

```
*****
1338 Fri Aug 2 17:18:11 2013
new/usr/src/grub/capability
3966 zfs lz4 compression (etc) should have bumped grub capability VERSION
*****
```

1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 # Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
22 # Copyright (c) 2012 by Delphix. All rights reserved.
23 #
24 # This file defines the current capabilities of GRUB over and above that
25 # supported by the standard distribution
26 #
27 # The version field contains the version of the associated GRUB software. The
28 # version is incremented by 1 each time there is a bugfix or enhancement to
29 # GRUB necessitating that the boot blocks be reinstalled for that fix or
30 # enhancement to take effect.
31 # The version field contains the version of the associated GRUB software.
32 # The version is incremented by .1 (minor version number) each time there
33 # is a bugfix or enhancement of GRUB. In addition, the major version number
34 # is bumped up by 1 every time a release boundary is crossed. Thus if in S11
35 # the starting version is 3, in S12 the starting version will be 4.
36 # Note that the first major number in each sequence is a whole integer
37 # i.e. 2.0 is truncated to 2 and 3.0 is truncated to 3.
38 #
39 # VERSION=22
40 # NOTE: Live Upgrade is currently unable to handle decimal fractions (i.e.
41 # minor version numbers) so the version number is being bumped up in
42 # integer increments until Live Upgrade is fixed.
43 #
44 # This file and the associated version are Solaris specific and are
45 # not a part of the open source distribution of GRUB.
46 #
47 # VERSION=21
48 dboot
49 xvm
50 zfs
51 findroot

new/usr/src/grub/grub-0.97/stage2/fsys\_zfs.c

1

```
*****
43642 Fri Aug 2 17:18:11 2013
new/usr/src/grub/grub-0.97/stage2/fsys_zfs.c
3966 zfs lz4 compression (etc) should have bumped grub capability VERSION
*****
_____unchanged_portion_omitted_____
959 /*
960 * List of pool features that the grub implementation of ZFS supports for
961 * read. Note that features that are only required for write do not need
962 * to be listed here since grub opens pools in read-only mode.
963 */
964 * When this list is updated the version number in usr/src/grub/capability
965 * must be incremented to ensure the new grub gets installed.
966 #endif /* ! codereview */
967 */
968 static const char *spa_feature_names[] = {
969     "org.illumos:lz4_compress",
970     NULL
971 };
972 */
973 * Checks whether the MOS features that are active are supported by this
974 * (GRUB's) implementation of ZFS.
975 *
976 */
977 * Return:
978 *     0: Success.
979 *     errnum: Failure.
980 */
981 static int
982 check_mos_features(dnode_phys_t *mosmdn, char *stack)
983 {
984     uint64_t objnum;
985     dnode_phys_t *dn;
986     uint8_t error = 0;
987
988     dn = (dnode_phys_t *)stack;
989     stack += DNODE_SIZE;
990
991     if ((errnum = dnode_get(mosmdn, DMU_POOL_DIRECTORY_OBJECT,
992         DMU_OT_OBJECT_DIRECTORY, dn, stack)) != 0)
993         return (errnum);
994
995     /*
996      * Find the object number for 'features_for_read' and retrieve its
997      * corresponding dnode. Note that we don't check features_for_write
998      * because GRUB is not opening the pool for write.
999     */
1000    if ((errnum = zap_lookup(dn, DMU_POOL_FEATURES_FOR_READ, &objnum,
1001        stack)) != 0)
1002        return (errnum);
1003
1004    if ((errnum = dnode_get(mosmdn, objnum, DMU_OTN_ZAP_METADATA,
1005        dn, stack)) != 0)
1006        return (errnum);
1007
1008    return (zap_iterate(dn, check_feature, spa_feature_names, stack));
1009 }
1010 */
1011 * Given a MOS metadnode, get the metadnode of a given filesystem name (fsname),
1012 * e.g. pool/rootfs, or a given object number (obj), e.g. the object number
1013 * of pool/rootfs.
1014 *
1015 */
1016 * If no fsname and no obj are given, return the DSL_DIR metadnode.
1017 * If fsname is given, return its metadnode and its matching object number.
```

new/usr/src/grub/grub-0.97/stage2/fsys\_zfs.c

2

```
1018     * If only obj is given, return the metadnode for this object number.
1019     *
1020     * Return:
1021     *     0 - success
1022     *     errnum - failure
1023     */
1024 static int
1025 get_objset_mdn(dnode_phys_t *mosmdn, char *fsname, uint64_t *obj,
1026                 dnode_phys_t *mdn, char *stack)
1027 {
1028     uint64_t objnum, headobj;
1029     char *cname, ch;
1030     blkptr_t *bp;
1031     objset_phys_t *osp;
1032     int issnapshot = 0;
1033     char *snapname;
1034
1035     if (fsname == NULL && obj) {
1036         headobj = *obj;
1037         goto skip;
1038     }
1039
1040     if (errnum = dnode_get(mosmdn, DMU_POOL_DIRECTORY_OBJECT,
1041         DMU_OT_OBJECT_DIRECTORY, mdn, stack))
1042         return (errnum);
1043
1044     if (errnum = zap_lookup(mdn, DMU_POOL_ROOT_DATASET, &objnum,
1045         stack))
1046         return (errnum);
1047
1048     if (errnum = dnode_get(mosmdn, objnum, DMU_OT_DSL_DIR, mdn, stack))
1049         return (errnum);
1050
1051     if (fsname == NULL) {
1052         headobj =
1053             ((dsl_dir_phys_t *)DN_BONUS(mdn))->dd_head_dataset_obj;
1054         goto skip;
1055     }
1056
1057     /* take out the pool name */
1058     while (*fsname && !grub_isspace(*fsname) && *fsname != '/')
1059         fsname++;
1060
1061     while (*fsname && !grub_isspace(*fsname)) {
1062         uint64_t childobj;
1063
1064         while (*fsname == '/')
1065             fsname++;
1066
1067         cname = fsname;
1068         while (*fsname && !grub_isspace(*fsname) && *fsname != '/')
1069             fsname++;
1070         ch = *fsname;
1071         *fsname = 0;
1072
1073         snapname = cname;
1074         while (*snapname && !grub_isspace(*snapname) && *snapname !=
1075             '@')
1076             snapname++;
1077         if (*snapname == '@') {
1078             issnapshot = 1;
1079             *snapname = 0;
1080         }
1081         childobj =
1082             ((dsl_dir_phys_t *)DN_BONUS(mdn))->dd_child_dir_zapobj;
1083         if (errnum = dnode_get(mosmdn, childobj,
```

```

1084         DMU_OT_DSL_DIR_CHILD_MAP, mdn, stack))
1085         return (errnum);
1086
1087     if (zap_lookup(mdn, cname, &objnum, stack))
1088         return (ERR_FILESYSTEM_NOT_FOUND);
1089
1090     if (errnum = dnode_get(mosmdn, objnum, DMU_OT_DSL_DIR,
1091                           mdn, stack))
1092         return (errnum);
1093
1094     *fsname = ch;
1095     if (issnapshot)
1096         *snapname = '@';
1097
1098     headobj = ((dsl_dir_phys_t *)DN_BONUS(mdn))->dd_head_dataset_obj;
1099     if (obj)
1100         *obj = headobj;
1101
1102 skip:
1103     if (errnum = dnode_get(mosmdn, headobj, DMU_OT_DSL_DATASET, mdn, stack))
1104         return (errnum);
1105     if (issnapshot) {
1106         uint64_t snapobj;
1107
1108         snapobj = ((dsl_dataset_phys_t *)DN_BONUS(mdn))->
1109                     ds_snapnames_zapobj;
1110
1111         if (errnum = dnode_get(mosmdn, snapobj,
1112                               DMU_OT_DSL_DS_SNAP_MAP, mdn, stack))
1113             return (errnum);
1114         if (zap_lookup(mdn, snapname + 1, &headobj, stack))
1115             return (ERR_FILESYSTEM_NOT_FOUND);
1116         if (errnum = dnode_get(mosmdn, headobj,
1117                               DMU_OT_DSL_DATASET, mdn, stack))
1118             return (errnum);
1119         if (obj)
1120             *obj = headobj;
1121     }
1122
1123     bp = &((dsl_dataset_phys_t *)DN_BONUS(mdn))->ds_bp;
1124     osp = (objset_phys_t *)stack;
1125     stack += sizeof (objset_phys_t);
1126     if (errnum = zio_read(bp, osp, stack))
1127         return (errnum);
1128
1129     grub_memmove((char *)mdn, (char *)&osp->os_meta_dnode, DNODE_SIZE);
1130
1131     return (0);
1132 }
1133
1134 */
1135 * For a given XDR packed nvlist, verify the first 4 bytes and move on.
