

```

*****
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 <dandmcd@omniti.com>
Reviewed by: Richard PALO <richard@netbsd.org>
Reviewed by: Toomas Soome <tsoome@me.com>
*****
_____unchanged_portion_omitted_____

```

```

1922 const char *
1923 tcp_optname(private_t *pri, long val)
1924 {
1925     switch (val) {
1926     case TCP_NODELAY:           return ("TCP_NODELAY");
1927     case TCP_MAXSEG:           return ("TCP_MAXSEG");
1928     case TCP_KEEPAKIVE:        return ("TCP_KEEPAKIVE");
1929     case TCP_NOTIFY_THRESHOLD: return ("TCP_NOTIFY_THRESHOLD");
1930     case TCP_ABORT_THRESHOLD:  return ("TCP_ABORT_THRESHOLD");
1931     case TCP_CONN_NOTIFY_THRESHOLD: return ("TCP_CONN_NOTIFY_THRESHOLD");
1932     case TCP_CONN_ABORT_THRESHOLD: return ("TCP_CONN_ABORT_THRESHOLD");
1933     case TCP_RECVDSTADDR:      return ("TCP_RECVDSTADDR");
1934     case TCP_ANONPRIVBIND:     return ("TCP_ANONPRIVBIND");
1935     case TCP_EXCLBIND:         return ("TCP_EXCLBIND");
1936     case TCP_INIT_CWND:        return ("TCP_INIT_CWND");
1937     case TCP_KEEPAKIVE_THRESHOLD: return ("TCP_KEEPAKIVE_THRESHOLD");
1938     case TCP_KEEPAKIVE_ABORT_THRESHOLD:
1939         return ("TCP_KEEPAKIVE_ABORT_THRESHOLD");
1940     case TCP_CORK:             return ("TCP_CORK");
1941     case TCP_RTO_INITIAL:      return ("TCP_RTO_INITIAL");
1942     case TCP_RTO_MIN:          return ("TCP_RTO_MIN");
1943     case TCP_RTO_MAX:          return ("TCP_RTO_MAX");
1944     case TCP_LINGER2:          return ("TCP_LINGER2");
1945     case TCP_KEEPIDLE:         return ("TCP_KEEPIDLE");
1946     case TCP_KEEPCNT:          return ("TCP_KEEPCNT");
1947     case TCP_KEEPINTVL:        return ("TCP_KEEPINTVL");
1948 #endif /* ! codereview */
1949
1950     default:                   (void) snprintf(pri->code_buf,
1951         sizeof (pri->code_buf),
1952         "0x%lx", val);
1953     return (pri->code_buf);
1954 }
1955 }

```

```

1958 const char *
1959 sctp_optname(private_t *pri, long val)
1960 {
1961     switch (val) {
1962     case Sctp_RTOINFO:         return ("Sctp_RTOINFO");
1963     case Sctp_AssocINFO:       return ("Sctp_AssocINFO");
1964     case Sctp_InitMSG:         return ("Sctp_InitMSG");
1965     case Sctp_NodeLAY:         return ("Sctp_NodeLAY");
1966     case Sctp_AutoClose:       return ("Sctp_AutoClose");
1967     case Sctp_Set_Peer_Primary_Addr:
1968         return ("Sctp_Set_Peer_Primary_Addr");
1969     case Sctp_Primary_Addr:     return ("Sctp_Primary_Addr");
1970     case Sctp_Adaptation_Layer: return ("Sctp_Adaptation_Layer");
1971     case Sctp_Disable_Fragments: return ("Sctp_Disable_Fragments");
1972     case Sctp_Peer_Addr_Params: return ("Sctp_Peer_Addr_Params");
1973     case Sctp_Default_Send_Param: return ("Sctp_Default_Send_Param");
1974     case Sctp_Events:          return ("Sctp_Events");
1975     case Sctp_I_Want_Mapped_V4_Addr:
1976         return ("Sctp_I_Want_Mapped_V4_Addr");
1977     case Sctp_MaxSeg:          return ("Sctp_MaxSeg");

```

```

1978     case Sctp_Status:         return ("Sctp_Status");
1979     case Sctp_Get_Peer_Addr_Info:
1980         return ("Sctp_Get_Peer_Addr_Info");
1981
1982     case Sctp_Add_Addr:       return ("Sctp_Add_Addr");
1983     case Sctp_Rem_Addr:       return ("Sctp_Rem_Addr");
1984
1985     default:                   (void) snprintf(pri->code_buf,
1986         sizeof (pri->code_buf),
1987         "0x%lx", val);
1988     return (pri->code_buf);
1989 }

```

```

1992 const char *
1993 udp_optname(private_t *pri, long val)
1994 {
1995     switch (val) {
1996     case UDP_Checksum:         return ("UDP_Checksum");
1997     case UDP_AnonPrivBind:     return ("UDP_AnonPrivBind");
1998     case UDP_ExclBind:         return ("UDP_ExclBind");
1999     case UDP_RcvHdr:           return ("UDP_RcvHdr");
2000     case UDP_Nat_T_Endpoint:   return ("UDP_Nat_T_Endpoint");
2001
2002     default:                   (void) snprintf(pri->code_buf,
2003         sizeof (pri->code_buf), "0x%lx",
2004         val);
2005     return (pri->code_buf);
2006 }
2007 }

