turning random I/O into sequential I/O