1136 *
1137 * An XDR packed nvlist is encoded as (comments from nvs_xdr_create) :
1138 *
1139 *   encoding method/host endian      (4 bytes)
1140 *   nvl_version                    (4 bytes)
1141 *   nvl_nvflag                     (4 bytes)
1142 *
1143 *   encoded nvpairs:
1144 *     encoded size of the nvpair    (4 bytes)
1145 *     decoded size of the nvpair    (4 bytes)
1146 *     name string size            (4 bytes)
1147 *     name string data            (sizeof(NV_ALIGN4(string)))
1148 *     data type                   (4 bytes)
1149 *     # of elements in the nvpair  (4 bytes)
1150 *     data

```

```

1150     *      2 zero's for the last nvpair
1151     *          (end of the entire list)           (8 bytes)
1152     *
1153     * Return:
1154     *     0 - success
1155     *     1 - failure
1156     */
1157 static int
1158 nvlist_unpack(char *nvlist, char **out)
1159 {
1160     /* Verify if the 1st and 2nd byte in the nvlist are valid. */
1161     if (nvlist[0] != NV_ENCODE_XDR || nvlist[1] != HOST_ENDIAN)
1162         return (1);
1163
1164     *out = nvlist + 4;
1165     return (0);
1166 }
1167
1168 static char *
1169 nvlist_array(char *nvlist, int index)
1170 {
1171     int i, encode_size;
1172
1173     for (i = 0; i < index; i++) {
1174         /* skip the header, nvl_version, and nvl_nvflag */
1175         nvlist = nvlist + 4 * 2;
1176
1177         while (encode_size = BSWAP_32(*(uint32_t *)nvlist))
1178             nvlist += encode_size; /* goto the next nvpair */
1179
1180         nvlist = nvlist + 4 * 2; /* skip the ending 2 zeros - 8 bytes */
1181     }
1182
1183     return (nvlist);
1184 }
1185
1186 /*
1187 * The nvlist_next_nvpair() function returns a handle to the next nvpair in the
1188 * list following nvpair. If nvpair is NULL, the first pair is returned. If
1189 * nvpair is the last pair in the nvlist, NULL is returned.
1190 */
1191 static char *
1192 nvlist_next_nvpair(char *nvl, char *nvpair)
1193 {
1194     char *cur, *prev;
1195     int encode_size;
1196
1197     if (nvl == NULL)
1198         return (NULL);
1199
1200     if (nvpair == NULL) {
1201         /* skip over nvl_version and nvl_nvflag */
1202         nvpair = nvl + 4 * 2;
1203     } else {
1204         /* skip to the next nvpair */
1205         encode_size = BSWAP_32(*(uint32_t *)nvpair);
1206         nvpair += encode_size;
1207     }
1208
1209     /* 8 bytes of 0 marks the end of the list */
1210     if ((*uint64_t *)nvpair == 0)
1211         return (NULL);
1212
1213     return (nvpair);
1214 }

```

```

1216 /*
1217  * This function returns 0 on success and 1 on failure. On success, a string
1218  * containing the name of nvpair is saved in buf.
1219  */
1220 static int
1221 nvpair_name(char *nvp, char *buf, int buflen)
1222 {
1223     int len;
1224
1225     /* skip over encode/decode size */
1226     nvp += 4 * 2;
1227
1228     len = BSWAP_32(*(uint32_t *)nvp);
1229     if (buflen < len + 1)
1230         return (1);
1231
1232     grub_memmove(buf, nvp + 4, len);
1233     buf[len] = '\0';
1234
1235     return (0);
1236 }
1237
1238 /*
1239  * This function retrieves the value of the nvpair in the form of enumerated
1240  * type data_type_t. This is used to determine the appropriate type to pass to
1241  * nvpair_value().
