```
**********************************************************
  133737 Tue Oct 27 21:06:48 2015
new/usr/src/uts/common/fs/zfs/zfs_vnops.c
6334 Cannot unlink files when over quota
**********************************************************
______unchanged_portion_omitted_

1607 /*
1608  * Remove an entry from a directory.
1609  *
1610  *      IN:     dvp     - vnode of directory to remove entry from.
1611  *              name    - name of entry to remove.
1612  *              cr      - credentials of caller.
1613  *              ct      - caller context
1614  *              flags   - case flags
1615  *
1616  *      RETURN: 0 on success, error code on failure.
1617  *
1618  * Timestamps:
1619  *      dvp - ctime|mtime
1620  *       vp - ctime (if nlink > 0)
1621  */

1623 uint64_t null_xattr = 0;

1625 /*ARGSUSED*/
1626 static int
1627 zfs_remove(vnode_t *dvp, char *name, cred_t *cr, caller_context_t *ct,
1628     int flags)
1629 {
1630         znode_t         *zp, *dzp = VTOZ(dvp);
1631         znode_t         *xzp;
1632         vnode_t         *vp;
1633         zfsvfs_t        *zfsvfs = dzp->z_zfsvfs;
1634         zilog_t         *zilog;
1635         uint64_t        acl_obj, xattr_obj;
1636         uint64_t        xattr_obj_unlinked = 0;
1637         uint64_t        obj = 0;
1638         zfs_dirlock_t   *dl;
1639         dmu_tx_t        *tx;
1640         boolean_t       may_delete_now, delete_now = FALSE;
1641         boolean_t       unlinked, toobig = FALSE;
1642         uint64_t        txtype;
1643         pathname_t      *realnmp = NULL;
1644         pathname_t      realnm;
1645         int             error;
1646         int             zflg = ZEXISTS;
1647         boolean_t       waited = B_FALSE;

1649         ZFS_ENTER(zfsvfs);
1650         ZFS_VERIFY_ZP(dzp);
1651         zilog = zfsvfs->z_log;

1653         if (flags & FIGNORECASE) {
1654                 zflg |= ZCILOOK;
1655                 pn_alloc(&realnm);
1656                 realnmp = &realnm;
1657         }

1659 top:
1660         xattr_obj = 0;
1661         xzp = NULL;
1662         /*
1663          * Attempt to lock directory; fail if entry doesn't exist.
1664          */
1665         if (error = zfs_dirent_lock(&dl, dzp, name, &zp, zflg,
```



```
1666             NULL, realnmp)) {
1667                 if (realnmp)
1668                         pn_free(realnmp);
1669                 ZFS_EXIT(zfsvfs);
1670                 return (error);
1671         }

1673         vp = ZTOV(zp);

1675         if (error = zfs_zaccess_delete(dzp, zp, cr)) {
1676                 goto out;
1677         }

1679         /*
1680          * Need to use rmdir for removing directories.
1681          */
1682         if (vp->v_type == VDIR) {
1683                 error = SET_ERROR(EPERM);
1684                 goto out;
1685         }

1687         vnevent_remove(vp, dvp, name, ct);

1689         if (realnmp)
1690                 dnlc_remove(dvp, realnmp->pn_buf);
1691         else
1692                 dnlc_remove(dvp, name);

1694         mutex_enter(&vp->v_lock);
1695         may_delete_now = vp->v_count == 1 && !vn_has_cached_data(vp);
1696         mutex_exit(&vp->v_lock);

1698         /*
1699          * We may delete the znode now, or we may put it in the unlinked set;
1700          * it depends on whether we're the last link, and on whether there are
1701          * other holds on the vnode.  So we dmu_tx_hold() the right things to
1702          * allow for either case.
1703          */
1704         obj = zp->z_id;
1705         tx = dmu_tx_create(zfsvfs->z_os);
1706         dmu_tx_hold_zap(tx, dzp->z_id, FALSE, name);
1707         dmu_tx_hold_sa(tx, zp->z_sa_hdl, B_FALSE);
1708         zfs_sa_upgrade_txholds(tx, zp);
1709         zfs_sa_upgrade_txholds(tx, dzp);
1710         if (may_delete_now) {
1711                 toobig =
1712                     zp->z_size > zp->z_blksz * DMU_MAX_DELETEBLKCNT;
1713                 /* if the file is too big, only hold_free a token amount */
1714                 dmu_tx_hold_free(tx, zp->z_id, 0,
1715                     (toobig ? DMU_MAX_ACCESS : DMU_OBJECT_END));
1716         }

1718         /* are there any extended attributes? */
1719         error = sa_lookup(zp->z_sa_hdl, SA_ZPL_XATTR(zfsvfs),
1720             &xattr_obj, sizeof (xattr_obj));
1721         if (error == 0 && xattr_obj) {
1722                 error = zfs_zget(zfsvfs, xattr_obj, &xzp);
1723                 ASSERT0(error);
1724                 dmu_tx_hold_sa(tx, zp->z_sa_hdl, B_TRUE);
1725                 dmu_tx_hold_sa(tx, xzp->z_sa_hdl, B_FALSE);
1726         }

1728         mutex_enter(&zp->z_lock);
1729         if ((acl_obj = zfs_external_acl(zp)) != 0 && may_delete_now)
1730                 dmu_tx_hold_free(tx, acl_obj, 0, DMU_OBJECT_END);
1731         mutex_exit(&zp->z_lock);
```

```
1733         /* charge as an update -- would be nice not to charge at all */
1734         dmu_tx_hold_zap(tx, zfsvfs->z_unlinkedobj, FALSE, NULL);
1735
1736         /*
1737          * Mark this transaction as typically resulting in a net free of space
1738          * Mark this transaction as typically resulting in a net free of
1739          * space, unless object removal will be delayed indefinitely
1740          * (due to active holds on the vnode due to the file being open).
