```
**********************************************************
   67781 Mon Sep 14 07:48:18 2015
new/usr/src/cmd/truss/print.c
6227 truss(1M) is not showing TCP_KEEPIDLE, TCP_KEEPCNT, and TCP_KEEPINTVL TCP o
Reviewed by: Dan McDonald <danmcd@omniti.com>
Reviewed by: Richard PALO <richard@netbsd.org>
Reviewed by: Toomas Soome <tsoome@me.com>
**********************************************************
______unchanged_portion_omitted_

const char *
tcp_optname(private_t *pri, long val)
{
        switch (val) {
        case TCP_NODELAY:               return ("TCP_NODELAY");
        case TCP_MAXSEG:                return ("TCP_MAXSEG");
        case TCP_KEEPALIVE:             return ("TCP_KEEPALIVE");
        case TCP_NOTIFY_THRESHOLD:      return ("TCP_NOTIFY_THRESHOLD");
        case TCP_ABORT_THRESHOLD:       return ("TCP_ABORT_THRESHOLD");
        case TCP_CONN_NOTIFY_THRESHOLD: return ("TCP_CONN_NOTIFY_THRESHOLD");
        case TCP_CONN_ABORT_THRESHOLD:  return ("TCP_CONN_ABORT_THRESHOLD");
        case TCP_RECVDSTADDR:           return ("TCP_RECVDSTADDR");
        case TCP_ANONPRIVBIND:          return ("TCP_ANONPRIVBIND");
        case TCP_EXCLBIND:              return ("TCP_EXCLBIND");
        case TCP_INIT_CWND:             return ("TCP_INIT_CWND");
        case TCP_KEEPALIVE_THRESHOLD:   return ("TCP_KEEPALIVE_THRESHOLD");
        case TCP_KEEPALIVE_ABORT_THRESHOLD:
                return ("TCP_KEEPALIVE_ABORT_THRESHOLD");
        case TCP_CORK:                  return ("TCP_CORK");
        case TCP_RTO_INITIAL:           return ("TCP_RTO_INITIAL");
        case TCP_RTO_MIN:               return ("TCP_RTO_MIN");
        case TCP_RTO_MAX:               return ("TCP_RTO_MAX");
        case TCP_LINGER2:               return ("TCP_LINGER2");
        case TCP_KEEPIDLE:              return ("TCP_KEEPIDLE");
        case TCP_KEEPCNT:               return ("TCP_KEEPCNT");
        case TCP_KEEPINTVL:             return ("TCP_KEEPINTVL");
#endif /* ! codereview */

        default:                        (void) snprintf(pri->code_buf,
                                            sizeof (pri->code_buf),
                                            "0x%lx", val);
                                        return (pri->code_buf);
        }
}


const char *
sctp_optname(private_t *pri, long val)
{
        switch (val) {
        case SCTP_RTOINFO:              return ("SCTP_RTOINFO");
        case SCTP_ASSOCINFO:            return ("SCTP_ASSOCINFO");
        case SCTP_INITMSG:              return ("SCTP_INITMSG");
        case SCTP_NODELAY:              return ("SCTP_NODELAY");
        case SCTP_AUTOCLOSE:            return ("SCTP_AUTOCLOSE");
        case SCTP_SET_PEER_PRIMARY_ADDR:
                return ("SCTP_SET_PEER_PRIMARY_ADDR");
        case SCTP_PRIMARY_ADDR:         return ("SCTP_PRIMARY_ADDR");
        case SCTP_ADAPTATION_LAYER:     return ("SCTP_ADAPTATION_LAYER");
        case SCTP_DISABLE_FRAGMENTS:    return ("SCTP_DISABLE_FRAGMENTS");
        case SCTP_PEER_ADDR_PARAMS:     return ("SCTP_PEER_ADDR_PARAMS");
        case SCTP_DEFAULT_SEND_PARAM:   return ("SCTP_DEFAULT_SEND_PARAM");
        case SCTP_EVENTS:               return ("SCTP_EVENTS");
        case SCTP_I_WANT_MAPPED_V4_ADDR:
                return ("SCTP_I_WANT_MAPPED_V4_ADDR");
        case SCTP_MAXSEG:               return ("SCTP_MAXSEG");
```

```
        case SCTP_STATUS:               return ("SCTP_STATUS");
        case SCTP_GET_PEER_ADDR_INFO:   return ("SCTP_GET_PEER_ADDR_INFO");

        case SCTP_ADD_ADDR:             return ("SCTP_ADD_ADDR");
        case SCTP_REM_ADDR:             return ("SCTP_REM_ADDR");

        default:                        (void) snprintf(pri->code_buf,
                                            sizeof (pri->code_buf),
                                            "0x%lx", val);
                                        return (pri->code_buf);
        }
}


const char *
udp_optname(private_t *pri, long val)
{
        switch (val) {
        case UDP_CHECKSUM:              return ("UDP_CHECKSUM");
        case UDP_ANONPRIVBIND:          return ("UDP_ANONPRIVBIND");
        case UDP_EXCLBIND:              return ("UDP_EXCLBIND");
        case UDP_RCVHDR:                return ("UDP_RCVHDR");
        case UDP_NAT_T_ENDPOINT:        return ("UDP_NAT_T_ENDPOINT");

        default:                        (void) snprintf(pri->code_buf,
                                            sizeof (pri->code_buf), "0x%lx",
                                            val);
                                        return (pri->code_buf);
        }
}


/*
 * Print setsockopt()/getsockopt() 3rd argument.