```

```

2010 /*
2011  * Print setsockopt()/getsockopt() 3rd argument.
2012  */
2013 /*ARGSUSED*/
2014 void
2015 prt_son(private_t *pri, int raw, long val)
2016 {
2017     /* cheating -- look at the level */
2018     switch (pri->sys_args[1]) {
2019     case SOL_SOCKET:           outstring(pri, sol_optname(pri, val));
2020                                 break;
2021     case SOL_ROUTE:           outstring(pri, route_optname(pri, val));
2022                                 break;
2023     case IPPROTO_TCP:         outstring(pri, tcp_optname(pri, val));
2024                                 break;
2025     case IPPROTO_UDP:         outstring(pri, udp_optname(pri, val));
2026                                 break;
2027     case IPPROTO_SCTP:        outstring(pri, sctp_optname(pri, val));
2028                                 break;
2029     default:                   prt_dec(pri, 0, val);
2030     }
2031     break;
2032 }

```

```

2035 /*
2036  * Print utrap type
2037  */
2038 /*ARGSUSED*/
2039 void
2040 prt_utt(private_t *pri, int raw, long val)
2041 {
2042     const char *s = NULL;

```

```

2044 #ifdef __sparc
2045     if (!raw) {
2046         switch (val) {
2047             case UT_INSTRUCTION_DISABLED:
2048                 s = "UT_INSTRUCTION_DISABLED"; break;
2049             case UT_INSTRUCTION_ERROR:
2050                 s = "UT_INSTRUCTION_ERROR"; break;
2051             case UT_INSTRUCTION_PROTECTION:
2052                 s = "UT_INSTRUCTION_PROTECTION"; break;
2053             case UT_ILLTRAP_INSTRUCTION:
2054                 s = "UT_ILLTRAP_INSTRUCTION"; break;
2055             case UT_ILLEGAL_INSTRUCTION:
2056                 s = "UT_ILLEGAL_INSTRUCTION"; break;
2057             case UT_PRIVILEGED_OPCODE:
2058                 s = "UT_PRIVILEGED_OPCODE"; break;
2059             case UT_FP_DISABLED:
2060                 s = "UT_FP_DISABLED"; break;
2061             case UT_FP_EXCEPTION_IEEE_754:
2062                 s = "UT_FP_EXCEPTION_IEEE_754"; break;
2063             case UT_FP_EXCEPTION_OTHER:
2064                 s = "UT_FP_EXCEPTION_OTHER"; break;
2065             case UT_TAG_OVERFLOW:
2066                 s = "UT_TAG_OVERFLOW"; break;
2067             case UT_DIVISION_BY_ZERO:
2068                 s = "UT_DIVISION_BY_ZERO"; break;
2069             case UT_DATA_EXCEPTION:
2070                 s = "UT_DATA_EXCEPTION"; break;
2071             case UT_DATA_ERROR:
2072                 s = "UT_DATA_ERROR"; break;
2073             case UT_DATA_PROTECTION:
2074                 s = "UT_DATA_PROTECTION"; break;
2075             case UT_MEM_ADDRESS_NOT_ALIGNED:
2076                 s = "UT_MEM_ADDRESS_NOT_ALIGNED"; break;
2077             case UT_PRIVILEGED_ACTION:
2078                 s = "UT_PRIVILEGED_ACTION"; break;
2079             case UT_ASYNC_DATA_ERROR:
2080                 s = "UT_ASYNC_DATA_ERROR"; break;
2081             case UT_TRAP_INSTRUCTION_16:
2082                 s = "UT_TRAP_INSTRUCTION_16"; break;
2083             case UT_TRAP_INSTRUCTION_17:
2084                 s = "UT_TRAP_INSTRUCTION_17"; break;
2085             case UT_TRAP_INSTRUCTION_18:
2086                 s = "UT_TRAP_INSTRUCTION_18"; break;
2087             case UT_TRAP_INSTRUCTION_19:
2088                 s = "UT_TRAP_INSTRUCTION_19"; break;
2089             case UT_TRAP_INSTRUCTION_20:
2090                 s = "UT_TRAP_INSTRUCTION_20"; break;
2091             case UT_TRAP_INSTRUCTION_21:
2092                 s = "UT_TRAP_INSTRUCTION_21"; break;
2093             case UT_TRAP_INSTRUCTION_22:
2094                 s = "UT_TRAP_INSTRUCTION_22"; break;
2095             case UT_TRAP_INSTRUCTION_23:
2096                 s = "UT_TRAP_INSTRUCTION_23"; break;
2097             case UT_TRAP_INSTRUCTION_24:
2098                 s = "UT_TRAP_INSTRUCTION_24"; break;
2099             case UT_TRAP_INSTRUCTION_25:
2100                 s = "UT_TRAP_INSTRUCTION_25"; break;
2101             case UT_TRAP_INSTRUCTION_26:
2102                 s = "UT_TRAP_INSTRUCTION_26"; break;
2103             case UT_TRAP_INSTRUCTION_27:
2104                 s = "UT_TRAP_INSTRUCTION_27"; break;
2105             case UT_TRAP_INSTRUCTION_28:
2106                 s = "UT_TRAP_INSTRUCTION_28"; break;
2107             case UT_TRAP_INSTRUCTION_29:
2108                 s = "UT_TRAP_INSTRUCTION_29"; break;
2109             case UT_TRAP_INSTRUCTION_30:

```

```

2110                 s = "UT_TRAP_INSTRUCTION_30"; break;
2111                 case UT_TRAP_INSTRUCTION_31:
2112                     s = "UT_TRAP_INSTRUCTION_31"; break;
2113             }
2114         }
2115 #endif /* __sparc */

2117     if (s == NULL)
2118         prt_dec(pri, 0, val);
2119     else
2120         outstring(pri, s);
2121 }

2124 /*
2125  * Print utrap handler
2126  */
2127 void
2128 prt_uth(private_t *pri, int raw, long val)
2129 {
2130     const char *s = NULL;

2132     if (!raw) {
2133         switch (val) {
2134             case (long)UTH_NOCHANGE:
2135                 s = "UTH_NOCHANGE"; break;
2136         }
2137     }

2138     if (s == NULL)
2139         prt_hex(pri, 0, val);
2140     else
2141         outstring(pri, s);
2142 }

2144 const char *
2145 access_flags(private_t *pri, long arg)
2146 {
2147 #define E_OK 010
2148     char *str = pri->code_buf;

2150     if (arg & ~(R_OK|W_OK|X_OK|E_OK))
2151         return (NULL);

2153     /* NB: F_OK == 0 */
2154     if (arg == F_OK)
2155         return ("F_OK");
2156     if (arg == E_OK)
2157         return ("F_OK|E_OK");

2159     *str = '\0';
2160     if (arg & R_OK)
2161         (void) strlcat(str, "|R_OK", sizeof (pri->code_buf));
2162     if (arg & W_OK)
2163         (void) strlcat(str, "|W_OK", sizeof (pri->code_buf));
2164     if (arg & X_OK)
2165         (void) strlcat(str, "|X_OK", sizeof (pri->code_buf));
2166     if (arg & E_OK)
2167         (void) strlcat(str, "|E_OK", sizeof (pri->code_buf));
2168     return ((const char *) (str + 1));
2169 #undef E_OK
2170 }

2172 /*
2173  * Print access() flags.
2174  */
2175 void

```

```

2176 prt_acc(private_t *pri, int raw, long val)
2177 {
2178     const char *s = raw? NULL : access_flags(pri, val);
2180     if (s == NULL)
2181         prt_dex(pri, 0, val);
2182     else
2183         outstring(pri, s);
2184 }
2186 /*
2187 * Print shutdown() "how" (2nd argument
2188 */
2189 void
2190 prt_sht(private_t *pri, int raw, long val)
2191 {
2192     if (raw) {
2193         prt_dex(pri, 0, val);
2194         return;
2195     }
2196     switch (val) {
2197     case SHUT_RD:    outstring(pri, "SHUT_RD");    break;
2198     case SHUT_WR:    outstring(pri, "SHUT_WR");    break;
2199     case SHUT_RDWR: outstring(pri, "SHUT_RDWR"); break;
2200     default:        prt_dec(pri, 0, val);         break;
2201     }
2202 }
2204 /*
2205 * Print fcntl() F_SETFL flags (3rd argument or fdsync flag (2nd arg)
2206 */
2207 static struct fcntl_flags {
2208     long    val;
2209     const char *name;
2210 } fcntl_flags[] = {
2211 #define FC_FL(flag)    { (long)flag, "|" # flag }
2212     FC_FL(FREVKED),
2213     FC_FL(FREAD),
2214     FC_FL(FWRITE),
2215     FC_FL(FNDELAY),
2216     FC_FL(FAPPEND),
2217     FC_FL(FSYNC),
2218     FC_FL(FDSYNC),
2219     FC_FL(FRSYNC),
2220     FC_FL(FOFFMAX),
2221     FC_FL(FNONBLOCK),
2222     FC_FL(FCREAT),
2223     FC_FL(FTRUNC),
2224     FC_FL(FEXCL),
2225     FC_FL(FNOCTTY),
2226     FC_FL(FXATTR),
2227     FC_FL(FASYNC),
2228     FC_FL(FNODSYNC)
2229 #undef FC_FL
2230 };
2232 void
2233 prt_ffg(private_t *pri, int raw, long val)
2234 {
2235 #define CBSIZE    sizeof (pri->code_buf)
2236     char *s = pri->code_buf;
2237     size_t used = 1;
2238     struct fcntl_flags *fp;
2240     if (raw) {
2241         (void) snprintf(s, CBSIZE, "0x%lx", val);

