sysctlinfo

Explore the FreeBSD sysctl MIB
and get object info

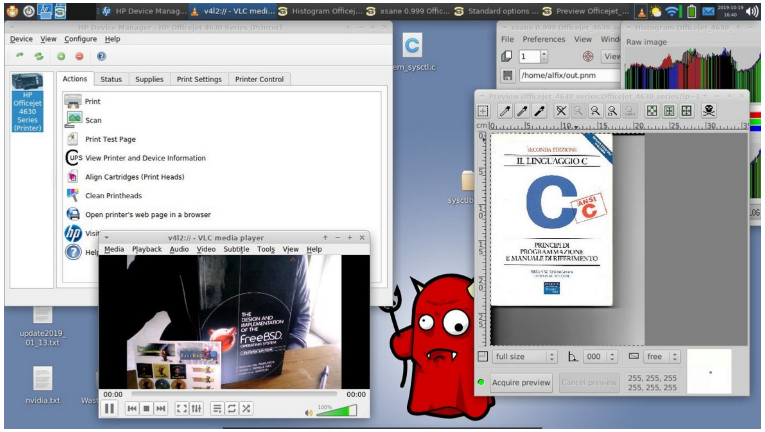
BSDCan 2020
The Technical BSD Conference
High value. Low cost. Something for everyone.

About me

- Alfonso S. Siciliano
- Computer programmer
- FreeBSD Contributor
- Daily FreeBSD Laptop User

About me

- Alfonso S. Siciliano
- Computer programmer
- FreeBSD Contributor
- Daily FreeBSD Laptop User



sysctlinfo

sysctlinfo is a new interface
to explore the sysctl MIB and
to get the info of an object

Prerequisite

sysctl()

system call

sysctl() system call

- 4.4BSD introduced the *sysctl()* system call
- get or set the state of the system
  - example: set *maxsockets*
  - example: get *numopensockets*

sysctl design

- The kernel exposes the parameters for sysctl as objects of a Management Information Base (“MIB”)
- Each object has a number so an Object Identifier (“OID”) is a series of integers separated by periods
- This is a convenient hierarchical notation for the kernel namespace

Example MIB

- [1] kern
  - [1.1] kern.ostype = “FreeBSD”
  - [1.2] kern.osrelease = “13.0-CURRENT”
  - [1.3] kern.osrevision = 199506

- [4.2.0] net.inet.ip
  - [4.2.0.1] net.inet.ip.forwarding = 0
  - [4.2.0.2] net.inet.ip.redirect = 1
  - [4.2.0.3] net.inet.ip.ttl = 64

sysctl API

```
int
sysctl(const int *id, u_int idlevel,
            void *oldp,  size_t *oldlenp,
            const void *newp, size_t newlen);
```

sysctl API

OID

```
int
sysctl(const int *id, u_int idlevel,

        void *oldp,  size_t *oldlenp,

        const void *newp, size_t newlen);
```

sysctl API

Object Value buffer

```
int
sysctl(const int *id, u_int idlevel,

          void *oldp,  size_t *oldlenp,

          const void *newp, size_t newlen);
```

sysctl API

Old value

```
int
sysctl(const int *id, u_int idlevel,

        void *oldp,  size_t *oldlenp,

        const void *newp, size_t newlen);
```

New value

sysctl get value

- Hostname OID = [1 . 1]   (kern.hostname)

```
int oid[2] = {KERN, HOSTNAME};
char buf[100];
size_t buflen = 100;
sysctl(oid, 2, buf, &buflen, NULL, 0);
printf(“VALUE: %s, %u\n”, buf, buflen);
```

```
%> ./example_sysctl
VALUE: fbsd.laptop, 12
```

sysctl set value

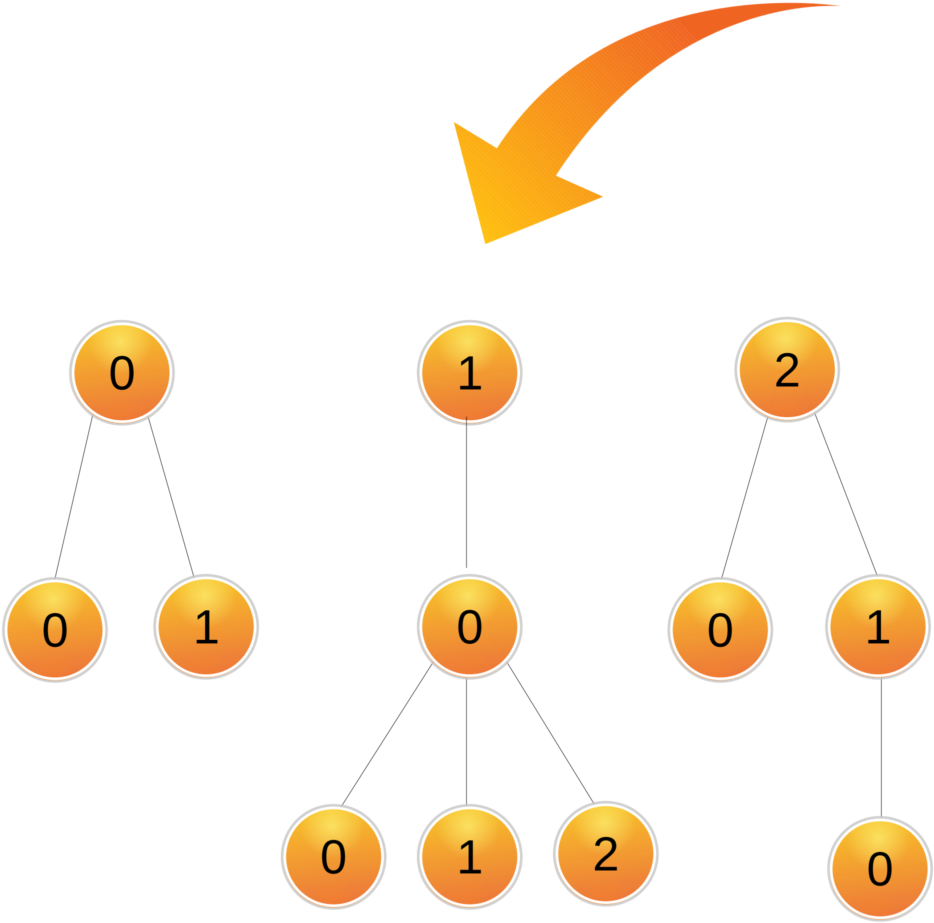
- Hostname OID = [1 . 1]   (kern.hostname)

```
int oid[2] = {KERN, HOSTNAME};
char oldbuf[100];
size_t oldbuflen = 100;
char *newbuf = “new.hostname”;

sysctl(oid, 2, buf, &buflen, newbuf, strlen(newbuf) + 1);
printf(“OLD:  %s, %u\n”, oldbuf, oldbuflen);
printf(“NEW: %s, %u\n”, newbuf, newbuflen);
```

```
%> ./example_sysctl
OLD: fbsd.laptop, 12
NEW: new.hostname, 13
```

MIB Implementation

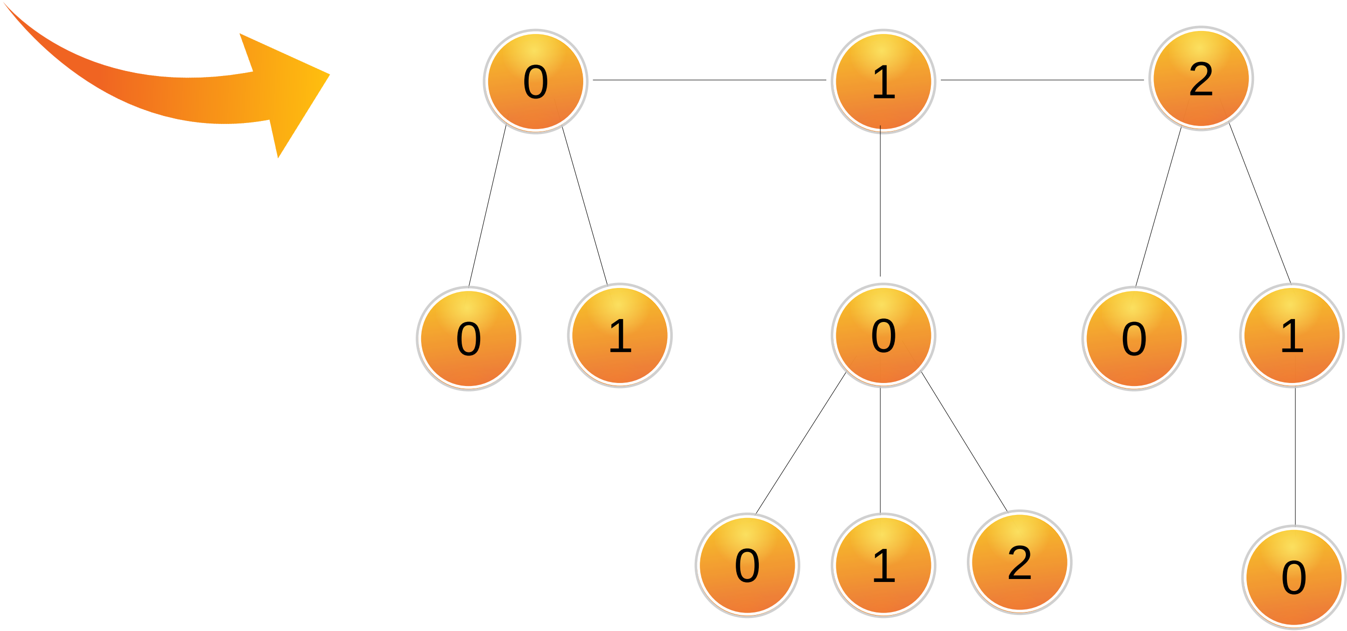
- The MIB is implemented by a collection of trees



MIB Implementation

- The MIB is implemented by a collection of trees

- The roots are entries of a list (SLIST)

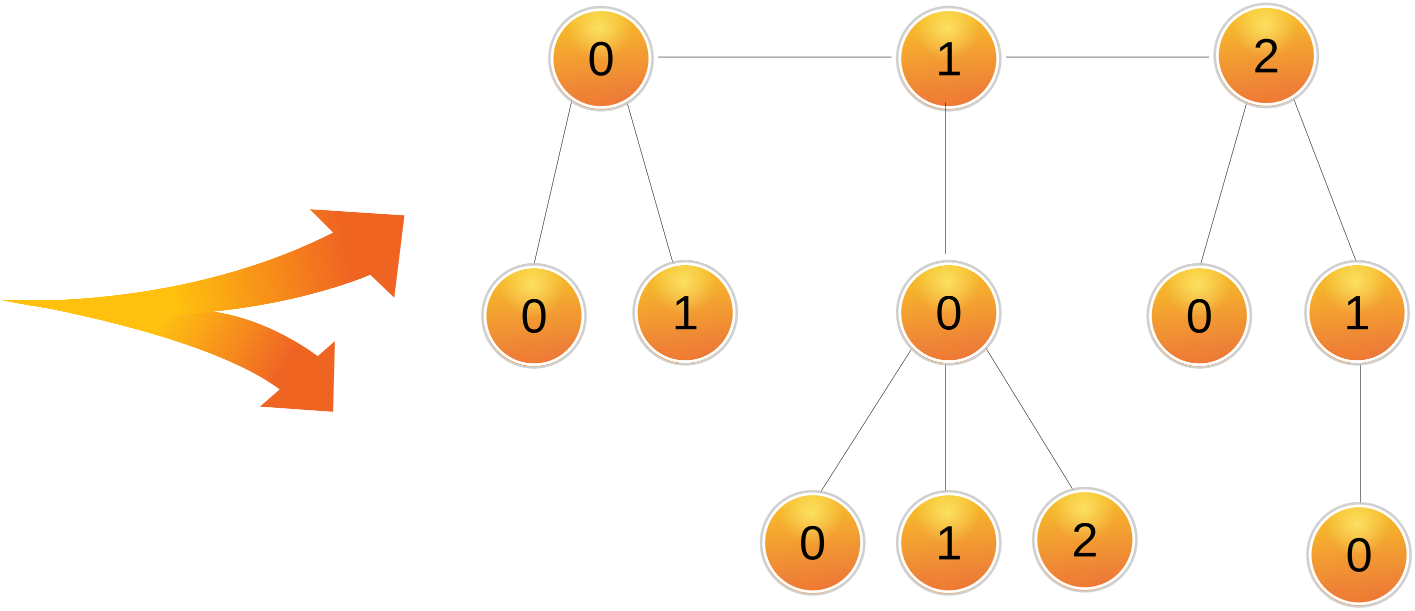


MIB Implementation

- The MIB is implemented by a collection of trees

- The roots are entries of a list (SLIST)

- Every node represents an object



Object Implementation

```
struct sysctl_oid {
        struct sysctl_oid_list oid_children;
        struct sysctl_oid_list *oid_parent;
        SLIST_ENTRY(sysctl_oid) oid_link;
        int             oid_number;
        u_int           oid_kind;
        void            *oid_arg1;
        intmax_t        oid_arg2;
        const char      *oid_name;
        int             (*oid_handler)(SYSCTL_HANDLER_ARGS);
        const char      *oid_fmt;
        int             oid_refcnt;
        u_int           oid_running;
        const char      *oid_descr;
        const char      *oid_label;
};
```

OID

kern_sysctl.c

- sysctl() explores the MIB to find the object by its OID
- sysctl() calls the handler of the object
- the handler can read or write the buffers

Object Implementation

handler

```
struct sysctl_oid {
        struct sysctl_oid_list oid_children;
        struct sysctl_oid_list *oid_parent;
        SLIST_ENTRY(sysctl_oid) oid_link;
        int             oid_number;
        u_int           oid_kind;
        void            *oid_arg1;
        intmax_t        oid_arg2;
        const char      *oid_name;
        int             (*oid_handler)(SYSCTL_HANDLER_ARGS);
        const char      *oid_fmt;
        int             oid_refcnt;
        u_int           oid_running;
        const char      *oid_descr;
        const char      *oid_label;
};
```

The sysctl(8) utility

/sbin/sysctl can get or set the system state

```
sysctl [-bdehiNnoTtqWx] [-B bufsize][-f filename]
        name[=value[,value]] ...

sysctl [-bdehNnoTtqWx] [-B bufsize] -a
```

The sysctl(8) utility

```
%> sysctl -a
kern.ostype: FreeBSD
kern.osrelease: 13.0-CURRENT
kern.osrevision: 199506
kern.version: FreeBSD 13.0-CURRENT r352742 GENERIC
kern.maxvnodes: 140219
kern.maxproc: 9124
kern.maxfiles: 119348
kern.argmax: 262144
kern.securelevel: -1
kern.hostname: fbsd.lab
kern.hostid: 345698765
kern.clockrate: { hz = 1000, tick = 1000, profhz = 8128, stathz = 127 }
...
...
```

... Thousands of objects ...