 */
/*ARGSUSED*/
void
prt_son(private_t *pri, int raw, long val)
{
        /* cheating -- look at the level */
        switch (pri->sys_args[1]) {
        case SOL_SOCKET:        outstring(pri, sol_optname(pri, val));
                                break;
        case SOL_ROUTE:         outstring(pri, route_optname(pri, val));
                                break;
        case IPPROTO_TCP:       outstring(pri, tcp_optname(pri, val));
                                break;
        case IPPROTO_UDP:       outstring(pri, udp_optname(pri, val));
                                break;
        case IPPROTO_SCTP:      outstring(pri, sctp_optname(pri, val));
                                break;
        default:                prt_dec(pri, 0, val);
                                break;
        }
}


/*
 * Print utrap type
 */
/*ARGSUSED*/
void
prt_utt(private_t *pri, int raw, long val)
{
        const char *s = NULL;
```

```
#ifdef __sparc
	if (!raw) {
		switch (val) {
		case UT_INSTRUCTION_DISABLED:
			s = "UT_INSTRUCTION_DISABLED"; break;
		case UT_INSTRUCTION_ERROR:
			s = "UT_INSTRUCTION_ERROR"; break;
		case UT_INSTRUCTION_PROTECTION:
			s = "UT_INSTRUCTION_PROTECTION"; break;
		case UT_ILLTRAP_INSTRUCTION:
			s = "UT_ILLTRAP_INSTRUCTION"; break;
		case UT_ILLEGAL_INSTRUCTION:
			s = "UT_ILLEGAL_INSTRUCTION"; break;
		case UT_PRIVILEGED_OPCODE:
			s = "UT_PRIVILEGED_OPCODE"; break;
		case UT_FP_DISABLED:
			s = "UT_FP_DISABLED"; break;
		case UT_FP_EXCEPTION_IEEE_754:
			s = "UT_FP_EXCEPTION_IEEE_754"; break;
		case UT_FP_EXCEPTION_OTHER:
			s = "UT_FP_EXCEPTION_OTHER"; break;
		case UT_TAG_OVERFLOW:
			s = "UT_TAG_OVERFLOW"; break;
		case UT_DIVISION_BY_ZERO:
			s = "UT_DIVISION_BY_ZERO"; break;
		case UT_DATA_EXCEPTION:
			s = "UT_DATA_EXCEPTION"; break;
		case UT_DATA_ERROR:
			s = "UT_DATA_ERROR"; break;
		case UT_DATA_PROTECTION:
			s = "UT_DATA_PROTECTION"; break;
		case UT_MEM_ADDRESS_NOT_ALIGNED:
			s = "UT_MEM_ADDRESS_NOT_ALIGNED"; break;
		case UT_PRIVILEGED_ACTION:
			s = "UT_PRIVILEGED_ACTION"; break;
		case UT_ASYNC_DATA_ERROR:
			s = "UT_ASYNC_DATA_ERROR"; break;
		case UT_TRAP_INSTRUCTION_16:
			s = "UT_TRAP_INSTRUCTION_16"; break;
		case UT_TRAP_INSTRUCTION_17:
			s = "UT_TRAP_INSTRUCTION_17"; break;
		case UT_TRAP_INSTRUCTION_18:
			s = "UT_TRAP_INSTRUCTION_18"; break;
		case UT_TRAP_INSTRUCTION_19:
			s = "UT_TRAP_INSTRUCTION_19"; break;
		case UT_TRAP_INSTRUCTION_20:
			s = "UT_TRAP_INSTRUCTION_20"; break;
		case UT_TRAP_INSTRUCTION_21:
			s = "UT_TRAP_INSTRUCTION_21"; break;
		case UT_TRAP_INSTRUCTION_22:
			s = "UT_TRAP_INSTRUCTION_22"; break;
		case UT_TRAP_INSTRUCTION_23:
			s = "UT_TRAP_INSTRUCTION_23"; break;
		case UT_TRAP_INSTRUCTION_24:
			s = "UT_TRAP_INSTRUCTION_24"; break;
		case UT_TRAP_INSTRUCTION_25:
			s = "UT_TRAP_INSTRUCTION_25"; break;
		case UT_TRAP_INSTRUCTION_26:
			s = "UT_TRAP_INSTRUCTION_26"; break;
		case UT_TRAP_INSTRUCTION_27:
			s = "UT_TRAP_INSTRUCTION_27"; break;
		case UT_TRAP_INSTRUCTION_28:
			s = "UT_TRAP_INSTRUCTION_28"; break;
