```
**********************************************************
   39929 Tue Jun 21 14:47:39 2016
new/usr/src/uts/common/fs/nfs/nfs4_srv_readdir.c
7123 encode timestamps in rfs4_op_readdir()
**********************************************************
______unchanged_portion_omitted_

#define IS_MIN_ATTR_MASK(m)     (((m) & ~MINIMAL_RD_ATTRS) == 0)
/*
 * If readdir only needs to return FILEID, we can take it from the
 * dirent struct and save doing the lookup.
 */
/* ARGSUSED */
void
rfs4_op_readdir(nfs_argop4 *argop, nfs_resop4 *resop,
        struct svc_req *req, struct compound_state *cs)
{
        READDIR4args *args = &argop->nfs_argop4_u.opreaddir;
        READDIR4res *resp = &resop->nfs_resop4_u.opreaddir;
        struct exportinfo *newexi = NULL;
        int error;
        mblk_t *mp;
        uint_t mpcount;
        int alloc_err = 0;
        vnode_t *dvp = cs->vp;
        vnode_t *vp;
        vattr_t va;
        struct dirent64 *dp;
        rfs4_sb_encode_t dsbe, sbe;
        int vfs_different;
        int rddir_data_len, rddir_result_size;
        caddr_t rddir_data;
        offset_t rddir_next_offset;
        int dircount;
        int no_space;
        int iseofdir;
        uint_t eof;
        struct iovec iov;
        struct uio uio;
        int tsize;
        int check_visible;
        int expseudo = 0;

        uint32_t *ptr, *ptr_redzone;
        uint32_t *beginning_ptr;
        uint32_t *lastentry_ptr;
        uint32_t *attrmask_ptr;
        uint32_t *attr_offset_ptr;
        uint32_t attr_length;
        uint32_t rndup;
        uint32_t namelen;
        uint32_t rddirattr_error = 0;
        int nents;
        bitmap4 ar = args->attr_request & NFS4_SRV_RDDIR_SUPPORTED_ATTRS;
        bitmap4 ae;
        rfs4_pc_encode_t dpce, pce;
        ulong_t pc_val;
        uint64_t maxread;
        uint64_t maxwrite;
        uint_t true = TRUE;
        uint_t false = FALSE;
        uid_t lastuid;
        gid_t lastgid;
        int lu_set, lg_set;
        utf8string owner, group;
        int owner_error, group_error;
```



```
        struct sockaddr *ca;
        char *name = NULL;

        DTRACE_NFSV4_2(op__readdir__start, struct compound_state *, cs,
            READDIR4args *, args);

        lu_set = lg_set = 0;
        owner.utf8string_len = group.utf8string_len = 0;
        owner.utf8string_val = group.utf8string_val = NULL;

        resp->mblk = NULL;

        /* Maximum read and write size */
        maxread = maxwrite = rfs4_tsize(req);

        if (dvp == NULL) {
                *cs->statusp = resp->status = NFS4ERR_NOFILEHANDLE;
                goto out;
        }

        /*
         * If there is an unshared filesystem mounted on this vnode,
         * do not allow readdir in this directory.
         */
        if (vn_ismntpt(dvp)) {
                *cs->statusp = resp->status = NFS4ERR_ACCESS;
                goto out;
        }

        if (dvp->v_type != VDIR) {
                *cs->statusp = resp->status = NFS4ERR_NOTDIR;
                goto out;
        }

        if (args->maxcount <= RFS4_MINLEN_RDDIR4) {
                *cs->statusp = resp->status = NFS4ERR_TOOSMALL;
                goto out;
        }

        /*
         * If write-only attrs are requested, then fail the readdir op
         */
        if (args->attr_request &
            (FATTR4_TIME_MODIFY_SET_MASK | FATTR4_TIME_ACCESS_SET_MASK)) {
                *cs->statusp = resp->status = NFS4ERR_INVAL;
                goto out;
        }

        error = VOP_ACCESS(dvp, VREAD, 0, cs->cr, NULL);
        if (error) {
                *cs->statusp = resp->status = puterrno4(error);
                goto out;
        }

        if (args->cookieverf != Readdir4verf) {
                *cs->statusp = resp->status = NFS4ERR_NOT_SAME;
                goto out;
        }

        /* Is there pseudo-fs work that is needed for this readdir? */
        check_visible = PSEUDO(cs->exi) ||
            ! is_exported_sec(cs->nfsflavor, cs->exi) ||
            cs->access & CS_ACCESS_LIMITED;

        /* Check the requested attributes and only do the work if needed */
```

```
503         if (ar & (FATTR4_MAXFILESIZE_MASK |
504             FATTR4_MAXLINK_MASK |
505             FATTR4_MAXNAME_MASK)) {
506                 if (error = rfs4_get_pc_encode(cs->vp, &dpce, ar, cs->cr)) {
507                         *cs->statusp = resp->status = puterrno4(error);
508                         goto out;
509                 }
510                 pce = dpce;
511         }

513         /* If there is statvfs data requested, pick it up once */
514         if (ar &
515             (FATTR4_FILES_AVAIL_MASK |
516             FATTR4_FILES_FREE_MASK |
517             FATTR4_FILES_TOTAL_MASK |
518             FATTR4_FILES_AVAIL_MASK |
519             FATTR4_FILES_FREE_MASK |
520             FATTR4_FILES_TOTAL_MASK)) {
521                 if (error = rfs4_get_sb_encode(dvp->v_vfsp, &dsbe)) {
522                         *cs->statusp = resp->status = puterrno4(error);
523                         goto out;
524                 }
525                 sbe = dsbe;
526         }

528         /*
529          * Max transfer size of the server is the absolute limite.
530          * If the client has decided to max out with something really
531          * tiny, then return toosmall.  Otherwise, move forward and
532          * see if a single entry can be encoded.
533          */
534         tsize = rfs4_tsize(req);
535         if (args->maxcount > tsize)
536                 args->maxcount = tsize;
537         else if (args->maxcount < RFS4_MINLEN_RDDIR_BUF) {
538                 if (args->maxcount < RFS4_MINLEN_ENTRY4) {
539                         *cs->statusp = resp->status = NFS4ERR_TOOSMALL;
540                         goto out;
541                 }
542         }

544         /*
545          * How large should the mblk be for outgoing encoding.
