puffs - Pass-to-Userspace Framework File System

AsiaBSDCon 2007 Tokyo, Japan

Antti Kantee

`pooka@cs.hut.fi`

Helsinki University of Technology

Talk structure

- what is puffs?
- why do we care?
- puffs architecture overview
- kernel and transport mechanism
- userspace components
- example file systems
- measured performance figures
- compatibility
- future work
- conclusions

Introduction to puffs

Pass-to-Userspace Framework File System

- passes file system interface to userspace and provides a framework
- kernel interface: VFS
- userspace interface: almost VFS
- userspace library provides convenience functions such as continuation support
- NetBSD-current (4.0 will have *some* support)

Why the name *puffs*?

- puff pastry, increases in volume when baked

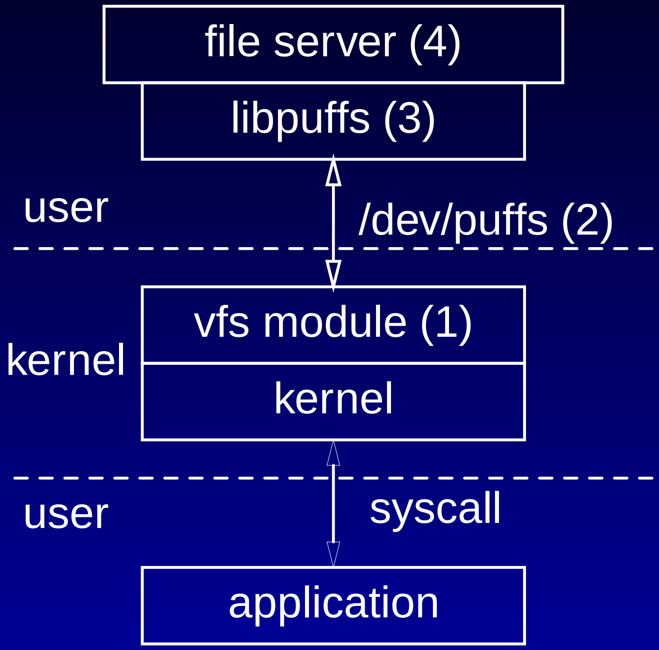
Why userspace file systems

- fault tolerance and isolation: one error doesn't bring the system down
- easier to program
  - easier to test
  - easier to debug, single-step and do iteration
- do we really need all the error-prone namespace management for example for procfs in the kernel?
- libraries and pre-existing software: most of the time written against POSIX instead of the BSD kernel

puffs architecture

1. vfs module marshalls request
2. requests are transported to userspace
3. library decodes and dispatches request
4. file server handles request

- result passed back



VFS module

- attach puffs to kernel like all file systems
- interpret incoming requests, convert to transport-suitable format and queue request to file server
- police duty making sure file server plays nice
- vnode -> file server node -> vnode handled with cookies, file server selects cookie value when it creates a node
- short-circuit unimplemented operations
- integrate to UBC
- snapshot support

Messaging format

- nothing to write a slide about .... yet
- a bunch of structs with manual accessors, no real constructors or destructors or anything of the sort
- all structs "subclassed" from the transport frame header `struct puffs_req`
- used within the kernel and libpuffs, actual file systems get a decoded interface

Transport: `/dev/puffs`

- device opened once per file system instance
- file server driven operation
  - get: fetch a request, move it to queue waiting for responses
  - put: results for a request fetched by getop, not done for all requests
  - flush: flush or purge kernel cache
  - suspend: file system snapshots
- can transport multiple requests per single getop or putop kernel call
- tries to minimize amount of copys required

User library

- provides basic programming interface for the library, plus a bunch of convenience routines
- file system implementation is a bunch of callbacks, much like with vfs
- file server should call `puffs_mount()`, execute necessary operations and either pass control the puffs or fetch and put requests by itself using library functions
  - some backends require constant fondling such as with TCP sucket buffers
  - other backends always execute everything "instantly"

file system interface

- almost vfs, not quite
- missing some operations such as `revoke()` and `get/putpages()`
- all operations get `struct puffs_cc *` as an opaque library context
- vnode operations additionally receive cookie value: either parent directory cookie or node cookie, depending on operation
- rest of the parameters mimic their kernel counterparts, e.g. `kauth_cred_t` -> `puffs_cred *`