		case UT_TRAP_INSTRUCTION_29:
			s = "UT_TRAP_INSTRUCTION_29"; break;
		case UT_TRAP_INSTRUCTION_30:
```

```
			s = "UT_TRAP_INSTRUCTION_30"; break;
		case UT_TRAP_INSTRUCTION_31:
			s = "UT_TRAP_INSTRUCTION_31"; break;
		}
	}
#endif /* __sparc */

	if (s == NULL)
		prt_dec(pri, 0, val);
	else
		outstring(pri, s);
}


/*
 * Print utrap handler
 */
void
prt_uth(private_t *pri, int raw, long val)
{
	const char *s = NULL;

	if (!raw) {
		switch (val) {
		case (long)UTH_NOCHANGE:	s = "UTH_NOCHANGE"; break;
		}
	}

	if (s == NULL)
		prt_hex(pri, 0, val);
	else
		outstring(pri, s);
}

const char *
access_flags(private_t *pri, long arg)
{
#define E_OK 010
	char *str = pri->code_buf;

	if (arg & ~(R_OK|W_OK|X_OK|E_OK))
		return (NULL);

	/* NB: F_OK == 0 */
	if (arg == F_OK)
		return ("F_OK");
	if (arg == E_OK)
		return ("F_OK|E_OK");

	*str = '\0';
	if (arg & R_OK)
		(void) strlcat(str, "|R_OK", sizeof (pri->code_buf));
	if (arg & W_OK)
		(void) strlcat(str, "|W_OK", sizeof (pri->code_buf));
	if (arg & X_OK)
		(void) strlcat(str, "|X_OK", sizeof (pri->code_buf));
	if (arg & E_OK)
		(void) strlcat(str, "|E_OK", sizeof (pri->code_buf));
	return ((const char *)(str + 1));
#undef E_OK
}

/*
 * Print access() flags.
 */
void
```

```
prt_acc(private_t *pri, int raw, long val)
{
        const char *s = raw? NULL : access_flags(pri, val);

        if (s == NULL)
                prt_dex(pri, 0, val);
        else
                outstring(pri, s);
}

/*
 * Print shutdown() "how" (2nd) argument
 */
void
prt_sht(private_t *pri, int raw, long val)
{
        if (raw) {
                prt_dex(pri, 0, val);
                return;
        }
        switch (val) {
        case SHUT_RD:   outstring(pri, "SHUT_RD");      break;
        case SHUT_WR:   outstring(pri, "SHUT_WR");      break;
        case SHUT_RDWR: outstring(pri, "SHUT_RDWR");    break;
        default:        prt_dec(pri, 0, val);           break;
        }
}

/*
 * Print fcntl() F_SETFL flags (3rd) argument or fdsync flag (2nd arg)
 */
static struct fcntl_flags {
        long            val;
        const char      *name;
} fcntl_flags[] = {
#define FC_FL(flag)     { (long)flag, "|" # flag }
        FC_FL(FREVOKED),
        FC_FL(FREAD),
        FC_FL(FWRITE),
        FC_FL(FNDELAY),
        FC_FL(FAPPEND),
        FC_FL(FSYNC),
        FC_FL(FDSYNC),
        FC_FL(FRSYNC),
        FC_FL(FOFFMAX),
        FC_FL(FNONBLOCK),
        FC_FL(FCREAT),
        FC_FL(FTRUNC),
        FC_FL(FEXCL),
        FC_FL(FNOCTTY),
        FC_FL(FXATTR),
        FC_FL(FASYNC),
        FC_FL(FNODSYNC)
#undef FC_FL
};

void
prt_ffg(private_t *pri, int raw, long val)
{
#define CBSIZE  sizeof (pri->code_buf)
        char *s = pri->code_buf;
        size_t used = 1;
        struct fcntl_flags *fp;

        if (raw) {
                (void) snprintf(s, CBSIZE, "0x%lx", val);
```

```
                outstring(pri, s);
                return;
        }
        if (val == 0) {
                outstring(pri, "(no flags)");
                return;
        }