546          */
547         if (args->maxcount < MAXBSIZE)
548                 mpcount = MAXBSIZE;
549         else
550                 mpcount = args->maxcount;

552         /*
553          * mp will contain the data to be sent out in the readdir reply.
554          * It will be freed after the reply has been sent.
555          * Let's roundup the data to a BYTES_PER_XDR_UNIX multiple,
556          * so that the call to xdrmblk_putmblk() never fails.
557          */
558         mp = allocb(RNDUP(mpcount), BPRI_MED);

560         if (mp == NULL) {
561                 /*
562                  * The allocation of the client's requested size has
563                  * failed.  It may be that the size is too large for
564                  * current system utilization; step down to a "common"
565                  * size and wait for the allocation to occur.
566                  */
567                 if (mpcount > MAXBSIZE)
568                         args->maxcount = mpcount = MAXBSIZE;
```



```
569                 mp = allocb_wait(RNDUP(mpcount), BPRI_MED,
570                     STR_NOSIG, &alloc_err);
571         }

573         ASSERT(mp != NULL);
574         ASSERT(alloc_err == 0);

576         resp->mblk = mp;

578         ptr = beginning_ptr = (uint32_t *)mp->b_datap->db_base;

580         /*
581          * The "redzone" at the end of the encoding buffer is used
582          * to deal with xdr encoding length.  Instead of checking
583          * each encoding of an attribute value before it is done,
584          * make the assumption that it will fit into the buffer and
585          * check occasionally.
586          *
587          * The largest block of attributes that are encoded without
588          * checking the redzone is 18 * BYTES_PER_XDR_UNIT (72 bytes)
589          * "round" to 128 as the redzone size.
590          */
591         if (args->maxcount < (mpcount - 128))
592                 ptr_redzone =
593                     (uint32_t *)(((char *)ptr) + RNDUP(args->maxcount));
594         else
595                 ptr_redzone =
596                     (uint32_t *)((((char *)ptr) + RNDUP(mpcount)) - 128);

598         /*
599          * Set the dircount; this will be used as the size for the
600          * readdir of the underlying filesystem.  First make sure
601          * that it is large enough to do a reasonable readdir (client
602          * may have short changed us - it is an advisory number);
603          * then make sure that it isn't too large.
604          * After all of that, if maxcount is "small" then just use
605          * that for the dircount number.
606          */
607         dircount = (args->dircount < MAXBSIZE) ? MAXBSIZE : args->dircount;
608         dircount = (dircount > tsize) ? tsize : dircount;
609         if (dircount > args->maxcount)
610                 dircount = args->maxcount;
611         if (args->maxcount <= MAXBSIZE) {
612                 if (args->maxcount < RFS4_MINLEN_RDDIR_BUF)
613                         dircount = RFS4_MINLEN_RDDIR_BUF;
614                 else
615                         dircount = args->maxcount;
616         }

618         /* number of entries fully encoded in outgoing buffer */
619         nents = 0;

621         /* ENCODE READDIR4res.cookieverf */
622         IXDR_PUT_HYPER(ptr, Readdir4verf);

624         rddir_data_len = dircount;
625         rddir_data = kmem_alloc(rddir_data_len, KM_NOSLEEP);
626         if (rddir_data == NULL) {
627                 /* The allocation failed; downsize and wait for it this time */
628                 if (rddir_data_len > MAXBSIZE)
629                         rddir_data_len = dircount = MAXBSIZE;
630                 rddir_data = kmem_alloc(rddir_data_len, KM_SLEEP);
631         }

633         rddir_next_offset = (offset_t)args->cookie;
```

```
635         ca = (struct sockaddr *)svc_getrpccaller(req->rq_xprt)->buf;

637 readagain:

639         no_space = FALSE;
640         iseofdir = FALSE;

642         vp = NULL;

644         /* Move on to reading the directory contents */
645         iov.iov_base = rddir_data;
646         iov.iov_len = rddir_data_len;
647         uio.uio_iov = &iov;
648         uio.uio_iovcnt = 1;
649         uio.uio_segflg = UIO_SYSSPACE;
650         uio.uio_extflg = UIO_COPY_CACHED;
651         uio.uio_loffset = rddir_next_offset;
652         uio.uio_resid = rddir_data_len;

654         (void) VOP_RWLOCK(dvp, V_WRITELOCK_FALSE, NULL);

656         error = VOP_READDIR(dvp, &uio, cs->cr, &iseofdir, NULL, 0);

658         VOP_RWUNLOCK(dvp, V_WRITELOCK_FALSE, NULL);

660         if (error) {
661                 kmem_free((caddr_t)rddir_data, rddir_data_len);
662                 freeb(resp->mblk);
663                 resp->mblk = NULL;
664                 resp->data_len = 0;
665                 *cs->statusp = resp->status = puterrno4(error);
666                 goto out;
667         }


670         rddir_result_size = rddir_data_len - uio.uio_resid;

672         /* No data were read. Check if we reached the end of the directory. */
673         if (rddir_result_size == 0) {
674                 /* encode the BOOLEAN marking no further entries */
675                 IXDR_PUT_U_INT32(ptr, false);
676                 /* encode the BOOLEAN signifying end of directory */
677                 IXDR_PUT_U_INT32(ptr, iseofdir ? true : false);
678                 resp->data_len = (char *)ptr - (char *)beginning_ptr;
679                 resp->mblk->b_wptr += resp->data_len;
680                 kmem_free((caddr_t)rddir_data, rddir_data_len);
681                 *cs->statusp = resp->status = NFS4_OK;
682                 goto out;
683         }

685         lastentry_ptr = ptr;
686         no_space = 0;
687         for (dp = (struct dirent64 *)rddir_data;
688             !no_space && rddir_result_size > 0; dp = nextdp(dp)) {

690                 /* reset expseudo */
691                 expseudo = 0;

693                 if (vp) {
694                         VN_RELE(vp);
695                         vp = NULL;
696                 }

698                 if (newexi)
699                         newexi = NULL;
```



```
701                 rddir_result_size -= dp->d_reclen;

703                 /* skip "." and ".." entries */
704                 if (dp->d_ino == 0 || NFS_IS_DOTNAME(dp->d_name)) {
705                         rddir_next_offset = dp->d_off;
706                         continue;
707                 }

709                 if (check_visible &&
710                     !nfs_visible_inode(cs->exi, dp->d_ino, &expseudo)) {
711                         rddir_next_offset = dp->d_off;
712                         continue;
713                 }

715                 /*
716                  * Only if the client requested attributes...
