Day 10
Advanced Linux Programming
Part III

Joel Isaacson
Ascender Technologies Ltd.
http://ascender.com

Today's Lectures

- Today we will continue to examine the Linux API
  - Semaphores (Futexes)
  - Fuser FS
  - Ssh
  - Signals
  - UIO

Synchronization
Posix Semaphores - posix.c

```
#include <semaphore.h>
int main(int argc, char *argv[])
{
  sem_t sem;
  int i;
  sem_init(&sem, 1, 1);
  for(i=0;i<10000000;i++) {
    sem_wait(&sem); // Down
    sem_post(&sem); // Up
  }
}
```

Synchronization
Sys V (ATT) Semaphores

```
#include <linux/sem.h>
struct sembuf op;
int main(int argc, char *argv[])
{
  int semid, i;
  union semun arg;

  semid = semget(1234, 1, 0666|IPC_CREAT);
  arg.val = 1; semctl(semid, 0, SETVAL, arg);
  for(i=0;i<10000000;i++) {
    op.sem_op = -1; semop(semid, &op, 1); // Down
    op.sem_op = 1;  semop(semid, &op, 1); // Up
  }
}
```

Sys V vs Posix Semaphores

- The manual pages for Posix and Sys V semaphores don't seem much different.
- The programs do exactly the same things.
- The Sys V programs run about 24 time slower than the Posix program on Linux.
- The Linux implementation is at least 3 times faster than OpenSolaris.

Sys V vs. Posix Semaphores Linux vs OpenSolaris

	SYS V	POSIX
Linux	17.26, 2.89, 14.38	0.71, 0.71, 0.0
OpenSolaris	51.38, 24.33, 26.98	45.13, 30.74, 14.34

Total time, user time, system time

Sys V vs. Posix Semaphores

- The Sys V implementation is totally kernel based. Each semaphore call takes one system call.
- The Posix implementation is a user/kernel optimized mix based on kernel **futex**'es. Only when a semaphore is contested a system call is needed.

Futex's

- A futex (short for "fast userspace mutex") is a basic tool to implement locking and building higher-level locking abstractions such as semaphores on Linux.

Futex's

- A futex consists of a piece of memory (an aligned integer) that can be shared among processes.
- It can be incremented and decremented by atomic assembler instructions, and processes can wait for the value to become positive.

Futex's

- Futex operations are done almost entirely in userspace.
- The kernel is only involved when a contended case requires arbitration.
- This allows locking primitives implemented using futexes to be very efficient.

Futex's

- Most operations do not require arbitration between processes.
- Most operations can be performed without needing to perform a (relatively expensive) system call.
- Only processor supplied atomic access/ modify instructions are used for userspace synchronization.

FUSE Filesystem

- Filesystem in Userspace (FUSE) is a loadable kernel module for Unix-like computer operating systems, that allows non-privileged users to create their own file systems without editing the kernel code.

