NinePea — A Small 9P Library for Arduino and Plan 9

Eli Cohen
echoline@gmail.com

ABSTRACT

NinePea is a tiny 9P library specifically designed for use with Arduino Mega boards connected to Unix-like operating systems and Plan 9.

1. Introduction

NinePea was originally a project for Arduino. It implements the 9P filesystem protocol for servers running on the Arduino board. Compared to other 9P libraries it is more compact, sacrificing a lot of features they would have for minimalism. It only supports the 9P2000 version of 9P and does not support auth. It does not do much error checking, and is only meant to be used locally.

2. What are synthetic filesystems?

Most people who have used a modern computer are familiar with what a JPEG file is. It's a picture in storage, you can open it, delete it, etc. A synthetic file is a file created on the fly. Imagine you open a JPEG and it's the most recent image from a webcam. An implementation of exactly this is `V4LFS` for Linux which uses *NinePea* directly. It layers *NinePea* over Linux's video4linux subsystem for webcams. It presents only one synthetic file, `jpeg` , which is a synthetic jpeg generated by taking a picture from the webcam. Linux users may be familiar with the `/proc` filesystem which is a synthetic filesystem from the Linux kernel exposing information about running processes and other system information.

3. Unix Design Philosophy

There are several key ideas behind the design of a truly Unix-like operating system. Everything is a file. The screen, keyboard, and mouse are represented in the `/dev` directory by special device files. Programs are small, they do one thing each and do it well. The programs each handle one part of a task and are used together by piping together standard input and output in scripts. Efficiency is not always the main goal. Simplicity of design, interoperability of component parts, and flexibility of reuse are the goals of a truly Unix-like OS. Programs are simple filters from standard input to standard output and designed to be used in pipelines, a classic example being to take a screenshot on Plan 9:

```
cat /dev/screen | topng > screenshot.png
```

4. Arduino Boards

Arduinos are simple open-source electronics prototyping boards. Arduino started with simple Atmel AVR chips. The Arduino Mega 2560 is based around an Atmega2560 chip, a programmable MCU with only 8 KB RAM, 256 KB of programmable flash, and 4 KB of EEPROM storage. It has 16 analog-to-digital inputs, 70 pins that can be used for digital I/O, and a lot of other functionality for prototyping. These boards can drive pulse-width modulation signals, be used as controllers for SPI or I2C, and have 4 TTL serial ports. The Arduino project also encompasses a simple IDE in Java, which compiles C++ sketches to software for the AVR chip. This of course doesn't run on Plan 9, at least at the time of this writing. Plan 9 doesn't support Java or C++. All of these examples were compiled, uploaded, and used on Linux.

5. What is 9P?

9P is the protocol that makes Plan 9 beautiful engineering. It is a bytestream protocol for filesystem implementation. 9P is a stream of bytes representing filesystem opens, read, deletions, etc. It's more of an API than a filesystem in a more traditional sense. It boils the concept of a file down to a bytestream. A file is simply a system object that can be opened, closed, read from, written to, etc. On more traditional systems a file is usually some data which is stored. On Unix, one of the design principles was that "everything is a file." Data are files on storage, but devices are also represented as files. In `/dev` there are device files such as the disks, keyboard, or a webcam. Some files are synthetically created on the fly, some are on storage. Some are even remote. 9P extends this concept. It is a bytestream which can be transported in any way, over TCP/IP, inside another file, or in this case over the serial port. The Plan 9 manual page section 5 describes 9P in detail and all of the messages that can be in the bytestream.