717                  * If the VOP_LOOKUP fails ENOENT, then skip this entry
718                  * for the readdir response.  If there was another error,
719                  * then set the rddirattr_error and the error will be
720                  * encoded later in the "attributes" section.
721                  */
722                 ae = ar;
723                 if (ar == 0)
724                         goto reencode_attrs;

726                 error = nfs4_readdir_getvp(dvp, dp->d_name,
727                     &vp, &newexi, req, cs, expseudo);
728                 if (error == ENOENT) {
729                         rddir_next_offset = dp->d_off;
730                         continue;
731                 }

733                 rddirattr_error = error;

735                 /*
736                  * The vp obtained from above may be from a
737                  * different filesystem mount and the vfs-like
738                  * attributes should be obtained from that
739                  * different vfs; only do this if appropriate.
740                  */
741                 if (vp &&
742                     (vfs_different = (dvp->v_vfsp != vp->v_vfsp))) {
743                         if (ar & (FATTR4_FILES_AVAIL_MASK |
744                             FATTR4_FILES_FREE_MASK |
745                             FATTR4_FILES_TOTAL_MASK |
746                             FATTR4_FILES_AVAIL_MASK |
747                             FATTR4_FILES_FREE_MASK |
748                             FATTR4_FILES_TOTAL_MASK)) {
749                                 if (error =
750                                     rfs4_get_sb_encode(dvp->v_vfsp,
751                                     &sbe)) {
752                                         /* Remove attrs from encode */
753                                         ae &= ~(FATTR4_FILES_AVAIL_MASK |
754                                             FATTR4_FILES_FREE_MASK |
755                                             FATTR4_FILES_TOTAL_MASK |
756                                             FATTR4_FILES_AVAIL_MASK |
757                                             FATTR4_FILES_FREE_MASK |
758                                             FATTR4_FILES_TOTAL_MASK);
759                                         rddirattr_error = error;
760                                 }
761                         }
762                         if (ar & (FATTR4_MAXFILESIZE_MASK |
763                             FATTR4_MAXLINK_MASK |
764                             FATTR4_MAXNAME_MASK)) {
765                                 if (error = rfs4_get_pc_encode(cs->vp,
766                                     &pce, ar, cs->cr)) {
```

```
767                                     ar &= ~(FATTR4_MAXFILESIZE_MASK |
768                                         FATTR4_MAXLINK_MASK |
769                                         FATTR4_MAXNAME_MASK);
770                                     rddirattr_error = error;
771                                 }
772                         }
773                 }

775 reencode_attrs:
776                 /* encode the BOOLEAN for the existence of the next entry */
777                 IXDR_PUT_U_INT32(ptr, true);
778                 /* encode the COOKIE for the entry */
779                 IXDR_PUT_U_HYPER(ptr, dp->d_off);

781                 name = nfscmd_convname(ca, cs->exi, dp->d_name,
782                     NFSCMD_CONV_OUTBOUND, MAXPATHLEN + 1);

784                 if (name == NULL) {
785                         rddir_next_offset = dp->d_off;
786                         continue;
787                 }
788                 /* Calculate the dirent name length */
789                 namelen = strlen(name);

791                 rndup = RNDUP(namelen) / BYTES_PER_XDR_UNIT;

793                 /* room for LENGTH + string ? */
794                 if ((ptr + (1 + rndup)) > ptr_redzone) {
795                         no_space = TRUE;
796                         continue;
797                 }

799                 /* encode the LENGTH of the name */
800                 IXDR_PUT_U_INT32(ptr, namelen);
801                 /* encode the RNDUP FILL first */
802                 ptr[rndup - 1] = 0;
803                 /* encode the NAME of the entry */
804                 bcopy(name, (char *)ptr, namelen);
805                 /* now bump the ptr after... */
806                 ptr += rndup;

808                 if (name != dp->d_name)
809                         kmem_free(name, MAXPATHLEN + 1);

811                 /*
812                  * Keep checking on the dircount to see if we have
813                  * reached the limit; from the RFC, dircount is to be
814                  * the XDR encoded limit of the cookie plus name.
815                  * So the count is the name, XDR_UNIT of length for
816                  * that name and 2 * XDR_UNIT bytes of cookie;
817                  * However, use the regular DIRENT64 to match most
818                  * client's APIs.
819                  */
820                 dircount -= DIRENT64_RECLEN(namelen);
821                 if (nents != 0 && dircount < 0) {
822                         no_space = TRUE;
823                         continue;
824                 }

826                 /*
827                  * Attributes requested?
828                  * Gather up the attribute info and the previous VOP_LOOKUP()
829                  * succeeded; if an error occurs on the VOP_GETATTR() then
830                  * return just the error (again if it is requested).
831                  * Note that the previous VOP_LOOKUP() could have failed
832                  * itself which leaves this code without anything for
```



```
833                  * a VOP_GETATTR().
834                  * Also note that the readdir_attr_error is left in the
835                  * encoding mask if requested and so is the mounted_on_fileid.
