

freebsd-performance: [tjr@freebsd.org: Port of Niels Provos's file descriptor allocation code]

## [tjr@freebsd.org: Port of Niels Provos's file descriptor allocation code]

**Source:** <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/performance/2003-11/0022.html>

---

**From:** Sean Chittenden ([seanc\\_at\\_FreeBSD.org](mailto:seanc_at_FreeBSD.org))

**Date:** 11/28/03

Date: Thu, 27 Nov 2003 16:54:57 -0800

To: performance@FreeBSD.org

This is of more interest here I think so I've forwarded the message here. In short, Tim's ported NetBSD's code new FD allocation code and here's a graph of the resulting performance boost:

<http://www.citi.umich.edu/u/provos/benchmark/netbsd-fdalloc.jpg>

For the various parties interested on this list (of which I'm sure there are many), enjoy! -sc

--

Sean Chittenden

**attached mail follows:**

---

Date: Thu, 27 Nov 2003 18:02:39 +1100

To: current@freebsd.org

I've ported Niels Provos's file descriptor allocation code to FreeBSD in case anyone wants to try it out & run some benchmarks. If the performance boost turns out to be worth the added complexity, I might clean it up a bit and commit it.

See <http://mail-index.netbsd.org/tech-perform/2003/10/28/0001.html> and Banga & Mogul's USENIX paper (linked to from the other URL) for the details.

<http://perforce.freebsd.org/chv.cgi?CH=43066>

--- //depot/user/tjr/freebsd-tjr/src/sys/kern/init\_main.c 2003/10/05 17:21:48

+++ //depot/user/tjr/freebsd-tjr/src/sys/kern/init\_main.c 2003/11/26 19:42:24

@@ -415,6 +415,8 @@

fdp->fd\_fd.fd\_ofiles = fdp->fd\_dfiles;

fdp->fd\_fd.fd\_ofileflags = fdp->fd\_dfileflags;

fdp->fd\_fd.fd\_nfiles = NDFILE;

+ fdp->fd\_fd.fd\_himap = fdp->fd\_dhimap;

[tjr@freebsd.org: Port of Niels Provos's file descriptor allocation code]

freebsd-performance: [tjr@freebsd.org: Port of Niels Provos's file descriptor allocation code]

```
+ fdp->fd_fd.fd_lomap = fdp->fd_dlomap;

    /* Create the limits structures. */
    p->p_limit = &limit0;
--- //depot/user/tjr/freebsd-tjr/src/sys/kern/kern_descrip.c 2003/10/27 00:31:20
+++ //depot/user/tjr/freebsd-tjr/src/sys/kern/kern_descrip.c 2003/11/26 19:42:24
@@ -96,6 +96,7 @@

static int do_dup(struct thread *td, enum dup_type type, int old, int new,
    register_t *retval);
+static __inline int find_next_zero(uint32_t *, int, u_int);

/*
 * Descriptor management.
@@ -105,6 +106,62 @@
struct sx filelist_lock; /* sx to protect filelist */
struct mtx sigio_lock; /* mtx to protect pointers to sigio */

+static __inline int
+find_next_zero(uint32_t *bitmap, int want, u_int bits)
+{
+ int i, off, maxoff;
+ uint32_t sub;
+
+ if (want > bits)
+ return -1;
+
+ off = want >> NDENTRYSHIFT;
+ i = want & NDENTRYMASK;
+ if (i) {
+ sub = bitmap[off] | ((u_int)~0 >> (NDENTRIES - i));
+ if (sub != ~0)
+ goto found;
+ off++;
+ }
+
+ maxoff = NDLOSLOTS(bits);
+ while (off < maxoff) {
+ if ((sub = bitmap[off]) != ~0)
+ goto found;
+ off++;
+ }
+
+ return (-1);
+
+found:
+ return (off << NDENTRYSHIFT) + ffs(~sub) - 1;
+}
+
+int
+fd_find_last_set(struct filedesc *fd, int last)
```