1242 */
1243 static int
1244 nvpair_type(char *nvp)
1245 {
1246     int name_len, type;
1247
1248     /* skip over encode/decode size */
1249     nvp += 4 * 2;
1250
1251     /* skip over name_len */
1252     name_len = BSWAP_32(*uint32_t *)nvp;
1253     nvp += 4;
1254
1255     /* skip over name */
1256     nvp = nvp + ((name_len + 3) & ~3); /* align */
1257
1258     type = BSWAP_32(*uint32_t *)nvp;
1259
1260     return (type);
1261 }
1262
1263 static int
1264 nvpair_value(char *nvp, void *val, int valtype, int *nelmp)
1265 {
1266     int name_len, type, slen;
1267     char *strval = val;
1268     uint64_t *intval = val;
1269
1270     /* skip over encode/decode size */
1271     nvp += 4 * 2;
1272
1273     /* skip over name_len */
1274     name_len = BSWAP_32(*uint32_t *)nvp;
1275     nvp += 4;
1276
1277     /* skip over name */
1278     nvp = nvp + ((name_len + 3) & ~3); /* align */
1279
1280     /* skip over type */
1281     type = BSWAP_32(*uint32_t *)nvp;

```

```

1282     nvp += 4;
1283
1284     if (type == valtype) {
1285         int nelm;
1286
1287         nelm = BSWAP_32(*uint32_t *)nvp;
1288         if (valtype != DATA_TYPE_BOOLEAN && nelm < 1)
1289             return (1);
1290         nvp += 4;
1291
1292         switch (valtype) {
1293             case DATA_TYPE_BOOLEAN:
1294                 return (0);
1295
1296             case DATA_TYPE_STRING:
1297                 slen = BSWAP_32(*uint32_t *)nvp;
1298                 nvp += 4;
1299                 grub_memmove(strval, nvp, slen);
1300                 strval[slen] = '\0';
1301                 return (0);
1302
1303             case DATA_TYPE_UINT64:
1304                 *intval = BSWAP_64(*uint64_t *)nvp;
1305                 return (0);
1306
1307             case DATA_TYPE_NVLIST:
1308                 *(void **)val = (void *)nvp;
1309                 return (0);
1310
1311             case DATA_TYPE_NVLIST_ARRAY:
1312                 *(void **)val = (void *)nvp;
1313                 if (nelmp)
1314                     *nelmp = nelm;
1315                 return (0);
1316         }
1317     }
1318
1319     return (1);
1320 }
1321
1322 static int
1323 nvlist_lookup_value(char *nvlist, char *name, void *val, int valtype,
1324                      int *nelmp)
1325 {
1326     char *nvpair;
1327
1328     for (nvpair = nvlist_next_nvpair(nvlist, NULL);
1329          nvpair != NULL;
1330          nvpair = nvlist_next_nvpair(nvlist, nvpair)) {
1331         int name_len = BSWAP_32(*uint32_t *)(nvpair + 4 * 2);
1332         char *nvp_name = nvpair + 4 * 3;
1333
1334         if ((grub_strncmp(nvp_name, name, name_len) == 0) &&
1335             nvpair_type(nvpair) == valtype) {
1336             return (nvpair_value(nvpair, val, valtype, nelmp));
1337         }
1338     }
1339
1340     return (1);
1341
1342 /*
1343  * Check if this vdev is online and is in a good state.
1344 */
1345 static int
1346 vdev_validate(char *nv)
1347 {

```

```

1348     uint64_t ival;
1349
1350     if (nvlist_lookup_value(nv, ZPOOL_CONFIG_OFFLINE, &ival,
1351         DATA_TYPE_UINT64, NULL) == 0 ||
1352         nvlist_lookup_value(nv, ZPOOL_CONFIG_FAULTED, &ival,
1353         DATA_TYPE_UINT64, NULL) == 0 ||
1354         nvlist_lookup_value(nv, ZPOOL_CONFIG_REMOVED, &ival,
1355         DATA_TYPE_UINT64, NULL) == 0)
1356         return (ERR_DEV_VALUES);
1357
1358     return (0);
1359 }
1360 */
1361 /* Get a valid vdev pathname/devid from the boot device.
1362 * The caller should already allocate MAXPATHLEN memory for bootpath and devid.
1363 */
1364 static int
1365 vdev_get_bootpath(char *nv, uint64_t inguid, char *devid, char *bootpath,
1366     int is_spare)
1367 {
1368     char type[16];
1369
1370     if (nvlist_lookup_value(nv, ZPOOL_CONFIG_TYPE, &type, DATA_TYPE_STRING,
1371         NULL))
1372         return (ERR_FSYS_CORRUPT);
1373
1374     if (grub_strcmp(type, VDEV_TYPE_DISK) == 0) {
1375         uint64_t guid;
1376
1377         if (vdev_validate(nv) != 0)
1378             return (ERR_NO_BOOTPATH);
1379
1380         if (nvlist_lookup_value(nv, ZPOOL_CONFIG_GUID,
1381             &guid, DATA_TYPE_UINT64, NULL) != 0)
1382             return (ERR_NO_BOOTPATH);
1383
1384         if (guid != inguid)
1385             return (ERR_NO_BOOTPATH);
1386
1387         /* for a spare vdev, pick the disk labeled with "is_spare" */
1388         if (is_spare) {
1389             uint64_t spare = 0;
1390             (void) nvlist_lookup_value(nv, ZPOOL_CONFIG_IS_SPARE,
1391                 &spare, DATA_TYPE_UINT64, NULL);
1392             if (!spare)
1393                 return (ERR_NO_BOOTPATH);
1394         }
1395
1396         if (nvlist_lookup_value(nv, ZPOOL_CONFIG_PHYS_PATH,
1397             bootpath, DATA_TYPE_STRING, NULL) != 0)
1398             bootpath[0] = '\0';
1399
1400         if (nvlist_lookup_value(nv, ZPOOL_CONFIG_DEVID,
1401             devid, DATA_TYPE_STRING, NULL) != 0)
1402             devid[0] = '\0';
1403
1404         if (grub_strlen(bootpath) >= MAXPATHLEN ||
1405             grub_strlen(devid) >= MAXPATHLEN)
1406             return (ERR_WONT_FIT);
1407
1408         return (0);
1409
1410     } else if (grub_strcmp(type, VDEV_TYPE_MIRROR) == 0 ||
1411         grub_strcmp(type, VDEV_TYPE_REPLACE) == 0 ||
1412         (is_spare = (grub_strcmp(type, VDEV_TYPE_SPARE) == 0))) {

```

```

1414     int nelm, i;
1415     char *child;
1416
1417     if (nvlist_lookup_value(nv, ZPOOL_CONFIG_CHILDREN, &child,
1418         DATA_TYPE_NVLIST_ARRAY, &nelm))
1419         return (ERR_FSYS_CORRUPT);
1420
1421     for (i = 0; i < nelm; i++) {
1422         char *child_i;
1423
1424         child_i = nvlist_array(child, i);
1425         if (vdev_get_bootpath(child_i, inguid, devid,
1426             bootpath, is_spare) == 0)
1427             return (0);
1428     }
1429
1430     return (ERR_NO_BOOTPATH);
1431 }
1432
1433 /*
1434 * Check the disk label information and retrieve needed vdev name-value pairs.