836                  */
837                 if (ae != 0) {
838                         if (!vp) {
839                                 ae = ar & (FATTR4_RDATTR_ERROR_MASK |
840                                     FATTR4_MOUNTED_ON_FILEID_MASK);
841                         } else {
842                                 va.va_mask = AT_ALL;
843                                 rddirattr_error =
844                                     VOP_GETATTR(vp, &va, 0, cs->cr, NULL);
845                                 if (rddirattr_error) {
846                                         ae = ar & (FATTR4_RDATTR_ERROR_MASK |
847                                             FATTR4_MOUNTED_ON_FILEID_MASK);
848                                 } else {
849                                         /*
850                                          * We may lie about the object
851                                          * type for a referral
852                                          */
853                                         if (vn_is_nfs_reparse(vp, cs->cr) &&
854                                             client_is_downrev(req))
855                                                 va.va_type = VLNK;
856                                 }
857                         }
858                 }

860                 /* START OF ATTRIBUTE ENCODING */

862                 /* encode the LENGTH of the BITMAP4 array */
863                 IXDR_PUT_U_INT32(ptr, 2);
864                 /* encode the BITMAP4 */
865                 attrmask_ptr = ptr;
866                 IXDR_PUT_HYPER(ptr, ae);
867                 attr_offset_ptr = ptr;
868                 /* encode the default LENGTH of the attributes for entry */
869                 IXDR_PUT_U_INT32(ptr, 0);

871                 if (ptr > ptr_redzone) {
872                         no_space = TRUE;
873                         continue;
874                 }

876                 /* Check if any of the first 32 attributes are being encoded */
877                 if (ae & 0xffffffff00000000) {
878                         /*
879                          * Redzone check is done at the end of this section.
880                          * This particular section will encode a maximum of
881                          * 18 * BYTES_PER_XDR_UNIT of data
882                          */
883                         if (ae &
884                             (FATTR4_SUPPORTED_ATTRS_MASK |
885                             FATTR4_TYPE_MASK |
886                             FATTR4_FH_EXPIRE_TYPE_MASK |
887                             FATTR4_CHANGE_MASK |
888                             FATTR4_SIZE_MASK |
889                             FATTR4_LINK_SUPPORT_MASK |
890                             FATTR4_SYMLINK_SUPPORT_MASK |
891                             FATTR4_NAMED_ATTR_MASK |
892                             FATTR4_FSID_MASK |
893                             FATTR4_UNIQUE_HANDLES_MASK |
894                             FATTR4_LEASE_TIME_MASK |
895                             FATTR4_RDATTR_ERROR_MASK)) {

897                                 if (ae & FATTR4_SUPPORTED_ATTRS_MASK) {
898                                         IXDR_PUT_INT32(ptr, 2);
```

```
899                                     IXDR_PUT_HYPER(ptr,
900                                         rfs4_supported_attrs);
901                             }
902                             if (ae & FATTR4_TYPE_MASK) {
903                                     uint_t ftype = vt_to_nf4[va.va_type];
904                                     if (dvp->v_flag & V_XATTRDIR) {
905                                             if (va.va_type == VDIR)
906                                                     ftype = NF4ATTRDIR;
907                                             else
908                                                     ftype = NF4NAMEDATTR;
909                                     }
910                                     IXDR_PUT_U_INT32(ptr, ftype);
911                             }
912                             if (ae & FATTR4_FH_EXPIRE_TYPE_MASK) {
913                                     uint_t expire_type = FH4_PERSISTENT;
914                                     IXDR_PUT_U_INT32(ptr, expire_type);
915                             }
916                             if (ae & FATTR4_CHANGE_MASK) {
917                                     u_longlong_t change;
918                                     NFS4_SET_FATTR4_CHANGE(change,
919                                         va.va_ctime);
920                                     IXDR_PUT_HYPER(ptr, change);
921                             }
922                             if (ae & FATTR4_SIZE_MASK) {
923                                     u_longlong_t size = va.va_size;
924                                     IXDR_PUT_HYPER(ptr, size);
925                             }
926                             if (ae & FATTR4_LINK_SUPPORT_MASK) {
927                                     IXDR_PUT_U_INT32(ptr, true);
928                             }
929                             if (ae & FATTR4_SYMLINK_SUPPORT_MASK) {
930                                     IXDR_PUT_U_INT32(ptr, true);
931                             }
932                             if (ae & FATTR4_NAMED_ATTR_MASK) {
933                                     uint_t isit;
934                                     pc_val = FALSE;
935                                     int sattr_error;

937                                     if (!(vp->v_vfsp->vfs_flag &
938                                         VFS_XATTR)) {
939                                             isit = FALSE;
940                                     } else {
941                                             sattr_error = VOP_PATHCONF(vp,
942                                                 _PC_SATTR_EXISTS,
943                                                 &pc_val, cs->cr, NULL);
944                                             if (sattr_error || pc_val == 0)
945                                                     (void) VOP_PATHCONF(vp,
946                                                         _PC_XATTR_EXISTS,
947                                                         &pc_val,
948                                                         cs->cr, NULL);
949                                     }
950                                     isit = (pc_val ? TRUE : FALSE);
951                                     IXDR_PUT_U_INT32(ptr, isit);
952                             }
953                             if (ae & FATTR4_FSID_MASK) {
954                                     u_longlong_t major, minor;
955                                     struct exportinfo *exi;

957                                     exi = newexi ? newexi : cs->exi;
958                                     if (exi->exi_volatile_dev) {
959                                             int *pmaj = (int *)&major;

961                                             pmaj[0] = exi->exi_fsid.val[0];
962                                             pmaj[1] = exi->exi_fsid.val[1];
963                                             minor = 0;
964                                     } else {
```



```
965                                             major = getmajor(va.va_fsid);
966                                             minor = getminor(va.va_fsid);
967                                     }
968                                     IXDR_PUT_HYPER(ptr, major);
969                                     IXDR_PUT_HYPER(ptr, minor);
970                             }
971                             if (ae & FATTR4_UNIQUE_HANDLES_MASK) {
972                                     IXDR_PUT_U_INT32(ptr, false);
973                             }
974                             if (ae & FATTR4_LEASE_TIME_MASK) {
975                                     uint_t lt = rfs4_lease_time;
976                                     IXDR_PUT_U_INT32(ptr, lt);
977                             }
978                             if (ae & FATTR4_RDATTR_ERROR_MASK) {
979                                     rddirattr_error =
980                                         (rddirattr_error == 0 ?
981                                         0 : puterrno4(rddirattr_error));
982                                     IXDR_PUT_U_INT32(ptr, rddirattr_error);
983                             }

985                             /* Check the redzone boundary */
986                             if (ptr > ptr_redzone) {
987                                     if (nents || IS_MIN_ATTR_MASK(ar)) {
988                                             no_space = TRUE;
989                                             continue;
990                                     }
991                                     MINIMIZE_ATTR_MASK(ar);
992                                     ae = ar;
993                                     ptr = lastentry_ptr;
994                                     goto reencode_attrs;
995                             }
996                     }
997                     /*
998                      * Redzone check is done at the end of this section.