1738          */
1741         if (may_delete_now)
1739         dmu_tx_mark_netfree(tx);
1740
1741         error = dmu_tx_assign(tx, waited ? TXG_WAITED : TXG_NOWAIT);
1742         if (error) {
1743                 zfs_dirent_unlock(dl);
1744                 VN_RELE(vp);
1745                 if (xzp)
1746                         VN_RELE(ZTOV(xzp));
1747                 if (error == ERESTART) {
1748                         waited = B_TRUE;
1749                         dmu_tx_wait(tx);
1750                         dmu_tx_abort(tx);
1751                         goto top;
1752                 }
1753                 if (realnmp)
1754                         pn_free(realnmp);
1755                 dmu_tx_abort(tx);
1756                 ZFS_EXIT(zfsvfs);
1757                 return (error);
1758         }
1759
1760         /*
1761          * Remove the directory entry.
1762          */
1763         error = zfs_link_destroy(dl, zp, tx, zflg, &unlinked);
1764
1765         if (error) {
1766                 dmu_tx_commit(tx);
1767                 goto out;
1768         }
1769
1770         if (unlinked) {
1771                 /*
1772                  * Hold z_lock so that we can make sure that the ACL obj
1773                  * hasn't changed.  Could have been deleted due to
1774                  * zfs_sa_upgrade().
1775                  */
1776                 mutex_enter(&zp->z_lock);
1777                 mutex_enter(&vp->v_lock);
1778                 (void) sa_lookup(zp->z_sa_hdl, SA_ZPL_XATTR(zfsvfs),
1779                     &xattr_obj_unlinked, sizeof (xattr_obj_unlinked));
1780                 delete_now = may_delete_now && !toobig &&
1781                     vp->v_count == 1 && !vn_has_cached_data(vp) &&
1782                     xattr_obj == xattr_obj_unlinked && zfs_external_acl(zp) ==
1783                     acl_obj;
1784                 mutex_exit(&vp->v_lock);
1785         }
1786
1787         if (delete_now) {
1788                 if (xattr_obj_unlinked) {
1789                         ASSERT3U(xzp->z_links, ==, 2);
1790                         mutex_enter(&xzp->z_lock);
1791                         xzp->z_unlinked = 1;
1792                         xzp->z_links = 0;
1793                         error = sa_update(xzp->z_sa_hdl, SA_ZPL_LINKS(zfsvfs),
```



```
1794                             &xzp->z_links, sizeof (xzp->z_links), tx);
1795                         ASSERT3U(error,  ==,  0);
1796                         mutex_exit(&xzp->z_lock);
1797                         zfs_unlinked_add(xzp, tx);
1798
1799                         if (zp->z_is_sa)
1800                                 error = sa_remove(zp->z_sa_hdl,
1801                                     SA_ZPL_XATTR(zfsvfs), tx);
1802                         else
1803                                 error = sa_update(zp->z_sa_hdl,
1804                                     SA_ZPL_XATTR(zfsvfs), &null_xattr,
1805                                     sizeof (uint64_t), tx);
1806                         ASSERT0(error);
1807                 }
1808                 mutex_enter(&vp->v_lock);
1809                 vp->v_count--;
1810                 ASSERT0(vp->v_count);
1811                 mutex_exit(&vp->v_lock);
1812                 mutex_exit(&zp->z_lock);
1813                 zfs_znode_delete(zp, tx);
1814         } else if (unlinked) {
1815                 mutex_exit(&zp->z_lock);
1816                 zfs_unlinked_add(zp, tx);
1817         }
1818
1819         txtype = TX_REMOVE;
1820         if (flags & FIGNORECASE)
1821                 txtype |= TX_CI;
1822         zfs_log_remove(zilog, tx, txtype, dzp, name, obj);
1823
1824         dmu_tx_commit(tx);
1825 out:
1826         if (realnmp)
1827                 pn_free(realnmp);
1828
1829         zfs_dirent_unlock(dl);
1830
1831         if (!delete_now)
1832                 VN_RELE(vp);
1833         if (xzp)
1834                 VN_RELE(ZTOV(xzp));
1835
1836         if (zfsvfs->z_os->os_sync == ZFS_SYNC_ALWAYS)
1837                 zil_commit(zilog, 0);
1838
1839         ZFS_EXIT(zfsvfs);
1840         return (error);
1841 }
```

*unchanged_portion_omitted_*