```

```

2242         outstring(pri, s);
2243         return;
2244     }
2245     if (val == 0) {
2246         outstring(pri, "(no flags)");
2247         return;
2248     }
2250     *s = '\0';
2251     for (fp = fcntl_flags;
2252          fp < &fcntl_flags[sizeof (fcntl_flags) / sizeof (*fp)]; fp++) {
2253         if (val & fp->val) {
2254             used = strlcat(s, fp->name, CBSIZE);
2255             val &= ~fp->val;
2256         }
2257     }
2259     if (val != 0 && used <= CBSIZE)
2260         used += snprintf(s + used, CBSIZE - used, "|0x%lx", val);
2262     if (used >= CBSIZE)
2263         (void) snprintf(s + 1, CBSIZE-1, "0x%lx", val);
2264     outstring(pri, s + 1);
2265 #undef CBSIZE
2266 }
2268 void
2269 prt_prs(private_t *pri, int raw, long val)
2270 {
2271     static size_t setsize;
2272     priv_set_t *set = priv_allocset();
2274     if (setsize == 0) {
2275         const priv_impl_info_t *info = getprivimplinfo();
2276         if (info != NULL)
2277             setsize = info->priv_setsize * sizeof (priv_chunk_t);
2278     }
2280     if (setsize != 0 && !raw && set != NULL &&
2281         Pread(Proc, set, setsize, val) == setsize) {
2282         int i;
2284         outstring(pri, "{");
2285         for (i = 0; i < setsize / sizeof (priv_chunk_t); i++) {
2286             char buf[9]; /* 8 hex digits + '\0' */
2287             (void) snprintf(buf, sizeof (buf), "%08x",
2288                 ((priv_chunk_t *)set)[i]);
2289             outstring(pri, buf);
2290         }
2292         outstring(pri, "}");
2293     } else {
2294         prt_hex(pri, 0, val);
2295     }
2297     if (set != NULL)
2298         priv_freeset(set);
2299 }
2301 /*
2302 * Print privilege set operation.
2303 */
2304 void
2305 prt_pro(private_t *pri, int raw, long val)
2306 {
2307     const char *s = NULL;

```

```

2309     if (!raw) {
2310         switch ((priv_op_t)val) {
2311             case PRIV_ON:          s = "PRIV_ON";          break;
2312             case PRIV_OFF:        s = "PRIV_OFF";         break;
2313             case PRIV_SET:        s = "PRIV_SET";         break;
2314         }
2315     }
2317     if (s == NULL)
2318         prt_dec(pri, 0, val);
2319     else
2320         outstring(pri, s);
2321 }
2323 /*
2324 * Print privilege set name
2325 */
2326 void
2327 prt_prn(private_t *pri, int raw, long val)
2328 {
2329     const char *s = NULL;
2331     if (!raw)
2332         s = priv_getsetbynum((int)val);
2334     if (s == NULL)
2335         prt_dec(pri, 0, val);
2336     else {
2337         char *dup = strdup(s);
2338         char *q;
2340         /* Do the best we can in this case */
2341         if (dup == NULL) {
2342             outstring(pri, s);
2343             return;
2344         }
2346         outstring(pri, "PRIV_");
2348         q = dup;
2350         while (*q != '\0') {
2351             *q = toupper(*q);
2352             q++;
2353         }
2354         outstring(pri, dup);
2355         free(dup);
2356     }
2357 }
2359 /*
2360 * Print process flag names.
2361 */
2362 void
2363 prt_pfl(private_t *pri, int raw, long val)
2364 {
2365     const char *s = NULL;
2367     if (!raw) {
2368         switch ((int)val) {
2369             case PRIV_DEBUG:      s = "PRIV_DEBUG";      break;
2370             case PRIV_AWARE:      s = "PRIV_AWARE";      break;
2371             case PRIV_XPOLICY:    s = "PRIV_XPOLICY";    break;
2372             case PRIV_AWARE_RESET: s = "PRIV_AWARE_RESET"; break;
2373             case PRIV_PFEEXEC:    s = "PRIV_PFEEXEC";    break;

```

```

2374         case NET_MAC_AWARE:      s = "NET_MAC_AWARE";   break;
2375         case NET_MAC_AWARE_INHERIT:
2376             s = "NET_MAC_AWARE_INHERIT";
2377             break;
2378     }
2379 }
2381     if (s == NULL)
2382         prt_dec(pri, 0, val);
2383     else
2384         outstring(pri, s);
2385 }
2387 /*
2388 * Print lgrp_affinity_{get,set}() arguments.
2389 */
2390 /*ARGSUSED*/
2391 void
2392 prt_laf(private_t *pri, int raw, long val)
2393 {
2394     lgrp_affinity_args_t laff;
2396     if (Pread(Proc, &laff, sizeof (lgrp_affinity_args_t), val) !=
2397         sizeof (lgrp_affinity_args_t)) {
2398         prt_hex(pri, 0, val);
2399         return;
2400     }
2401     /*
2402      * arrange the arguments in the order that user calls with
2403      */
2404     prt_dec(pri, 0, laff.idtype);
2405     outstring(pri, ", ");
2406     prt_dec(pri, 0, laff.id);
2407     outstring(pri, ", ");
2408     prt_dec(pri, 0, laff.lgrp);
2409     outstring(pri, ", ");
2410     if (pri->sys_args[0] == LGRP_SYS_AFFINITY_SET)
2411         prt_dec(pri, 0, laff.aff);
2412 }
2414 /*
2415 * Print a key_t as IPC_PRIVATE if it is 0.
2416 */
2417 void
2418 prt_key(private_t *pri, int raw, long val)
2419 {
2420     if (!raw && val == 0)
2421         outstring(pri, "IPC_PRIVATE");
2422     else
2423         prt_dec(pri, 0, val);
2424 }
2427 /*
2428 * Print zone_getattr() attribute types.
2429 */
2430 void
2431 prt_zga(private_t *pri, int raw, long val)
2432 {
2433     const char *s = NULL;
2435     if (!raw) {
2436         switch ((int)val) {
2437             case ZONE_ATTR_NAME:  s = "ZONE_ATTR_NAME";  break;
2438             case ZONE_ATTR_ROOT:  s = "ZONE_ATTR_ROOT";  break;
2439             case ZONE_ATTR_STATUS: s = "ZONE_ATTR_STATUS"; break;