        *s = '\0';
        for (fp = fcntl_flags;
            fp < &fcntl_flags[sizeof (fcntl_flags) / sizeof (*fp)]; fp++) {
                if (val & fp->val) {
                        used = strlcat(s, fp->name, CBSIZE);
                        val &= ~fp->val;
                }
        }

        if (val != 0 && used <= CBSIZE)
                used += snprintf(s + used, CBSIZE - used, "|0x%lx", val);

        if (used >= CBSIZE)
                (void) snprintf(s + 1, CBSIZE-1, "0x%lx", val);
        outstring(pri, s + 1);
#undef CBSIZE
}

void
prt_prs(private_t *pri, int raw, long val)
{
        static size_t setsize;
        priv_set_t *set = priv_allocset();

        if (setsize == 0) {
                const priv_impl_info_t *info = getprivimplinfo();
                if (info != NULL)
                        setsize = info->priv_setsize * sizeof (priv_chunk_t);
        }

        if (setsize != 0 && !raw && set != NULL &&
            Pread(Proc, set, setsize, val) == setsize) {
                int i;

                outstring(pri, "{");
                for (i = 0; i < setsize / sizeof (priv_chunk_t); i++) {
                        char buf[9];    /* 8 hex digits + '\0' */
                        (void) snprintf(buf, sizeof (buf), "%08x",
                            ((priv_chunk_t *)set)[i]);
                        outstring(pri, buf);
                }

                outstring(pri, "}");
        } else {
                prt_hex(pri, 0, val);
        }

        if (set != NULL)
                priv_freeset(set);
}

/*
 * Print privilege set operation.
 */
void
prt_pro(private_t *pri, int raw, long val)
{
        const char *s = NULL;
	if (!raw) {
		switch ((priv_op_t)val) {
		case PRIV_ON:		s = "PRIV_ON";		break;
		case PRIV_OFF:		s = "PRIV_OFF";		break;
		case PRIV_SET:		s = "PRIV_SET";		break;
		}
	}

	if (s == NULL)
		prt_dec(pri, 0, val);
	else
		outstring(pri, s);
}

/*
 * Print privilege set name
 */
void
prt_prn(private_t *pri, int raw, long val)
{
	const char *s = NULL;

	if (!raw)
		s = priv_getsetbynum((int)val);

	if (s == NULL)
		prt_dec(pri, 0, val);
	else {
		char *dup = strdup(s);
		char *q;

		/* Do the best we can in this case */
		if (dup == NULL) {
			outstring(pri, s);
			return;
		}

		outstring(pri, "PRIV_");

		q = dup;

		while (*q != '\0') {
			*q = toupper(*q);
			q++;
		}
		outstring(pri, dup);
		free(dup);
	}
}

/*
 * Print process flag names.
 */
void
prt_pfl(private_t *pri, int raw, long val)
{
	const char *s = NULL;

	if (!raw) {
		switch ((int)val) {
		case PRIV_DEBUG:	s = "PRIV_DEBUG";	break;
		case PRIV_AWARE:	s = "PRIV_AWARE";	break;
		case PRIV_XPOLICY:	s = "PRIV_XPOLICY";	break;
		case PRIV_AWARE_RESET:  s = "PRIV_AWARE_RESET"; break;
		case PRIV_PFEXEC:	s = "PRIV_PFEXEC";	break;
```

```
		case NET_MAC_AWARE:	s =  "NET_MAC_AWARE";	break;
		case NET_MAC_AWARE_INHERIT:
			s = "NET_MAC_AWARE_INHERIT";
			break;
		}
	}

	if (s == NULL)
		prt_dec(pri, 0, val);
	else
		outstring(pri, s);
}

/*
 * Print lgrp_affinity_{get,set}() arguments.