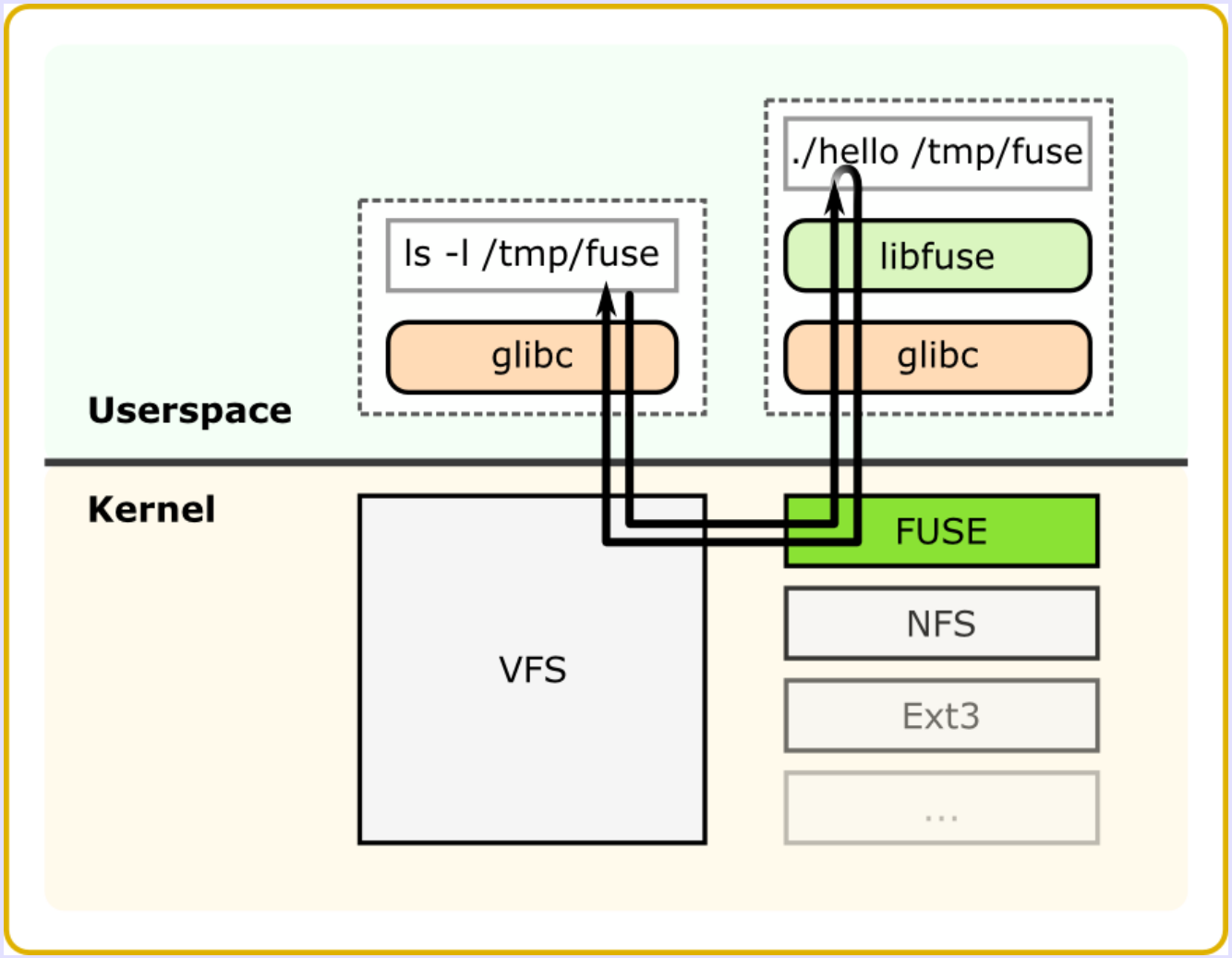
FUSE Filesystem

- This is achieved by running the file system code in user space, while the FUSE module only provides a "bridge" to the actual kernel interfaces.
- FUSE was officially merged into the mainstream Linux kernel tree in kernel version 2.6.14.

FUSE Filesystem

- FUSE is particularly useful for writing virtual file systems.
- Unlike traditional filesystems, which essentially save data to and retrieve data from disk, virtual filesystems do not actually store data themselves.
- They act as a view or translation of an existing filesystem or storage device.

FUSE Filesystem



FUSE: Hello World Example Definitions and libraries

```
#define FUSE_USE_VERSION 26

#include <stdlib.h>
#include <fuse.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>

static const char *hello_str = "Hello World!\n";
static const char *hello_path = "/hello";
```

FUSE: Hello World Example getattr() function

```
static int hello_getattr(const char *path, struct stat *stbuf)
{
      int res = 0;
      memset(stbuf, 0, sizeof(struct stat));
      if(strcmp(path, "/") == 0) {
          stbuf->st_mode = S_IFDIR | 0755;
          stbuf->st_nlink = 2;
      }
      else if(strcmp(path, hello_path) == 0) {
          stbuf->st_mode = S_IFREG | 0444;
          stbuf->st_nlink = 1;
          stbuf->st_size = strlen(hello_str);
      }
      else
          res = -ENOENT;

      return res;
}
```