999                      * This particular section will encode a maximum of
1000                     * 4 * BYTES_PER_XDR_UNIT of data.
1001                     * NOTE: that if ACLs are supported that the
1002                     * redzone calculations will need to change.
1003                     */
1004                    if (ae &
1005                        (FATTR4_ACL_MASK |
1006                        FATTR4_ACLSUPPORT_MASK |
1007                        FATTR4_ARCHIVE_MASK |
1008                        FATTR4_CANSETTIME_MASK |
1009                        FATTR4_CASE_INSENSITIVE_MASK |
1010                        FATTR4_CASE_PRESERVING_MASK |
1011                        FATTR4_CHOWN_RESTRICTED_MASK)) {

1013                            if (ae & FATTR4_ACL_MASK) {
1014                                    ASSERT(0);
1015                            }
1016                            if (ae & FATTR4_ACLSUPPORT_MASK) {
1017                                    ASSERT(0);
1018                            }
1019                            if (ae & FATTR4_ARCHIVE_MASK) {
1020                                    ASSERT(0);
1021                            }
1022                            if (ae & FATTR4_CANSETTIME_MASK) {
1023                                    IXDR_PUT_U_INT32(ptr, true);
1024                            }
1025                            if (ae & FATTR4_CASE_INSENSITIVE_MASK) {
1026                                    IXDR_PUT_U_INT32(ptr, false);
1027                            }
1028                            if (ae & FATTR4_CASE_PRESERVING_MASK) {
1029                                    IXDR_PUT_U_INT32(ptr, true);
1030                            }
```

```
1031                                if (ae & FATTR4_CHOWN_RESTRICTED_MASK) {
1032                                        uint_t isit;
1033                                        pc_val = FALSE;
1034                                        (void) VOP_PATHCONF(vp,
1035                                            _PC_CHOWN_RESTRICTED,
1036                                            &pc_val, cs->cr, NULL);
1037                                        isit = (pc_val ? TRUE : FALSE);
1038                                        IXDR_PUT_U_INT32(ptr, isit);
1039                                }
1040                                /* Check the redzone boundary */
1041                                if (ptr > ptr_redzone) {
1042                                        if (nents || IS_MIN_ATTR_MASK(ar)) {
1043                                                no_space = TRUE;
1044                                                continue;
1045                                        }
1046                                        MINIMIZE_ATTR_MASK(ar);
1047                                        ae = ar;
1048                                        ptr = lastentry_ptr;
1049                                        goto reencode_attrs;
1050                                }
1051                        }
1052                        /*
1053                         * Redzone check is done before the filehandle
1054                         * is encoded.
1055                         */
1056                        if (ae &
1057                            (FATTR4_FILEHANDLE_MASK |
1058                            FATTR4_FILEID_MASK)) {

1060                                if (ae & FATTR4_FILEHANDLE_MASK) {
1061                                        struct {
1062                                                uint_t len;
1063                                                char *val;
1064                                                char fh[NFS_FH4_LEN];
1065                                        } fh;
1066                                        fh.len = 0;
1067                                        fh.val = fh.fh;
1068                                        (void) makefh4((nfs_fh4 *)&fh, vp,
1069                                            (newexi ? newexi : cs->exi));

1071                                        if (dvp->v_flag & V_XATTRDIR)
1072                                                set_fh4_flag((nfs_fh4 *)&fh,
1073                                                    FH4_NAMEDATTR);

1075                                        if (!xdr_inline_encode_nfs_fh4(
1076                                            &ptr, ptr_redzone,
1077                                            (nfs_fh4_fmt_t *)fh.val)) {
1078                                                if (nents ||
1079                                                    IS_MIN_ATTR_MASK(ar)) {
1080                                                        no_space = TRUE;
1081                                                        continue;
1082                                                }
1083                                                MINIMIZE_ATTR_MASK(ar);
1084                                                ae = ar;
1085                                                ptr = lastentry_ptr;
1086                                                goto reencode_attrs;
1087                                        }
1088                                }
1089                                if (ae & FATTR4_FILEID_MASK) {
1090                                        IXDR_PUT_HYPER(ptr, va.va_nodeid);
1091                                }
1092                                /* Check the redzone boundary */
1093                                if (ptr > ptr_redzone) {
1094                                        if (nents || IS_MIN_ATTR_MASK(ar)) {
1095                                                no_space = TRUE;
1096                                                continue;
```



```
1097                                        }
1098                                        MINIMIZE_ATTR_MASK(ar);
1099                                        ae = ar;
1100                                        ptr = lastentry_ptr;
1101                                        goto reencode_attrs;
1102                                }
1103                        }
1104                        /*
1105                         * Redzone check is done at the end of this section.
1106                         * This particular section will encode a maximum of
1107                         * 15 * BYTES_PER_XDR_UNIT of data.