[tjr@freebsd.org: Port of Niels Provos's file descriptor allocation code]

```

+{
+ int off, i;
+ struct file **ofiles = fd->fd_ofiles;
+ uint32_t *bitmap = fd->fd_lomap;
+
+ off = (last - 1) >> NDENTRYSHIFT;
+
+ while (!bitmap[off] && off >= 0)
+ off--;
+
+ if (off < 0)
+ return (0);
+
+ i = ((off + 1) << NDENTRYSHIFT) - 1;
+ if (i >= last)
+ i = last - 1;
+
+ while (i > 0 && ofiles[i] == NULL)
+ i--;
+
+ return (i);
+}
+
+/*
+ * System calls on descriptors.
+ */
@@ -505,13 +562,8 @@
+ * avoid this case.
+ */
+ if (fdp->fd_ofiles[old] != fp) {
- if (fdp->fd_ofiles[new] == NULL) {
- if (new < fdp->fd_freefile)
- fdp->fd_freefile = new;
- while (fdp->fd_lastfile > 0 &&
- fdp->fd_ofiles[fdp->fd_lastfile] == NULL)
- fdp->fd_lastfile--;
- }
+ if (fdp->fd_ofiles[new] == NULL)
+ fd_unused(fdp, new);
+ FILEDESC_UNLOCK(fdp);
+ fdrop(fp, td);
+ return (EBADF);
@@ -545,8 +597,7 @@
+ */
+ fdp->fd_ofiles[new] = fp;
+ fdp->fd_ofileflags[new] = fdp->fd_ofileflags[old] &~ UF_EXCLOSE;
- if (new > fdp->fd_lastfile)
- fdp->fd_lastfile = new;
+ fd_used(fdp, new);
+ FILEDESC_UNLOCK(fdp);
+ *retval = new;