6. What is Plan 9?

Plan 9 is a 9P multiplexer and not much else. On Plan 9, everything is a file more than even on a Unix system. 9P is the mechanism which allows this. 9P is used to share files over the network like NFS does, but also internally. On Plan 9 there is no single root directory. The root directory is a container, or namespace, different for different processes. The kernel exports a root of 9P filesystems from drivers at locations starting with # , such as `#p` for proc and `#A` for the audio device, but user programs can also post 9P servers to `#s` which is commonly bound at `/srv`. Each process has its own filesystem namespace, instead of having a static root directory like Unix. Each process can use the `mount(1)` and `bind(1)` programs to arrange its view of the filesystem however it wants. Instead of a static root and virtual filesystem, each process group has its own namespace. Mount is much different on Plan 9 than a Unix, it only speaks 9P. For example, disks in Plan 9 show up under #S , which is commonly bound to `/dev` , much like a disk would show up on Unix. However, to mount a disk another 9P file-server is needed. To mount a DOS partition the user first needs dossrv running, which translates from the DOS storage to 9P. This makes a 9P server show up at `/srv/dos` which can be mounted with the 9P-speaking Plan 9 mount command. Mount doesn't access the disk directly, it accesses the 9P server. The process to mount the DOS partition ends up more like this:

```
dossrv
mount /srv/dos /n/dos /dev/dospartition
```

Many people new to Plan 9 from Unix have made the mistake of running:

```
mount /dev/dospartition /n/dos
```

This is how it would work on Unix. Plan 9 commands share names but sometimes work differently. In this case, Plan 9 writes 9P directly to the start of the DOS partition, corrupting the storage. Plan 9 truly adheres to the Unix design principle of giving the user plenty of rope to do anything with, including making catastrophic mistakes. Everything is comprised in a powerful paradigm of simple system components with interoperable bytestreams, revolving around extensive use of 9P and per-process filesystem namespaces which can be rearranged with `mount` and `bind`. Plan 9 was never as fully developed by so many contributers as an OS like Linux. This has been good and bad for it. It does '10% as much 5 times better.' Plan 9 still remains much more minimal, and true to the Unix design philosophy.

7. What is NinePea?

NinePea is a very compact 9P server library intended for use with Arduino Megas. Almost everything runs on the Atmega chip on the Arduino board. 9P typically has a maximum of 8 KB per message, and these chips have only 8 KB of RAM. Linux's 9P support has a minimum of 4 KB message size supported, which barely fits here. On a Linux computer the included tty9p program ensures that an entire 9P message is sent or received at a time, but other than that everything runs on the Arduino. Basically the serial port becomes a 9P fileserver endpoint. Here is a usage case on Linux using the 9mount convenience program and netcat to listen on TCP/IP:

```
$ mkfifo pipe
$ ./tty9p /dev/ttyUSB0 < pipe | nc -l -p 2000 >> pipe &
$ 9mount tcp!localhost!2000 /mnt/arduino
$ ls /mnt/arduino
d00 d05 d10 d15 d20 d25 d30 d35 d40 d45 d50 d55 d60 d65
d01 d06 d11 d16 d21 d26 d31 d36 d41 d46 d51 d56 d61 d66
d02 d07 d12 d17 d22 d27 d32 d37 d42 d47 d52 d57 d62 d67
d03 d08 d13 d18 d23 d28 d33 d38 d43 d48 d53 d58 d63 d68
d04 d09 d14 d19 d24 d29 d34 d39 d44 d49 d54 d59 d64 d69
```

This is the pinfs example running on the Arduino. First a fifo pipe is made with the standard mkfifo command. The tty9p utility is used to set serial port parameters and packetize the 9P messages on the serial port, and it is looped through the fifo pipe and netcat to listen on TCP port 2000. 9mount is a convenience program for making 9P mounting simpler on Linux, and here it is being used to mount the serial port through the above command. Then the filesystem can just be accessed with the standard Linux tools. Shown here is a listing of all the synthetic files being generated by the Arduino board. This example is running on Linux, but a Plan 9 computer can easily mount the service on the Linux computer over TCP. The user can write a 1 or 0 to d13 to turn on or off the LED on the Arduino board, and can read or write any of the 70 files to access the digital I/O state of any pin.