The sysctl(8) utility

```
%> sysctl kern.ostype
kern.ostype: FreeBSD


%> sysctl -d kern.ostype
kern.ostype: Operating system type


%> sysctl -t kern.ostype
kern.ostype: string
```

The prometheus_sysctl_exporter(8) utility

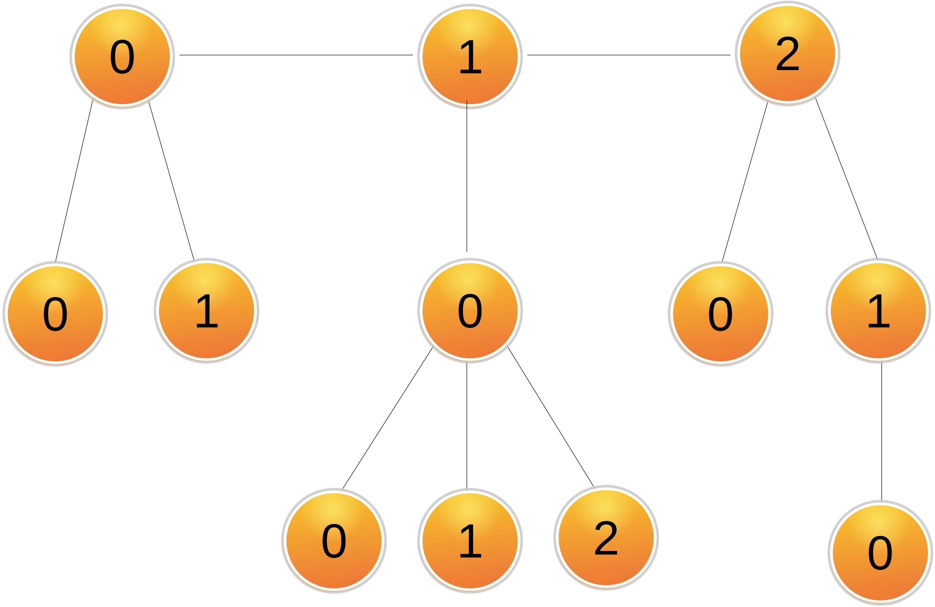
- */sbin/prometheus_sysctl_exporter*
- Addressing modern cloud computing requirements
- Added the label info to an object

```
%> prometheus_sysctl_exporter kern.features.compat_freebsd7
sysctl_kern_features{feature="compat_freebsd7"} 1
%>
```

label

The Problem

KERNEL SPACE

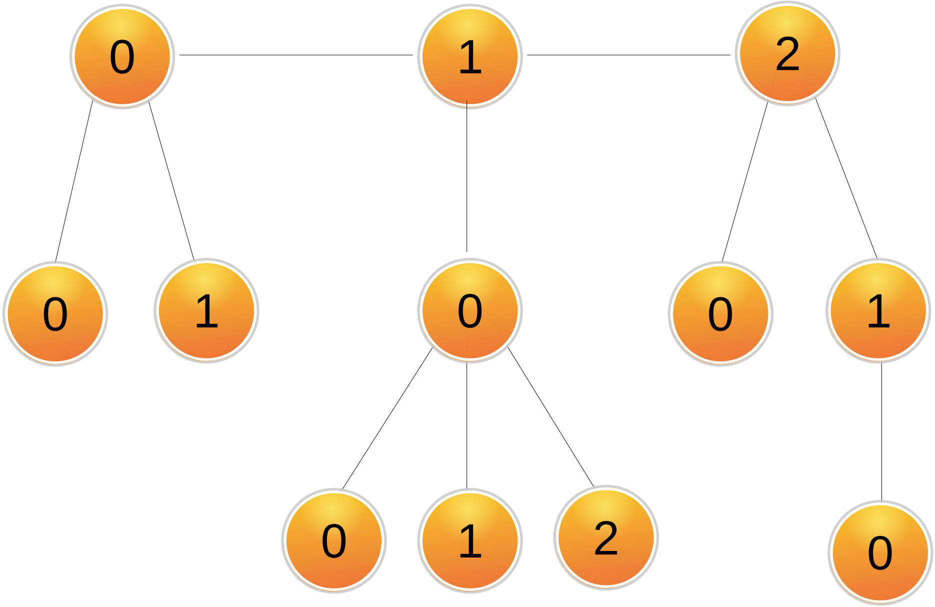


The Problem

USERLAND

KERNEL SPACE

```
%> sysctl -a
```

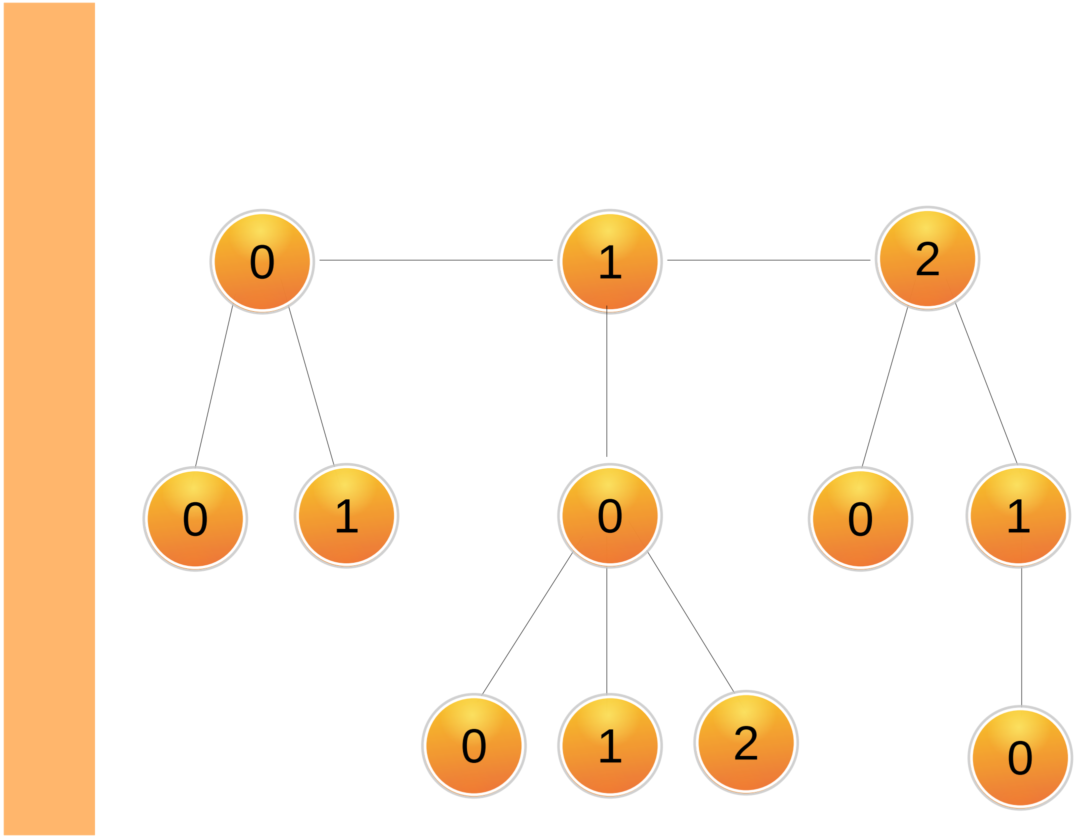


The Problem

USERLAND

sysctl():
OID → value

KERNEL SPACE



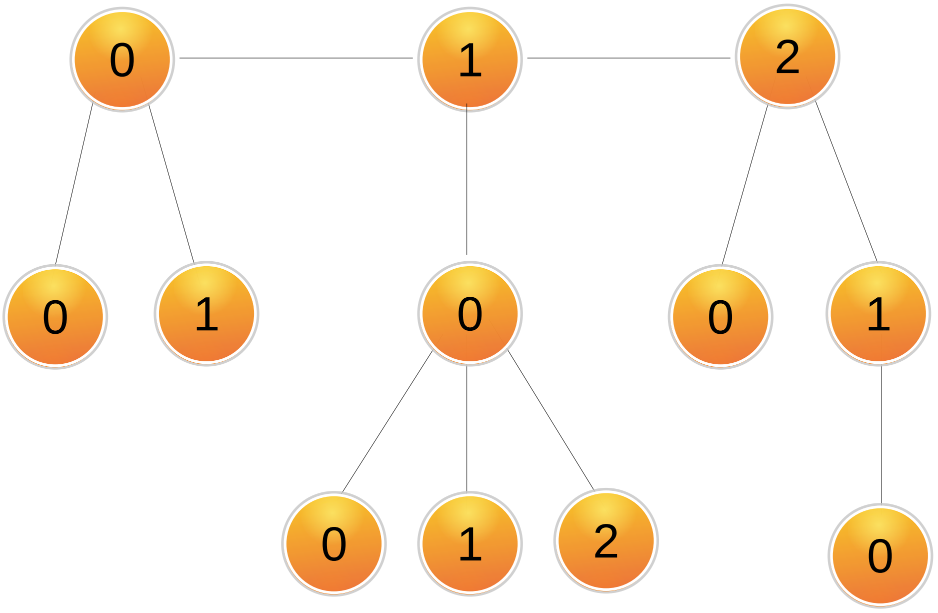
The Problem

USERLAND

sysctl():
OID → value

KERNEL SPACE

```
%> sysctl kern.ostype
```



The Problem

USERLAND

sysctl():

OID → handler()

KERNEL SPACE

```
%> sysctl -d kern.ostype
```

?

```
%> sysctl -t kern.ostype
```

?

```
%> prometheus_sysctl_exporter
```

?

```
struct sysctl_oid {
        struct sysctl_oid_list oid_children;
        struct sysctl_oid_list *oid_parent;
        SLIST_ENTRY(sysctl_oid) oid_link;
        int oid_number;
        u_int oid_kind;
        void *oid_arg1;
        intmax_t oid_arg2;
        const char *oid_name;
        int (*oid_handler)
                        (SYSCTL_HANDLER_ARGS);
        const char *oid_fmt;
        int oid_refcnt;
        u_int oid_running;
        const char *oid_descr;
        const char *oid_label;
};
```

The current interface

- The kernel provides an undocumented interface in *kern_sysctl.c*
- Introduced over 20 years ago
- For getting the info of a node: name, type, format, next leaf and OID by name
  - latter: description and label

Current interface implementation

- The interface is implemented by “internal” nodes.
- Their handlers find the wanted object then pass the info to userland

The current interface

- Undocumentd interface, *kern_sysctl.c*

```
/*
 * "Staff-functions"
 *
 * These functions implement a presently undocumented interface used by
 * the sysctl program to walk the tree, and get the type so it can print the value.
 * This interface is under work and consideration, and should probably be killed
 * with a big axe by the first person who can find the time.
 * Be aware though, that the proper interface isn't as obvious as it may seem,
 * there are various conflicting requirements.
 *
 * {0,0}        printf the entire MIB-tree.
 * {0,1,...}    return the name of the "..." OID.
 * {0,2,...}    return the next OID.
 * {0,3}        return the OID of the name in "new"
 * {0,4,...}    return the kind & format info for the "..." OID.
 * {0,5,...}    return the description of the "..." OID.
 * {0,6,...}    return the aggregation label of the "..." OID.
 */
```

Current interface implementation

- 0.1 *sysctl.name*
- 0.2 *sysctl.next*
- 0.3 *sysctl.name2oid*
- 0.4 *sysctl.oidfmt*
- 0.5 *sysctl.oiddescr*
- 0.6 *sysctl.oidlabel*

```
struct sysctl_oid {
    struct sysctl_oid_list oid_children;
    struct sysctl_oid_list *oid_parent;
    SLIST_ENTRY(sysctl_oid) oid_link;
    int oid_number;
    u_int oid_kind;
    void *oid_arg1;
    intmax_t oid_arg2;
    const char *oid_name;
    int (*oid_handler)
        (SYSCTL_HANDLER_ARGS);
    const char *oid_fmt;
    int oid_refcnt;
    u_int oid_running;
    const char *oid_descr;
    const char *oid_label;
};
```

Current interface API

- The internal nodes are CTLTYPE_NODEs with a not-NULL handler (except *sysctl.name2oid*)
- The desired node is specified exending the OID of the internal node

Current interface API

- Example: get the description of the object with id/idlevel

```
internal_oid[0] = 0;
internal_oid[1] = 5;
memcpy(internal_oid+2, id, idlevel * sizeof(int));
sysctl(internal_oid, 2 + idlevel, buf, &buflen, NULL, 0);
```

[0.5.X.Y.Z]

Current interface API

- *sysctl.name2oid uses the buffers*

```
internal_oid[0] = 0;
internal_oid[0] = 3;
sysctl(internal_oid, 2, id, &idlevel, name, strlen(name) + 1);
```

Limitations

- The CTL_MAXNAME, in sys/sysctl.h, defines the max level, currently 24

- sysctl(9) can add a node with 24 levels

  X1.x2.x3.x4.x5.x6.x7.x8.x9.x10.x11.
  x12.x13.x14.x15.x16.x17.x18.x19.
  x20.x21.x22.x23.x24

- and sysctl(3) syscall can get/set its value,

Limitations

- Unfortunatly the current interface, except sysctl.name2oid, can manage a node up to CTL_MAXNAME – 2 levels

```
internal_oid[0] = 0;
internal_oid[1] = 5;
memcpy(internal_oid+2, id, idlevele * sizeof(int));
sysctl(internal_oid, 2 + idlevel, buf, &buflen, NULL, 0);
```