1108                         */
1109                        if (ae &
1110                            (FATTR4_FILES_AVAIL_MASK |
1111                            FATTR4_FILES_FREE_MASK |
1112                            FATTR4_FILES_TOTAL_MASK |
1113                            FATTR4_FS_LOCATIONS_MASK |
1114                            FATTR4_HIDDEN_MASK |
1115                            FATTR4_HOMOGENEOUS_MASK |
1116                            FATTR4_MAXFILESIZE_MASK |
1117                            FATTR4_MAXLINK_MASK |
1118                            FATTR4_MAXNAME_MASK |
1119                            FATTR4_MAXREAD_MASK |
1120                            FATTR4_MAXWRITE_MASK)) {

1122                                if (ae & FATTR4_FILES_AVAIL_MASK) {
1123                                        IXDR_PUT_HYPER(ptr, sbe.fa);
1124                                }
1125                                if (ae & FATTR4_FILES_FREE_MASK) {
1126                                        IXDR_PUT_HYPER(ptr, sbe.ff);
1127                                }
1128                                if (ae & FATTR4_FILES_TOTAL_MASK) {
1129                                        IXDR_PUT_HYPER(ptr, sbe.ft);
1130                                }
1131                                if (ae & FATTR4_FS_LOCATIONS_MASK) {
1132                                        ASSERT(0);
1133                                }
1134                                if (ae & FATTR4_HIDDEN_MASK) {
1135                                        ASSERT(0);
1136                                }
1137                                if (ae & FATTR4_HOMOGENEOUS_MASK) {
1138                                        IXDR_PUT_U_INT32(ptr, true);
1139                                }
1140                                if (ae & FATTR4_MAXFILESIZE_MASK) {
1141                                        IXDR_PUT_HYPER(ptr, pce.maxfilesize);
1142                                }
1143                                if (ae & FATTR4_MAXLINK_MASK) {
1144                                        IXDR_PUT_U_INT32(ptr, pce.maxlink);
1145                                }
1146                                if (ae & FATTR4_MAXNAME_MASK) {
1147                                        IXDR_PUT_U_INT32(ptr, pce.maxname);
1148                                }
1149                                if (ae & FATTR4_MAXREAD_MASK) {
1150                                        IXDR_PUT_HYPER(ptr, maxread);
1151                                }
1152                                if (ae & FATTR4_MAXWRITE_MASK) {
1153                                        IXDR_PUT_HYPER(ptr, maxwrite);
1154                                }
1155                                /* Check the redzone boundary */
1156                                if (ptr > ptr_redzone) {
1157                                        if (nents || IS_MIN_ATTR_MASK(ar)) {
1158                                                no_space = TRUE;
1159                                                continue;
1160                                        }
1161                                        MINIMIZE_ATTR_MASK(ar);
1162                                        ae = ar;
```

```
1163                                        ptr = lastentry_ptr;
1164                                        goto reencode_attrs;
1165                                }
1166                        }
1167                }
1168                if (ae & 0x00000000ffffffff) {
1169                        /*
1170                         * Redzone check is done at the end of this section.
1171                         * This particular section will encode a maximum of
1172                         * 3 * BYTES_PER_XDR_UNIT of data.
1173                         */
1174                        if (ae &
1175                            (FATTR4_MIMETYPE_MASK |
1176                            FATTR4_MODE_MASK |
1177                            FATTR4_NO_TRUNC_MASK |
1178                            FATTR4_NUMLINKS_MASK)) {

1180                                if (ae & FATTR4_MIMETYPE_MASK) {
1181                                        ASSERT(0);
1182                                }
1183                                if (ae & FATTR4_MODE_MASK) {
1184                                        uint_t m = va.va_mode;
1185                                        IXDR_PUT_U_INT32(ptr, m);
1186                                }
1187                                if (ae & FATTR4_NO_TRUNC_MASK) {
1188                                        IXDR_PUT_U_INT32(ptr, true);
1189                                }
1190                                if (ae & FATTR4_NUMLINKS_MASK) {
1191                                        IXDR_PUT_U_INT32(ptr, va.va_nlink);
1192                                }
1193                                /* Check the redzone boundary */
1194                                if (ptr > ptr_redzone) {
1195                                        if (nents || IS_MIN_ATTR_MASK(ar)) {
1196                                                no_space = TRUE;
1197                                                continue;
1198                                        }
1199                                        MINIMIZE_ATTR_MASK(ar);
1200                                        ae = ar;
1201                                        ptr = lastentry_ptr;
1202                                        goto reencode_attrs;
1203                                }
1204                        }
1205                        /*
1206                         * Redzone check is done before the encoding of the
1207                         * owner string since the length is indeterminate.
1208                         */
1209                        if (ae & FATTR4_OWNER_MASK) {
1210                                if (!lu_set) {
1211                                        owner_error = nfs_idmap_uid_str(
1212                                            va.va_uid, &owner, TRUE);
1213                                        if (!owner_error) {
1214                                                lu_set = TRUE;
1215                                                lastuid = va.va_uid;
1216                                        }
1217                                } else  if (va.va_uid != lastuid) {
1218                                        if (owner.utf8string_len != 0) {
1219                                                kmem_free(owner.utf8string_val,
1220                                                    owner.utf8string_len);
1221                                                owner.utf8string_len = 0;
1222                                                owner.utf8string_val = NULL;
1223                                        }
1224                                        owner_error = nfs_idmap_uid_str(
1225                                            va.va_uid, &owner, TRUE);
1226                                        if (!owner_error) {
1227                                                lastuid = va.va_uid;
1228                                        } else {
```



```
1229                                                lu_set = FALSE;
1230                                        }
1231                                }
1232                                if (!owner_error) {
1233                                        if ((ptr +
1234                                            (owner.utf8string_len /
1235                                            BYTES_PER_XDR_UNIT)
1236                                            + 2) > ptr_redzone) {
1237                                                if (nents ||
1238                                                    IS_MIN_ATTR_MASK(ar)) {
1239                                                        no_space = TRUE;
1240                                                        continue;
1241                                                }
1242                                                MINIMIZE_ATTR_MASK(ar);
1243                                                ae = ar;
1244                                                ptr = lastentry_ptr;
1245                                                goto reencode_attrs;
1246                                        }
1247                                        /* encode the LENGTH of owner string */
1248                                        IXDR_PUT_U_INT32(ptr,
1249                                            owner.utf8string_len);
1250                                        /* encode the RNDUP FILL first */
1251                                        rndup = RNDUP(owner.utf8string_len) /
1252                                            BYTES_PER_XDR_UNIT;
1253                                        ptr[rndup - 1] = 0;
1254                                        /* encode the OWNER */
1255                                        bcopy(owner.utf8string_val, ptr,
1256                                            owner.utf8string_len);
1257                                        ptr += rndup;
1258                                }
1259                        }
1260                        /*
1261                         * Redzone check is done before the encoding of the
1262                         * group string since the length is indeterminate.