1435 *
1436 * Return:
1437 *   0 - success
1438 *   ERR_* - failure
1439 */
1440 static int
1441 check_pool_label(uint64_t sector, char *stack, char *outdevid,
1442     char *outpath, uint64_t *outguid, uint64_t *outashift, uint64_t *outversion)
1443 {
1444     vdev_phys_t *vdev;
1445     uint64_t pool_state, txg = 0;
1446     char *nvlist, *nv, *features;
1447     uint64_t diskguid;
1448
1449     sector += (VDEV_SKIP_SIZE >> SPA_MINBLOCKSHIFT);
1450
1451     /* Read in the vdev name-value pair list (112K). */
1452     if (devread(sector, 0, VDEV_PHYS_SIZE, stack) == 0)
1453         return (ERR_READ);
1454
1455     vdev = (vdev_phys_t *)stack;
1456     stack += sizeof (vdev_phys_t);
1457
1458     if (nvlist_unpack(vdev->vp_nvlist, &nvlist))
1459         return (ERR_FSYS_CORRUPT);
1460
1461     if (nvlist_lookup_value(nvlist, ZPOOL_CONFIG_POOL_STATE, &pool_state,
1462         DATA_TYPE_UINT64, NULL))
1463         return (ERR_FSYS_CORRUPT);
1464
1465     if (pool_state == POOL_STATE_DESTROYED)
1466         return (ERR_FILESYSTEM_NOT_FOUND);
1467
1468     if (nvlist_lookup_value(nvlist, ZPOOL_CONFIG_POOL_NAME,
1469         current_rootpool, DATA_TYPE_STRING, NULL))
1470         return (ERR_FSYS_CORRUPT);
1471
1472     if (nvlist_lookup_value(nvlist, ZPOOL_CONFIG_POOL_TXG, &txg,
1473         DATA_TYPE_UINT64, NULL))
1474         return (ERR_FSYS_CORRUPT);
1475
1476     /* not an active device */
1477     if (txg == 0)
1478         return (ERR_NO_BOOTPATH);
1479

```

```

1481     if (nvlist_lookup_value(nvlist, ZPOOL_CONFIG_VERSION, outversion,
1482         DATA_TYPE_UINT64, NULL))
1483         return (ERR_FSYS_CORRUPT);
1484     if (!SPA_VERSION_IS_SUPPORTED(*outversion))
1485         return (ERR_NEWER_VERSION);
1486     if (nvlist_lookup_value(nvlist, ZPOOL_CONFIG_VDEV_TREE, &nv,
1487         DATA_TYPE_NVLIST, NULL))
1488         return (ERR_FSYS_CORRUPT);
1489     if (nvlist_lookup_value(nvlist, ZPOOL_CONFIG_GUID, &diskguid,
1490         DATA_TYPE_UINT64, NULL))
1491         return (ERR_FSYS_CORRUPT);
1492     if (nvlist_lookup_value(nv, ZPOOL_CONFIG_ASHIFT, outashift,
1493         DATA_TYPE_UINT64, NULL) != 0)
1494         return (ERR_FSYS_CORRUPT);
1495     if (vdev_get_bootpath_nv(diskguid, outdevid, outpath, 0))
1496         return (ERR_NO_BOOTPATH);
1497     if (nvlist_lookup_value(nvlist, ZPOOL_CONFIG_POOL_GUID, outguid,
1498         DATA_TYPE_UINT64, NULL))
1499         return (ERR_FSYS_CORRUPT);

1501     if (nvlist_lookup_value(nvlist, ZPOOL_CONFIG_FEATURES_FOR_READ,
1502         &features, DATA_TYPE_NVLIST, NULL) == 0) {
1503         char *nvp;
1504         char *name = stack;
1505         stack += MAXNAMELEN;
1506
1507         for (nvp = nvlist_next_nvpair(features, NULL);
1508             nvp != NULL;
1509             nvp = nvlist_next_nvpair(features, nvp)) {
1510             zap_attribute_t za;
1511
1512             if (nvpair_name(nvp, name, MAXNAMELEN) != 0)
1513                 return (ERR_FSYS_CORRUPT);
1514
1515             za.za_integer_length = 8;
1516             za.za_num_integers = 1;
1517             za.za_first_integer = 1;
1518             za.za_name = name;
1519             if (check_feature(&za, spa_feature_names, stack) != 0)
1520                 return (ERR_NEWER_VERSION);
1521         }
1522     }
1523
1524     return (0);
1525 }

1526 /* zfs_mount() locates a valid uberblock of the root pool and read in its MOS
1527 * to the memory address MOS.
1528 */
1529 *
1530 * Return:
1531 *   1 - success
1532 *   0 - failure
1533 */
1534
1535 int
1536 zfs_mount(void)
1537 {
1538     char *stack, *ub_array;
1539     int label = 0;
1540     uberblock_t *ubbest;
1541     objset_phys_t *osp;
1542     char tmp_bootpath[MAXNAMELEN];
1543     char tmp_devid[MAXNAMELEN];
1544     uint64_t tmp_guid, ashift, version;
1545     uint64_t adjpl = (uint64_t)part_length << SPA_MINBLOCKSHIFT;

```

```

1546     int err = errnum; /* preserve previous errnum state */
1547
1548     /* if it's our first time here, zero the best_uberblock out */
1549     if (best_drive == 0 && best_part == 0 && find_best_root) {
1550         grub_memset(&current_uberblock, 0, sizeof (uberblock_t));
1551         pool_guid = 0;
1552     }
1553
1554     stackbase = ZFS_SCRATCH;
1555     stack = stackbase;
1556     ub_array = stack;
1557     stack += VDEV_UBERBLOCK_RING;
1558
1559     osp = (objset_phys_t *)stack;
1560     stack += sizeof (objset_phys_t);
1561     adjpl = P2ALIGN(adjpl, (uint64_t)sizeof (vdev_label_t));
1562
1563     for (label = 0; label < VDEV_LABELS; label++) {
1564
1565         /*
1566         * some eltorito stacks don't give us a size and
1567         * we end up setting the size to MAXUINT, further
1568         * some of these devices stop working once a single
1569         * read past the end has been issued. Checking
1570         * for a maximum part_length and skipping the backup
1571         * labels at the end of the slice/partition/device
1572         * avoids breaking down on such devices.
1573         */
1574         if (part_length == MAXUINT && label == 2)
1575             break;
1576
1577         uint64_t sector = vdev_label_start(adjpl,
1578             label) >> SPA_MINBLOCKSHIFT;
1579
1580         /* Read in the uberblock ring (128K). */
1581         if (devread(sector +
1582             ((VDEV_SKIP_SIZE + VDEV_PHYS_SIZE) >> SPA_MINBLOCKSHIFT),
1583             0, VDEV_UBERBLOCK_RING, ub_array) == 0)
1584             continue;
1585
1586         if (check_pool_label(sector, stack, tmp_devid,
1587             tmp_bootpath, &tmp_guid, &ashift, &version))
1588             continue;
1589
1590         if (pool_guid == 0)
1591             pool_guid = tmp_guid;
1592
1593         if ((ubbest = find_bestub(ub_array, ashift, sector)) == NULL ||
1594             zio_read(&ubbest->ub_rootbp, osp, stack) != 0)
1595             continue;
1596
1597         VERIFY_OS_TYPE(osp, DMU_OST_META);
1598
1599         if (version >= SPA_VERSION_FEATURES &&
1600             check_mos_features(&osp->os_meta_dnode, stack) != 0)
1601             continue;
1602
1603         if (find_best_root && ((pool_guid != tmp_guid) ||
1604             vdev_uberblock_compare(ubbest, &(current_uberblock)) <= 0))
1605             continue;
1606
1607         /* Got the MOS. Save it at the memory addr MOS. */
1608         grub_memmove(MOS, &osp->os_meta_dnode, DNODE_SIZE);
1609         grub_memmove(&current_uberblock, ubbest, sizeof (uberblock_t));
1610         grub_memmove(current_bootpath, tmp_bootpath, MAXNAMELEN);
1611         grub_memmove(current_devid, tmp_devid, grub_strlen(tmp_devid));

```

```

1612         is_zfs_mount = 1;
1613     }
1614 }
1615 /*
1616  * While some fs impls. (tftp) rely on setting and keeping
1617  * global errnums set, others won't reset it and will break
1618  * when issuing rawreads. The goal here is to simply not
1619  * have zfs mount attempts impact the previous state.