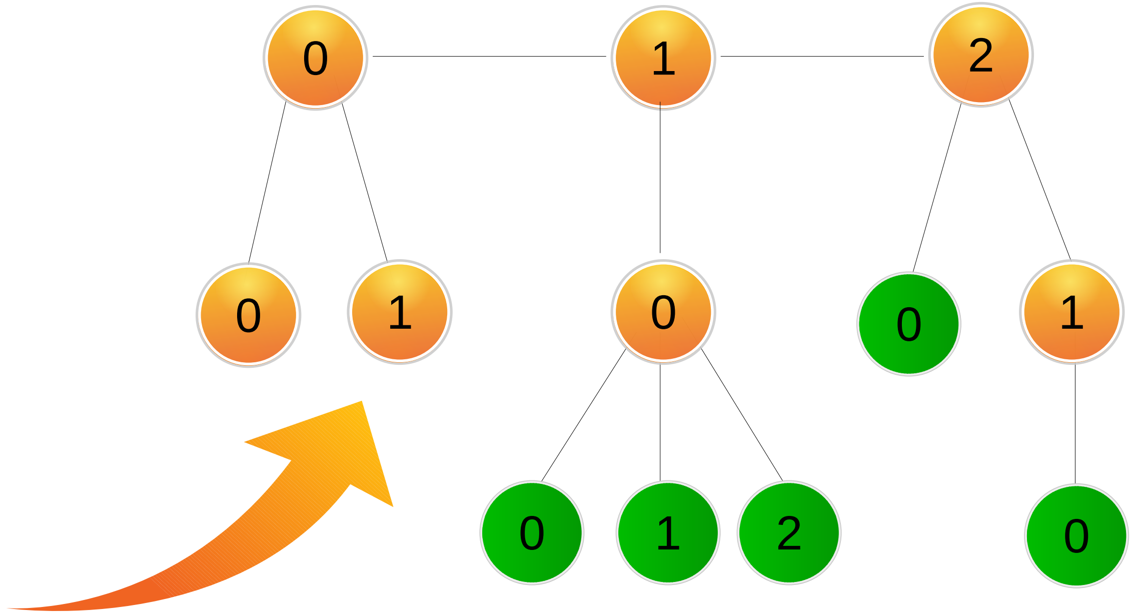
Limitations

- sysctl utility false negative

```
%> sysctl x1
sysctl: sysctl(getnext) -1 88: Cannot allocate memory

%> sysctl x1.x2.x3.x4.x5.x6.x7.x8.x9.x10.x11.x12.x13.x14.
x15.x16.x17.x18.x19.x20.x21.x22.x23.x24
sysctl: sysctl fmt -1 1024 22: Invalid argument
```

Limitations

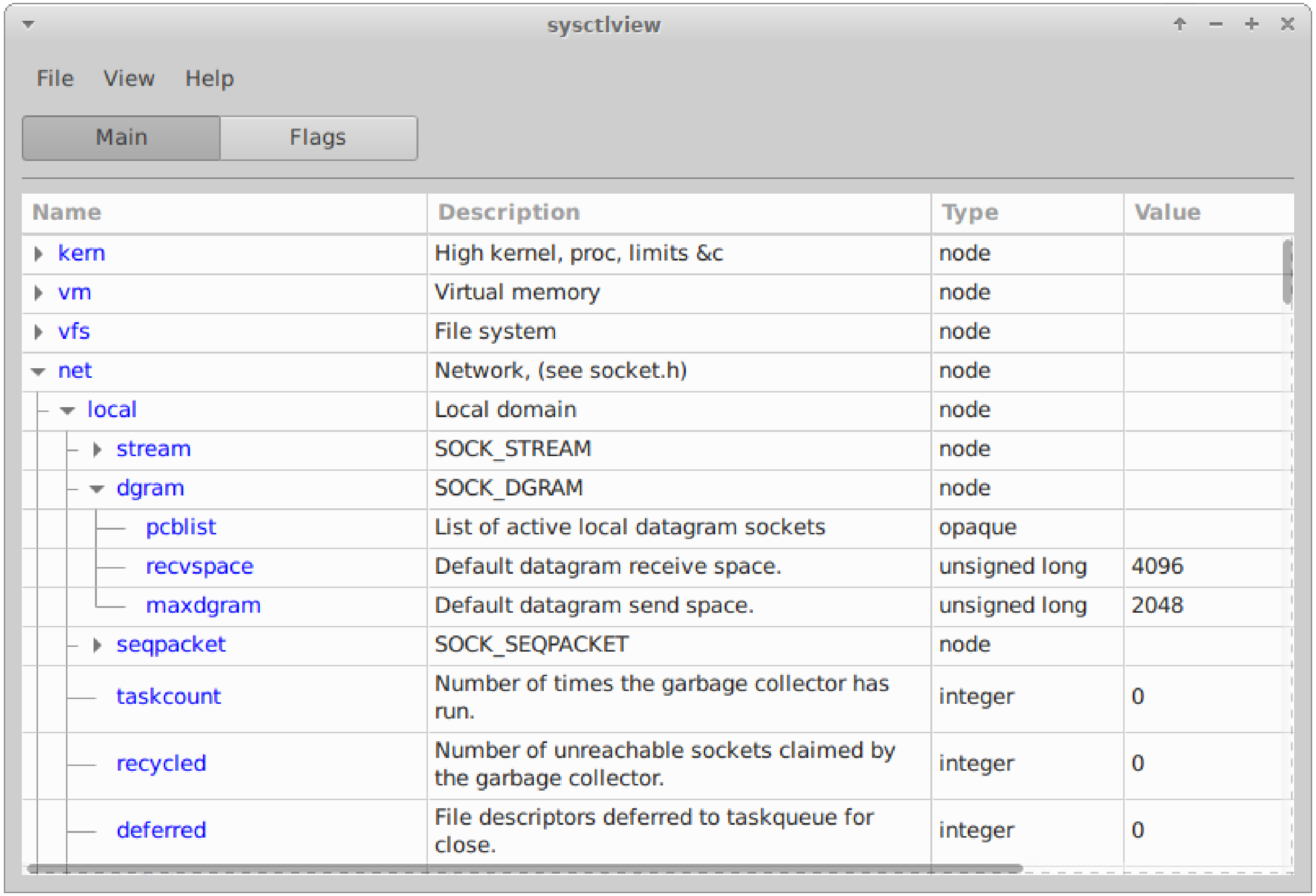
- The current interface provides *sysctl.next* to explore the MIB, it gets the next-leaf



Example:
Start 0.1

Limitations

- sysctlview, the graphical sysctl MIB explorer, needs to get also the internal nodes



Name	Description	Type	Value
kern	High kernel, proc, limits &c	node	
vm	Virtual memory	node	
vfs	File system	node	
net	Network, (see socket.h)	node	
local	Local domain	node	
stream	SOCK_STREAM	node	
dgram	SOCK_DGRAM	node	
pcblist	List of active local datagram sockets	opaque	
recvspace	Default datagram receive space.	unsigned long	4096
maxdgram	Default datagram send space.	unsigned long	2048
seqpacket	SOCK_SEQPACKET	node	
taskcount	Number of times the garbage collector has run.	integer	0
recycled	Number of unreachable sockets claimed by the garbage collector.	integer	0
deferred	File descriptors deferred to taskqueue for close.	integer	0

Limitations

- sysctlview <1.5 wasted computation comparing 2 OIDs to retrieve the internal nodes

Limitations

- *sysctl.name* gets the name by the OID
  - [1.1] → “kern.ostype”

- If no node has the specified OID *sysctl.name* returns always *zero* building a fake name up to 10 digits
  - ✗ [1.1.1000000000XXX] → “kern.ostype.1000000000”
  - ✗ totally non-existent OID
    [3000.4000.5000] → “3000.4000.5000”
  - [1.14.1.0] → “kern.ipc.pid.0” ?

Limitations

A node to build only real names can be useful, example:

- The sysctlmibinfo library wraps the current interface and provides a high level API to explore the MIB
  - *sysctlmif_name()* uses *sysctl.name* to get the name of an object by its OID

Limitations

- sysctlbyname(3) manual:

The sysctlbyname() function accepts an ASCII representation of the name and internally looks up the integer name vector. Apart from that, it behaves the same as the standard sysctl() function.

Limitations

- The sysctl(3) and sysctlbyname(3) manual uses the object: 1.30.1.<pid> = “kern.proc.pid.<pid>” in the Example Section.

- sysctl(3) [1.30.1.<pid>] ✓
- sysctlbyname(3) “kern.proc.pid.<pid>” ✗

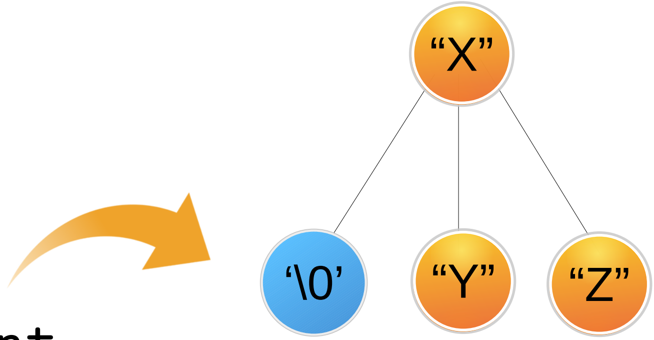
- *sysctl.name2oid* can not manage an extened name for a CTLTYPE_NODE with a no-NULL handler

Limitations

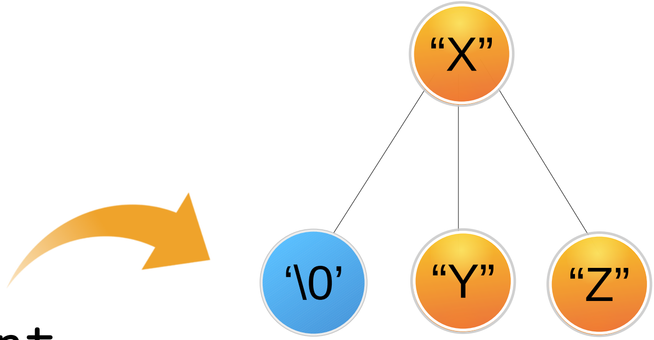
- if some level-name is just the ‘\0’ character *sysctl.name2oid* gets an incomplete OID and returns 0
- Then sysctlbyname() could get/set the value of an unwanted object
  - Probably false negative, because the ancestor is not readable/writable

Limitations

```
%> sysctl security.jail.param.allow.mount.
```

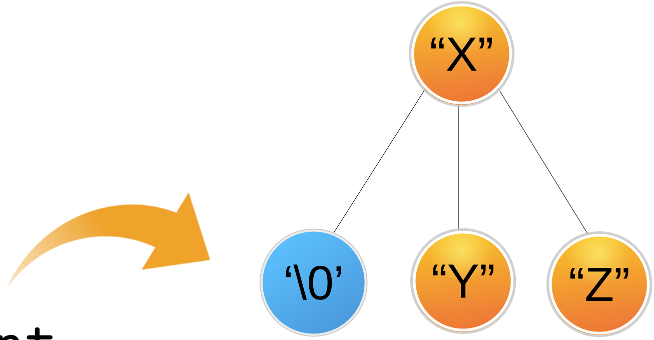


Limitations



```
%> sysctl security.jail.param.allow.mount.
security.jail.param.allow.mount.tmpfs: 0
security.jail.param.allow.mount.debugfs: 0
security.jail.param.allow.mount.anon_inodefs: 0
security.jail.param.allow.mount.procfs: 0
security.jail.param.allow.mount.devfs: 0
security.jail.param.allow.mount.: 0  ✓
```

Limitations



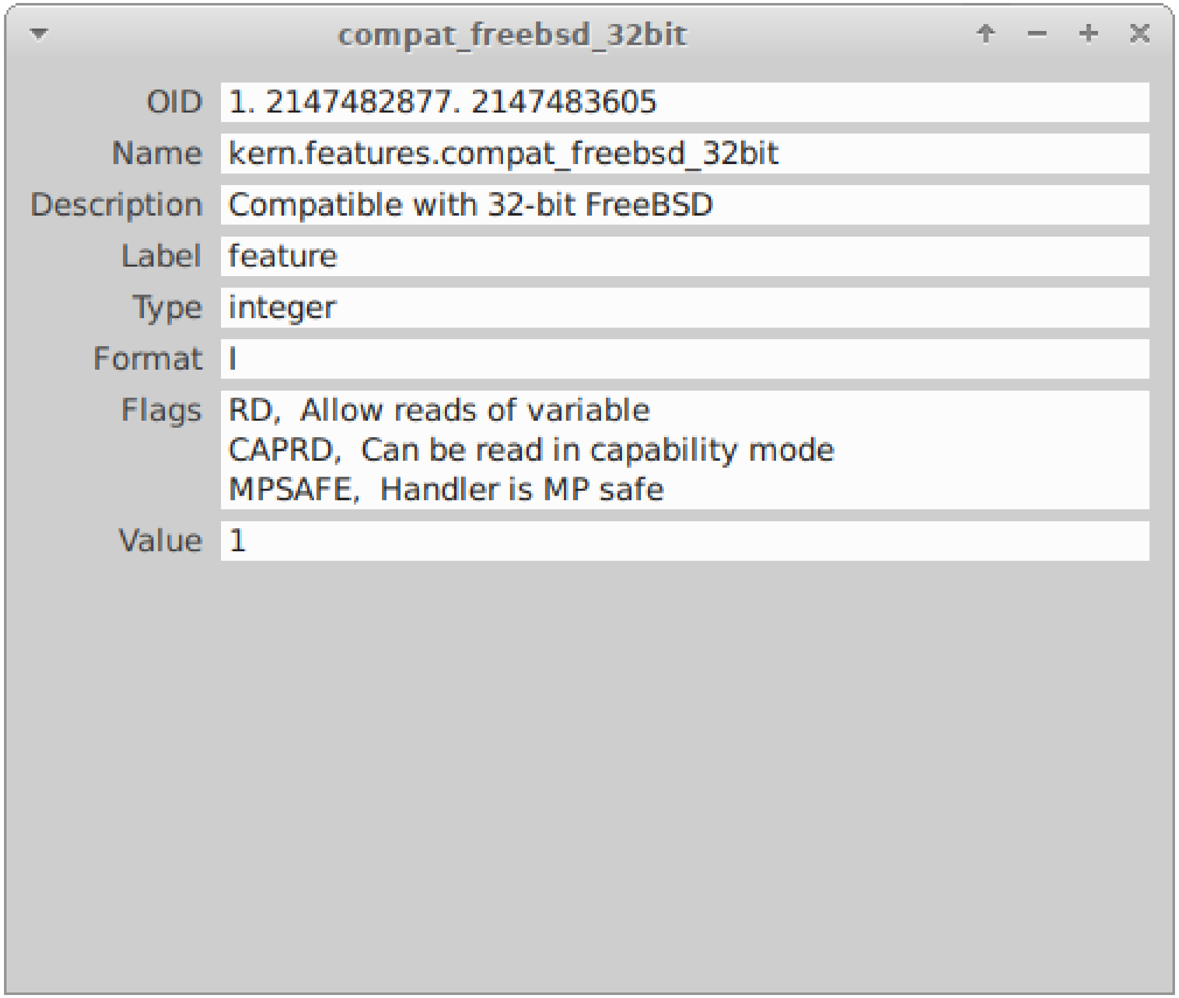
```
%> sysctl security.jail.param.allow.mount.
security.jail.param.allow.mount.tmpfs: 0              ✗
security.jail.param.allow.mount.debugfs: 0
security.jail.param.allow.mount.anon_inodefs: 0
security.jail.param.allow.mount.procfs: 0             ✗
security.jail.param.allow.mount.devfs: 0
security.jail.param.allow.mount.: 0  ✓
```