```

```

2440     case ZONE_ATTR_PRIVSET: s = "ZONE_ATTR_PRIVSET"; break;
2441     case ZONE_ATTR_UNIQID:  s = "ZONE_ATTR_UNIQID"; break;
2442     case ZONE_ATTR_POOLID:  s = "ZONE_ATTR_POOLID"; break;
2443     case ZONE_ATTR_INITPID: s = "ZONE_ATTR_INITPID"; break;
2444     case ZONE_ATTR_SLBL:    s = "ZONE_ATTR_SLBL"; break;
2445     case ZONE_ATTR_INITNAME: s = "ZONE_ATTR_INITNAME"; break;
2446     case ZONE_ATTR_BOOTARGS: s = "ZONE_ATTR_BOOTARGS"; break;
2447     case ZONE_ATTR_BRAND:    s = "ZONE_ATTR_BRAND"; break;
2448     case ZONE_ATTR_FLAGS:    s = "ZONE_ATTR_FLAGS"; break;
2449     case ZONE_ATTR_PHYS_MCAP: s = "ZONE_ATTR_PHYS_MCAP"; break;
2450     }
2451 }

2453 if (s == NULL)
2454     prt_dec(pri, 0, val);
2455 else
2456     outstring(pri, s);
2457 }

2459 /*
2460  * Print a file descriptor as AT_FDCWD if necessary
2461  */
2462 void
2463 prt_atc(private_t *pri, int raw, long val)
2464 {
2465     if ((int)val == AT_FDCWD) {
2466         if (raw)
2467             prt_hex(pri, 0, (uint_t)AT_FDCWD);
2468         else
2469             outstring(pri, "AT_FDCWD");
2470     } else {
2471         prt_dec(pri, 0, val);
2472     }
2473 }

2475 /*
2476  * Print Trusted Networking database operation codes (labelsys; tn*)
2477  */
2478 static void
2479 prt_tnd(private_t *pri, int raw, long val)
2480 {
2481     const char *s = NULL;

2483     if (!raw) {
2484         switch ((tsol_dbops_t)val) {
2485             case TNDB_NOOP:      s = "TNDB_NOOP";      break;
2486             case TNDB_LOAD:     s = "TNDB_LOAD";      break;
2487             case TNDB_DELETE:   s = "TNDB_DELETE";   break;
2488             case TNDB_FLUSH:    s = "TNDB_FLUSH";    break;
2489             case TNDB_GET:      s = "TNDB_GET";      break;
2490         }
2491     }

2493     if (s == NULL)
2494         prt_dec(pri, 0, val);
2495     else
2496         outstring(pri, s);
2497 }

2499 /*
2500  * Print LIO_XX flags
2501  */
2502 void
2503 prt_lio(private_t *pri, int raw, long val)
2504 {
2505     if (raw)