```

```
@@ -836,6 +887,7 @@
#endif
    fdp->fd_ofiles[fd] = NULL;
    fdp->fd_ofileflags[fd] = 0;
+ fd_unused(fdp, fd);
    if (td->td_proc->p_fdtol != NULL) {
        /*
         * Ask fdfree() to sleep to ensure that all relevant
@@ -849,10 +901,6 @@
        * we now hold the fp reference that used to be owned by the descriptor
        * array.
        */
- while (fdp->fd_lastfile > 0 && fdp->fd_ofiles[fdp->fd_lastfile] == NULL)
- fdp->fd_lastfile--;
- if (fd < fdp->fd_freefile)
- fdp->fd_freefile = fd;
    if (fd < fdp->fd_knlistsize) {
        FILEDESC_UNLOCK(fdp);
        knote_fdclose(td, fd);
@@ -1052,9 +1100,11 @@
    struct proc *p = td->td_proc;
    struct filedesc *fdp = td->td_proc->p_fdp;
    int i;
- int lim, last, nfiles;
+ int lim, last, nfiles, oldnfiles;
    struct file **newofile, **oldofile;
    char *newofileflags;
+ uint32_t *newhimap, *newlomap, *oldhimap, *oldlomap;
+ u_int off, new;

    FILEDESC_LOCK_ASSERT(fdp, MA_OWNED);

@@ -1066,12 +1116,28 @@
    lim = min((int)p->p_rlimit[RLIMIT_NOFILE].rlim_cur, maxfilesperproc);
    for (;;) {
        last = min(fdp->fd_nfiles, lim);
+again:
        i = max(want, fdp->fd_freefile);
- for (; i < last; i++) {
- if (fdp->fd_ofiles[i] == NULL) {
- fdp->fd_ofileflags[i] = 0;
- if (i > fdp->fd_lastfile)
- fdp->fd_lastfile = i;
+ off = i >> NDENTRYSHIFT;
+ new = find_next_zero(fdp->fd_himap, off,
+ (last + NDENTRIES - 1) >> NDENTRYSHIFT);
+ if (new != -1) {
+ i = find_next_zero(&fdp->fd_lomap[new],
+ new > off ? 0 : i & NDENTRYMASK, NDENTRIES);
+ if (i == -1) {
```

```

+ /*
+ * free file descriptor in this block was
+ * below want, try again with higher want.
+ */
+ want = (new + 1) << NDENTRYSHIFT;
+ goto again;
+ }
+ i += (new << NDENTRYSHIFT);
+ if (i < last) {
+ KASSERT(fdp->fd_ofiles[i] == NULL,
+ ("free descriptor isn't"));
+ fdp->fd_ofileflags[i] = 0; /* XXX needed? */
+ fd_used(fdp, i);
+         if (want <= fdp->fd_freefile)
+             fdp->fd_freefile = i;
+         *result = i;
@@ -1082,7 +1148,7 @@
+         /*
+          * No space in current array. Expand?
+          */
- if (i >= lim)
+ if (fdp->fd_nfiles >= lim)
+     return (EMFILE);
+     if (fdp->fd_nfiles < NDEXTENT)
+         nfiles = NDEXTENT;
@@ -1090,8 +1156,16 @@
+         nfiles = 2 * fdp->fd_nfiles;
+         while (nfiles < want)
+             nfiles <<= 1;
+ oldnfiles = fdp->fd_nfiles;
+ FILEDESC_UNLOCK(fdp);
+ newofile = malloc(nfiles * OFILESIZE, M_FILEDESC, M_WAITOK);
+ if (NDHISLOTS(nfiles) > NDHISLOTS(oldnfiles)) {
+ newhimap = malloc(NDHISLOTS(nfiles) * sizeof(uint32_t),
+ M_FILEDESC, M_WAITOK);
+ newlomap = malloc(NDLOSLOTS(nfiles) * sizeof(uint32_t),
+ M_FILEDESC, M_WAITOK);
+ } else
+ newhimap = newlomap = NULL;

+         /*
+          * Deal with file-table extend race that might have
@@ -1101,6 +1175,10 @@
+         if (fdp->fd_nfiles >= nfiles) {
+             FILEDESC_UNLOCK(fdp);
+             free(newofile, M_FILEDESC);
+ if (newhimap != NULL)
+ free(newhimap, M_FILEDESC);
+ if (newlomap != NULL)
+ free(newlomap, M_FILEDESC);
+             FILEDESC_LOCK(fdp);

```

```

        continue;
    }
@@ -1122,11 +1200,33 @@
        oldofile = NULL;
        fdp->fd_ofiles = newofile;
        fdp->fd_ofileflags = newofileflags;
+ oldlomap = oldhimap = NULL;
+ if (NDHISLOTS(nfiles) > NDHISLOTS(oldnfiles)) {
+ memcpy(newhimap, fdp->fd_himap,
+ (i = NDHISLOTS(oldnfiles) * sizeof(uint32_t)));
+ memset((char *)newhimap + i, 0,
+ NDHISLOTS(nfiles) * sizeof(uint32_t) - i);
+ memcpy(newlomap, fdp->fd_lomap,
+ (i = NDLOSLOTS(oldnfiles) * sizeof(uint32_t)));
+ memset((char *)newlomap + i, 0,
+ NDLOSLOTS(nfiles) * sizeof(uint32_t) - i);
+ if (NDHISLOTS(oldnfiles) > NDHISLOTS(NDFILE)) {
+ oldhimap = fdp->fd_himap;
+ oldlomap = fdp->fd_lomap;
+ }
+ fdp->fd_himap = newhimap;
+ fdp->fd_lomap = newlomap;
+ }
        fdp->fd_nfiles = nfiles;
        fdexpand++;
- if (oldofile != NULL) {
+ if (oldofile != NULL || oldlomap != NULL || oldhimap != NULL) {
        FILEDESC_UNLOCK(fdp);
- free(oldofile, M_FILEDESC);
+ if (oldofile != NULL)
+ free(oldofile, M_FILEDESC);
+ if (oldlomap != NULL)
+ free(oldlomap, M_FILEDESC);
+ if (oldhimap != NULL)
+ free(oldhimap, M_FILEDESC);
        FILEDESC_LOCK(fdp);
    }
}
@@ -1276,6 +1376,8 @@
    newfdp->fd_fd.fd_ofileflags = newfdp->fd_dfileflags;
    newfdp->fd_fd.fd_nfiles = NDFILE;
    newfdp->fd_fd.fd_knlistsize = -1;
+ newfdp->fd_fd.fd_himap = newfdp->fd_dhimap;
+ newfdp->fd_fd.fd_lomap = newfdp->fd_dlomap;
    return (&newfdp->fd_fd);
}

@@ -1340,6 +1442,10 @@
    newfdp->fd_ofiles = ((struct filedesc0 *) newfdp)->fd_dfiles;
    newfdp->fd_ofileflags =
        ((struct filedesc0 *) newfdp)->fd_dfileflags;