Limitations

- The current interface does not take care about security
- Capability
  - CTLFLAG_CAPWR
  - CTLFLAG_CAPRD

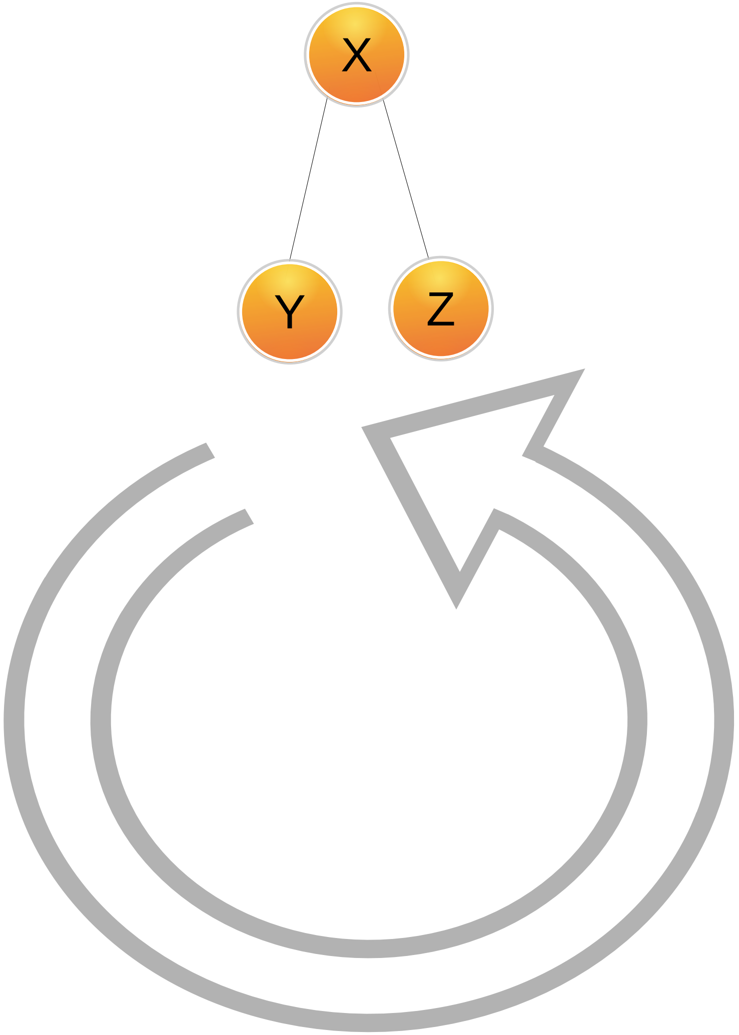
Limitations

- Inefficient



compat_freebsd_32bit

OID	1. 2147482877. 2147483605
Name	kern.features.compat_freebsd_32bit
Description	Compatible with 32-bit FreeBSD
Label	feature
Type	integer
Format	I
Flags	RD, Allow reads of variable CAPRD, Can be read in capability mode MPSAFE, Handler is MP safe
Value	1



A new interface

- sysctlinfo (info not value)
- Address limitations
- Improve efficiency
- New features
- Doc: readme, examples, manual
- Constants, no magical numbers
- Interfaces have to coexist

Features

Primarily sysctlinfo provides a new set of internal nodes to manage an object up to CTL_MAXNAME levels

- 0.1 *sysctl.name*
- 0.2 *sysctl.next*
- 0.3 *sysctl.name2oid*
- 0.4 *sysctl.oidfmt*
- 0.5 *sysctl.oiddescr*
- 0.6 *sysctl.oidlabel*

- *sysctl.entryfakename*
- *sysctl.entrynextleaf*
- *sysctl.entryfakeidbyname*
- *sysctl.entrykind*
- *sysctl.entryfmt*
- *sysctl.entrydesc*
- *sysctl.entrylabel*

Features

- sysctl utility converted to use sysctlinfo

```
%> sysctl x1
x1.x2.x3.x4.x5.x6.x7.x8.x9.x10.x11.x12.x13.x14.x15.x16.x17.x18.
x19.x20.x21.x22.x23.x24: 24
```

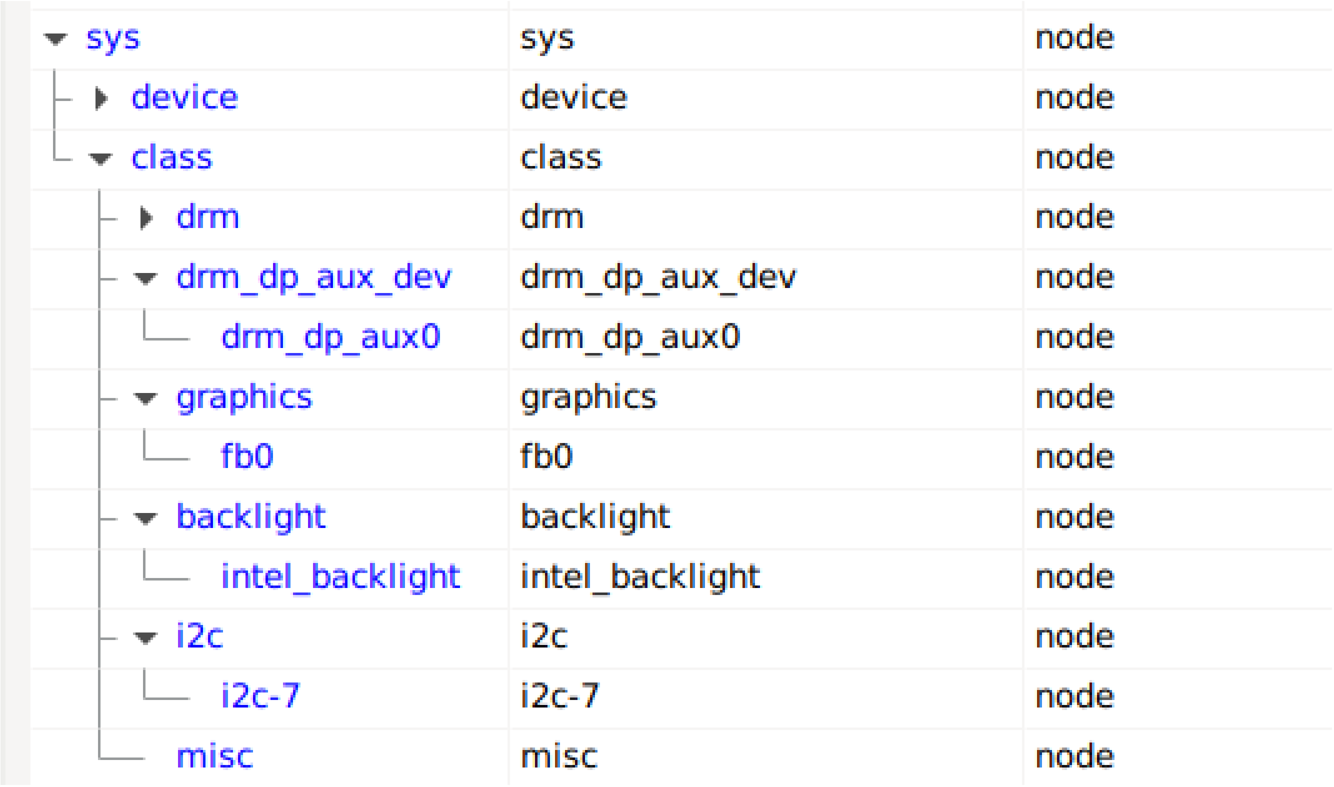
✓

```
%> sysctl x1.x2.x3.x4.x5.x6.x7.x8.x9.x10.x11.x12.x13.x14.x15.x16.
x17.x18.x19.x20.x21.x22.x23.x24
x1.x2.x3.x4.x5.x6.x7.x8.x9.x10.x11.x12.x13.x14.x15.x16.x17.x18.
x19.x20.x21.x22.x23.x24: 24
```

✓

Features

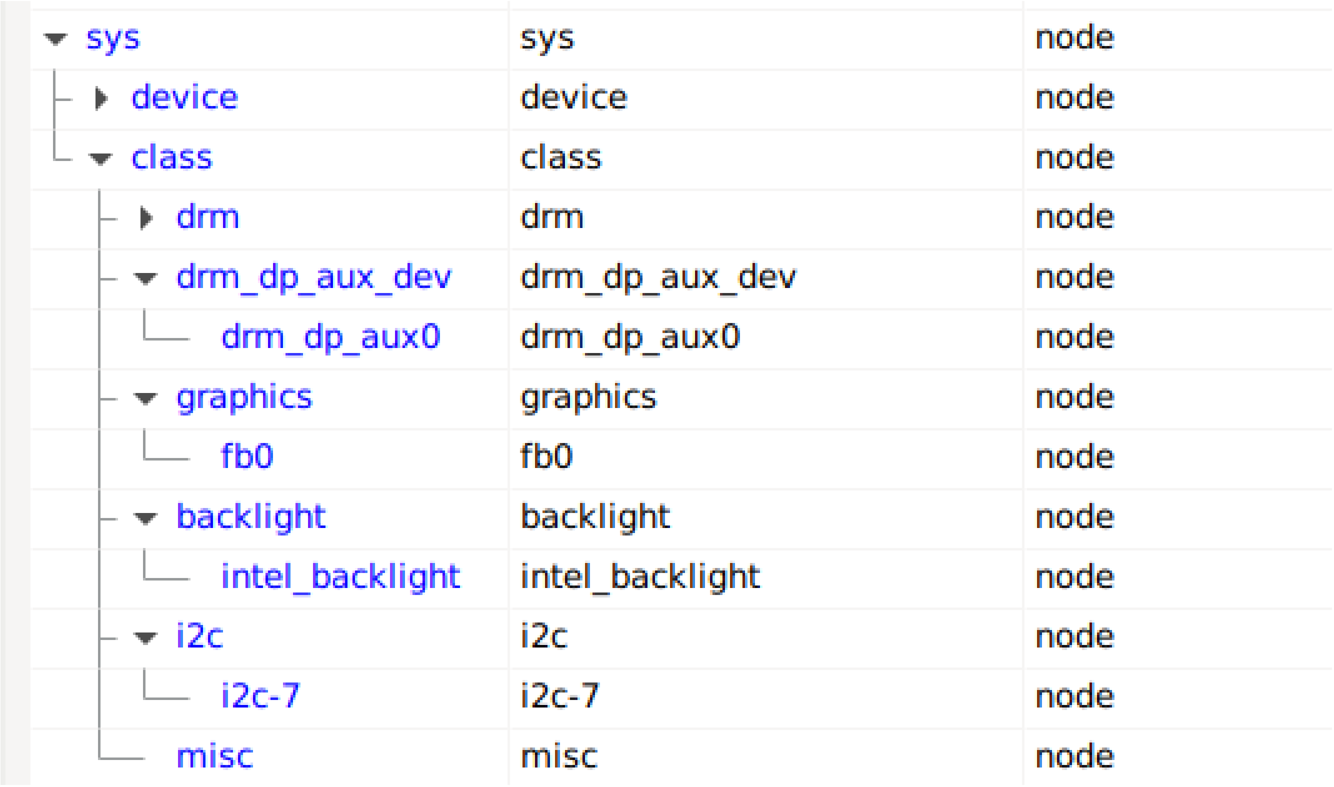
- New feature: *sysctl.entrynextnode* to get the next leaf or the next internal node



sys	sys	node
device	device	node
class	class	node
drm	drm	node
drm_dp_aux_dev	drm_dp_aux_dev	node
drm_dp_aux0	drm_dp_aux0	node
graphics	graphics	node
fb0	fb0	node
backlight	backlight	node
intel_backlight	intel_backlight	node
i2c	i2c	node
i2c-7	i2c-7	node
misc	misc	node

Features

- New feature: *sysctl.entrynextnode* to get the next leaf or the next internal node



sys	sys	node
device	device	node
class	class	node
drm	drm	node
drm_dp_aux_dev	drm_dp_aux_dev	node
drm_dp_aux0	drm_dp_aux0	node
graphics	graphics	node
fb0	fb0	node
backlight	backlight	node
intel_backlight	intel_backlight	node
i2c	i2c	node
i2c-7	i2c-7	node
misc	misc	node

Subtree of CTLTYPE_NODE

Features

- New feature:*sysctl.entryidbyname* to build the OID by name also if some level-name is just ‘\0’, unlike:
  - *sysctl.name2oid*
  - *sysctl.entryfakeidbyname*

Features

- sysctl converted on sysctlinfo

```
%> sysctl security.jail.param.allow.mount.
security.jail.param.allow.mount.: 0
```

✓

- sysctl in BASE on the current interface

```
%> sysctl security.jail.param.allow.mount.
security.jail.param.allow.mount.tmpfs: 0
security.jail.param.allow.mount.debugfs: 0
security.jail.param.allow.mount.anon_inodefs: 0
security.jail.param.allow.mount.procfs: 0
security.jail.param.allow.mount.devfs: 0
security.jail.param.allow.mount.: 0
```