pathnames

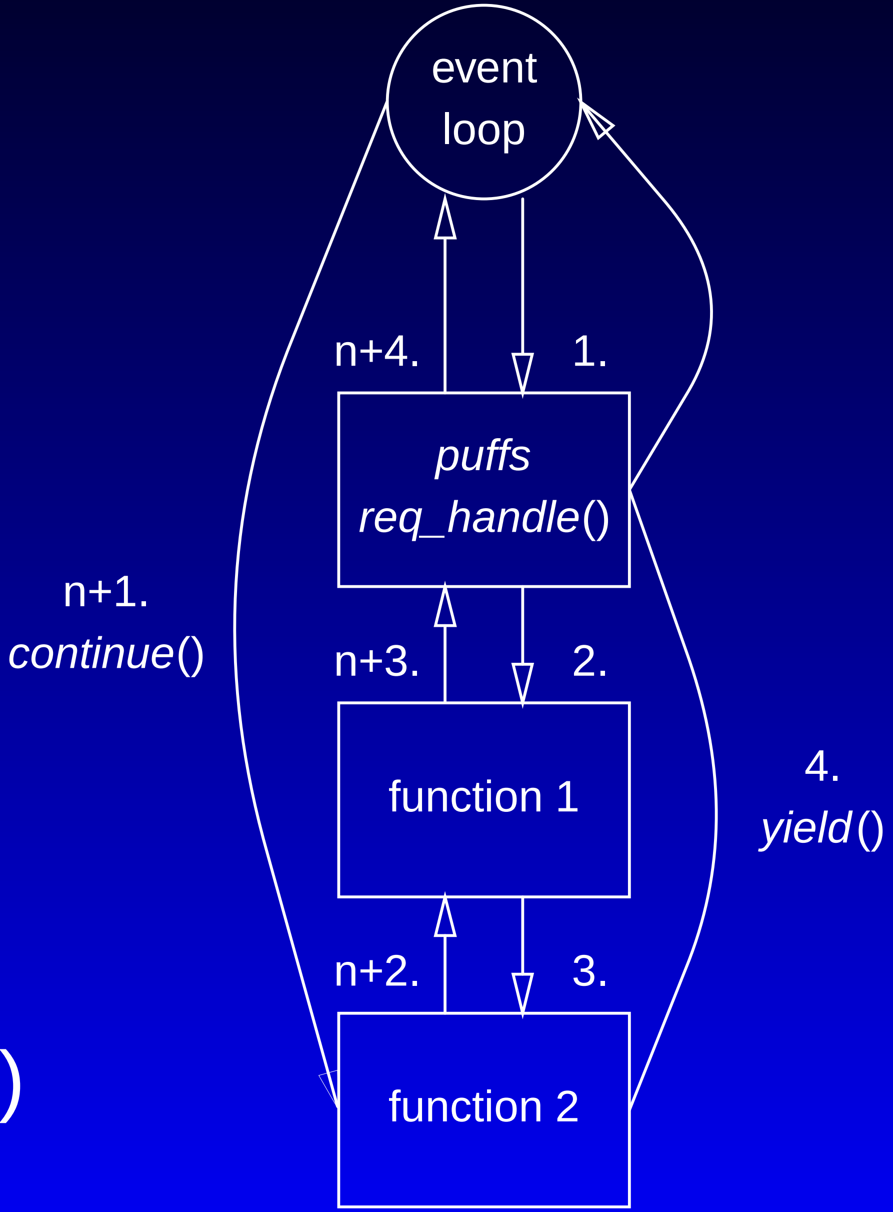
- kernel file systems operate on the concept that `lookup` provides a node and then forget about pathnames except for operations which operate in a directory
- for some user file servers, full pathnames are useful, e.g. sshfs
- puffs provides them as an optional component under the same interface
- also possible to provide own path-generating routines, such as for "rot13fs", or even something completely different like sysctl MIB names

continuations

- all file system operations do not finish instantly, usually no point in waiting synchronously
- threads could be used .... but they suck
- support continuations in libpuffs
- like threads, but explicitly scheduled with `puffs_yield()` and `puffs_continue()`
- file systems need to implement some hook from request response to continue
- need to drive file system backend I/O and puffs requests from an event loop
  - there's only one thread, remember