 */
/*ARGSUSED*/
void
prt_laf(private_t *pri, int raw, long val)
{
	lgrp_affinity_args_t	laff;

	if (Pread(Proc, &laff, sizeof (lgrp_affinity_args_t), val) !=
	    sizeof (lgrp_affinity_args_t)) {
		prt_hex(pri, 0, val);
		return;
	}
	/*
	 * arrange the arguments in the order that user calls with
	 */
	prt_dec(pri, 0, laff.idtype);
	outstring(pri, ", ");
	prt_dec(pri, 0, laff.id);
	outstring(pri, ", ");
	prt_dec(pri, 0, laff.lgrp);
	outstring(pri, ", ");
	if (pri->sys_args[0] == LGRP_SYS_AFFINITY_SET)
		prt_dec(pri, 0, laff.aff);
}

/*
 * Print a key_t as IPC_PRIVATE if it is 0.
 */
void
prt_key(private_t *pri, int raw, long val)
{
	if (!raw && val == 0)
		outstring(pri, "IPC_PRIVATE");
	else
		prt_dec(pri, 0, val);
}


/*
 * Print zone_getattr() attribute types.
 */
void
prt_zga(private_t *pri, int raw, long val)
{
	const char *s = NULL;

	if (!raw) {
		switch ((int)val) {
		case ZONE_ATTR_NAME:	s = "ZONE_ATTR_NAME";	break;
		case ZONE_ATTR_ROOT:	s = "ZONE_ATTR_ROOT";	break;
		case ZONE_ATTR_STATUS:  s = "ZONE_ATTR_STATUS"; break;
		case ZONE_ATTR_PRIVSET: s = "ZONE_ATTR_PRIVSET"; break;
		case ZONE_ATTR_UNIQID:  s = "ZONE_ATTR_UNIQID"; break;
		case ZONE_ATTR_POOLID:  s = "ZONE_ATTR_POOLID"; break;
		case ZONE_ATTR_INITPID: s = "ZONE_ATTR_INITPID"; break;
		case ZONE_ATTR_SLBL:    s = "ZONE_ATTR_SLBL"; break;
		case ZONE_ATTR_INITNAME:	s = "ZONE_ATTR_INITNAME"; break;
		case ZONE_ATTR_BOOTARGS:	s = "ZONE_ATTR_BOOTARGS"; break;
		case ZONE_ATTR_BRAND:   s = "ZONE_ATTR_BRAND"; break;
		case ZONE_ATTR_FLAGS:   s = "ZONE_ATTR_FLAGS"; break;
		case ZONE_ATTR_PHYS_MCAP: s = "ZONE_ATTR_PHYS_MCAP"; break;
		}
	}

	if (s == NULL)
		prt_dec(pri, 0, val);
	else
		outstring(pri, s);
}

/*
 * Print a file descriptor as AT_FDCWD if necessary
 */
void
prt_atc(private_t *pri, int raw, long val)
{
	if ((int)val == AT_FDCWD) {
		if (raw)
			prt_hex(pri, 0, (uint_t)AT_FDCWD);
		else
			outstring(pri, "AT_FDCWD");
	} else {
		prt_dec(pri, 0, val);
	}
}

/*
 * Print Trusted Networking database operation codes (labelsys; tn*)
 */
static void
prt_tnd(private_t *pri, int raw, long val)
{
	const char *s = NULL;

	if (!raw) {
		switch ((tsol_dbops_t)val) {
		case TNDB_NOOP:		s = "TNDB_NOOP";	break;
		case TNDB_LOAD:		s = "TNDB_LOAD";	break;
		case TNDB_DELETE:	s = "TNDB_DELETE";	break;
		case TNDB_FLUSH:	s = "TNDB_FLUSH";	break;
		case TNDB_GET:		s = "TNDB_GET";		break;
		}
	}

	if (s == NULL)
		prt_dec(pri, 0, val);
	else
		outstring(pri, s);
}

/*
 * Print LIO_XX flags
 */
void
prt_lio(private_t *pri, int raw, long val)
{
	if (raw)
```

```
		prt_dec(pri, 0, val);
	else if (val == LIO_WAIT)
		outstring(pri, "LIO_WAIT");
	else if (val == LIO_NOWAIT)
		outstring(pri, "LIO_NOWAIT");
	else
		prt_dec(pri, 0, val);
}

const char *
door_flags(private_t *pri, long val)
{
	door_attr_t attr = (door_attr_t)val;
	char *str = pri->code_buf;

	*str = '\0';
#define PROCESS_FLAG(flg)						\
	if (attr & flg) {						\
		(void) strlcat(str, "|" #flg, sizeof (pri->code_buf));	\
		attr &= ~flg;						\
	}

	PROCESS_FLAG(DOOR_UNREF);
	PROCESS_FLAG(DOOR_UNREF_MULTI);
	PROCESS_FLAG(DOOR_PRIVATE);
	PROCESS_FLAG(DOOR_REFUSE_DESC);
	PROCESS_FLAG(DOOR_NO_CANCEL);
	PROCESS_FLAG(DOOR_LOCAL);
	PROCESS_FLAG(DOOR_REVOKED);
	PROCESS_FLAG(DOOR_IS_UNREF);
#undef PROCESS_FLAG

	if (attr != 0 || *str == '\0') {
		size_t len = strlen(str);
		(void) snprintf(str + len, sizeof (pri->code_buf) - len,
		    "|0x%X", attr);
	}

	return (str + 1);
}

/*
 * Print door_create() flags
 */
void
prt_dfl(private_t *pri, int raw, long val)