FUSE: Hello World Example readdir() function

```
static int hello_readdir(const char *path, void *buf,
                         fuse_fill_dir_t filler,
                         off_t offset,
                         struct fuse_file_info *fi)
{
      (void) offset;
      (void) fi;

      if(strcmp(path, "/") != 0)
          return -ENOENT;

      filler(buf, ".", NULL, 0);
      filler(buf, "..", NULL, 0);
      filler(buf, hello_path + 1, NULL, 0);

      return 0;
}
```

FUSE: Hello World Example open() function

```
static int hello_open(const char *path,
                      struct fuse_file_info *fi)
{
      if(strcmp(path, hello_path) != 0)
          return -ENOENT;

      if((fi->flags & 3) != O_RDONLY)
          return -EACCES;

      return 0;
}
```

FUSE: Hello World Example read() function

```
static int hello_read(const char *path, char *buf,
                      size_t size, off_t offset,
                      struct fuse_file_info *fi)
{
      size_t len;
      (void) fi;
      if(strcmp(path, hello_path) != 0)
          return -ENOENT;
      len = strlen(hello_str);
      if (offset < len) {
          if (offset + size > len)
              size = len - offset;
          memcpy(buf, hello_str + offset, size);
      } else
          size = 0;
        return size;
}
```

FUSE: Hello World Example main() function

```
static struct fuse_operations hello_oper = {
      .getattr   = hello_getattr,
      .readdir = hello_readdir,
      .open   = hello_open,
      .read   = hello_read,
};

  int main(int argc, char *argv[])
{
      return fuse_main(argc, argv, &hello_oper, NULL);
}
```

AVFS - A Virtual File System

- AVFS is a system, which enables all programs to look inside archived or compressed files, or access remote files without recompiling the programs or changing the kernel.
- At the moment it supports floppies, tar and gzip files, zip, bzip2, ar and rar files, ftp sessions, http, webdav, rsh/rcp, ssh/scp.

AVFS - A Virtual File System

- Open up a tar file:

  cd test.tar#

- Read a file via ftp:

  cat .avfs/#ftp:ascender.com/pub/README

- Read a file via http:

  cat .avfs/#http:ascender.com|index.php

Diffie-Hellman Key Exchange

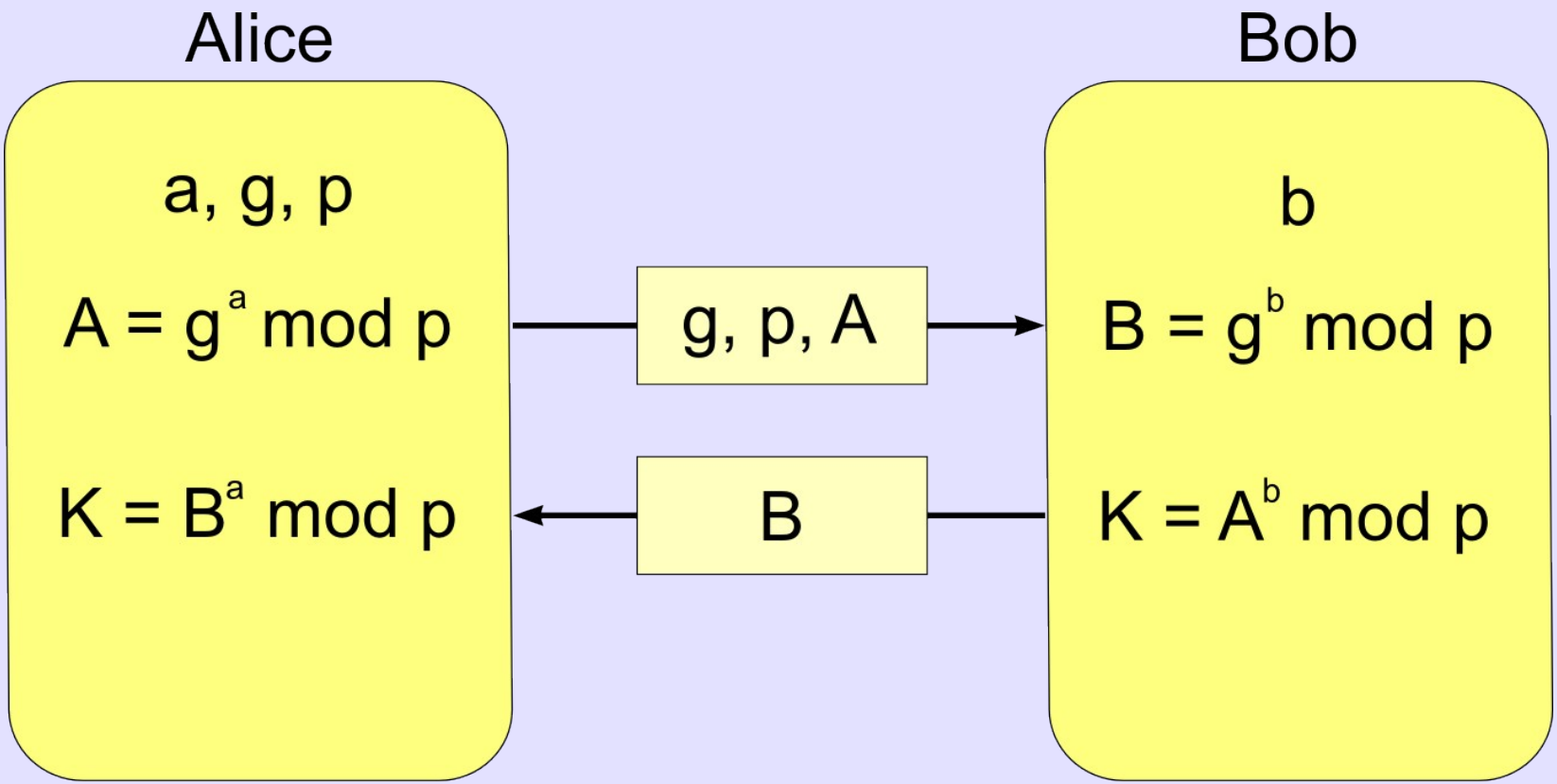
- The Diffie-Hellman key exchange procedure allows two parties that have not previously exchanged any information to agree on a “shared secret”.
- The amazing thing is even if all communications between the parties is intercepted the “man-in-the-middle” can not calculate the “shared secret”

Diffie-Hellman Key Exchange

1) Alice and Bob agree to use a prime number p=23 and base g=5.
2) Alice chooses a secret integer a=6, then sends Bob $g^a \bmod p = 5^6 \bmod 23 = 8$.
3) Bob chooses a secret integer b=15, then sends Alice $g^b \bmod p = 5^{15} \bmod 23 = 19$.
4) Alice computes $(g^b \bmod p)^a \bmod p = 19^6 \bmod 23 = 2$.
5) Bob computes $(g^a \bmod p)^b \bmod p = 8^{15} \bmod 23 = 2$.

Both Alice and Bob have arrived at the same value, because g^{ab} and g^{ba} are equal.

Diffie-Hellman Key Exchange



$$K = A^b \bmod p = (g^a \bmod p)^b \bmod p = g^{ab} \bmod p = (g^b \bmod p)^a \bmod p = B^a \bmod p$$

http://en.wikipedia.org/wiki/Image:Diffie-Hellman-Schl%C3%BCsselaustausch.svg

Diffie-Hellman Key Exchange

- Now let us analyze the key exchange from an observer that see all traffic between Bob and Alice.
- The man-in-the-middle will see *g, p, A, B.*
- I order to find the “shared secret”, *K*, we have to know either *a* or *b*.
- This problem is the *discrete logarithm problem*, believed to be intractable.

SSH Transport Layer Protocol

- The Secure Shell (SSH) is a protocol for secure remote login and other network services over an insecure network.
- The protocol can be used as a basis for a number of secure network services. It provides strong encryption, server authentication, and integrity protection.
- It may also provide compression.

