```
**********************************************************
   41455 Tue Feb 25 17:31:19 2014
new/usr/src/lib/libzfs/common/libzfs_import.c
4626 libzfs memleak in zpool_in_use()
Reviewed by: Tony Nguyen <tony.nguyen@nexenta.com>
Reviewed by: Saso Kiselkov <sasa.kiselkov@nexenta.com>
**********************************************************
______unchanged_portion_omitted_

1521 /*
1522  * Determines if the pool is in use.  If so, it returns true and the state of
1523  * the pool as well as the name of the pool.  Both strings are allocated and
1524  * must be freed by the caller.
1525  */
1526 int
1527 zpool_in_use(libzfs_handle_t *hdl, int fd, pool_state_t *state, char **namestr,
1528     boolean_t *inuse)
1529 {
1530         nvlist_t *config;
1531         char *name;
1532         boolean_t ret;
1533         uint64_t guid, vdev_guid;
1534         zpool_handle_t *zhp;
1535         nvlist_t *pool_config;
1536         uint64_t stateval, isspare;
1537         aux_cbdata_t cb = { 0 };
1538         boolean_t isactive;

1540         *inuse = B_FALSE;

1542         if (zpool_read_label(fd, &config) != 0) {
1543                 (void) no_memory(hdl);
1544                 return (-1);
1545         }

1547         if (config == NULL)
1548                 return (0);

1550         verify(nvlist_lookup_uint64(config, ZPOOL_CONFIG_POOL_STATE,
1551             &stateval) == 0);
1552         verify(nvlist_lookup_uint64(config, ZPOOL_CONFIG_GUID,
1553             &vdev_guid) == 0);

1555         if (stateval != POOL_STATE_SPARE && stateval != POOL_STATE_L2CACHE) {
1556                 verify(nvlist_lookup_string(config, ZPOOL_CONFIG_POOL_NAME,
1557                     &name) == 0);
1558                 verify(nvlist_lookup_uint64(config, ZPOOL_CONFIG_POOL_GUID,
1559                     &guid) == 0);
1560         }

1562         switch (stateval) {
1563         case POOL_STATE_EXPORTED:
1564                 /*
1565                  * A pool with an exported state may in fact be imported
1566                  * read-only, so check the in-core state to see if it's
1567                  * active and imported read-only.  If it is, set
1568                  * its state to active.
1569                  */
1570                 if (pool_active(hdl, name, guid, &isactive) == 0 && isactive &&
1571                     (zhp = zpool_open_canfail(hdl, name)) != NULL) {
1572                         if (zpool_get_prop_int(zhp, ZPOOL_PROP_READONLY, NULL))
1571                     (zhp = zpool_open_canfail(hdl, name)) != NULL &&
1572                     zpool_get_prop_int(zhp, ZPOOL_PROP_READONLY, NULL))
1573                                 stateval = POOL_STATE_ACTIVE;

1575                         zpool_close(zhp);
```



```
1576                 }

1578 #endif /* ! codereview */
1579                 ret = B_TRUE;
1580                 break;

1582         case POOL_STATE_ACTIVE:
1583                 /*
1584                  * For an active pool, we have to determine if it's really part
1585                  * of a currently active pool (in which case the pool will exist
1586                  * and the guid will be the same), or whether it's part of an
1587                  * active pool that was disconnected without being explicitly
1588                  * exported.
1589                  */
1590                 if (pool_active(hdl, name, guid, &isactive) != 0) {
1591                         nvlist_free(config);
1592                         return (-1);
1593                 }

1595                 if (isactive) {
1596                         /*
1597                          * Because the device may have been removed while
1598                          * offlined, we only report it as active if the vdev is
1599                          * still present in the config.  Otherwise, pretend like
1600                          * it's not in use.
1601                          */
1602                         if ((zhp = zpool_open_canfail(hdl, name)) != NULL &&
1603                             (pool_config = zpool_get_config(zhp, NULL))
1604                             != NULL) {
1605                                 nvlist_t *nvroot;

1607                                 verify(nvlist_lookup_nvlist(pool_config,
1608                                     ZPOOL_CONFIG_VDEV_TREE, &nvroot) == 0);
1609                                 ret = find_guid(nvroot, vdev_guid);
1610                         } else {
1611                                 ret = B_FALSE;
1612                         }

1614                         /*
1615                          * If this is an active spare within another pool, we
1616                          * treat it like an unused hot spare.  This allows the
1617                          * user to create a pool with a hot spare that currently
1618                          * in use within another pool.  Since we return B_TRUE,
1619                          * libdiskmgt will continue to prevent generic consumers
1620                          * from using the device.
1621                          */
1622                         if (ret && nvlist_lookup_uint64(config,
1623                             ZPOOL_CONFIG_IS_SPARE, &isspare) == 0 && isspare)
1624                                 stateval = POOL_STATE_SPARE;

1626                         if (zhp != NULL)
1627                                 zpool_close(zhp);
1628                 } else {
1629                         stateval = POOL_STATE_POTENTIALLY_ACTIVE;
1630                         ret = B_TRUE;
1631                 }
1632                 break;

1634         case POOL_STATE_SPARE:
1635                 /*
1636                  * For a hot spare, it can be either definitively in use, or
1637                  * potentially active.  To determine if it's in use, we iterate
1638                  * over all pools in the system and search for one with a spare
1639                  * with a matching guid.
1640                  *
1641                  * Due to the shared nature of spares, we don't actually report
```

```
1642                  * the potentially active case as in use.  This means the user
1643                  * can freely create pools on the hot spares of exported pools,
1644                  * but to do otherwise makes the resulting code complicated, and
1645                  * we end up having to deal with this case anyway.
1646                  */
1647                 cb.cb_zhp = NULL;
1648                 cb.cb_guid = vdev_guid;
1649                 cb.cb_type = ZPOOL_CONFIG_SPARES;
1650                 if (zpool_iter(hdl, find_aux, &cb) == 1) {
1651                         name = (char *)zpool_get_name(cb.cb_zhp);
1652                         ret = TRUE;
1653                 } else {
1654                         ret = FALSE;
1655                 }
1656                 break;

1658         case POOL_STATE_L2CACHE:

1660                 /*
1661                  * Check if any pool is currently using this l2cache device.
1662                  */
1663                 cb.cb_zhp = NULL;
1664                 cb.cb_guid = vdev_guid;
1665                 cb.cb_type = ZPOOL_CONFIG_L2CACHE;
1666                 if (zpool_iter(hdl, find_aux, &cb) == 1) {
1667                         name = (char *)zpool_get_name(cb.cb_zhp);
1668                         ret = TRUE;
1669                 } else {
1670                         ret = FALSE;
1671                 }
1672                 break;

1674         default:
1675                 ret = B_FALSE;
1676         }


1679         if (ret) {
1680                 if ((*namestr = zfs_strdup(hdl, name)) == NULL) {
1681                         if (cb.cb_zhp)
1682                                 zpool_close(cb.cb_zhp);
1683                         nvlist_free(config);
1684                         return (-1);
1685                 }
1686                 *state = (pool_state_t)stateval;
1687         }

1689         if (cb.cb_zhp)
1690                 zpool_close(cb.cb_zhp);

1692         nvlist_free(config);
1693         *inuse = ret;
1694         return (0);
1695 }
```