```

```

+ newfdp->fd_himap =
+ ((struct filedesc0 *) newfdp)->fd_dhimap;
+ newfdp->fd_lomap =
+ ((struct filedesc0 *) newfdp)->fd_dlomap;
+     i = NDFILE;
+     } else {
+         /*
@@ -1354,6 +1460,17 @@
+         FILEDESC_UNLOCK(fdp);
+         MALLOC(newfdp->fd_ofiles, struct file **, i * OFILESIZE,
+         M_FILEDESC, M_WAITOK);
+ if (NDHISLOTS(i) <= NDHISLOTS(NDFILE)) {
+ newfdp->fd_himap =
+ ((struct filedesc0 *) newfdp)->fd_dhimap;
+ newfdp->fd_lomap =
+ ((struct filedesc0 *) newfdp)->fd_dlomap;
+ } else {
+ newfdp->fd_himap = malloc(NDHISLOTS(i) * sizeof(uint32_t),
+ M_FILEDESC, M_WAITOK);
+ newfdp->fd_lomap = malloc(NDLOSLOTS(i) * sizeof(uint32_t),
+ M_FILEDESC, M_WAITOK);
+ }
+
+     FILEDESC_LOCK(fdp);
+     newfdp->fd_lastfile = fdp->fd_lastfile;
+     newfdp->fd_nfiles = fdp->fd_nfiles;
@@ -1377,6 +1494,10 @@
+     newfdp->fd_nfiles = i;
+     bcopy(fdp->fd_ofiles, newfdp->fd_ofiles, i * sizeof(struct file **));
+     bcopy(fdp->fd_ofileflags, newfdp->fd_ofileflags, i * sizeof(char));
+ if (i < NDENTRIES * NDENTRIES)
+ i = NDENTRIES * NDENTRIES; /* size of inlined bitmaps */
+ memcpy(newfdp->fd_himap, fdp->fd_himap, NDHISLOTS(i)*sizeof(uint32_t));
+ memcpy(newfdp->fd_lomap, fdp->fd_lomap, NDLOSLOTS(i)*sizeof(uint32_t));
+
+ /*
+  * kq descriptors cannot be copied.
@@ -1385,12 +1506,9 @@
+     fpp = &newfdp->fd_ofiles[newfdp->fd_lastfile];
+     for (i = newfdp->fd_lastfile; i >= 0; i--, fpp--) {
+         if (*fpp != NULL && (*fpp)->f_type == DTYPE_KQUEUE) {
+ fd_unused(newfdp, i);
+         *fpp = NULL;
+ - if (i < newfdp->fd_freefile)
+ - newfdp->fd_freefile = i;
+         }
+ - if (*fpp == NULL && i == newfdp->fd_lastfile && i > 0)
+ - newfdp->fd_lastfile--;
+         }
+         newfdp->fd_knlist = NULL;
+         newfdp->fd_knlistsize = -1;
@@ -1526,6 +1644,10 @@