1620  */
1621 errnum = err;
1622 return (0);
1623 }

1624 */

1625 /* zfs_open() locates a file in the rootpool by following the
1626 * MOS and places the dnode of the file in the memory address DNODE.
1627 *
1628 * Return:
1629 *   1 - success
1630 *   0 - failure
1631 */
1632 int
1633 zfs_open(char *filename)
1634 {
1635     char *stack;
1636     dnode_phys_t *mdn;
1637
1638     file_buf = NULL;
1639     stackbase = ZFS_SCRATCH;
1640     stack = stackbase;
1641
1642     mdn = (dnode_phys_t *)stack;
1643     stack += sizeof(dnode_phys_t);
1644
1645     dnode_mdn = NULL;
1646     dnode_buf = (dnode_phys_t *)stack;
1647     stack += 1<<DNODE_BLOCK_SHIFT;
1648
1649 /*
1650  * menu.lst is placed at the root pool filesystem level,
1651  * do not goto 'current_bootfs'.
1652  */
1653 if (is_top_dataset_file(filename)) {
1654     if (errnum = get_objset_mdn(MOS, NULL, NULL, mdn, stack))
1655         return (0);
1656
1657     current_bootfs_obj = 0;
1658 } else {
1659     if (current_bootfs[0] == '\0') {
1660         /* Get the default root filesystem object number */
1661         if (errnum = get_default_bootfsobj(MOS,
1662             &current_bootfs_obj, stack))
1663             return (0);
1664
1665         if (errnum = get_objset_mdn(MOS, NULL,
1666             &current_bootfs_obj, mdn, stack))
1667             return (0);
1668
1669     } else {
1670         if (errnum = get_objset_mdn(MOS, current_bootfs,
1671             &current_bootfs_obj, mdn, stack)) {
1672             grub_memset(current_bootfs, 0, MAXNAMELEN);
1673             return (0);
1674         }
1675     }
1676 }
1677 }
```

```

1679     if (dnode_get_path(mdn, filename, DNODE, stack)) {
1680         errnum = ERR_FILE_NOT_FOUND;
1681         return (0);
1682     }
1683
1684     /* get the file size and set the file position to 0 */
1685
1686     /*
1687      * For DMU_OT_SA we will need to locate the SIZE attribute
1688      * attribute, which could be either in the bonus buffer
1689      * or the "spill" block.
1690      */
1691     if (DNODE->dn_bonustype == DMU_OT_SA) {
1692         sa_hdr_phys_t *sahdrp;
1693         int hdrsize;
1694
1695         if (DNODE->dn_bonuslen != 0) {
1696             sahdrp = (sa_hdr_phys_t *)DN_BONUS(DNODE);
1697         } else {
1698             if (DNODE->dn_flags & DNODE_FLAG_SPILL_BLKPTR) {
1699                 blkptr_t *bp = &DNODE->dn_spill;
1700                 void *buf;
1701
1702                 buf = (void *)stack;
1703                 stack += BP_GET_LSIZE(bp);
1704
1705                 /* reset errnum to rawread() failure */
1706                 errnum = 0;
1707                 if (zio_read(bp, buf, stack) != 0) {
1708                     return (0);
1709                 }
1710                 sahdrp = buf;
1711             } else {
1712                 errnum = ERR_FSYS_CORRUPT;
1713                 return (0);
1714             }
1715         }
1716         hdrsize = SA_HDR_SIZE(sahdrp);
1717         filemax = *(uint64_t *)((char *)sahdrp + hdrsize +
1718             SA_SIZE_OFFSET);
1719     } else {
1720         filemax = ((znode_phys_t *)DN_BONUS(DNODE))->zp_size;
1721     }
1722     filepos = 0;
1723
1724     dnode_buf = NULL;
1725     return (1);
1726 }

1727 /*
1728  * zfs_read reads in the data blocks pointed by the DNODE.
1729  *
1730  * Return:
1731  *   len - the length successfully read in to the buffer
1732  *   0 - failure
1733  */
1734 int
1735 zfs_read(char *buf, int len)
1736 {
1737     char *stack;
1738     int blksz, length, movesize;
1739
1740     if (file_buf == NULL) {
1741         file_buf = stackbase;
1742         stackbase += SPA_MAXBLOCKSIZE;
1743     }
1744
1745     if (length > movesize) {
1746         if (length > blksz) {
1747             length = blksz;
1748         }
1749         if (movesize > blksz) {
1750             movesize = blksz;
1751         }
1752         if (length > movesize) {
1753             length = movesize;
1754         }
1755     }
1756
1757     if (length > 0) {
1758         if (length > blksz) {
1759             length = blksz;
1760         }
1761         if (movesize > blksz) {
1762             movesize = blksz;
1763         }
1764         if (length > movesize) {
1765             length = movesize;
1766         }
1767     }
1768
1769     if (length > 0) {
1770         if (length > blksz) {
1771             length = blksz;
1772         }
1773         if (movesize > blksz) {
1774             movesize = blksz;
1775         }
1776         if (length > movesize) {
1777             length = movesize;
1778         }
1779     }
1780
1781     if (length > 0) {
1782         if (length > blksz) {
1783             length = blksz;
1784         }
1785         if (movesize > blksz) {
1786             movesize = blksz;
1787         }
1788         if (length > movesize) {
1789             length = movesize;
1790         }
1791     }
1792
1793     if (length > 0) {
1794         if (length > blksz) {
1795             length = blksz;
1796         }
1797         if (movesize > blksz) {
1798             movesize = blksz;
1799         }
1800         if (length > movesize) {
1801             length = movesize;
1802         }
1803     }
1804
1805     if (length > 0) {
1806         if (length > blksz) {
1807             length = blksz;
1808         }
1809         if (movesize > blksz) {
1810             movesize = blksz;
1811         }
1812         if (length > movesize) {
1813             length = movesize;
1814         }
1815     }
1816
1817     if (length > 0) {
1818         if (length > blksz) {
1819             length = blksz;
1820         }
1821         if (movesize > blksz) {
1822             movesize = blksz;
1823         }
1824         if (length > movesize) {
1825             length = movesize;
1826         }
1827     }
1828
1829     if (length > 0) {
1830         if (length > blksz) {
1831             length = blksz;
1832         }
1833         if (movesize > blksz) {
1834             movesize = blksz;
1835         }
1836         if (length > movesize) {
1837             length = movesize;
1838         }
1839     }
1840
1841     if (length > 0) {
1842         if (length > blksz) {
1843             length = blksz;
1844         }
1845         if (movesize > blksz) {
1846             movesize = blksz;
1847         }
1848         if (length > movesize) {
1849             length = movesize;
1850         }
1851     }
1852
1853     if (length > 0) {
1854         if (length > blksz) {
1855             length = blksz;
1856         }
1857         if (movesize > blksz) {
1858             movesize = blksz;
1859         }
1860         if (length > movesize) {
1861             length = movesize;
1862         }
1863     }
1864
1865     if (length > 0) {
1866         if (length > blksz) {
1867             length = blksz;
1868         }
1869         if (movesize > blksz) {
1870             movesize = blksz;
1871         }
1872         if (length > movesize) {
1873             length = movesize;
1874         }
1875     }
1876
1877     if (length > 0) {
1878         if (length > blksz) {
1879             length = blksz;
1880         }
1881         if (movesize > blksz) {
1882             movesize = blksz;
1883         }
1884         if (length > movesize) {
1885             length = movesize;
1886         }
1887     }
1888
1889     if (length > 0) {