```

```

2506         prt_dec(pri, 0, val);
2507     else if (val == LIO_WAIT)
2508         outstring(pri, "LIO_WAIT");
2509     else if (val == LIO_NOWAIT)
2510         outstring(pri, "LIO_NOWAIT");
2511     else
2512         prt_dec(pri, 0, val);
2513 }

2515 const char *
2516 door_flags(private_t *pri, long val)
2517 {
2518     door_attr_t attr = (door_attr_t)val;
2519     char *str = pri->code_buf;

2521     *str = '\0';
2522 #define PROCESS_FLAG(flag) \
2523     if (attr & flag) { \
2524         (void) strcat(str, "|" #flag, sizeof (pri->code_buf)); \
2525         attr &= ~flag; \
2526     }

2528     PROCESS_FLAG(DOOR_UNREF);
2529     PROCESS_FLAG(DOOR_UNREF_MULTI);
2530     PROCESS_FLAG(DOOR_PRIVATE);
2531     PROCESS_FLAG(DOOR_REFUSE_DESC);
2532     PROCESS_FLAG(DOOR_NO_CANCEL);
2533     PROCESS_FLAG(DOOR_LOCAL);
2534     PROCESS_FLAG(DOOR_REVOKED);
2535     PROCESS_FLAG(DOOR_IS_UNREF);
2536 #undef PROCESS_FLAG

2538     if (attr != 0 || *str == '\0') {
2539         size_t len = strlen(str);
2540         (void) snprintf(str + len, sizeof (pri->code_buf) - len,
2541             "|0x%X", attr);
2542     }

2544     return (str + 1);
2545 }

2547 /*
2548  * Print door_create() flags
2549  */
2550 void
2551 prt_dfl(private_t *pri, int raw, long val)
2552 {
2553     if (raw)
2554         prt_hex(pri, 0, val);
2555     else
2556         outstring(pri, door_flags(pri, val));
2557 }

2559 /*
2560  * Print door_*param() param argument
2561  */
2562 void
2563 prt_dpm(private_t *pri, int raw, long val)
2564 {
2565     if (raw)
2566         prt_hex(pri, 0, val);
2567     else if (val == DOOR_PARAM_DESC_MAX)
2568         outstring(pri, "DOOR_PARAM_DESC_MAX");
2569     else if (val == DOOR_PARAM_DATA_MIN)
2570         outstring(pri, "DOOR_PARAM_DATA_MIN");
2571     else if (val == DOOR_PARAM_DATA_MAX)

```

```

2572     outstring(pri, "DOOR_PARAM_DATA_MAX");
2573     else
2574         prt_hex(pri, 0, val);
2575 }

2577 /*
2578  * Print rctlsys subcodes
2579  */
2580 void
2581 prt_rsc(private_t *pri, int raw, long val) /* print utssys code */
2582 {
2583     const char *s = raw? NULL : rctlsyscode(val);

2585     if (s == NULL)
2586         prt_dec(pri, 0, val);
2587     else
2588         outstring(pri, s);
2589 }

2591 /*
2592  * Print getrctl flags
2593  */
2594 void
2595 prt_rgf(private_t *pri, int raw, long val)
2596 {
2597     long action = val & (~RCTLSYS_ACTION_MASK);

2599     if (raw)
2600         prt_hex(pri, 0, val);
2601     else if (action == RCTL_FIRST)
2602         outstring(pri, "RCTL_FIRST");
2603     else if (action == RCTL_NEXT)
2604         outstring(pri, "RCTL_NEXT");
2605     else if (action == RCTL_USAGE)
2606         outstring(pri, "RCTL_USAGE");
2607     else
2608         prt_hex(pri, 0, val);
2609 }

2611 /*
2612  * Print setrctl flags
2613  */
2614 void
2615 prt_rsf(private_t *pri, int raw, long val)
2616 {
2617     long action = val & (~RCTLSYS_ACTION_MASK);
2618     long pval = val & RCTL_LOCAL_ACTION_MASK;
2619     char *s = pri->code_buf;

2621     if (raw) {
2622         prt_hex(pri, 0, val);
2623         return;
2624     } else if (action == RCTL_INSERT)
2625         (void) strcpy(s, "RCTL_INSERT");
2626     else if (action == RCTL_DELETE)
2627         (void) strcpy(s, "RCTL_DELETE");
2628     else if (action == RCTL_REPLACE)
2629         (void) strcpy(s, "RCTL_REPLACE");
2630     else {
2631         prt_hex(pri, 0, val);
2632         return;
2633     }

2635     if (pval & RCTL_USE_RECIPIENT_PID) {
2636         pval ^= RCTL_USE_RECIPIENT_PID;
2637         (void) strcat(s, "|RCTL_USE_RECIPIENT_PID",

```

```

2638         sizeof (pri->code_buf));
2639     }

2641     if ((pval & RCTLSYS_ACTION_MASK) != 0)
2642         prt_hex(pri, 0, val);
2643     else if (*s != '\0')
2644         outstring(pri, s);
2645     else
2646         prt_hex(pri, 0, val);
2647 }

2649 /*
2650  * Print rctlctl flags
2651  */
2652 void
2653 prt_rcf(private_t *pri, int raw, long val)
2654 {
2655     long action = val & (~RCTLSYS_ACTION_MASK);

2657     if (raw)
2658         prt_hex(pri, 0, val);
2659     else if (action == RCTLCTL_GET)
2660         outstring(pri, "RCTLCTL_GET");
2661     else if (action == RCTLCTL_SET)
2662         outstring(pri, "RCTLCTL_SET");
2663     else
2664         prt_hex(pri, 0, val);
2665 }

2667 /*
2668  * Print setprojctl flags
2669  */
2670 void
2671 prt_spf(private_t *pri, int raw, long val)
2672 {
2673     long action = val & TASK_PROJ_MASK;

2675     if (!raw && (action == TASK_PROJ_PURGE))
2676         outstring(pri, "TASK_PROJ_PURGE");
2677     else
2678         prt_hex(pri, 0, val);
2679 }

2681 /*
2682  * Print forkx() flags
2683  */
2684 void
2685 prt_fxf(private_t *pri, int raw, long val)
2686 {
2687     char *str;

2689     if (val == 0)
2690         outstring(pri, "0");
2691     else if (raw || (val & ~(FORK_NOSIGCHLD | FORK_WAITPID)))
2692         prt_hhx(pri, 0, val);
2693     else {
2694         str = pri->code_buf;
2695         *str = '\0';
2696         if (val & FORK_NOSIGCHLD)
2697             (void) strcat(str, "|FORK_NOSIGCHLD",
2698                 sizeof (pri->code_buf));
2699         if (val & FORK_WAITPID)
2700             (void) strcat(str, "|FORK_WAITPID",
2701                 sizeof (pri->code_buf));
2702         outstring(pri, str + 1);
2703     }