✗

Features

- New feature: *sysctl.entryidinputyname* can manage a name extended with an input for the handler
- If the name is expanded the object has to be a CTLTYPE_NODE with a not-NULL handler
- To improve sysctlbyname()

```
sysctlbyname("kern.proc.pid.1") -1 Error! ✗
sysctlbyname_improved("kern.proc.pid.1") 0 OK! ✓
```

Features

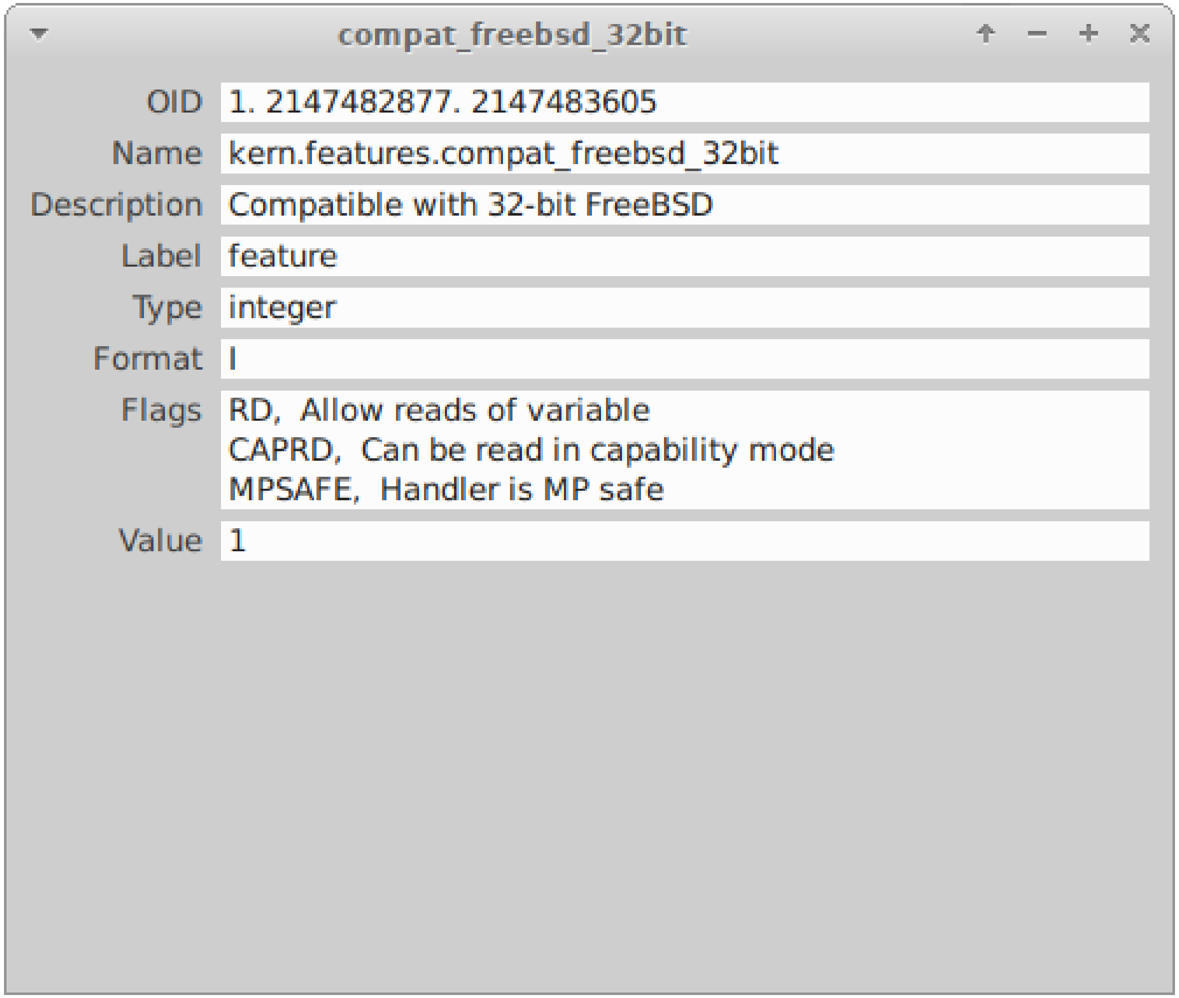
- The new interface is still inefficient:
  - a single info at a time
  - the kernel needs to find the same objects many times

Features

- Introduced
  - *sysctl.entryallinfo*
  - *sysctl.entryallinfo_withnextnode*
  - *sysctl.entryallinfo_withnextleaf*
- 33% (30-35) more efficient

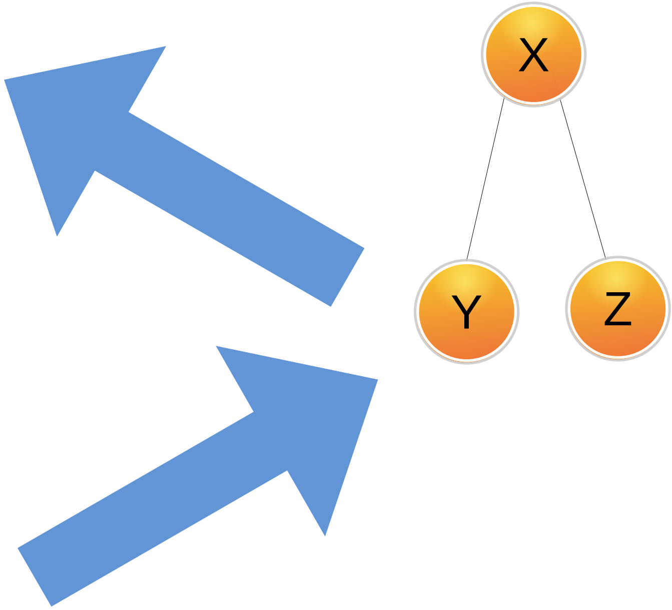
Features

- Efficient



compat_freebsd_32bit

OID	1. 2147482877. 2147483605
Name	kern.features.compat_freebsd_32bit
Description	Compatible with 32-bit FreeBSD
Label	feature
Type	integer
Format	I
Flags	RD, Allow reads of variable CAPRD, Can be read in capability mode MPSAFE, Handler is MP safe
Value	1



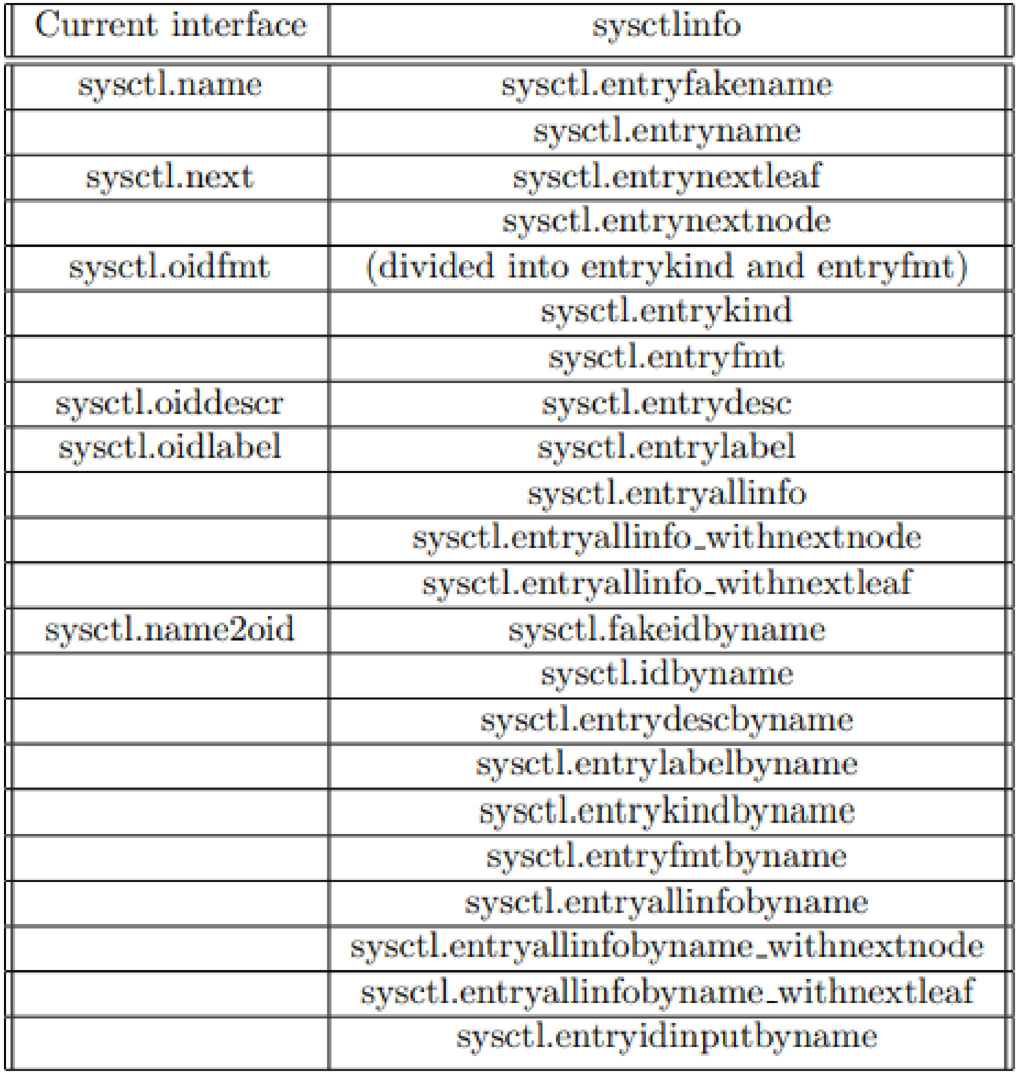
Features

- *****_byname_ nodes search the object by its name
- avoid to call sysctl.name2oid (or similar) to explore the MIB just to find the corresponding OID
    - sysctl.entrydescbyname,
    - sysctl.entrylabelbyname
    - sysctl.entrykindbyname
    - sysctl.entryfmtbyname
    - sysctl.entryallinfobyname
    - sysctl.entryallinfobyname_withnextnode
    - sysctl.entryallinfobyname_withnextleaf

Features

- Support capability mode
- After *cap_enter(2)* check
  - *CTLFLAG_CAPWR*
  - *CTLFLAG_CAPRD*

Comparison



Current interface	sysctlinfo
sysctl.name	sysctl.entryfakename
	sysctl.entryname
sysctl.next	sysctl.entrynextleaf
	sysctl.entrynextnode
sysctl.oidfmt	(divided into entrykind and entryfmt)
	sysctl.entrykind
	sysctl.entryfmt
sysctl.oiddescr	sysctl.entrydesc
sysctl.oidlabel	sysctl.entrylabel
	sysctl.entryallinfo
	sysctl.entryallinfo_withnextnode
	sysctl.entryallinfo_withnextleaf
sysctl.name2oid	sysctl.fakeidbyname
	sysctl.idbyname
	sysctl.entrydescbyname
	sysctl.entrylabelbyname
	sysctl.entrykindbyname
	sysctl.entryfmtbyname
	sysctl.entryallinfobyname
	sysctl.entryallinfobyname_withnextnode
	sysctl.entryallinfobyname_withnextleaf
	sysctl.entryidinputbyname

sysctlinfo API

```
int
SYSCTLINFO(int *id, size_t idlevel, int prop[2],
    void *buf, size_t *buflen);

int
SYSCTLINFO_BYNAME(char *name, int prop[2],
    void *buf, size_t *buflen);
```

API

Wanted object (OID)

```
int
SYSCTLINFO(int *id, size_t idlevel,

           int prop[2],

           void *buf, size_t *buflen);
```

API

Prop[0] = CTL_SYSCTL
Prop[1] = What info?
ENTRYNAME, ENTRYDESC,...

```
int
SYSCTLINFO(int *id, size_t idlevel,

           int prop[2],

           void *buf, size_t *buflen);
```

API

```
int
SYSCTLINFO(int *id, size_t idlevel,

           int prop[2],

           void *buf, size_t *buflen);
```

Info

API

Wanted object (name)

```
int
SYSCTLINFO_BYNAME(char *name,
                  int prop[2],
                  void *buf, size_t *buflen);
```

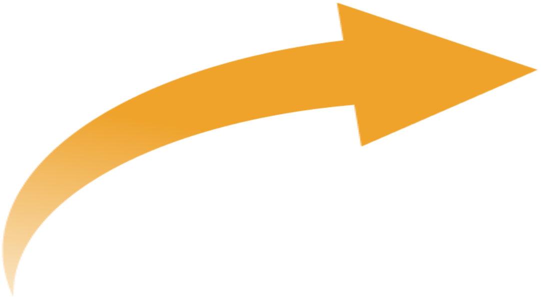
API

Prop[0] = CTL_SYSCTL
Prop[1] = What info?
ENTRYDESCBYNAME, ...

```
int
SYSCTLINFO_BYNAME(char *name,

                  int prop[2],

                  void *buf, size_t *buflen);
```

API

```
int
SYSCTLINFO_BYNAME(char *name,
                  int prop[2],
                  void *buf, size_t *buflen);
```