1890         if (length > blksz) {
1891             length = blksz;
1892         }
1893         if (movesize > blksz) {
1894             movesize = blksz;
1895         }
1896         if (length > movesize) {
1897             length = movesize;
1898         }
1899     }
1900
1901     if (length > 0) {
1902         if (length > blksz) {
1903             length = blksz;
1904         }
1905         if (movesize > blksz) {
1906             movesize = blksz;
1907         }
1908         if (length > movesize) {
1909             length = movesize;
1910         }
1911     }
1912
1913     if (length > 0) {
1914         if (length > blksz) {
1915             length = blksz;
1916         }
1917         if (movesize > blksz) {
1918             movesize = blksz;
1919         }
1920         if (length > movesize) {
1921             length = movesize;
1922         }
1923     }
1924
1925     if (length > 0) {
1926         if (length > blksz) {
1927             length = blksz;
1928         }
1929         if (movesize > blksz) {
1930             movesize = blksz;
1931         }
1932         if (length > movesize) {
1933             length = movesize;
1934         }
1935     }
1936
1937     if (length > 0) {
1938         if (length > blksz) {
1939             length = blksz;
1940         }
1941         if (movesize > blksz) {
1942             movesize = blksz;
1943         }
1944         if (length > movesize) {
1945             length = movesize;
1946         }
1947     }
1948
1949     if (length > 0) {
1950         if (length > blksz) {
1951             length = blksz;
1952         }
1953         if (movesize > blksz) {
1954             movesize = blksz;
1955         }
1956         if (length > movesize) {
1957             length = movesize;
1958         }
1959     }
1960
1961     if (length > 0) {
1962         if (length > blksz) {
1963             length = blksz;
1964         }
1965         if (movesize > blksz) {
1966             movesize = blksz;
1967         }
1968         if (length > movesize) {
1969             length = movesize;
1970         }
1971     }
1972
1973     if (length > 0) {
1974         if (length > blksz) {
1975             length = blksz;
1976         }
1977         if (movesize > blksz) {
1978             movesize = blksz;
1979         }
1980         if (length > movesize) {
1981             length = movesize;
1982         }
1983     }
1984
1985     if (length > 0) {
1986         if (length > blksz) {
1987             length = blksz;
1988         }
1989         if (movesize > blksz) {
1990             movesize = blksz;
1991         }
1992         if (length > movesize) {
1993             length = movesize;
1994         }
1995     }
1996
1997     if (length > 0) {
1998         if (length > blksz) {
1999             length = blksz;
2000         }
2001         if (movesize > blksz) {
2002             movesize = blksz;
2003         }
2004         if (length > movesize) {
2005             length = movesize;
2006         }
2007     }
2008
2009     if (length > 0) {
2010         if (length > blksz) {
2011             length = blksz;
2012         }
2013         if (movesize > blksz) {
2014             movesize = blksz;
2015         }
2016         if (length > movesize) {
2017             length = movesize;
2018         }
2019     }
2020
2021     if (length > 0) {
2022         if (length > blksz) {
2023             length = blksz;
2024         }
2025         if (movesize > blksz) {
2026             movesize = blksz;
2027         }
2028         if (length > movesize) {
2029             length = movesize;
2030         }
2031     }
2032
2033     if (length > 0) {
2034         if (length > blksz) {
2035             length = blksz;
2036         }
2037         if (movesize > blksz) {
2038             movesize = blksz;
2039         }
2040         if (length > movesize) {
2041             length = movesize;
2042         }
2043     }
2044
2045     if (length > 0) {
2046         if (length > blksz) {
2047             length = blksz;
2048         }
2049         if (movesize > blksz) {
2050             movesize = blksz;
2051         }
2052         if (length > movesize) {
2053             length = movesize;
2054         }
2055     }
2056
2057     if (length > 0) {
2058         if (length > blksz) {
2059             length = blksz;
2060         }
2061         if (movesize > blksz) {
2062             movesize = blksz;
2063         }
2064         if (length > movesize) {
2065             length = movesize;
2066         }
2067     }
2068
2069     if (length > 0) {
2070         if (length > blksz) {
2071             length = blksz;
2072         }
2073         if (movesize > blksz) {
2074             movesize = blksz;
2075         }
2076         if (length > movesize) {
2077             length = movesize;
2078         }
2079     }
2080
2081     if (length > 0) {
2082         if (length > blksz) {
2083             length = blksz;
2084         }
2085         if (movesize > blksz) {
2086             movesize = blksz;
2087         }
2088         if (length > movesize) {
2089             length = movesize;
2090         }
2091     }
2092
2093     if (length > 0) {
2094         if (length > blksz) {
2095             length = blksz;
2096         }
2097         if (movesize > blksz) {
2098             movesize = blksz;
2099         }
2100         if (length > movesize) {
2101             length = movesize;
2102         }
2103     }
2104
2105     if (length > 0) {
2106         if (length > blksz) {
2107             length = blksz;
2108         }
2109         if (movesize > blksz) {
2110             movesize = blksz;
2111         }
2112         if (length > movesize) {
2113             length = movesize;
2114         }
2115     }
2116
2117     if (length > 0) {
2118         if (length > blksz) {
2119             length = blksz;
2120         }
2121         if (movesize > blksz) {
2122             movesize = blksz;
2123         }
2124         if (length > movesize) {
2125             length = movesize;
2126         }
2127     }
2128
2129     if (length > 0) {
2130         if (length > blksz) {
2131             length = blksz;
2132         }
2133         if (movesize > blksz) {
2134             movesize = blksz;
2135         }
2136         if (length > movesize) {
2137             length = movesize;
2138         }
2139     }
2140
2141     if (length > 0) {
2142         if (length > blksz) {
2143             length = blksz;
2144         }
2145         if (movesize > blksz) {
2146             movesize = blksz;
2147         }
2148         if (length > movesize) {
2149             length = movesize;
2150         }
2151     }
2152
2153     if (length > 0) {
2154         if (length > blksz) {
2155             length = blksz;
2156         }
2157         if (movesize > blksz) {
2158             movesize = blksz;
2159         }
2160         if (length > movesize) {
2161             length = movesize;
2162         }
2163     }
2164
2165     if (length > 0) {
2166         if (length > blksz) {
2167             length = blksz;
2168         }
2169         if (movesize > blksz) {
2170             movesize = blksz;
2171         }
2172         if (length > movesize) {
2173             length = movesize;
2174         }
2175     }
2176
2177     if (length > 0) {
2178         if (length > blksz) {
2179             length = blksz;
2180         }