```

```

2704 }

2706 /*
2707  * Print faccessat() flag
2708  */
2709 void
2710 prt_fat(private_t *pri, int raw, long val)
2711 {
2712     if (val == 0)
2713         outstring(pri, "0");
2714     else if (!raw && val == AT_EACCESS)
2715         outstring(pri, "AT_EACCESS");
2716     else
2717         prt_hex(pri, 0, val);
2718 }

2720 /*
2721  * Print unlinkat() flag
2722  */
2723 void
2724 prt_uat(private_t *pri, int raw, long val)
2725 {
2726     if (val == 0)
2727         outstring(pri, "0");
2728     else if (!raw && val == AT_REMOVEDIR)
2729         outstring(pri, "AT_REMOVEDIR");
2730     else
2731         prt_hex(pri, 0, val);
2732 }

2734 /*
2735  * Print AT_SYMLINK_NOFOLLOW / AT_SYMLINK_FOLLOW flag
2736  */
2737 void
2738 prt_snf(private_t *pri, int raw, long val)
2739 {
2740     if (val == 0)
2741         outstring(pri, "0");
2742     else if (!raw && val == AT_SYMLINK_NOFOLLOW)
2743         outstring(pri, "AT_SYMLINK_NOFOLLOW");
2744     else if (!raw && val == AT_SYMLINK_FOLLOW)
2745         outstring(pri, "AT_SYMLINK_FOLLOW");
2746     else
2747         prt_hex(pri, 0, val);
2748 }

2750 void
2751 prt_grf(private_t *pri, int raw, long val)
2752 {
2753     int first = 1;

2755     if (raw != 0 || val == 0 ||
2756         (val & ~(GRND_NONBLOCK | GRND_RANDOM)) != 0) {
2757         outstring(pri, "0");
2758         return;
2759     }

2761     if (val & GRND_NONBLOCK) {
2762         outstring(pri, "|GRND_NONBLOCK" + first);
2763         first = 0;
2764     }
2765     if (val & GRND_RANDOM) {
2766         outstring(pri, "|GRND_RANDOM" + first);
2767         first = 0;
2768     }
2769 }

```

```

2771 /*
2772  * Array of pointers to print functions, one for each format.
2773  */
2774 void (* const Print[])() = {
2775     prt_nov, /* NOV -- no value */
2776     prt_dec, /* DEC -- print value in decimal */
2777     prt_oct, /* OCT -- print value in octal */
2778     prt_hex, /* HEX -- print value in hexadecimal */
2779     prt_dex, /* DEX -- print value in hexadecimal if big enough */
2780     prt_stg, /* STG -- print value as string */
2781     prt_ioc, /* IOC -- print ioctl code */
2782     prt_fcn, /* FCN -- print fcntl code */
2783     prt_s86, /* S86 -- print sysi86 code */
2784     prt_uts, /* UTS -- print utssys code */
2785     prt_opn, /* OPN -- print open code */
2786     prt_sig, /* SIG -- print signal name plus flags */
2787     prt_uat, /* UAT -- print unlinkat() flag */
2788     prt_msc, /* MSC -- print msgsys command */
2789     prt_msf, /* MSF -- print msgsys flags */
2790     prt_smc, /* SMC -- print semsys command */
2791     prt_sef, /* SEF -- print semsys flags */
2792     prt_shc, /* SHC -- print shmsys command */
2793     prt_shf, /* SHF -- print shmsys flags */
2794     prt_fat, /* FAT -- print faccessat( flag */
2795     prt_sfs, /* SFS -- print sysfs code */
2796     prt_rst, /* RST -- print string returned by syscall */
2797     prt_smf, /* SMF -- print streams message flags */
2798     prt_ioa, /* IOA -- print ioctl argument */
2799     prt_pip, /* PIP -- print pipe flags */
2800     prt_mtf, /* MTF -- print mount flags */
2801     prt_mft, /* MFT -- print mount file system type */
2802     prt_iob, /* IOB -- print contents of I/O buffer */
2803     prt_hhx, /* HHX -- print value in hexadecimal (half size) */
2804     prt_wop, /* WOP -- print waitsys() options */
2805     prt_spm, /* SPM -- print sigprocmask argument */
2806     prt_rlk, /* RLK -- print readlink buffer */
2807     prt_mpr, /* MPR -- print mmap()/mprotect() flags */
2808     prt_mty, /* MTY -- print mmap() mapping type flags */
2809     prt_mcf, /* MCF -- print memcntl() function */
2810     prt_mc4, /* MC4 -- print memcntl() (fourth) argument */
2811     prt_mc5, /* MC5 -- print memcntl() (fifth) argument */
2812     prt_mad, /* MAD -- print madvise() argument */
2813     prt_ulm, /* ULM -- print ulimit() argument */
2814     prt_rlm, /* RLM -- print get/setrlimit() argument */
2815     prt_cnf, /* CNF -- print sysconfig() argument */
2816     prt_inf, /* INF -- print sysinfo() argument */
2817     prt_ptc, /* PTC -- print pathconf/fpathconf() argument */
2818     prt_fui, /* FUI -- print fusers() input argument */
2819     prt_idt, /* IDT -- print idtype_t, waitid() argument */
2820     prt_lwf, /* LWF -- print lwp_create() flags */
2821     prt_itm, /* ITM -- print [get|set]itimer() arg */
2822     prt_llo, /* LLO -- print long long offset arg */
2823     prt_mod, /* MOD -- print modctl() subcode */
2824     prt_wnh, /* WHN -- print lseek() whence argument */
2825     prt_acl, /* ACL -- print acl() code */
2826     prt_aio, /* AIO -- print kaio() code */
2827     prt_aud, /* AUD -- print auditsys() code */
2828     prt_uns, /* DEC -- print value in unsigned decimal */
2829     prt_clc, /* CLC -- print cladm command argument */
2830     prt_clf, /* CLF -- print cladm flag argument */
2831     prt_cor, /* COR -- print corectl() subcode */
2832     prt_cco, /* CCO -- print corectl() options */
2833     prt_ccc, /* CCC -- print corectl() content */
2834     prt_rcc, /* RCC -- print corectl() returned content */
2835     prt_cpc, /* CPC -- print cpc() subcode */