Info

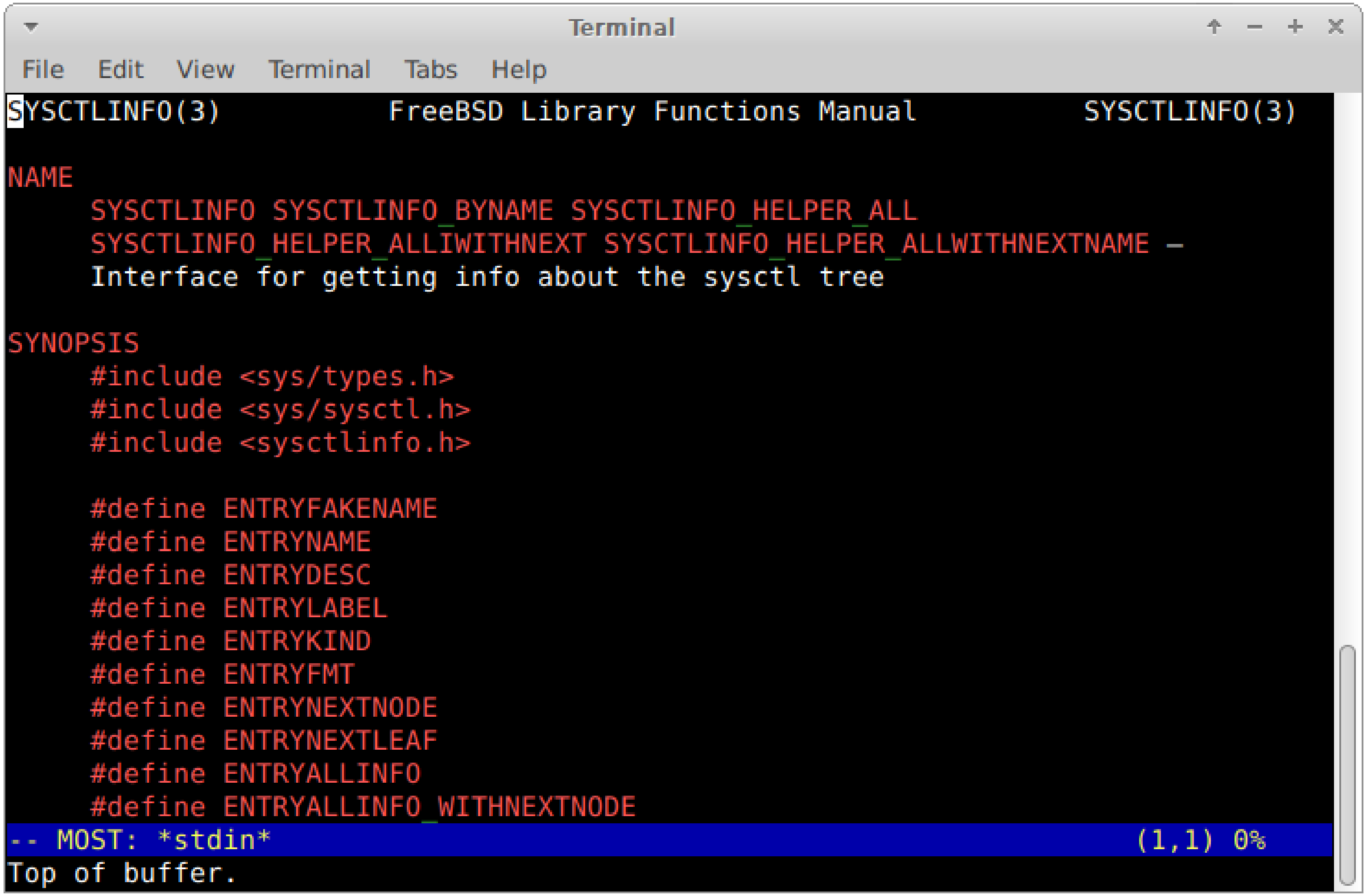
API

```
prop[0] = CTL_SYSCTL;
prop[1] = ENTRYDESCBYNAME;
SYSCTLINFO_BYNAME(“kern.ostype”, prop, buf, &buflen);
printf(“%s\n”, buf);
```

```
%> ./sysctlinfo_example_description
Operating system type
```

API

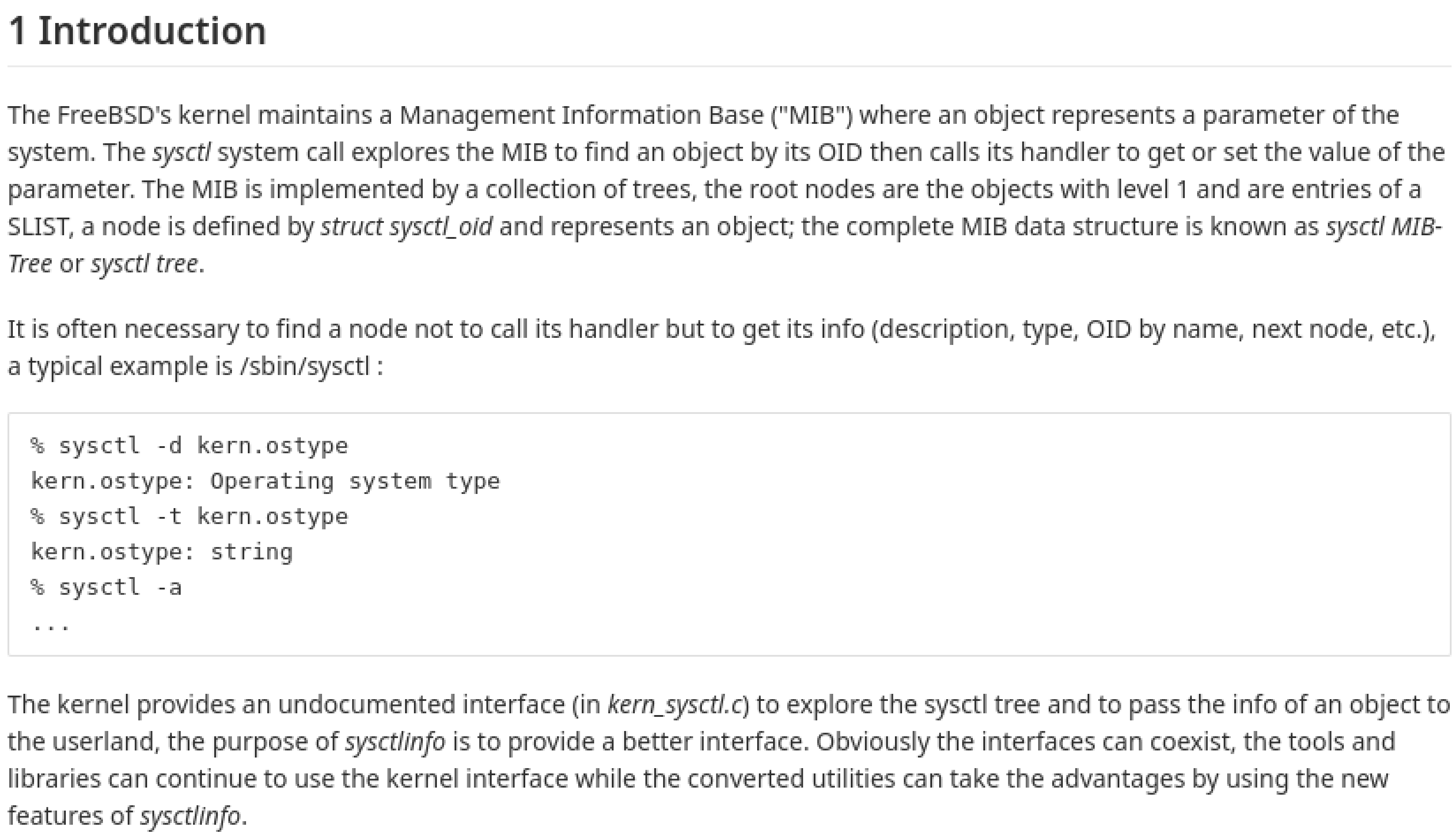
- Manuals sysctlinfo.3/.4



```
SYSCTLINFO(3)          FreeBSD Library Functions Manual          SYSCTLINFO(3)

NAME
     SYSCTLINFO SYSCTLINFO_BYNAME SYSCTLINFO_HELPER_ALL
     SYSCTLINFO_HELPER_ALLIWITHNEXT SYSCTLINFO_HELPER_ALLWITHNEXTNAME –
     Interface for getting info about the sysctl tree

SYNOPSIS
     #include <sys/types.h>
     #include <sys/sysctl.h>
     #include <sysctlinfo.h>

     #define ENTRYFAKENAME
     #define ENTRYNAME
     #define ENTRYDESC
     #define ENTRYLABEL
     #define ENTRYKIND
     #define ENTRYFMT
     #define ENTRYNEXTNODE
     #define ENTRYNEXTLEAF
     #define ENTRYALLINFO
     #define ENTRYALLINFO_WITHNEXTNODE
-- MOST: *stdin*                                                  (1,1) 0%
Top of buffer.
```

API

- README



1 Introduction

The FreeBSD's kernel maintains a Management Information Base ("MIB") where an object represents a parameter of the system. The *sysctl* system call explores the MIB to find an object by its OID then calls its handler to get or set the value of the parameter. The MIB is implemented by a collection of trees, the root nodes are the objects with level 1 and are entries of a SLIST, a node is defined by *struct sysctl_oid* and represents an object; the complete MIB data structure is known as *sysctl MIB-Tree* or *sysctl tree*.

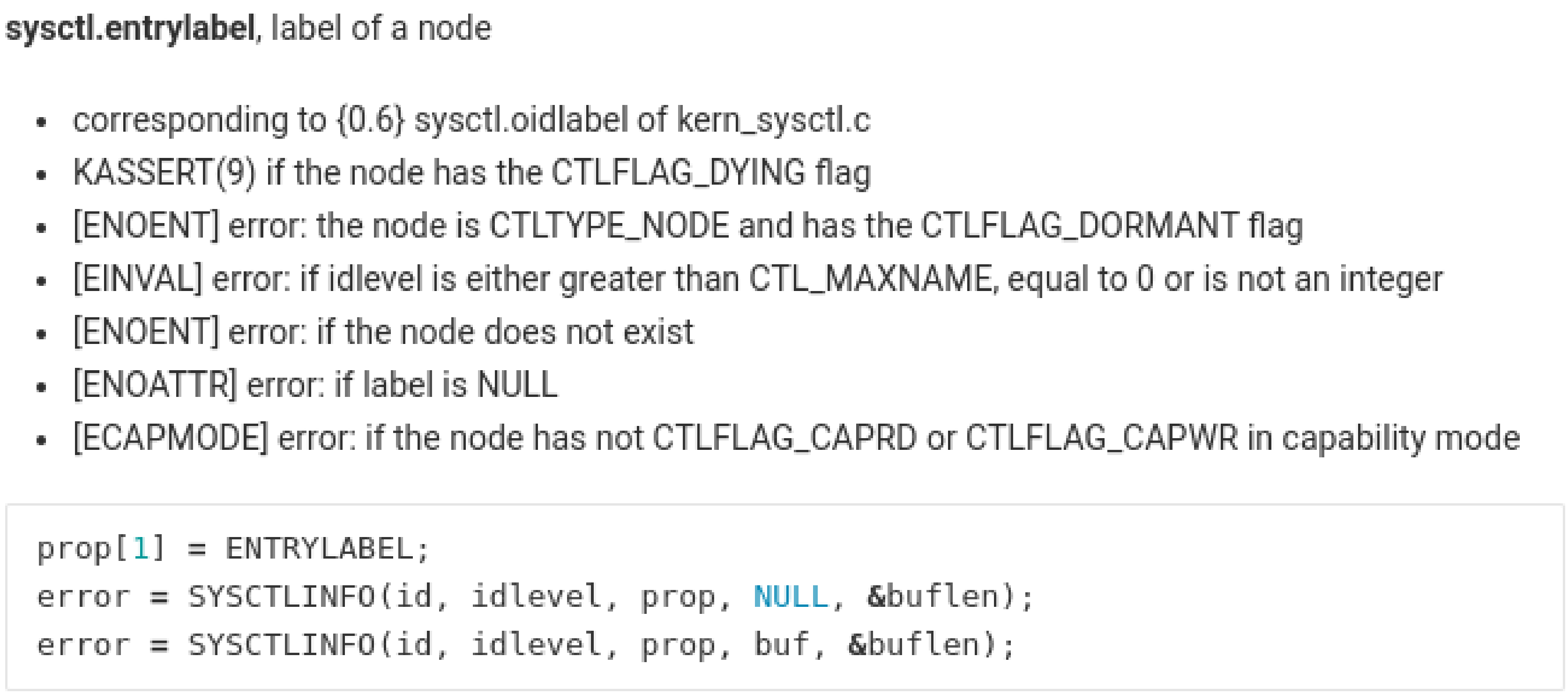
It is often necessary to find a node not to call its handler but to get its info (description, type, OID by name, next node, etc.), a typical example is /sbin/sysctl :

```
% sysctl -d kern.ostype
kern.ostype: Operating system type
% sysctl -t kern.ostype
kern.ostype: string
% sysctl -a
...
```

The kernel provides an undocumented interface (in *kern_sysctl.c*) to explore the sysctl tree and to pass the info of an object to the userland, the purpose of *sysctlinfo* is to provide a better interface. Obviously the interfaces can coexist, the tools and libraries can continue to use the kernel interface while the converted utilities can take the advantages by using the new features of *sysctlinfo*.