1263                         */
1264                        if (ae & FATTR4_OWNER_GROUP_MASK) {
1265                                if (!lg_set) {
1266                                        group_error =
1267                                            nfs_idmap_gid_str(va.va_gid,
1268                                            &group, TRUE);
1269                                        if (!group_error) {
1270                                                lg_set = TRUE;
1271                                                lastgid = va.va_gid;
1272                                        }
1273                                } else if (va.va_gid != lastgid) {
1274                                        if (group.utf8string_len != 0) {
1275                                                kmem_free(
1276                                                    group.utf8string_val,
1277                                                    group.utf8string_len);
1278                                                group.utf8string_len = 0;
1279                                                group.utf8string_val = NULL;
1280                                        }
1281                                        group_error =
1282                                            nfs_idmap_gid_str(va.va_gid,
1283                                            &group, TRUE);
1284                                        if (!group_error)
1285                                                lastgid = va.va_gid;
1286                                        else
1287                                                lg_set = FALSE;
1288                                }
1289                                if (!group_error) {
1290                                        if ((ptr +
1291                                            (group.utf8string_len /
1292                                            BYTES_PER_XDR_UNIT)
1293                                            + 2) > ptr_redzone) {
1294                                                if (nents ||
```

```
1295                                            IS_MIN_ATTR_MASK(ar)) {
1296                                                no_space = TRUE;
1297                                                continue;
1298                                        }
1299                                        MINIMIZE_ATTR_MASK(ar);
1300                                        ae = ar;
1301                                        ptr = lastentry_ptr;
1302                                        goto reencode_attrs;
1303                                }
1304                                /* encode the LENGTH of owner string */
1305                                IXDR_PUT_U_INT32(ptr,
1306                                    group.utf8string_len);
1307                                /* encode the RNDUP FILL first */
1308                                rndup = RNDUP(group.utf8string_len) /
1309                                    BYTES_PER_XDR_UNIT;
1310                                ptr[rndup - 1] = 0;
1311                                /* encode the OWNER */
1312                                bcopy(group.utf8string_val, ptr,
1313                                    group.utf8string_len);
1314                                ptr += rndup;
1315                        }
1316                }
1317                if (ae &
1318                    (FATTR4_QUOTA_AVAIL_HARD_MASK |
1319                    FATTR4_QUOTA_AVAIL_SOFT_MASK |
1320                    FATTR4_QUOTA_USED_MASK)) {
1321                        if (ae & FATTR4_QUOTA_AVAIL_HARD_MASK) {
1322                                ASSERT(0);
1323                        }
1324                        if (ae & FATTR4_QUOTA_AVAIL_SOFT_MASK) {
1325                                ASSERT(0);
1326                        }
1327                        if (ae & FATTR4_QUOTA_USED_MASK) {
1328                                ASSERT(0);
1329                        }
1330                }
1331                /*
1332                 * Redzone check is done at the end of this section.
1333                 * This particular section will encode a maximum of
1334                 * 10 * BYTES_PER_XDR_UNIT of data.
1335                 */
1336                if (ae &
1337                    (FATTR4_RAWDEV_MASK |
1338                    FATTR4_SPACE_AVAIL_MASK |
1339                    FATTR4_SPACE_FREE_MASK |
1340                    FATTR4_SPACE_TOTAL_MASK |
1341                    FATTR4_SPACE_USED_MASK |
1342                    FATTR4_SYSTEM_MASK)) {

1344                        if (ae & FATTR4_RAWDEV_MASK) {
1345                                fattr4_rawdev rd;
1346                                rd.specdata1 =
1347                                    (uint32)getmajor(va.va_rdev);
1348                                rd.specdata2 =
1349                                    (uint32)getminor(va.va_rdev);
1350                                IXDR_PUT_U_INT32(ptr, rd.specdata1);
1351                                IXDR_PUT_U_INT32(ptr, rd.specdata2);
1352                        }
1353                        if (ae & FATTR4_SPACE_AVAIL_MASK) {
1354                                IXDR_PUT_HYPER(ptr, sbe.space_avail);
1355                        }
1356                        if (ae & FATTR4_SPACE_FREE_MASK) {
1357                                IXDR_PUT_HYPER(ptr, sbe.space_free);
1358                        }
1359                        if (ae & FATTR4_SPACE_TOTAL_MASK) {
1360                                IXDR_PUT_HYPER(ptr, sbe.space_total);
```



```
1361                        }
1362                        if (ae & FATTR4_SPACE_USED_MASK) {
1363                                u_longlong_t su;
1364                                su = (fattr4_space_used) DEV_BSIZE *
1365                                    (fattr4_space_used) va.va_nblocks;
1366                                IXDR_PUT_HYPER(ptr, su);
1367                        }
1368                        if (ae & FATTR4_SYSTEM_MASK) {
1369                                ASSERT(0);
1370                        }
1371                        /* Check the redzone boundary */
1372                        if (ptr > ptr_redzone) {
1373                                if (nents || IS_MIN_ATTR_MASK(ar)) {
1374                                        no_space = TRUE;
1375                                        continue;
1376                                }
1377                                MINIMIZE_ATTR_MASK(ar);
1378                                ae = ar;
1379                                ptr = lastentry_ptr;
1380                                goto reencode_attrs;
1381                        }
1382                }
1383                /*
1384                 * Redzone check is done at the end of this section.
1385                 * This particular section will encode a maximum of
1386                 * 14 * BYTES_PER_XDR_UNIT of data.