{
	if (raw)
		prt_hex(pri, 0, val);
	else
		outstring(pri, door_flags(pri, val));
}

/*
 * Print door_*param() param argument
 */
void
prt_dpm(private_t *pri, int raw, long val)
{
	if (raw)
		prt_hex(pri, 0, val);
	else if (val == DOOR_PARAM_DESC_MAX)
		outstring(pri, "DOOR_PARAM_DESC_MAX");
	else if (val == DOOR_PARAM_DATA_MIN)
		outstring(pri, "DOOR_PARAM_DATA_MIN");
	else if (val == DOOR_PARAM_DATA_MAX)
```

```
		outstring(pri, "DOOR_PARAM_DATA_MAX");
	else
		prt_hex(pri, 0, val);
}

/*
 * Print rctlsys subcodes
 */
void
prt_rsc(private_t *pri, int raw, long val)	/* print utssys code */
{
	const char *s = raw? NULL : rctlsyscode(val);

	if (s == NULL)
		prt_dec(pri, 0, val);
	else
		outstring(pri, s);
}

/*
 * Print getrctl flags
 */
void
prt_rgf(private_t *pri, int raw, long val)
{
	long action = val & (~RCTLSYS_ACTION_MASK);

	if (raw)
		prt_hex(pri, 0, val);
	else if (action == RCTL_FIRST)
		outstring(pri, "RCTL_FIRST");
	else if (action == RCTL_NEXT)
		outstring(pri, "RCTL_NEXT");
	else if (action == RCTL_USAGE)
		outstring(pri, "RCTL_USAGE");
	else
		prt_hex(pri, 0, val);
}

/*
 * Print setrctl flags
 */
void
prt_rsf(private_t *pri, int raw, long val)
{
	long action = val & (~RCTLSYS_ACTION_MASK);
	long pval = val & RCTL_LOCAL_ACTION_MASK;
	char *s = pri->code_buf;

	if (raw) {
		prt_hex(pri, 0, val);
		return;
	} else if (action == RCTL_INSERT)
		(void) strcpy(s, "RCTL_INSERT");
	else if (action == RCTL_DELETE)
		(void) strcpy(s, "RCTL_DELETE");
	else if (action == RCTL_REPLACE)
		(void) strcpy(s, "RCTL_REPLACE");
	else {
		prt_hex(pri, 0, val);
		return;
	}

	if (pval & RCTL_USE_RECIPIENT_PID) {
		pval ^= RCTL_USE_RECIPIENT_PID;
		(void) strlcat(s, "|RCTL_USE_RECIPIENT_PID",
```

```
		    sizeof (pri->code_buf));
	}

	if ((pval & RCTLSYS_ACTION_MASK) != 0)
		prt_hex(pri, 0, val);
	else if (*s != '\0')
		outstring(pri, s);
	else
		prt_hex(pri, 0, val);
}

/*
 * Print rctlctl flags
 */
void
prt_rcf(private_t *pri, int raw, long val)
{
	long action = val & (~RCTLSYS_ACTION_MASK);

	if (raw)
		prt_hex(pri, 0, val);
	else if (action == RCTLCTL_GET)
		outstring(pri, "RCTLCTL_GET");
	else if (action == RCTLCTL_SET)
		outstring(pri, "RCTLCTL_SET");
	else
		prt_hex(pri, 0, val);
}

/*
 * Print setprojrctl flags
 */
void
prt_spf(private_t *pri, int raw, long val)
{
	long action = val & TASK_PROJ_MASK;

	if (!raw && (action == TASK_PROJ_PURGE))
		outstring(pri, "TASK_PROJ_PURGE");
	else
		prt_hex(pri, 0, val);
}

/*
 * Print forkx() flags
 */
void
prt_fxf(private_t *pri, int raw, long val)
{
	char *str;

	if (val == 0)
		outstring(pri, "0");
	else if (raw || (val & ~(FORK_NOSIGCHLD | FORK_WAITPID)))
		prt_hhx(pri, 0, val);
	else {
		str = pri->code_buf;
		*str = '\0';
		if (val & FORK_NOSIGCHLD)
			(void) strlcat(str, "|FORK_NOSIGCHLD",
			    sizeof (pri->code_buf));
		if (val & FORK_WAITPID)
			(void) strlcat(str, "|FORK_WAITPID",
			    sizeof (pri->code_buf));
		outstring(pri, str + 1);
	}
```

```
}

/*
 * Print faccessat() flag
 */
void
prt_fat(private_t *pri, int raw, long val)
{
        if (val == 0)
                outstring(pri, "0");
        else if (!raw && val == AT_EACCESS)
                outstring(pri, "AT_EACCESS");
        else
                prt_hex(pri, 0, val);
}

/*
 * Print unlinkat() flag
 */
void
prt_uat(private_t *pri, int raw, long val)
{
        if (val == 0)
                outstring(pri, "0");
        else if (!raw && val == AT_REMOVEDIR)
                outstring(pri, "AT_REMOVEDIR");
        else