```

```
    if (fdp->fd_nfiles > NDFILE)
        FREE(fdp->fd_ofiles, M_FILEDESC);
+ if (NDHISLOTS(fdp->fd_nfiles) > NDHISLOTS(NDFILE)) {
+ free(fdp->fd_himap, M_FILEDESC);
+ free(fdp->fd_lomap, M_FILEDESC);
+ }
    if (fdp->fd_cdir)
        vrele(fdp->fd_cdir);
    if (fdp->fd_rdir)
@@ -1603,15 +1725,12 @@
        fp = fdp->fd_ofiles[i];
        fdp->fd_ofiles[i] = NULL;
        fdp->fd_ofileflags[i] = 0;
- if (i < fdp->fd_freefile)
- fdp->fd_freefile = i;
+ fd_unused(fdp, i);
        FILEDESC_UNLOCK(fdp);
        (void) closef(fp, td);
        FILEDESC_LOCK(fdp);
    }
}
- while (fdp->fd_lastfile > 0 && fdp->fd_ofiles[fdp->fd_lastfile] == NULL)
- fdp->fd_lastfile--;
    FILEDESC_UNLOCK(fdp);
}

@@ -1657,15 +1776,12 @@
        fp = fdp->fd_ofiles[i];
        fdp->fd_ofiles[i] = NULL;
        fdp->fd_ofileflags[i] = 0;
- if (i < fdp->fd_freefile)
- fdp->fd_freefile = i;
+ fd_unused(fdp, i);
        FILEDESC_UNLOCK(fdp);
        (void) closef(fp, td);
        FILEDESC_LOCK(fdp);
    }
}
- while (fdp->fd_lastfile > 0 && fdp->fd_ofiles[fdp->fd_lastfile] == NULL)
- fdp->fd_lastfile--;
    FILEDESC_UNLOCK(fdp);
}

@@ -1714,6 +1830,7 @@
        FILEDESC_LOCK(fdp);
        if (fdp->fd_ofiles[fd] == fp) {
            fdp->fd_ofiles[fd] = NULL;
+ fd_unused(fdp, fd);
            extraref = 1;
        }
}
```

```

        FILEDESC_UNLOCK(fdp);
@@ -2182,10 +2299,9 @@
#endif
        fdp->fd_ofiles[indx] = wfp;
        fdp->fd_ofileflags[indx] = fdp->fd_ofileflags[dfd];
+ fd_used(fdp, indx);
        fhold_locked(wfp);
        FILE_UNLOCK(wfp);
- if (indx > fdp->fd_lastfile)
- fdp->fd_lastfile = indx;
        if (fp != NULL)
            FILE_LOCK(fp);
        FILEDESC_UNLOCK(fdp);
@@ -2210,21 +2326,8 @@
        fdp->fd_ofiles[dfd] = NULL;
        fdp->fd_ofileflags[indx] = fdp->fd_ofileflags[dfd];
        fdp->fd_ofileflags[dfd] = 0;
-
- /*
- * Complete the clean up of the filedesc structure by
- * recomputing the various hints.
- */
- if (indx > fdp->fd_lastfile) {
- fdp->fd_lastfile = indx;
- } else {
- while (fdp->fd_lastfile > 0 &&
- fdp->fd_ofiles[fdp->fd_lastfile] == NULL) {
- fdp->fd_lastfile--;
- }
- if (dfd < fdp->fd_freefile)
- fdp->fd_freefile = dfd;
- }
+ fd_unused(fdp, dfd);
+ fd_used(fdp, indx);
        if (fp != NULL)
            FILE_LOCK(fp);
        FILEDESC_UNLOCK(fdp);
---- //depot/user/tjr/freebsd-tjr/src/sys/kern/sys_pipe.c 2003/11/12 14:04:51
+++ //depot/user/tjr/freebsd-tjr/src/sys/kern/sys_pipe.c 2003/11/26 19:42:24
@@ -258,6 +258,7 @@
        FILEDESC_LOCK(fdp);
        if (fdp->fd_ofiles[td->td_retval[0]] == rf) {
            fdp->fd_ofiles[td->td_retval[0]] = NULL;
+ fd_unused(fdp, td->td_retval[0]);
        FILEDESC_UNLOCK(fdp);
        fdrop(rf, td);
    } else
---- //depot/user/tjr/freebsd-tjr/src/sys/kern/uipc_syscalls.c 2003/11/19 22:23:06
+++ //depot/user/tjr/freebsd-tjr/src/sys/kern/uipc_syscalls.c 2003/11/26 19:42:24
@@ -124,6 +124,7 @@
    if (error) {