```

```
2836     prt_sqc,      /* SQC -- print sigqueue() si_code argument */
2837     prt_pc4,      /* PC4 -- print pricntlsys() (fourth) argument */
2838     prt_pc5,      /* PC5 -- print pricntlsys() (key, value) pairs */
2839     prt_pst,      /* PST -- print processor set id */
2840     prt_mif,      /* MIF -- print meminfo() arguments */
2841     prt_pfm,      /* PFM -- print so_socket() proto-family (1st) arg */
2842     prt_skt,      /* SKT -- print so_socket() socket-type (2nd) arg */
2843     prt_skp,      /* SKP -- print so_socket() protocol (3rd) arg */
2844     prt_skv,      /* SKV -- print socket version arg */
2845     prt_sol,      /* SOL -- print [sg]setsockopt() level (2nd) arg */
2846     prt_son,      /* SON -- print [sg]setsockopt() opt-name (3rd) arg */
2847     prt_utt,      /* UTT -- print utrap type */
2848     prt_uth,      /* UTH -- print utrap handler */
2849     prt_acc,      /* ACC -- print access() flags */
2850     prt_sht,      /* SHT -- print shutdown() how (2nd) argument */
2851     prt_ffg,      /* FFG -- print fcntl() flags (3rd) argument */
2852     prt_prs,      /* PRS -- print privilege set */
2853     prt_pro,      /* PRO -- print privilege set operation */
2854     prt_prn,      /* PRN -- print privilege set name */
2855     prt_pfl,      /* PFL -- print privilege/process flag name */
2856     prt_laf,      /* LAF -- print lgrp_affinity arguments */
2857     prt_key,      /* KEY -- print key_t 0 as IPC_PRIVATE */
2858     prt_zga,      /* ZGA -- print zone_getattr attribute types */
2859     prt_atc,      /* ATC -- print AT_FDCWD or file descriptor */
2860     prt_lio,      /* LIO -- print LIO_XX flags */
2861     prt_dfl,      /* DFL -- print door_create() flags */
2862     prt_dpm,      /* DPM -- print DOOR_PARAM_XX flags */
2863     prt_tnd,      /* TND -- print trusted network data base opcode */
2864     prt_rsc,      /* RSC -- print rctlsys() subcodes */
2865     prt_rgf,      /* RGF -- print getrctl() flags */
2866     prt_rsf,      /* RSF -- print setrctl() flags */
2867     prt_rcf,      /* RCF -- print rctlsys_ctl() flags */
2868     prt_fxf,      /* FXF -- print forkx() flags */
2869     prt_spf,      /* SPF -- print rctlsys_projset() flags */
2870     prt_unl,      /* UNL -- as prt_uns except for -1 */
2871     prt_mob,      /* MOB -- print mmapobj() flags */
2872     prt_snf,      /* SNF -- print AT_SYMLINK_[NO]FOLLOW flag */
2873     prt_skc,      /* SKC -- print sockconfig() subcode */
2874     prt_acf,      /* ACF -- print accept4 flags */
2875     prt_pfd,      /* PFD -- print pipe fds */
2876     prt_grf,      /* GRF -- print getrandom flags */
2877     prt_dec,      /* HID -- hidden argument, make this the last one */
2878 };
```