```
**********************************************************
   38881 Thu Jan 31 12:29:30 2013
new/usr/src/uts/common/os/kstat_fr.c
XXXX Check for uninitialized kernel statistics
**********************************************************
______unchanged_portion_omitted_

1132 /*
1133  * Activate a fully initialized kstat and make it visible to /dev/kstat.
1134  */
1135 void
1136 kstat_install(kstat_t *ksp)
1137 {
1138         zoneid_t zoneid = ((ekstat_t *)ksp)->e_zone.zoneid;

1140         /*
1141          * If this is a variable-size kstat, it MUST provide kstat data locking
1142          * to prevent data-size races with kstat readers.
1143          */
1144         if ((ksp->ks_flags & KSTAT_FLAG_VAR_SIZE) && ksp->ks_lock == NULL) {
1145                 panic("kstat_install('%s', %d, '%s'): "
1146                     "cannot create variable-size kstat without data lock",
1147                     ksp->ks_module, ksp->ks_instance, ksp->ks_name);
1148         }

1150         if (kstat_hold_bykid(ksp->ks_kid, zoneid) != ksp) {
1151                 cmn_err(CE_WARN, "kstat_install(%p): does not exist",
1152                     (void *)ksp);
1153                 return;
1154         }

1156         if (ksp->ks_type == KSTAT_TYPE_NAMED && ksp->ks_data != NULL) {
1157                 int has_long_strings = 0;
1158                 uint_t i;
1159                 kstat_named_t *knp = KSTAT_NAMED_PTR(ksp);

1161                 for (i = 0; i < ksp->ks_ndata; i++, knp++) {
1162                         if (knp->data_type == KSTAT_DATA_STRING) {
1163                                 has_long_strings = 1;
1164                                 break;
1165                         }
1166                 }
1167                 /*
1168                  * It is an error for a named kstat with fields of
1169                  * KSTAT_DATA_STRING to be non-virtual.
1170                  */
1171                 if (has_long_strings && !(ksp->ks_flags & KSTAT_FLAG_VIRTUAL)) {
1172                         panic("kstat_install('%s', %d, '%s'): "
1173                             "named kstat containing KSTAT_DATA_STRING "
1174                             "is not virtual",
1175                             ksp->ks_module, ksp->ks_instance,
1176                             ksp->ks_name);
1177                 }
1178                 /*
1179                  * The default snapshot routine does not handle KSTAT_WRITE
1180                  * for long strings.
1181                  */
1182                 if (has_long_strings && (ksp->ks_flags & KSTAT_FLAG_WRITABLE) &&
1183                     (ksp->ks_snapshot == default_kstat_snapshot)) {
1184                         panic("kstat_install('%s', %d, '%s'): "
1185                             "named kstat containing KSTAT_DATA_STRING "
1186                             "is writable but uses default snapshot routine",
1187                             ksp->ks_module, ksp->ks_instance, ksp->ks_name);
1188                 }
1189         }
```



```
1191         if (ksp->ks_flags & KSTAT_FLAG_DORMANT) {

1193                 /*
1194                  * We are reactivating a dormant kstat.  Initialize the
1195                  * caller's underlying data to the value it had when the
1196                  * kstat went dormant, and mark the kstat as active.
1197                  * Grab the provider's kstat lock if it's not already held.
1198                  */
1199                 kmutex_t *lp = ksp->ks_lock;
1200                 if (lp != NULL && MUTEX_NOT_HELD(lp)) {
1201                         mutex_enter(lp);
1202                         (void) KSTAT_UPDATE(ksp, KSTAT_WRITE);
1203                         mutex_exit(lp);
1204                 } else {
1205                         (void) KSTAT_UPDATE(ksp, KSTAT_WRITE);
1206                 }
1207                 ksp->ks_flags &= ~KSTAT_FLAG_DORMANT;
1208         }

1210 #ifdef DEBUG
1211         /*
1212          * Search for uninitialized kstats.
1213          */
1214         switch (ksp->ks_type) {
1215         case KSTAT_TYPE_NAMED: {
1216                 uint_t i;
1217                 kstat_named_t *knp = KSTAT_NAMED_PTR(ksp);

1219                 for (i = 0; i < ksp->ks_ndata; i++, knp++) {
1220                         if (knp->data_type > KSTAT_DATA_STRING) {
1221                                 cmn_err(CE_WARN,
1222                                     "kstat_install('%s', %d, '%s'): "
1223                                     "invalid data type",
1224                                     ksp->ks_module, ksp->ks_instance,
1225                                     ksp->ks_name);
1226                         }

1228                         /*
1229                          * If the name of this kstat is empty
1230                          * we assume it is uninitialized.