```

freebsd-performance: [tjr@freebsd.org: Port of Niels Provos's file descriptor allocation code]

```
    if (fdp->fd_ofiles[fd] == fp) {
        fdp->fd_ofiles[fd] = NULL;
+ fd_unused(fdp, fd);
        FILEDESC_UNLOCK(fdp);
        fdrop(fp, td);
    } else
@@ -389,6 +390,7 @@
        FILEDESC_LOCK(fdp);
        if (fdp->fd_ofiles[fd] == nfp) {
            fdp->fd_ofiles[fd] = NULL;
+ fd_unused(fdp, fd);
            FILEDESC_UNLOCK(fdp);
            fdrop(nfp, td);
        } else {
@@ -583,6 +585,7 @@
        FILEDESC_LOCK(fdp);
        if (fdp->fd_ofiles[sv[1]] == fp2) {
            fdp->fd_ofiles[sv[1]] = NULL;
+ fd_unused(fdp, sv[1]);
            FILEDESC_UNLOCK(fdp);
            fdrop(fp2, td);
        } else
@@ -592,6 +595,7 @@
        FILEDESC_LOCK(fdp);
        if (fdp->fd_ofiles[sv[0]] == fp1) {
            fdp->fd_ofiles[sv[0]] = NULL;
+ fd_unused(fdp, sv[0]);
            FILEDESC_UNLOCK(fdp);
            fdrop(fp1, td);
        } else
--- //depot/user/tjr/freebsd-tjr/src/sys/kern/vfs_syscalls.c 2003/11/12 14:04:51
+++ //depot/user/tjr/freebsd-tjr/src/sys/kern/vfs_syscalls.c 2003/11/26 19:42:24
@@ -998,6 +998,7 @@
        FILEDESC_LOCK(fdp);
        if (fdp->fd_ofiles[indx] == fp) {
            fdp->fd_ofiles[indx] = NULL;
+ fd_unused(fdp, indx);
            FILEDESC_UNLOCK(fdp);
            fdrop(fp, td);
        } else
@@ -1090,6 +1091,7 @@
        FILEDESC_LOCK(fdp);
        if (fdp->fd_ofiles[indx] == fp) {
            fdp->fd_ofiles[indx] = NULL;
+ fd_unused(fdp, indx);
            FILEDESC_UNLOCK(fdp);
            fdrop(fp, td);
        } else
@@ -3978,6 +3980,7 @@
        FILEDESC_LOCK(fdp);
        if (fdp->fd_ofiles[indx] == fp) {
```

[tjr@freebsd.org: Port of Niels Provos's file descriptor allocation code]