SSH Transport Layer Protocol

- Key exchange method, public key algorithm, symmetric encryption algorithm, message authentication algorithm, and hash algorithm are all negotiated.
- The SSH transport layer protocol employs the Diffie-Hellman key exchange method to agree on a shared session key.

SSH Message Authentication

- A cryptographic message authentication code (MAC) is a short piece of information used to authenticate a message.
- A MAC algorithm accepts as input a secret key and an arbitrary-length message to be authenticated, and outputs a MAC (sometimes known as a tag).

SSH Message Authentication

- The MAC value protects both a message's data integrity as well as its authenticity, by allowing verifiers (who also possess the secret key) to detect any changes to the message content, and so should be called Message Authentication and Integrity Code: (MAIC).

SSH – X11 Forwarding

- X11 forwarding allows the encryption of remote X windows traffic, so that nobody can snoop on your remote xterms or insert malicious commands.
- The program automatically sets DISPLAY on the server machine, and forwards any X11 connections over the secure channel.

SSH – X11 Forwarding

- Fake Xauthority information is automatically generated and forwarded to the remote machine; the local client automatically examines incoming X11 connections and replaces the fake authorization data with the real data (never telling the remote machine the real information).

SSH – Port Forwarding

- Port forwarding allows forwarding of TCP/IP connections to a remote machine over an encrypted channel.
- Standard Internet applications like POP can be secured with this.

```
ssh -L 8080:example.com:80 example.com
```

- Then connect to the URL http://localhost:8080/

Signals

- Signals are software interrupts that provide a mechanism for handling asynchronous events.
- These events might be outside the system such as the user pressing ^C.
- These events might be internal to the program such as dividing by zero.

Signals

- Signals are software interrupts that provide a mechanism for handling asynchronous events.
- These events might be outside the system such as the user pressing ^C.
- These events might be internal to the program such as dividing by zero.

Signals

- Signals have been with Unix from the very beginning
- Over time they have evolved to become more reliable (and complex).
- The Posix standard has made signal processing under Linux robust and standard.

Signals

- All programs have to deal at some level with signals (program termination, segmentation violations, etc.)
- It is in general a good design rule to minimize the use of signals.
- It is usually a mistake to use signals as a IPC mechanism.

Signals

- Signals begin life when they are *raised*.
- The kernel *stores* the signal until it is able to deliver it to the program.
- The signal is then *delivered*.

Signals

- The kernel can perform three actions with a signal:
  1) Ignore the signal – No action will be taken
  2) Catch the signal – Perform some action when the signal is delivered to the program.
  3) Perform the default action – Perform the default action which might be termination, ignoring, core dump, stopping or continuing.

Signals Supported

• SIGHUP	1	Term
SIGINT	2	Term
SIGQUIT	3	Core
SIGILL	4	Core
SIGABRT	6	Core
SIGFPE	8	Core
SIGKILL	9	Term

Signals Supported

• SIGSEGV	11	Core
SIGPIPE	13	Term
SIGALRM	14	Term
SIGTERM	15	Term
SIGUSR1	10	Term
SIGUSR2	12	Term
SIGCHLD	17	Ign

Signals Supported

• SIGCONT	18	Cont
SIGSTOP	19	Stop
SIGTSTP	20	Stop
SIGTTIN	21	Stop
SIGTTOU	22	Stop
SIGBUS	7	Core
SIGPOLL		Term

Signals Supported

• SIGPROF	٢٧	Term
SIGSYS		Core
SIGTRAP	٥	Core
SIGURG	٢٣	Ign
SIGVTALRM	٢٦	Term
SIGXCPU	٢٤	Core
SIGXFSZ	٢٥	Core

Signal - C89

- The simplest and oldest interface is the signal() function:

```
#include <signal.h>

typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);
```

Signal - Waiting

- You can wait for a signal to occur:

```
#include <unistd.h>
int pause(void);
```

Signals - Inheritance

- Any signal that isn't ignored will be reset to their default actions after a fork.
- The *clone* system call can let child processes inherit signal actions and handlers.

Sending a Signal

-