1231                          */
1232                         if (knp->name[0] == '\0') {
1233                                 cmn_err(CE_WARN,
1234                                     "kstat_install('%s', %d, '%s'): "
1235                                     "uninitialized kstat",
1236                                     ksp->ks_module, ksp->ks_instance,
1237                                     ksp->ks_name);
1238                         }
1239                 }

1241                 break;
1242         }
1243         default:
1244                 break;
1245         }
1246 #endif

1248 #endif /* ! codereview */
1249         /*
1250          * Now that the kstat is active, make it visible to the kstat driver.
1251          */
1252         ksp->ks_flags &= ~KSTAT_FLAG_INVALID;
1253         kstat_rele(ksp);
1254 }

1256 /*
```

```
 * Remove a kstat from the system.  Or, if it's a persistent kstat,
 * just update the data and mark it as dormant.
 */
void
kstat_delete(kstat_t *ksp)
{
        kmutex_t *lp;
        ekstat_t *e = (ekstat_t *)ksp;
        zoneid_t zoneid;
        kstat_zone_t *kz;

        ASSERT(ksp != NULL);

        if (ksp == NULL)
                return;

        zoneid = e->e_zone.zoneid;

        lp = ksp->ks_lock;

        if (lp != NULL && MUTEX_HELD(lp)) {
                panic("kstat_delete(%p): caller holds data lock %p",
                    (void *)ksp, (void *)lp);
        }

        if (kstat_hold_bykid(ksp->ks_kid, zoneid) != ksp) {
                cmn_err(CE_WARN, "kstat_delete(%p): does not exist",
                    (void *)ksp);
                return;
        }

        if (ksp->ks_flags & KSTAT_FLAG_PERSISTENT) {
                /*
                 * Update the data one last time, so that all activity
                 * prior to going dormant has been accounted for.
                 */
                KSTAT_ENTER(ksp);
                (void) KSTAT_UPDATE(ksp, KSTAT_READ);
                KSTAT_EXIT(ksp);

                /*
                 * Mark the kstat as dormant and restore caller-modifiable
                 * fields to default values, so the kstat is readable during
                 * the dormant phase.
                 */
                ksp->ks_flags |= KSTAT_FLAG_DORMANT;
                ksp->ks_lock = NULL;
                ksp->ks_update = default_kstat_update;
                ksp->ks_private = NULL;
                ksp->ks_snapshot = default_kstat_snapshot;
                kstat_rele(ksp);
                return;
        }

        /*
         * Remove the kstat from the framework's AVL trees,
         * free the allocated memory, and increment kstat_chain_id so
         * /dev/kstat clients can detect the event.
         */
        mutex_enter(&kstat_chain_lock);
        avl_remove(&kstat_avl_bykid, e);
        avl_remove(&kstat_avl_byname, e);
        kstat_chain_id++;
        mutex_exit(&kstat_chain_lock);

        kz = e->e_zone.next;
```



```
        while (kz != NULL) {
                kstat_zone_t *t = kz;

                kz = kz->next;
                kmem_free(t, sizeof (*t));
        }
        kstat_rele(ksp);
        kstat_free(e);
}

void
kstat_delete_byname_zone(const char *ks_module, int ks_instance,
    const char *ks_name, zoneid_t ks_zoneid)
{
        kstat_t *ksp;

        ksp = kstat_hold_byname(ks_module, ks_instance, ks_name, ks_zoneid);
        if (ksp != NULL) {
                kstat_rele(ksp);
                kstat_delete(ksp);
        }
}

void
kstat_delete_byname(const char *ks_module, int ks_instance, const char *ks_name)
{
        kstat_delete_byname_zone(ks_module, ks_instance, ks_name, ALL_ZONES);
}

/*
 * The sparc V9 versions of these routines can be much cheaper than
 * the poor 32-bit compiler can comprehend, so they're in sparcv9_subr.s.
 * For simplicity, however, we always feed the C versions to lint.