freebsd-performance: [tjr@freebsd.org: Port of Niels Provos's file descriptor allocation code]

```
        fdp->fd_ofiles[indx] = NULL;
+ fd_unused(fdp, indx);
        FILEDESC_UNLOCK(fdp);
        fdrop(fp, td);
    } else
--- //depot/user/tjr/freebsd-tjr/src/sys/sys/filedesc.h 2003/11/12 14:04:51
+++ //depot/user/tjr/freebsd-tjr/src/sys/sys/filedesc.h 2003/11/26 19:42:24
@@ -57,6 +57,11 @@
    */
#define NDFILE 20
#define NDEXTENT 50 /* 250 bytes in 256-byte alloc. */
+#define NDENTRIES 32 /* 32 fds per entry */
+#define NDENTRYMASK (NDENTRIES - 1)
+#define NDENTRYSHIFT 5 /* bits per entry */
+#define NDLOOTS(x) (((x) + NDENTRIES - 1) >> NDENTRYSHIFT)
+#define NDHISLOTS(x) ((NDLOOTS(x) + NDENTRIES - 1) >> NDENTRYSHIFT)

struct filedesc {
    struct file **fd_ofiles; /* file structures for open files */
@@ -65,6 +70,8 @@
    struct vnode *fd_rdir; /* root directory */
    struct vnode *fd_jdir; /* jail root directory */
    int fd_nfiles; /* number of open files allocated */
+ uint32_t *fd_himap; /* each bit points to 32 fds */
+ uint32_t *fd_lomap; /* bitmap of free fds */
    int fd_lastfile; /* high-water mark of fd_ofiles */
    int fd_freelfile; /* approx. next free file */
    u_short fd_cmask; /* mask for file creation */
@@ -91,6 +98,12 @@
    */
    struct file *fd_dfiles[NDFILE];
    char fd_dfileflags[NDFILE];
+ /*
+ * These arrays are used when the number of open files is
+ * <= 1024, and are then pointed to by the pointers above.
+ */
+ uint32_t fd_dhimap[NDENTRIES >> NDENTRYSHIFT];
+ uint32_t fd_dlomap[NDENTRIES];
};

@@ -143,6 +156,7 @@
int dupfdopen(struct thread *td, struct filedesc *fdp, int indx, int dfd,
    int mode, int error);
int falloc(struct thread *p, struct file **resultfp, int *resultfd);
+int fd_find_last_set(struct filedesc *, int);
int fdalloc(struct thread *p, int want, int *result);
int fdavail(struct thread *td, int n);
void fdcloseexec(struct thread *td);
@@ -170,6 +184,37 @@
    return ((u_int)fd >= (u_int)fdp->fd_nfiles ? NULL : fdp->fd_ofiles[fd]);
```

[tjr@freebsd.org: Port of Niels Provos's file descriptor allocation code]

freebsd-performance: [tjr@freebsd.org: Port of Niels Provos's file descriptor allocation code]

```
}

+static __inline void
+fd_used(struct filedesc *fdp, int fd)
+{
+  u_int off = fd >> NDENTRYSHIFT;
+
+  fdp->fd_lomap[off] |= 1 << (fd & NDENTRYMASK);
+  if (fdp->fd_lomap[off] == ~0)
+  fdp->fd_himap[off >> NDENTRYSHIFT] |= 1 << (off & NDENTRYMASK);
+
+  if (fd > fdp->fd_lastfile)
+  fdp->fd_lastfile = fd;
+}
+
+static __inline void
+fd_unused(struct filedesc *fdp, int fd)
+{
+  u_int off = fd >> NDENTRYSHIFT;
+
+  if (fd < fdp->fd_freefile)
+  fdp->fd_freefile = fd;
+  if (fdp->fd_lomap[off] == ~0)
+  fdp->fd_himap[off >> NDENTRYSHIFT] &= ~(1 << (off & NDENTRYMASK));
+  fdp->fd_lomap[off] &= ~(1 << (fd & NDENTRYMASK));
+  #ifdef KASSERT /* XXX */
+  KASSERT(fd <= fdp->fd_lastfile,
+  ("fd_unused: fd_lastfile inconsistent"));
+  #endif
+  if (fd == fdp->fd_lastfile)
+  fdp->fd_lastfile = fd_find_last_set(fdp, fd);
+}
+
+extern struct mtx fdesc_mtx;

#endif /* _KERNEL */
```

---

freebsd-current@freebsd.org mailing list

<http://lists.freebsd.org/mailman/listinfo/freebsd-current>

To unsubscribe, send any mail to "freebsd-current-unsubscribe@freebsd.org"

---

freebsd-performance@freebsd.org mailing list

<http://lists.freebsd.org/mailman/listinfo/freebsd-performance>

To unsubscribe, send any mail to "freebsd-performance-unsubscribe@freebsd.org"

[tjr@freebsd.org: Port of Niels Provos's file descriptor allocation code]