                prt_hex(pri, 0, val);
}

/*
 * Print AT_SYMLINK_NOFOLLOW / AT_SYMLINK_FOLLOW flag
 */
void
prt_snf(private_t *pri, int raw, long val)
{
        if (val == 0)
                outstring(pri, "0");
        else if (!raw && val == AT_SYMLINK_NOFOLLOW)
                outstring(pri, "AT_SYMLINK_NOFOLLOW");
        else if (!raw && val == AT_SYMLINK_FOLLOW)
                outstring(pri, "AT_SYMLINK_FOLLOW");
        else
                prt_hex(pri, 0, val);
}

void
prt_grf(private_t *pri, int raw, long val)
{
        int first = 1;

        if (raw != 0 || val == 0 ||
            (val & ~(GRND_NONBLOCK | GRND_RANDOM)) != 0) {
                outstring(pri, "0");
                return;
        }

        if (val & GRND_NONBLOCK) {
                outstring(pri, "|GRND_NONBLOCK" + first);
                first = 0;
        }
        if (val & GRND_RANDOM) {
                outstring(pri, "|GRND_RANDOM" + first);
                first = 0;
        }
}
```

```
/*
 * Array of pointers to print functions, one for each format.
 */
void (* const Print[])() = {
        prt_nov,        /* NOV -- no value */
        prt_dec,        /* DEC -- print value in decimal */
        prt_oct,        /* OCT -- print value in octal */
        prt_hex,        /* HEX -- print value in hexadecimal */
        prt_dex,        /* DEX -- print value in hexadecimal if big enough */
        prt_stg,        /* STG -- print value as string */
        prt_ioc,        /* IOC -- print ioctl code */
        prt_fcn,        /* FCN -- print fcntl code */
        prt_s86,        /* S86 -- print sysi86 code */
        prt_uts,        /* UTS -- print utssys code */
        prt_opn,        /* OPN -- print open code */
        prt_sig,        /* SIG -- print signal name plus flags */
        prt_uat,        /* UAT -- print unlinkat() flag */
        prt_msc,        /* MSC -- print msgsys command */
        prt_msf,        /* MSF -- print msgsys flags */
        prt_smc,        /* SMC -- print semsys command */
        prt_sef,        /* SEF -- print semsys flags */
        prt_shc,        /* SHC -- print shmsys command */
        prt_shf,        /* SHF -- print shmsys flags */
        prt_fat,        /* FAT -- print faccessat( flag */
        prt_sfs,        /* SFS -- print sysfs code */
        prt_rst,        /* RST -- print string returned by syscall */
        prt_smf,        /* SMF -- print streams message flags */
        prt_ioa,        /* IOA -- print ioctl argument */
        prt_pip,        /* PIP -- print pipe flags */
        prt_mtf,        /* MTF -- print mount flags */
        prt_mft,        /* MFT -- print mount file system type */
        prt_iob,        /* IOB -- print contents of I/O buffer */
        prt_hhx,        /* HHX -- print value in hexadecimal (half size) */
        prt_wop,        /* WOP -- print waitsys() options */
        prt_spm,        /* SPM -- print sigprocmask argument */
        prt_rlk,        /* RLK -- print readlink buffer */
        prt_mpr,        /* MPR -- print mmap()/mprotect() flags */
        prt_mty,        /* MTY -- print mmap() mapping type flags */
        prt_mcf,        /* MCF -- print memcntl() function */
        prt_mc4,        /* MC4 -- print memcntl() (fourth) argument */
        prt_mc5,        /* MC5 -- print memcntl() (fifth) argument */
        prt_mad,        /* MAD -- print madvise() argument */
        prt_ulm,        /* ULM -- print ulimit() argument */
        prt_rlm,        /* RLM -- print get/setrlimit() argument */