8. Randomfs Example Sketch

Another example of *NinePea* is the randomfs sketch for Arduino. It presents one file, `random`, which can be mounted over `/dev` to replace Plan 9's `/dev/random`. This sketch is a bare example of a 9P fileserver, it only presents one file. On read it reads each of the 16 analog inputs on the Arduino Mega and XORs them with the system milliseconds counter:

```
for (pin = A0; pin < (NUM_ANALOG_INPUTS + A0); pin++) {
    seed <<= 1;
    seed ^= analogRead(pin) ^ millis();
}
```

to seed a 32-bit XOR shift pseudo-random number generator. The file it presents can be bound before `/dev/random` to replace it with the random number generation from the Arduino. `AnalogRead(pin)` returns the ambient voltage on that pin which is XOR'ed with `millis()` which is used as the seed for a simple PRNG. `Millis()` returns the number of milliseconds the Arduino has been booted, and the ambient voltage is similar on each pin, so the randomness probably isn't as good as it would be with a cryptographically secure generator. However, the manual page for `/dev/random` specifically says it is only to be used on Plan 9 as a seed for better pseudo-random number generators in the programs themselves.

9. Other Uses

NinePea was originally meant for use on Arduino boards. It could be made to work with a wifi shield to present a slow ethernet device for Plan 9 without writing any drivers. One could add a speaker and have a simple audio device. Arduino is meant for electronics prototyping, and although *NinePea* only builds under the Arduino IDE, once the board is configured and programmed as desired it can be plugged into a Plan 9 system and mounted like any other 9P server. It can be used to gather information from I2C or SPI sensors, to construct or read a signal, or for many other electronics prototyping applications. *NinePea* is also flexible, the Arduino library is labelled a C++ file but it's really just portable C. It's just a header and a C file that can be included with a project for simple 9P support. It does have some drawbacks, it isn't meant to be public-facing. It doesn't do authentication and it barely does any error checking. The V4LFS program shows how it can be included and used on Linux to wrap a webcam as a synthetic JPEG.

10. Performance of 9P

This is an example of the pinfs sketch. One interesting use of *NinePea* was using Plan 9 methodologies to bind the networking stack of the Linux machine the Arduino was plugged into over /net of a computer across the country and mounting *NinePea* remotely:

```
linux$ ./tty9p /dev/ttyUSB0 < pipe | nc -l -p 2000 >> pipe

cpu% bind /mnt/term/net /net
cpu% srv tcp!localhost!2000 arduino
cpu% mount /srv/arduino /n/a
```

This command sequence serves the serial port on TCP port 2000 from Linux, switches over to using the Linux machine's networking stack on the remote Plan 9 computer, posts a 9P service for connecting to port 2000, and finally mounts the service. After doing so, writing a 0 or 1 across the country and back takes almost a full second of 9P traffic back and forth:

```
cpu% echo 1 > /n/a/d13
```

9P adds a lot of overhead of messages going back and forth, besides the data inside it. In this case only one byte was being sent, but the overhead of 9P and the Internet across the country and back caused the data to take quite a long time to be sent out across the Internet, return, and finally go out and back over the 115200 baud serial port. The serial port was not the main bottleneck in this case. 9P is still very slow over long

distances because of all the overhead going back and forth for each operation. Mounting *NinePea* locally on the Linux computer, the main bottleneck as expected was the serial port itself, sending 9P back and forth as quickly as it could. The Arduino has an LED on digital pin 13 and LEDs for serial recieve and transmit. When it was mounted locally the LEDs for the serial port stayed on continuously while blinking the pin 13 LED in a loop, whereas when it was mounted remotely there was a visible delay as each 9P message was received.

11. Conclusion

The *NinePea* library can be copied easily for use on other systems. V4LFS is an example for Linux available at `https://github.com/echoline/v4lfs`. *NinePea* with a few more examples for Arduino Megas are available at the link `https://github.com/echoline/NinePea`. 9P and Plan 9 are fantastic software engineering, and exploring them can only help anyone interested in developing for a Unix-type system.

12. References

`https://9p.cat-v.org` A site about 9P
`intro(5)` Introduction to manual section 5 of Plan 9
`https://github.com/echoline/NinePea` The source