API

- README



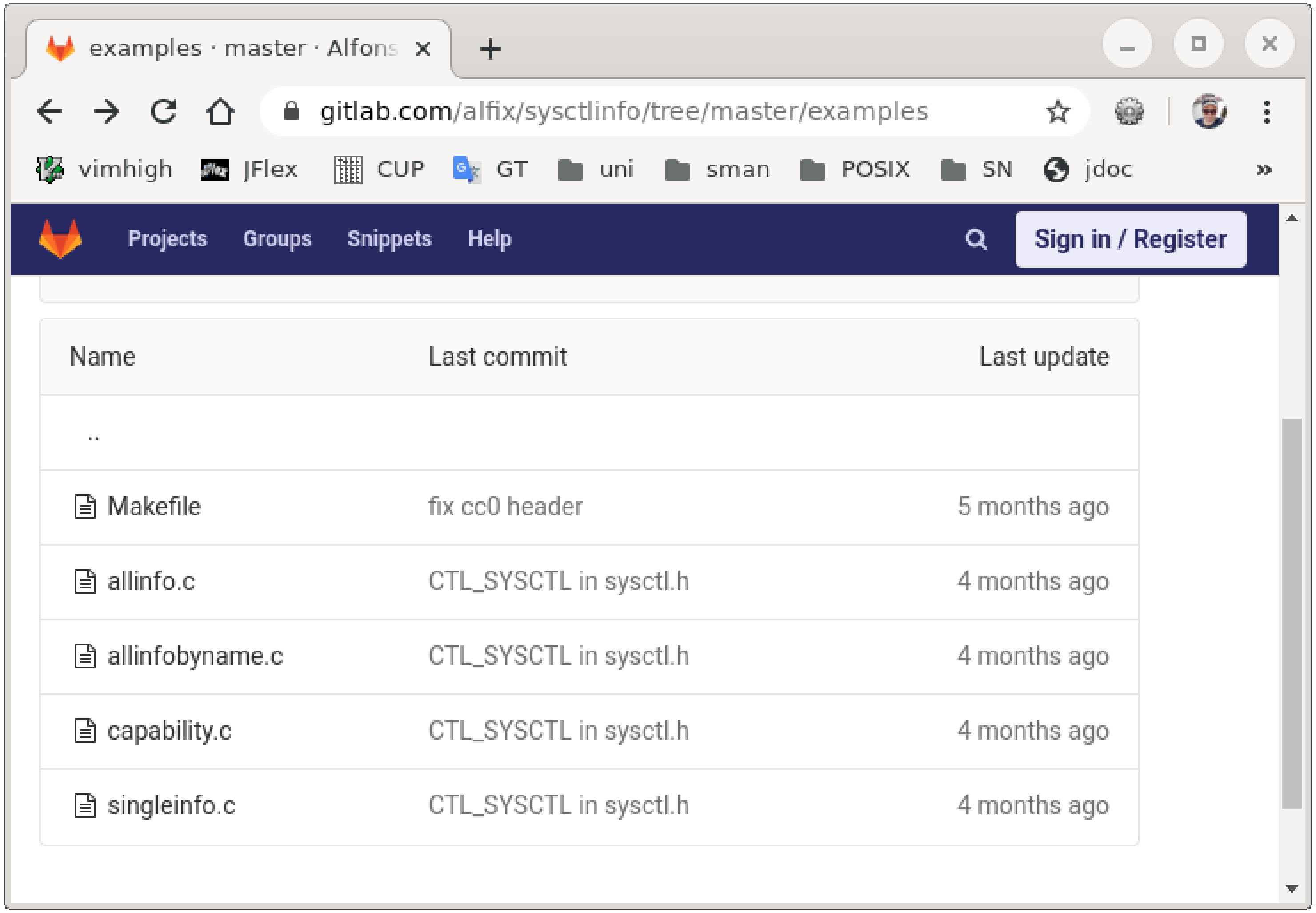
sysctl.entrylabel, label of a node

- corresponding to {0.6} sysctl.oidlabel of kern_sysctl.c
- KASSERT(9) if the node has the CTLFLAG_DYING flag
- [ENOENT] error: the node is CTLTYPE_NODE and has the CTLFLAG_DORMANT flag
- [EINVAL] error: if idlevel is either greater than CTL_MAXNAME, equal to 0 or is not an integer
- [ENOENT] error: if the node does not exist
- [ENOATTR] error: if label is NULL
- [ECAPMODE] error: if the node has not CTLFLAG_CAPRD or CTLFLAG_CAPWR in capability mode

```
prop[1] = ENTRYLABEL;
error = SYSCTLINFO(id, idlevel, prop, NULL, &buflen);
error = SYSCTLINFO(id, idlevel, prop, buf, &buflen);
```

API

- Examples to explore the MIB



Name	Last commit	Last update
..		
Makefile	fix cc0 header	5 months ago
allinfo.c	CTL_SYSCTL in sysctl.h	4 months ago
allinfobyname.c	CTL_SYSCTL in sysctl.h	4 months ago
capability.c	CTL_SYSCTL in sysctl.h	4 months ago
singleinfo.c	CTL_SYSCTL in sysctl.h	4 months ago

Implementation Note

- 1 function: sysctlinfo_interface()
- Nothing from kern_sysctl.c
  - Kernel module
  - Review: sys/sysctlinfo.h, kern_mib.c
- Lock: sysctl_wlock/sysctl_wunlock
  - Better: sysctl_rlock/sysctl_runlock
- *byname nodes almost implementation-free
- No capability:
  - sysctl.entryfakename
  - sysctl.entrynextleaf and sysctl.entrynextnode

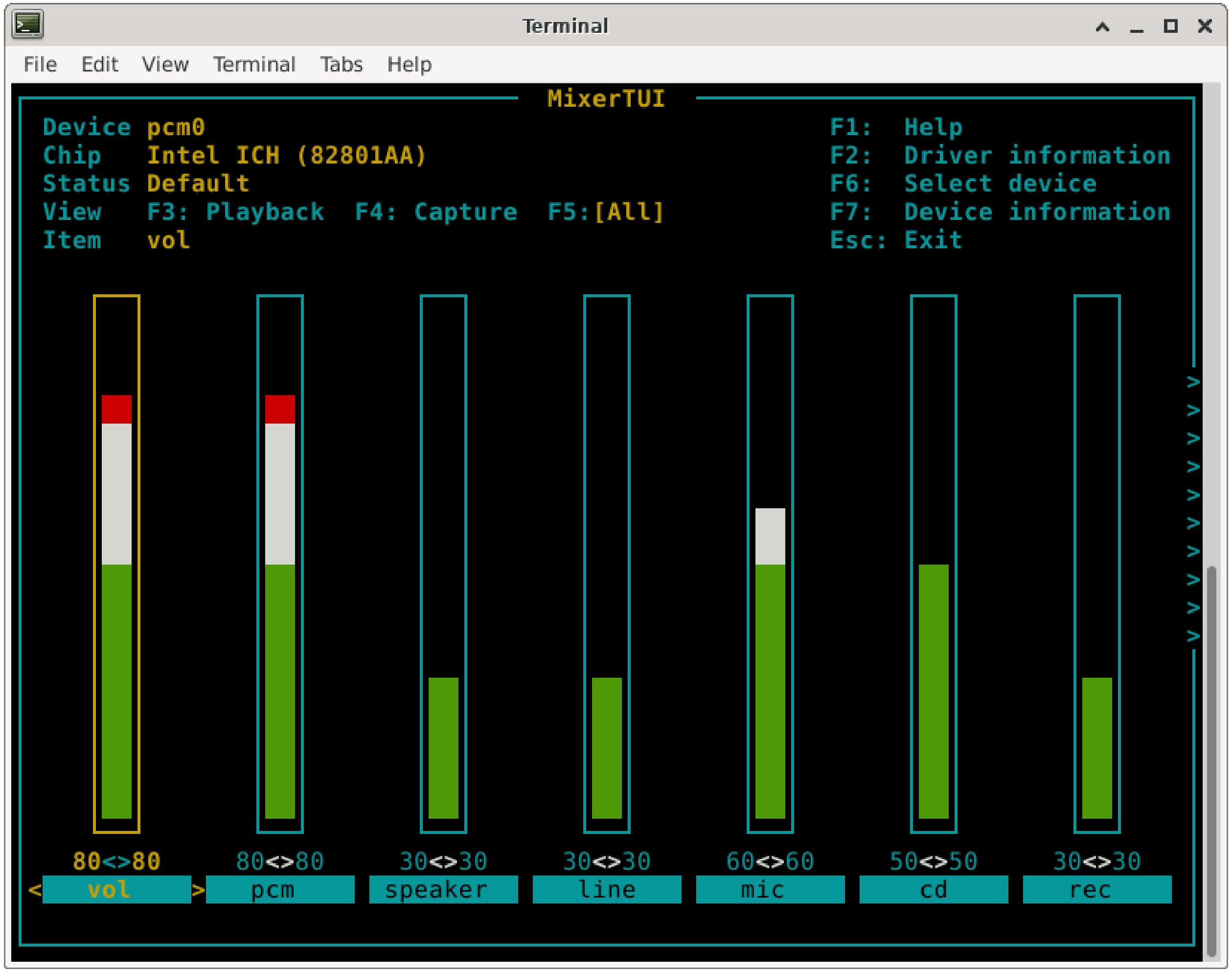
Real world use cases

- <sysutils/sysctlinfo-kmod>
- <sysutils/sysctlbyname-improved-kmod>

- <deskutils/sysctlview>
- <devel/libsysctlmibinfo2>
- <sysutils/nsysctl>
- <audio/mixertui>

Real world use case

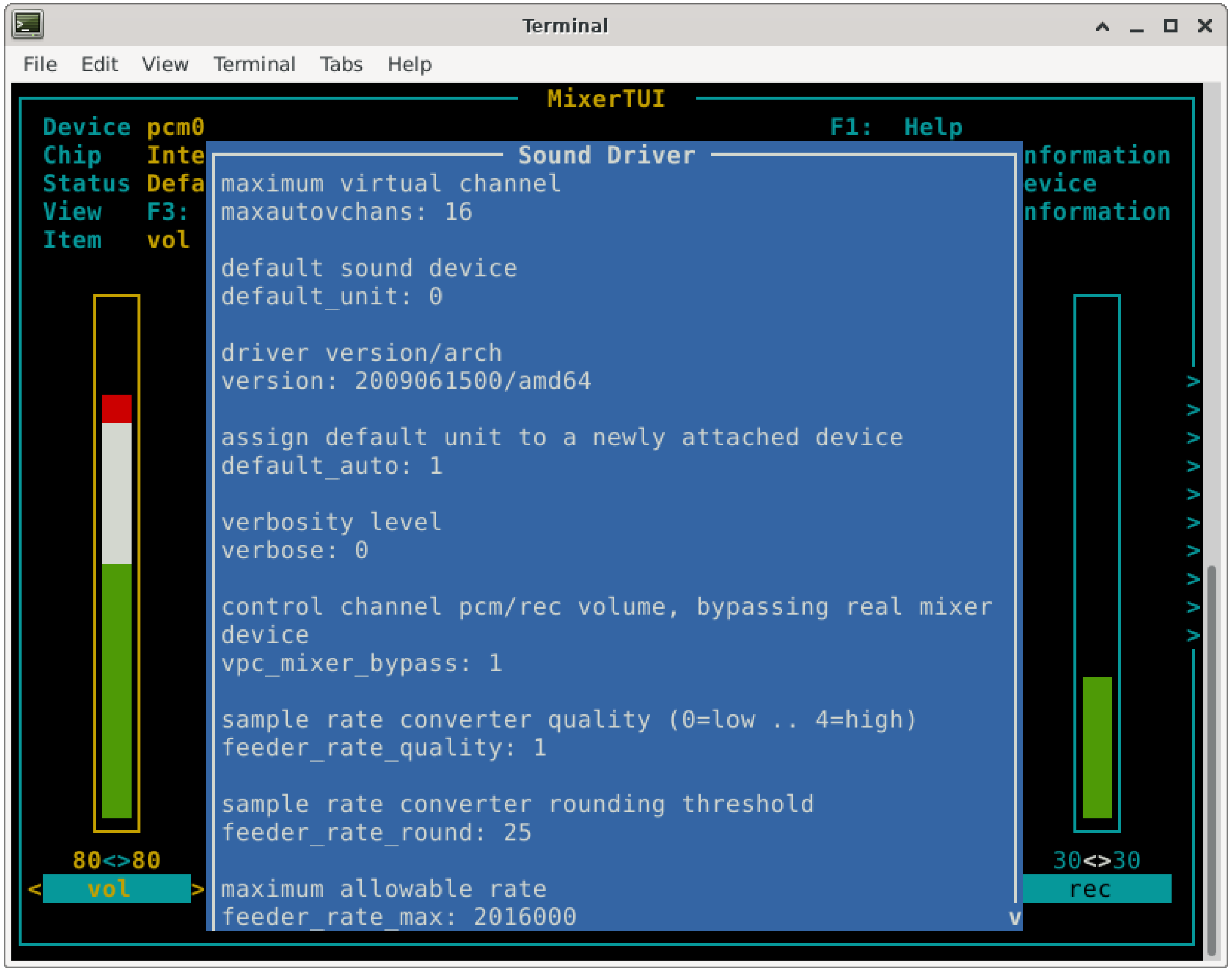
- mixertui



```
                              MixerTUI
 Device pcm0                                          F1:  Help
 Chip   Intel ICH (82801AA)                           F2:  Driver information
 Status Default                                       F6:  Select device
 View   F3: Playback  F4: Capture  F5:[All]           F7:  Device information
 Item   vol                                           Esc: Exit

  80<>80     80<>80     30<>30     30<>30     60<>60     50<>50     30<>30
<  vol   >    pcm      speaker      line       mic         cd         rec
```

Real world use case

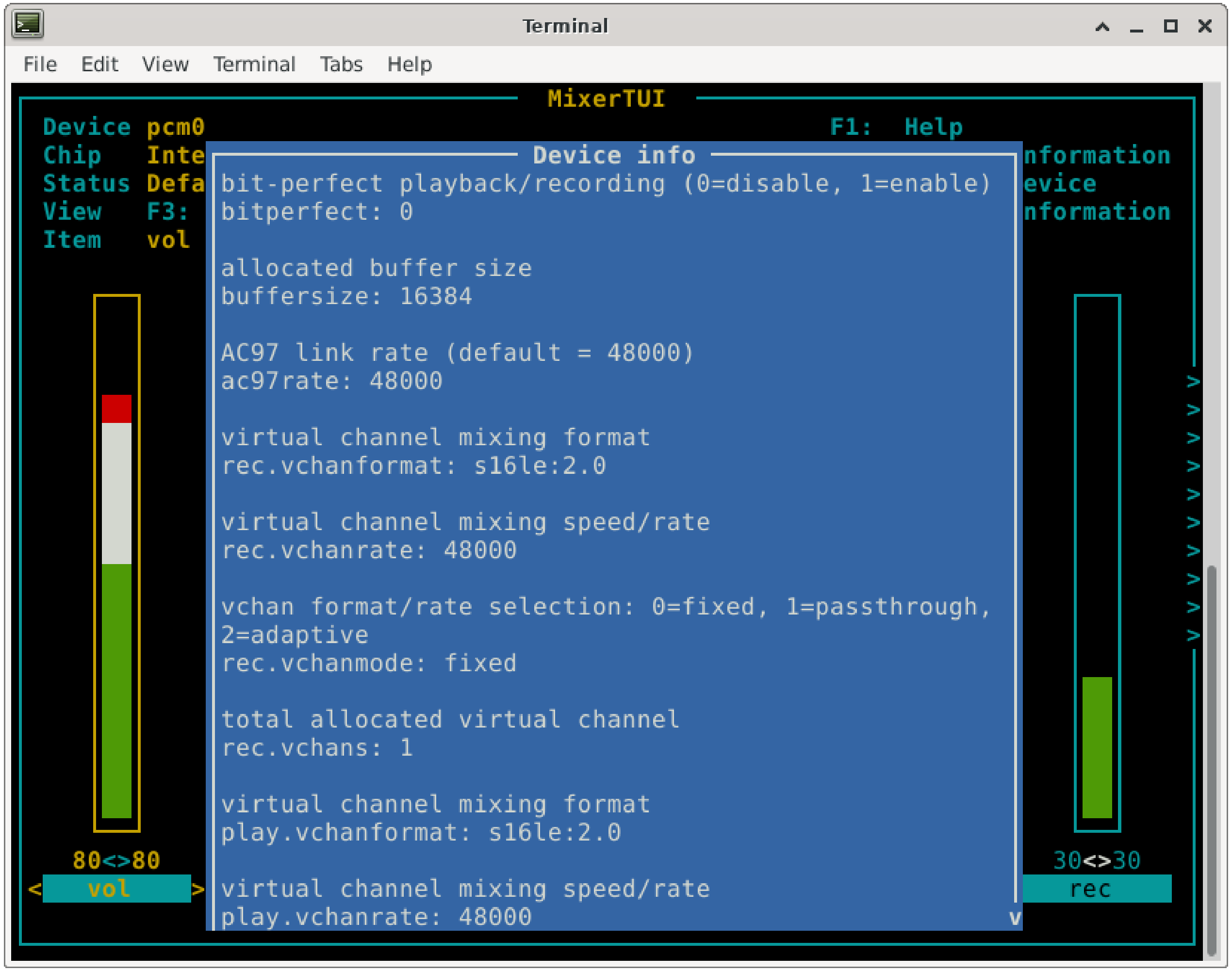
- mixertui



```
MixerTUI
Device pcm0                                   F1:  Help
Chip   Inte                                         nformation
Status Defa                                         evice
View   F3:                                          nformation
Item   vol

             Sound Driver
maximum virtual channel
maxautovchans: 16

default sound device
default_unit: 0

driver version/arch
version: 2009061500/amd64

assign default unit to a newly attached device
default_auto: 1

verbosity level
verbose: 0

control channel pcm/rec volume, bypassing real mixer
device
vpc_mixer_bypass: 1

sample rate converter quality (0=low .. 4=high)
feeder_rate_quality: 1

sample rate converter rounding threshold
feeder_rate_round: 25

maximum allowable rate
feeder_rate_max: 2016000

80<>80                                              30<>30
< vol >                                              rec
```