2181         if (movesize > blksz) {
2182             movesize = blksz;
2183         }
2184         if (length > movesize) {
2185             length = movesize;
2186         }
2187     }
2188
2189     if (length > 0) {
2190         if (length > blksz) {
2191             length = blksz;
2192         }
2193         if (movesize > blksz) {
2194             movesize = blksz;
2195         }
2196         if (length > movesize) {
2197             length = movesize;
2198         }
2199     }
2200
2201     if (length > 0) {
2202         if (length > blksz) {
2203             length = blksz;
2204         }
2205         if (movesize > blksz) {
2206             movesize = blksz;
2207         }
2208         if (length > movesize) {
2209             length = movesize;
2210         }
2211     }
2212
2213     if (length > 0) {
2214         if (length > blksz) {
2215             length = blksz;
2216         }
2217         if (movesize > blksz) {
2218             movesize = blksz;
2219         }
2220         if (length > movesize) {
2221             length = movesize;
2222         }
2223     }
2224
2225     if (length > 0) {
2226         if (length > blksz) {
2227             length = blksz;
2228         }
2229         if (movesize > blksz) {
2230             movesize = blksz;
2231         }
2232         if (length > movesize) {
2233             length = movesize;
2234         }
2235     }
2236
2237     if (length > 0) {
2238         if (length > blksz) {
2239             length = blksz;
2240         }
2241         if (movesize > blksz) {
2242             movesize = blksz;
2243         }
2244         if (length > movesize) {
2245             length = movesize;
2246         }
2247     }
2248
2249     if (length > 0) {
2250         if (length > blksz) {
2251             length = blksz;
2252         }
2253         if (movesize > blksz) {
2254             movesize = blksz;
2255         }
2256         if (length > movesize) {
2257             length = movesize;
2258         }
2259     }
2260
2261     if (length > 0) {
2262         if (length > blksz) {
2263             length = blksz;
2264         }
2265         if (movesize > blksz) {
2266             movesize = blksz;
2267         }
2268         if (length > movesize) {
2269             length = movesize;
2270         }
2271     }
2272
2273     if (length > 0) {
2274         if (length > blksz) {
2275             length = blksz;
2276         }
2277         if (movesize > blksz) {
2278             movesize = blksz;
2279         }
2280         if (length > movesize) {
2281             length = movesize;
2282         }
2283     }
2284
2285     if (length > 0) {
2286         if (length > blksz) {
2287             length = blksz;
2288         }
2289         if (movesize > blksz) {
2290             movesize = blksz;
2291         }
2292         if (length > movesize) {
2293             length = movesize;
2294         }
2295     }
2296
2297     if (length > 0) {
2298         if (length > blksz) {
2299             length = blksz;
2300         }
2301         if (movesize > blksz) {
2302             movesize = blksz;
2303         }
2304         if (length > movesize) {
2305             length = movesize;
2306         }
2307     }
2308
2309     if (length > 0) {
2310         if (length > blksz) {
2311             length = blksz;
2312         }
2313         if (movesize > blksz) {
2314             movesize = blksz;
2315         }
2316         if (length > movesize) {
2317             length = movesize;
2318         }
2319     }
2320
2321     if (length > 0) {
2322         if (length > blksz) {
2323             length = blksz;
2324         }
2325         if (movesize > blksz) {
2326             movesize = blksz;
2327         }
2328         if (length > movesize) {
2329             length = movesize;
2330         }
2331     }
2332
2333     if (length > 0) {
2334         if (length > blksz) {
2335             length = blksz;
2336         }
2337         if (movesize > blksz) {
2338             movesize = blksz;
2339         }
2340         if (length > movesize) {
2341             length = movesize;
2342         }
2343     }
2344
2345     if (length > 0) {
2346         if (length > blksz) {
2347             length = blksz;
2348         }
2349         if (movesize > blksz) {
2350             movesize = blksz;
2351         }
2352         if (length > movesize) {
2353             length = movesize;
2354         }
2355     }
2356
2357     if (length > 0) {
2358         if (length > blksz) {
2359             length = blksz;
2360         }
2361         if (movesize > blksz) {
2362             movesize = blksz;
2363         }
2364         if (length > movesize) {
2365             length = movesize;
2366         }
2367     }
2368
2369     if (length > 0) {
2370         if (length > blksz) {
2371             length = blksz;
2372         }
2373         if (movesize > blksz) {
2374             movesize = blksz;
2375         }
2376         if (length > movesize) {
2377             length = movesize;
2378         }
2379     }
2380
2381     if (length > 0) {
2382         if (length > blksz) {
2383             length = blksz;
2384         }
2385         if (movesize > blksz) {
2386             movesize = blksz;
2387         }
2388         if (length > movesize) {
2389             length = movesize;
2390         }
2391     }
2392
2393     if (length > 0) {
2394         if (length > blksz) {
2395             length = blksz;
2396         }
2397         if (movesize > blksz) {
2398             movesize = blksz;
2399         }
2400         if (length > movesize) {
2401             length = movesize;
2402         }
2403     }
2404
2405     if (length > 0) {
2406         if (length > blksz) {
2407             length = blksz;
2408         }
2409         if (movesize > blksz) {
2410             movesize = blksz;
2411         }
2412         if (length > movesize) {
2413             length = movesize;
2414         }
2415     }
2416
2417     if (length > 0) {
2418         if (length > blksz) {
2419             length = blksz;
2420         }
2421         if (movesize > blksz) {
2422             movesize = blksz;
2423         }
2424         if (length > movesize) {
2425             length = movesize;
2426         }
2427     }
2428
2429     if (length > 0) {
2430         if (length > blksz) {
2431             length = blksz;
2432         }
2433         if (movesize > blksz) {
2434             movesize = blksz;
2435         }
2436         if (length > movesize) {
2437             length = movesize;
2438         }
2439     }
2440
2441     if (length > 0) {
2442         if (length > blksz) {
2443             length = blksz;
2444         }
2445         if (movesize > blksz) {
2446             movesize = blksz;
2447         }
2448         if (length > movesize) {
2449             length = movesize;
2450         }
2451     }
2452
2453     if (length > 0) {
2454         if (length > blksz) {
2455             length = blksz;
2456         }
2457         if (movesize > blksz) {
2458             movesize = blksz;
2459         }
2460         if (length > movesize) {
2461             length = movesize;
2462         }
2463     }
2464
2465     if (length > 0) {
2466         if (length > blksz) {
2467             length = blksz;
2468         }
2469         if (movesize > blksz) {
2470             movesize = blksz;
2471         }