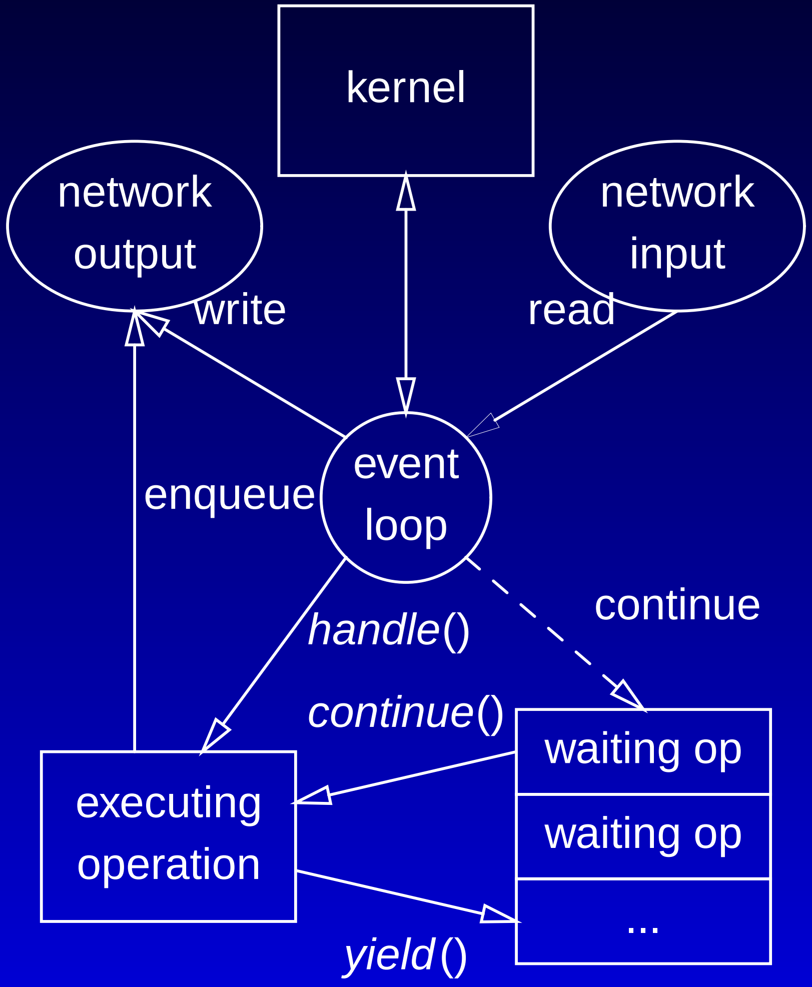
continuations continued

- automatically unwind stack to "top" of library
- jump right back in with local variables and entire stack like you left it
- library code was taxing to write, but programming is easy
- *yield*() + *continue*() "just work"



psshfs

- second version of sshfs written on top of puffs
- uses continuations
- multiple outstanding operations
- faster than nfs in some conditions



other file systems

- dtfs - delectable test file system
    - or détrempe file system, if you want to stay true to puffs
- sysctlfs - map sysctl namespace to a file system
- nullfs - operation like kernel nullfs. implemented in libpuffs with just a little frontend file system. nice for measurements
- rot13fs - present names and data of a mounted directory hierarchy as rot13

Development experiences

- some-other-namespace to file system can usually be written in about a day’s worth of work
    - this assumes a little familiarity with the system
- safe(ish ;-) to do file system development on desktop machine
- debugging nice and easy

Experimental results 1

- test extraction of kernel compilation directory (127MB, > 2000 files)

	tmpfs (s)	dtfs (s)	diff (%)
single	3.203	11.398	256%
double	5.536	22.350	303%
	ffs (s)	ffs+null (s)	diff (%)
single	47.677	53.826	12.9%
double	109.894	113.836	3.6%

Experimental results 2

- read of large file, uc : uncached, c : cached, bc : backend cached

	system (s)	wall (s)	cpu (%)
ffs (uc)	0.2	11.05	1.8
null (uc)	0.6	11.01	5.9
ffs (c)	0.2	0.21	100.0
null (c)	0.2	0.44	61.6
null (bc)	0.6	1.99	31.7

FUSE compatibility: refuse

Is it pronounced REfuse, reFUSE or REFuse?
who knows ;-)

- FUSE interface is widely spread
- supporting it is definitely a good thing, but don’t want to be limited by it
- solution: write compat layer on top of libpuffs
- agc initiated refuse project
- xtraeme added support to pkgsrc
- NetBSD can now run e.g. ntfs-3g installed from pkgsrc

Future work

- improve layering support in userspace
- make transport interface more generic
- write message specification in non-C
- support distributed vfs routing in userspace
  - and 9P while you're (I'm) at it
- (semi-)formally verify that vfs module does not expose anything dangerous to userspace
- make it clear what is expected of file systems, provide tools for it
  - currently it's only clear if you've written a couple of file systems

More work

- adapt kernel portion to NetBSD’s new locking primitives
- create tools for easy creating of file system namespaces
    - makes away with need to have homegrown struct array hacks in every fictional file system
- make interfaces more kernel-like (or make kernel more interface-like)
    - compile and run same code for kernel or userspace
    - simplification vs. unification

Wrapup

- userspace components provide isolation, fault tolerance and development comfort
- performance is the tradeoff, but usually hidden by I/O cost
  - and these days, most of the time you simply Just Don't Care
- current version of puffs works, but interfaces are not yet promised to be stable
- possible to run file systems taking advantage of the native interface or FUSE file systems using puffs + refuse

Interested? Get involved!

- if you're running NetBSD-current, add `MKPUFFS=yes` to `/etc/mk.conf`, try out `mount_psshfs` and pkgsrc stuff, file bug reports
- write new file systems (but do be prepared to change them slightly until the interface stabilizes)
- propose ideas for new features
- hype it so that people finally get rid of silly microkernel antipathies ;-)