 */
#if !defined(__sparc) || defined(lint) || defined(__lint)

void
kstat_waitq_enter(kstat_io_t *kiop)
{
        hrtime_t new, delta;
        ulong_t wcnt;

        new = gethrtime_unscaled();
        delta = new - kiop->wlastupdate;
        kiop->wlastupdate = new;
        wcnt = kiop->wcnt++;
        if (wcnt != 0) {
                kiop->wlentime += delta * wcnt;
                kiop->wtime += delta;
        }
}

void
kstat_waitq_exit(kstat_io_t *kiop)
{
        hrtime_t new, delta;
        ulong_t wcnt;

        new = gethrtime_unscaled();
        delta = new - kiop->wlastupdate;
        kiop->wlastupdate = new;
        wcnt = kiop->wcnt--;
        ASSERT((int)wcnt > 0);
        kiop->wlentime += delta * wcnt;
        kiop->wtime += delta;
}
```

```
1390 void
1391 kstat_runq_enter(kstat_io_t *kiop)
1392 {
1393         hrtime_t new, delta;
1394         ulong_t rcnt;

1396         new = gethrtime_unscaled();
1397         delta = new - kiop->rlastupdate;
1398         kiop->rlastupdate = new;
1399         rcnt = kiop->rcnt++;
1400         if (rcnt != 0) {
1401                 kiop->rlentime += delta * rcnt;
1402                 kiop->rtime += delta;
1403         }
1404 }

1406 void
1407 kstat_runq_exit(kstat_io_t *kiop)
1408 {
1409         hrtime_t new, delta;
1410         ulong_t rcnt;

1412         new = gethrtime_unscaled();
1413         delta = new - kiop->rlastupdate;
1414         kiop->rlastupdate = new;
1415         rcnt = kiop->rcnt--;
1416         ASSERT((int)rcnt > 0);
1417         kiop->rlentime += delta * rcnt;
1418         kiop->rtime += delta;
1419 }

1421 void
1422 kstat_waitq_to_runq(kstat_io_t *kiop)
1423 {
1424         hrtime_t new, delta;
1425         ulong_t wcnt, rcnt;

1427         new = gethrtime_unscaled();

1429         delta = new - kiop->wlastupdate;
1430         kiop->wlastupdate = new;
1431         wcnt = kiop->wcnt--;
1432         ASSERT((int)wcnt > 0);
1433         kiop->wlentime += delta * wcnt;
1434         kiop->wtime += delta;

1436         delta = new - kiop->rlastupdate;
1437         kiop->rlastupdate = new;
1438         rcnt = kiop->rcnt++;
1439         if (rcnt != 0) {
1440                 kiop->rlentime += delta * rcnt;
1441                 kiop->rtime += delta;
1442         }
1443 }

1445 void
1446 kstat_runq_back_to_waitq(kstat_io_t *kiop)
1447 {
1448         hrtime_t new, delta;
1449         ulong_t wcnt, rcnt;

1451         new = gethrtime_unscaled();

1453         delta = new - kiop->rlastupdate;
1454         kiop->rlastupdate = new;
```



```
1455         rcnt = kiop->rcnt--;
1456         ASSERT((int)rcnt > 0);
1457         kiop->rlentime += delta * rcnt;
1458         kiop->rtime += delta;

1460         delta = new - kiop->wlastupdate;
1461         kiop->wlastupdate = new;
1462         wcnt = kiop->wcnt++;
1463         if (wcnt != 0) {
1464                 kiop->wlentime += delta * wcnt;
1465                 kiop->wtime += delta;
1466         }
1467 }

1469 #endif

1471 void
1472 kstat_timer_start(kstat_timer_t *ktp)
1473 {
1474         ktp->start_time = gethrtime();
1475 }

1477 void
1478 kstat_timer_stop(kstat_timer_t *ktp)
1479 {
1480         hrtime_t        etime;
1481         u_longlong_t    num_events;

1483         ktp->stop_time = etime = gethrtime();
1484         etime -= ktp->start_time;
1485         num_events = ktp->num_events;
1486         if (etime < ktp->min_time || num_events == 0)
1487                 ktp->min_time = etime;
1488         if (etime > ktp->max_time)
1489                 ktp->max_time = etime;
1490         ktp->elapsed_time += etime;
1491         ktp->num_events = num_events + 1;
1492 }
```