2472         if (length > movesize) {
2473             length = movesize;
2474         }
2475     }
2476
2477     if (length > 0) {
2478         if (length > blksz) {
2479             length = blksz;
2480         }
2481         if (movesize > blksz) {
2482             movesize = blksz;
2483         }
2484         if (length > movesize) {
2485             length = movesize;
2486         }
2487     }
2488
2489     if (length > 0) {
2490         if (length > blksz) {
2491             length = blksz;
2492         }
2493         if (movesize > blksz) {
2494             movesize = blksz;
2495         }
2496         if (length > movesize) {
2497             length = movesize;
2498         }
2499     }
2500
2501     if (length > 0) {
2502         if (length > blksz) {
2503             length = blksz;
2504         }
2505         if (movesize > blksz) {
2506             movesize = blksz;
2507         }
2508         if (length > movesize) {
2509             length = movesize;
2510         }
2511     }
2512
2513     if (length > 0) {
2514         if (length > blksz) {
2515             length = blksz;
2516         }
2517         if (movesize > blksz) {
2518             movesize = blksz;
2519         }
2520         if (length > movesize) {
2521             length = movesize;
2522         }
2523     }
2524
2525     if (length > 0) {
2526         if (length > blksz) {
2527             length = blksz;
2528         }
2529         if (movesize > blksz) {
2530             movesize = blksz;
2531         }
2532         if (length > movesize) {
2533             length = movesize;
2534         }
2535     }
2536
2537     if (length > 0) {
2538         if (length > blksz) {
2539             length = blksz;
2540         }
2541         if (movesize > blksz) {
2542             movesize = blksz;
2543         }
2544         if (length > movesize) {
2545             length = movesize;
2546         }
2547     }
2548
2549     if (length > 0) {
2550         if (length > blksz) {
2551             length = blksz;
2552         }
2553         if (movesize > blksz) {
2554             movesize = blksz;
2555         }
2556         if (length > movesize) {
2557             length = movesize;
2558         }
2559     }
2560
2561     if (length > 0) {
2562         if (length > blksz) {
2563             length = blksz;
2564         }
2565         if (movesize > blksz) {
2566             movesize = blksz;
2567         }
2568         if (length > movesize) {
2569             length = movesize;
2570         }
2571     }
2572
2573     if (length > 0) {
2574         if (length > blksz) {
2575             length = blksz;
2576         }
2577         if (movesize > blksz) {
2578             movesize = blksz;
2579         }
2580         if (length > movesize) {
2581             length = movesize;
2582         }
2583     }
2584
2585     if (length > 0) {
2586         if (length > blksz) {
2587             length = blksz;
2588         }
2589         if (movesize > blksz) {
2590             movesize = blksz;
2591         }
2592         if (length > movesize) {
2593             length = movesize;
2594         }
2595     }
2596
2597     if (length > 0) {
2598         if (length > blksz) {
2599             length = blksz;
2600         }
2601         if (movesize > blksz) {
2602             movesize = blksz;
2603         }
2604         if (length > movesize) {
2605             length = movesize;
2606         }
2607     }
2608
2609     if (length > 0) {
2610         if (length > blksz) {
2611             length = blksz;
2612         }
2613         if (movesize > blksz) {
2614             movesize = blksz;
2615         }
2616         if (length > movesize) {
2617             length = movesize;
2618         }
2619     }
2620
2621     if (length > 0) {
2622         if (length > blksz) {
2623             length = blksz;
2624         }
2625         if (movesize > blksz) {
2626             movesize = blksz;
2627         }
2628         if (length > movesize) {
2629             length = movesize;
2630         }
2631     }
2632
2633     if (length > 0) {
2634         if (length > blksz) {
2635             length = blksz;
2636         }
2637         if (movesize > blksz) {
2638             movesize = blksz;
2639         }
2640         if (length > movesize) {
2641             length = movesize;
2642         }
2643     }
2644
2645     if (length > 0) {
2646         if (length > blksz) {
2647             length = blksz;
2648         }
2649         if (movesize > blksz) {
2650             movesize = blksz;
2651         }
2652         if (length > movesize) {
2653             length = movesize;
2654         }
2655     }
2656
2657     if (length > 0) {
2658         if (length > blksz) {
2659             length = blksz;
2660         }
2661         if (movesize > blksz) {
2662             movesize = blksz;
2663         }
2664         if (length > movesize) {
2665             length = movesize;
2666         }
2667     }
2668
2669     if (length > 0) {
2670         if (length > blksz) {
2671             length = blksz;
2672         }
2673         if (movesize > blksz) {
2674             movesize = blksz;
2675         }
2676         if (length > movesize) {
2677             length = movesize;
2678         }
2679     }
2680
2681     if (length > 0) {
2682         if (length > blksz) {
2683             length = blksz;
2684         }
2685         if (movesize > blksz) {
2686             movesize = blksz;
2687         }
2688         if (length > movesize) {
2689             length = movesize;
2690         }
2691     }
2692
2693     if (length > 0) {
2694         if (length > blksz) {
2695             length = blksz;
2696         }
2697         if (movesize > blksz) {
2698             movesize = blksz;
2699         }
2700         if (length > movesize) {
2701             length = movesize;
27
```

```
1744     file_start = file_end = 0;
1745 }
1746 stack = stackbase;
1747
1748 /*
1749  * If offset is in memory, move it into the buffer provided and return.
1750 */
1751 if (filepos >= file_start && filepos+len <= file_end) {
1752     grub_memmove(buf, file_buf + filepos - file_start, len);
1753     filepos += len;
1754     return (len);
1755 }
1756
1757 blksz = DNODE->dn_datablksize << SPA_MINBLOCKSHIFT;
1758 /*
1759  * Entire Dnode is too big to fit into the space available. We
1760  * will need to read it in chunks. This could be optimized to
1761  * read in as large a chunk as there is space available, but for
1762  * now, this only reads in one data block at a time.
1763 */
1764 length = len;
1765 while (length) {
1766     /*
1767      * Find requested blkid and the offset within that block.
1768      */
1769     uint64_t blkid = filepos / blksz;
1770
1771     if (errnum = dmu_read(DNODE, blkid, file_buf, stack))
1772         return (0);
1773
1774     file_start = blkid * blksz;
1775     file_end = file_start + blksz;
1776
1777     movesize = MIN(length, file_end - filepos);
1778
1779     grub_memmove(buf, file_buf + filepos - file_start,
1780                  movesize);
1781     buf += movesize;
1782     length -= movesize;
1783     filepos += movesize;
1784 }
1785
1786     return (len);
1787 }
1788
1789 /*
1790  * No-Op
1791 */
1792
1793 int
1794 zfs_embed(int *start_sector, int needed_sectors)
1795 {
1796     return (1);
1797 }
1798
1799 #endif /* FSYS_ZFS */
```