ufs_dirbad panic with mangled entries in ufs

FreeBSD's ufs usually does an excellent job in preventing file system corruption. But even the best system happens to mess up once in a while.

One thing you may eventually stumble accross are so called mangled entries, which are usually not fixable with fsck and result in kernel panics upon access.

Now these are usually a sign of severe file system corruption, often caused by hardware faults like bad memory modules, a faulty disk controller or even a deffective hard drive.

Consider checking and replacing your hardware if you encounter mangled entries on a frequent and recurring occasion.
You may actually succeed in fixing it following the steps outlined below, however it is very likely to happen again if you have faulty hardware. So in the end you'll end up curing the side-effects and not the actual reason, which may in turn lead to other, even more critical problems.

On the other hand, if you happen to have a corrupted file system like this very, very seldomly (as in "about once in a decade", it happened to myself only three times in 10 years that I've worked on some 200-300 servers in total) you may risk fixing it by means of the file system debugger.

I define this as a "minor corruption" to which the following usually applies:

- happens very, very seldomly
- happens as a result of a server deadlock/crash/power failure/etc
- limited to one or maybe two directory or file entries
- is not found by fsck eventually
- is not fixed by fsck
- causes the server to panic when accessing given file or directory

When the error happens
A typical error message thrown at you in this case may look like this (some output omitted):

```
/mnt/da1s1a: bad dir ino 16392 AT OFFSET 512: MANGLED ENTRY
panic: ufs_dirbad: bad dir
```

The message gives some essential information about the file system concerned (the actual mountpoint, not the device name itself) amd the inode of the directory or file.
First steps in recovery
So the next best thing to do in this situation is to reboot into single user mode.
From there have fsck inspect the device first.

```
# fsck -y /dev/da1s1a
** /dev/da1s1a
** Last Mounted on /mnt/da1s1a
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counters
** Phase 5 - Check Cyl groups
60040 files, 464657 used, 423186 free (43252 frags, 47493 blocks, 4.9% fragmentation)
```

```
***** FILE SYSTEM MARKED CLEAN *****
```

Now since mangled entries are usually not fixed by fsck, the term "FILE SYSTEM MARKED CLEAN" should not be trusted in.

You may risk to bring your system back up without any further work, however if it panics again with the same message (mind the inode number), you are likely to have unfixable (by means of fsck) corruption.
Optional but recommend: Try to crash the machine again
Personally I always try in crashing the system before I touch the file system with the debugger, however not without taking a current backup first if at all possible.

The reasons for doing so is simple:

- Trying to crash the machine will prove that the corruption still exists
- Finding the bad entry through manual search may indicate why and where it happened eventually
- It may reveal additional corruption
- It may proove that it cannot be fixed by fsck at all, no matter how often you run it

Finding the corrupted entry is easiest by walking the directory structure.

For this a simple command line like this usually works well enough. It should be run from single user mode and on the read-only mounted target device only to minimize all impacts.

```
# find / -type d -exec ls -ld {} ;
```

This will usually cause the system to panic again when accessing the corrupted directory.
If it does not, this method may:

```
# find / -type d -exec stat {} ;
```

If this still doesn't work, you may mount the device read-write so the afore mentioned commands can actually touch the file system to update file access times.

And if even that fails, try to create a dummy file inside each directory will do for sure:

```
# find / -type d -exec touch {}/mydummyfilenamewhichshouldnotexist ;
```

Now it must be noted that doing this on a already corrupted read-write file system _is_ dangerous.

I cannot stress this enough:

Don't take the risk if you don't have a backup!
Don't take the risk if you're not aware of the consequences!
Don't take the risk if you're a newbie!

A panic in this situation could make it even worse!
So, the system panics again...
Let's assume the system panics again with the same error message.

If you were lucky enough you even saw which directory was last accessed before the panic.
This may be valuable to know if you run some certain type of application and could reveal yet unknown application errors or even vulnerabilities like temporary file creation race conditions.

```
/mnt/da1s1a: bad dir ino 16392 AT OFFSET 512: MANGLED ENTRY
panic: ufs_dirbad: bad dir
```

So you now have proof that there is (still) an unfixed corruption on the file system.

You also have proof that it happened at the same inode than before.
If it's not the same inode, then you know for sure that there's either another corruption or faulty hardware which causes excessive errors.

For the latter case remember what I wrote before about faulty hardware.
Right, now how to fix it?
To fix it go back to single user mode and re-run fsck just to make sure. Keep your device mounted read-only.

Then start the file system debugger, fsdb:

```
# fsdb /dev/ad1s1a
** /dev/ad1s1a
Editing file system '/dev/ad1s1a'
Last mounted on /mnt/ad1s1a
[output omitted]
fsdb (inum: 2)>
```

Now go to the inode which was mentioned during kernel panic to get some additional information.

```
fsdb (inum: 2)> inode 16392
current inode: directory
I=16392 MODE=40755 SIZE=512
        BTIME=Oct 23 11:47:24 2006 [0 nsec]
        MTIME=Oct 23 11:47:24 2006 [0 nsec]
        CTIME=Oct 23 11:47:24 2006 [0 nsec]
        ATIME=Oct 23 11:47:24 2006 [0 nsec]
OWNER=root GRP=WHEEL LINKCNT=2 FLAGS=0 BLKCNT=4 GEN=157338b7
fsdb (inum: 16392)>
```

Even if it results in data loss, clearing the inode is the way to go to get rid of this.

```
fsdb (inum: 16392)> clri 16392
```

Then exit the debugger:

```
fsdb (inum: 16392)> quit

**** FILE SYSTEM STILL DIRTY *****
*** FILE SYSTEM MARKED DIRTY
*** BE SURE TO RUN FSDK TO CLEAN UP ANY DAMAGE
*** IF IT WAS MOUNTED, RE-MOUNT WITH -u -o reload
```

Run fsck as told:

```
** /dev/da1s1a
** Last Mounted on /mnt/da1s1a
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
UNALLOCATED  I=16392 OWNER=root MODE=0
SIZE=512 MTIME Oct 23 11:47:24 2006
NAME=/dsj????

REMOVE=YES
```

```
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counters
LINK COUNT DIR I=2  OWNER=root MODE=40755
SIZE=512 MTIME=Oct 23 11:47:24 2006  COUNT 21 SHOULD BE 20
ADJUST? yes

** Phase 5 - Check Cyl groups
FREE BLK COUNT(S) WRONG IN SUPERBLK
SALVAGE? yes

SUMMARY INFORMATION BAD
SAVLAGE? yes

BLK(S) MISSING IN BIT MAPS
SALVAGE? yes

60039 files, 464655 used, 423188 free (43248 frags, 47492 blocks, 4.9% fragmentation)

***** FILE SYSTEM MARKED CLEAN *****

***** FILE SYSTEM WAS MODIFIED *****
```

This is it?
Basically yes.

However I recommend rebooting the system once more into single user mode to rerun 'find'.
This will reveal if there is (no) further corruption. Also the reboot will ensure that the operating system can re-read the disklabel and file system properly. This is especially important after messing around with the file system debugger.
For this reason do run fsck once more just to make sure the file system is really clean.

Also try keeping to these premises:

- one corruption may happen once in a while and really mean nothing
- two is still possible but must be looked at critically
- three is a bad sign, there's usually more to come

Remember: The file system is at the heart of your server. Messing it up could compromise your data, your users and even your job. So care for it!