Real world use case

- mixertui



```
Device pcm0
Chip   Inte
Status Defa
View   F3:
Item   vol
```

```
Device info
bit-perfect playback/recording (0=disable, 1=enable)
bitperfect: 0

allocated buffer size
buffersize: 16384

AC97 link rate (default = 48000)
ac97rate: 48000

virtual channel mixing format
rec.vchanformat: s16le:2.0

virtual channel mixing speed/rate
rec.vchanrate: 48000

vchan format/rate selection: 0=fixed, 1=passthrough,
2=adaptive
rec.vchanmode: fixed

total allocated virtual channel
rec.vchans: 1

virtual channel mixing format
play.vchanformat: s16le:2.0

virtual channel mixing speed/rate
play.vchanrate: 48000
```

Real world use case

- nsysctl, sysctl clone
  - LibXo
  - Extra options

```
nsysctl [--libxo options [-r tagroot]] [-DdFGgIilmNpqTtWy]
        [-V | -v [h [b | o | x]]] [-e sep] [-B bufsize]
        [-f filename] name [=value[,value]] ...
nsysctl [--libxo options [-r tagroot]] [-DdFGgIlmNpqTtWy]
        [-V | -v [h [b | o | x]]] [-e sep] [-B bufsize] -A|-a|-X
```

Real world use case

```
%> nsysctl --libxo=xml,pretty -NldtFGv kern.features.compat_freebsd_32bit
<object>
  <name>kern.features.compat_freebsd_32bit</name>
  <label>feature</label>
  <description>Compatible with 32-bit FreeBSD</description>
  <type>integer</type>
  <format>I</format>
  <true-flags>
    <flag>RD</flag>
    <flag>MPSAFE</flag>
    <flag>CAPRD</flag>
  </true-flags>
  <value>1</value>
</object>
```

Real world use case

Debug:

```
%> nsysctl -aGImNt
```

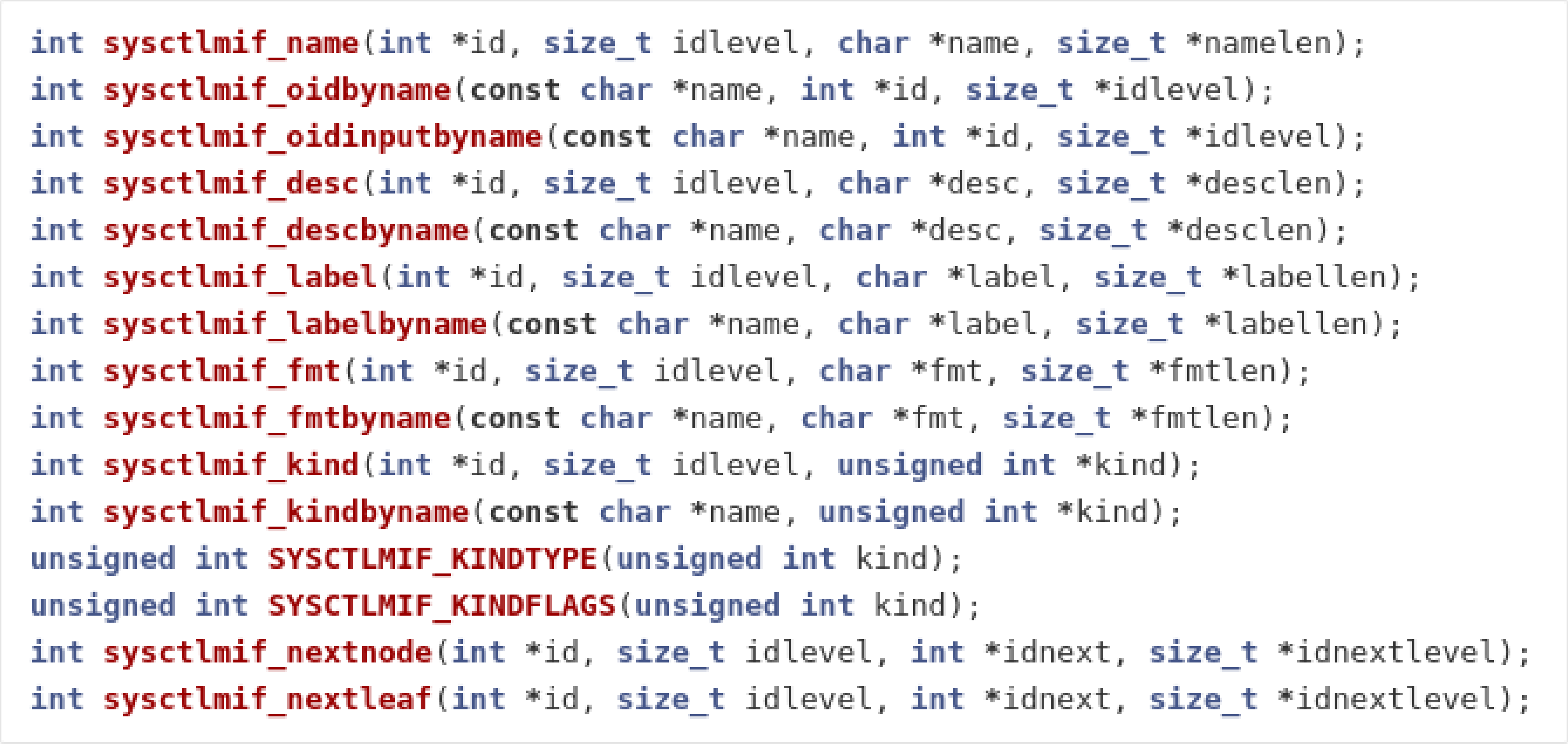
Avoid to recompile the kernel:

```
#ifdef SYSCTL_DEBUG
{0.0} sysctl.debug: printf the entire MIB
#endif
```

Real world use case

sysctlmibinfo2 library

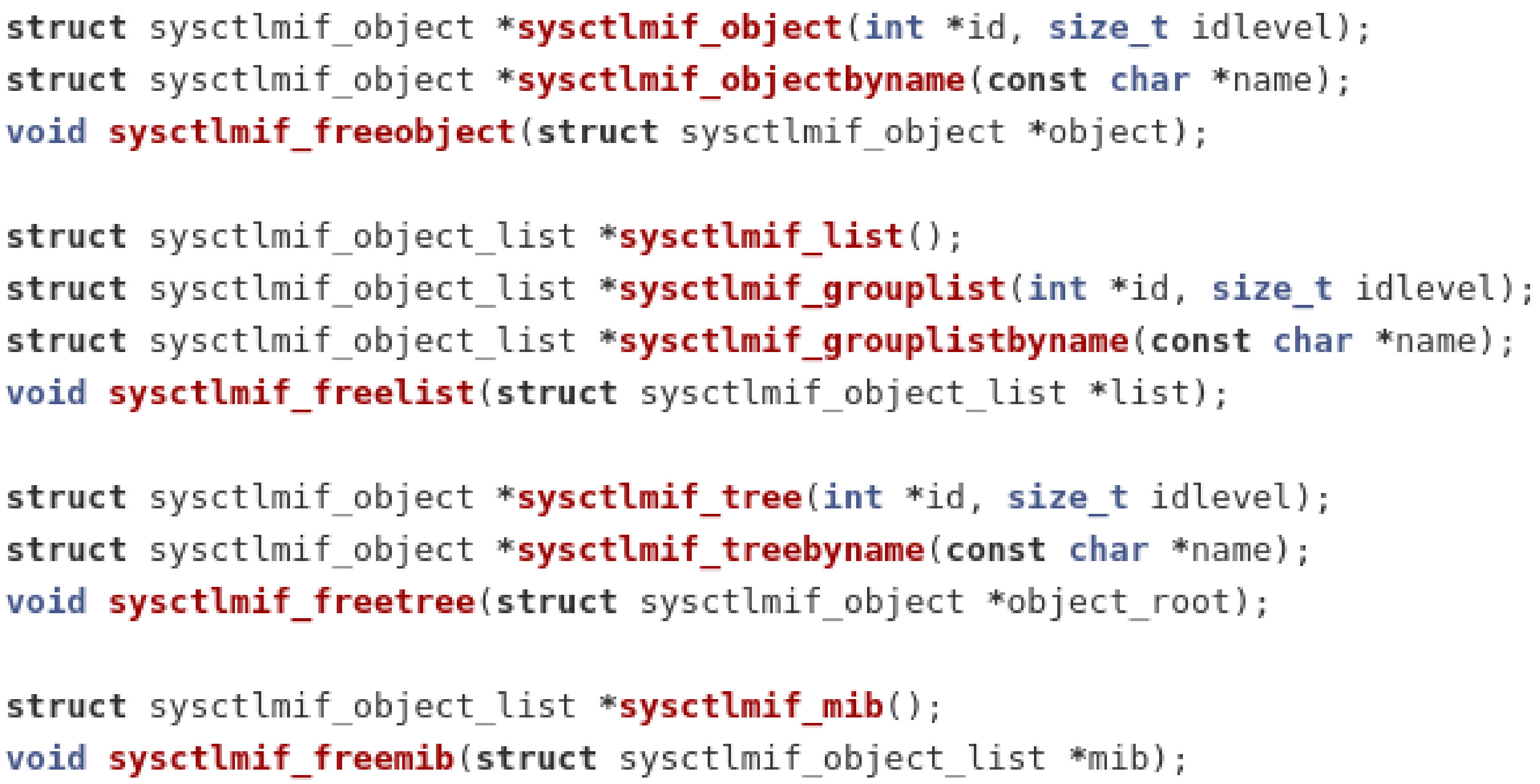
- wraps *sysctlinfo and sysctlbyname_improved*



```
int sysctlmif_name(int *id, size_t idlevel, char *name, size_t *namelen);
int sysctlmif_oidbyname(const char *name, int *id, size_t *idlevel);
int sysctlmif_oidinputbyname(const char *name, int *id, size_t *idlevel);
int sysctlmif_desc(int *id, size_t idlevel, char *desc, size_t *desclen);
int sysctlmif_descbyname(const char *name, char *desc, size_t *desclen);
int sysctlmif_label(int *id, size_t idlevel, char *label, size_t *labellen);
int sysctlmif_labelbyname(const char *name, char *label, size_t *labellen);
int sysctlmif_fmt(int *id, size_t idlevel, char *fmt, size_t *fmtlen);
int sysctlmif_fmtbyname(const char *name, char *fmt, size_t *fmtlen);
int sysctlmif_kind(int *id, size_t idlevel, unsigned int *kind);
int sysctlmif_kindbyname(const char *name, unsigned int *kind);
unsigned int SYSCTLMIF_KINDTYPE(unsigned int kind);
unsigned int SYSCTLMIF_KINDFLAGS(unsigned int kind);
int sysctlmif_nextnode(int *id, size_t idlevel, int *idnext, size_t *idnextlevel);
int sysctlmif_nextleaf(int *id, size_t idlevel, int *idnext, size_t *idnextlevel);
```

Real world use case

- sysctlmibinfo2 library
- high level API



```
struct sysctlmif_object *sysctlmif_object(int *id, size_t idlevel);
struct sysctlmif_object *sysctlmif_objectbyname(const char *name);
void sysctlmif_freeobject(struct sysctlmif_object *object);

struct sysctlmif_object_list *sysctlmif_list();
struct sysctlmif_object_list *sysctlmif_grouplist(int *id, size_t idlevel);
struct sysctlmif_object_list *sysctlmif_grouplistbyname(const char *name);
void sysctlmif_freelist(struct sysctlmif_object_list *list);

struct sysctlmif_object *sysctlmif_tree(int *id, size_t idlevel);
struct sysctlmif_object *sysctlmif_treebyname(const char *name);
void sysctlmif_freetree(struct sysctlmif_object *object_root);

struct sysctlmif_object_list *sysctlmif_mib();
void sysctlmif_freemib(struct sysctlmif_object_list *mib);
```

Links

- sysctinfo, code, README, examples, converted sysctl(8)
  - https://gitlab.com/alfix/sysctlinfo
- Review
  - review.freebsd.org/D21700
- sysctlbyname_improved
  - https://gitlab.com/alfix/sysctlbyname_improved

Thank you

- wiki.freebsd.org/AlfonsoSiciliano
- alfonso.siciliano@email.com
- Social
    - @alfsiciliano
    - @alfonsosiciliano