1387                 */
1388                if (ae &
1389                    (FATTR4_TIME_ACCESS_MASK |
1390                    FATTR4_TIME_ACCESS_SET_MASK |
1391                    FATTR4_TIME_BACKUP_MASK |
1392                    FATTR4_TIME_CREATE_MASK |
1393                    FATTR4_TIME_DELTA_MASK |
1394                    FATTR4_TIME_METADATA_MASK |
1395                    FATTR4_TIME_MODIFY_MASK |
1396                    FATTR4_TIME_MODIFY_SET_MASK |
1397                    FATTR4_MOUNTED_ON_FILEID_MASK)) {

1399                        if (ae & FATTR4_TIME_ACCESS_MASK) {
1400                                nfstime4 atime;
1401                                (void) nfs4_time_vton(&va.va_atime,
1402                                    &atime);
1403                                IXDR_PUT_HYPER(ptr, atime.seconds);
1404                                IXDR_PUT_INT32(ptr, atime.nseconds);
1400                                u_longlong_t sec =
1401                                    (u_longlong_t)va.va_atime.tv_sec;
1402                                uint_t nsec =
1403                                    (uint_t)va.va_atime.tv_nsec;
1404                                IXDR_PUT_HYPER(ptr, sec);
1405                                IXDR_PUT_INT32(ptr, nsec);
1405                        }
1406                        if (ae & FATTR4_TIME_ACCESS_SET_MASK) {
1407                                ASSERT(0);
1408                        }
1409                        if (ae & FATTR4_TIME_BACKUP_MASK) {
1410                                ASSERT(0);
1411                        }
1412                        if (ae & FATTR4_TIME_CREATE_MASK) {
1413                                ASSERT(0);
1414                        }
1415                        if (ae & FATTR4_TIME_DELTA_MASK) {
1416                                u_longlong_t sec = 0;
1417                                uint_t nsec = 1000;
1418                                IXDR_PUT_HYPER(ptr, sec);
1419                                IXDR_PUT_INT32(ptr, nsec);
1420                        }
```

```
1421                            if (ae & FATTR4_TIME_METADATA_MASK) {
1422                                    nfstime4 ctime;
1423                                    (void) nfs4_time_vton(&va.va_ctime,
1424                                        &ctime);
1425                                    IXDR_PUT_HYPER(ptr, ctime.seconds);
1426                                    IXDR_PUT_INT32(ptr, ctime.nseconds);
1423                                    u_longlong_t sec =
1424                                        (u_longlong_t)va.va_ctime.tv_sec;
1425                                    uint_t nsec =
1426                                        (uint_t)va.va_ctime.tv_nsec;
1427                                    IXDR_PUT_HYPER(ptr, sec);
1428                                    IXDR_PUT_INT32(ptr, nsec);
1427                            }
1428                            if (ae & FATTR4_TIME_MODIFY_MASK) {
1429                                    nfstime4 mtime;
1430                                    (void) nfs4_time_vton(&va.va_mtime,
1431                                        &mtime);
1432                                    IXDR_PUT_HYPER(ptr, mtime.seconds);
1433                                    IXDR_PUT_INT32(ptr, mtime.nseconds);
1431                                    u_longlong_t sec =
1432                                        (u_longlong_t)va.va_mtime.tv_sec;
1433                                    uint_t nsec =
1434                                        (uint_t)va.va_mtime.tv_nsec;
1435                                    IXDR_PUT_HYPER(ptr, sec);
1436                                    IXDR_PUT_INT32(ptr, nsec);
1434                            }
1435                            if (ae & FATTR4_TIME_MODIFY_SET_MASK) {
1436                                    ASSERT(0);
1437                            }
1438                            if (ae & FATTR4_MOUNTED_ON_FILEID_MASK) {
1439                                    IXDR_PUT_HYPER(ptr, dp->d_ino);
1440                            }
1441                            /* Check the redzone boundary */
1442                            if (ptr > ptr_redzone) {
1443                                    if (nents || IS_MIN_ATTR_MASK(ar)) {
1444                                            no_space = TRUE;
1445                                            continue;
1446                                    }
1447                                    MINIMIZE_ATTR_MASK(ar);
1448                                    ae = ar;
1449                                    ptr = lastentry_ptr;
1450                                    goto reencode_attrs;
1451                            }
1452                    }
1453            }

1455            /* Reset to directory's vfs info when encoding complete */
1456            if (vfs_different) {
1457                    dsbe = sbe;
1458                    dpce = pce;
1459                    vfs_different = 0;
1460            }

1462            /* "go back" and encode the attributes' length */
1463            attr_length =
1464                (char *)ptr -
1465                (char *)attr_offset_ptr -
1466                BYTES_PER_XDR_UNIT;
1467            IXDR_PUT_U_INT32(attr_offset_ptr, attr_length);

1469            /*
1470             * If there was trouble obtaining a mapping for either
1471             * the owner or group attributes, then remove them from
1472             * bitmap4 for this entry and reset the bitmap value
1473             * in the data stream.
1474             */
```



```
1475                    if (owner_error || group_error) {
1476                            if (owner_error)
1477                                    ae &= ~FATTR4_OWNER_MASK;
1478                            if (group_error)
1479                                    ae &= ~FATTR4_OWNER_GROUP_MASK;
1480                            IXDR_PUT_HYPER(attrmask_ptr, ae);
1481                    }

1483                    /* END OF ATTRIBUTE ENCODING */

1485                    lastentry_ptr = ptr;
1486                    nents++;
1487                    rddir_next_offset = dp->d_off;
1488            }

1490            /*
1491             * Check for the case that another VOP_READDIR() has to be done.
1492             * - no space encoding error
1493             * - no entry successfully encoded
1494             * - still more directory to read
1495             */
1496            if (!no_space && nents == 0 && !iseofdir)
1497                    goto readagain;

1499            *cs->statusp = resp->status = NFS4_OK;

1501            /*
1502             * If no_space is set then we terminated prematurely,
1503             * rewind to the last entry and this can never be EOF.
1504             */
1505            if (no_space) {
1506                    ptr = lastentry_ptr;
1507                    eof = FALSE; /* ended encoded prematurely */
1508            } else {
1509                    eof = (iseofdir ? TRUE : FALSE);
1510            }

1512            /*
1513             * If we have entries, always return them, otherwise only error
1514             * if we ran out of space.
1515             */
1516            if (nents || !no_space) {
1517                    ASSERT(ptr != NULL);
1518                    /* encode the BOOLEAN marking no further entries */
1519                    IXDR_PUT_U_INT32(ptr, false);
1520                    /* encode the BOOLEAN signifying end of directory */
1521                    IXDR_PUT_U_INT32(ptr, eof);

1523                    resp->data_len = (char *)ptr - (char *)beginning_ptr;
1524                    resp->mblk->b_wptr += resp->data_len;
1525            } else {
1526                    freeb(mp);
1527                    resp->mblk = NULL;
1528                    resp->data_len = 0;
1529                    *cs->statusp = resp->status = NFS4ERR_TOOSMALL;
1530            }

1532            kmem_free((caddr_t)rddir_data, rddir_data_len);
1533            if (vp)
1534                    VN_RELE(vp);
1535            if (owner.utf8string_len != 0)
1536                    kmem_free(owner.utf8string_val, owner.utf8string_len);
1537            if (group.utf8string_len != 0)
1538                    kmem_free(group.utf8string_val, group.utf8string_len);

1540 out:
```

```
1541         DTRACE_NFSV4_2(op__readdir__done, struct compound_state *, cs,
1542             READDIR4res *, resp);
1543 }
```

unchanged_portion_omitted_