        prt_cnf,        /* CNF -- print sysconfig() argument */
        prt_inf,        /* INF -- print sysinfo() argument */
        prt_ptc,        /* PTC -- print pathconf/fpathconf() argument */
        prt_fui,        /* FUI -- print fusers() input argument */
        prt_idt,        /* IDT -- print idtype_t, waitid() argument */
        prt_lwf,        /* LWF -- print lwp_create() flags */
        prt_itm,        /* ITM -- print [get|set]itimer() arg */
        prt_llo,        /* LLO -- print long long offset arg */
        prt_mod,        /* MOD -- print modctl() subcode */
        prt_whn,        /* WHN -- print lseek() whence arguiment */
        prt_acl,        /* ACL -- print acl() code */
        prt_aio,        /* AIO -- print kaio() code */
        prt_aud,        /* AUD -- print auditsys() code */
        prt_uns,        /* DEC -- print value in unsigned decimal */
        prt_clc,        /* CLC -- print cladm command argument */
        prt_clf,        /* CLF -- print cladm flag argument */
        prt_cor,        /* COR -- print corectl() subcode */
        prt_cco,        /* CCO -- print corectl() options */
        prt_ccc,        /* CCC -- print corectl() content */
        prt_rcc,        /* RCC -- print corectl() returned content */
        prt_cpc,        /* CPC -- print cpc() subcode */
	prt_sqc,	/* SQC -- print sigqueue() si_code argument */
	prt_pc4,	/* PC4 -- print priocntlsys() (fourth) argument */
	prt_pc5,	/* PC5 -- print priocntlsys() (key, value) pairs */
	prt_pst,	/* PST -- print processor set id */
	prt_mif,	/* MIF -- print meminfo() arguments */
	prt_pfm,	/* PFM -- print so_socket() proto-family (1st) arg */
	prt_skt,	/* SKT -- print so_socket() socket-type (2nd) arg */
	prt_skp,	/* SKP -- print so_socket() protocol (3rd) arg */
	prt_skv,	/* SKV -- print socket version arg */
	prt_sol,	/* SOL -- print [sg]etsockopt() level (2nd) arg */
	prt_son,	/* SON -- print [sg]etsockopt() opt-name (3rd) arg */
	prt_utt,	/* UTT -- print utrap type */
	prt_uth,	/* UTH -- print utrap handler */
	prt_acc,	/* ACC -- print access() flags */
	prt_sht,	/* SHT -- print shutdown() how (2nd) argument */
	prt_ffg,	/* FFG -- print fcntl() flags (3rd) argument */
	prt_prs,	/* PRS -- print privilege set */
	prt_pro,	/* PRO -- print privilege set operation */
	prt_prn,	/* PRN -- print privilege set name */
	prt_pfl,	/* PFL -- print privilege/process flag name */
	prt_laf,	/* LAF -- print lgrp_affinity arguments */
	prt_key,	/* KEY -- print key_t 0 as IPC_PRIVATE */
	prt_zga,	/* ZGA -- print zone_getattr attribute types */
	prt_atc,	/* ATC -- print AT_FDCWD or file descriptor */
	prt_lio,	/* LIO -- print LIO_XX flags */
	prt_dfl,	/* DFL -- print door_create() flags */
	prt_dpm,	/* DPM -- print DOOR_PARAM_XX flags */
	prt_tnd,	/* TND -- print trusted network data base opcode */
	prt_rsc,	/* RSC -- print rctlsys() subcodes */
	prt_rgf,	/* RGF -- print getrctl() flags */
	prt_rsf,	/* RSF -- print setrctl() flags */
	prt_rcf,	/* RCF -- print rctlsys_ctl() flags */
	prt_fxf,	/* FXF -- print forkx() flags */
	prt_spf,	/* SPF -- print rctlsys_projset() flags */
	prt_un1,	/* UN1 -- as prt_uns except for -1 */
	prt_mob,	/* MOB -- print mmapobj() flags */
	prt_snf,	/* SNF -- print AT_SYMLINK_[NO]FOLLOW flag */
	prt_skc,	/* SKC -- print sockconfig() subcode */
	prt_acf,	/* ACF -- print accept4 flags */
	prt_pfd,	/* PFD -- print pipe fds */
	prt_grf,	/* GRF -- print getrandom flags */
	prt_dec,	/* HID -- hidden argument, make this the last one */
};
```