

```

/* The kernel call implemented in this file:
 * m_type: SYS_FORK
 *
 * The parameters for this kernel call are:
 * m1_i1: PR_SLOT (child's process table slot)
 * m1_i2: PR_ENDPT (parent, process that forked)
 */

#include "../system.h"
#include <signal.h>
#if (CHIP == INTEL)
#include "../protect.h"
#endif

#include <minix/endpoint.h>

#if USE_FORK

/*-----*
 *          do_fork          *
 *-----*/
PUBLIC int do_fork(m_ptr)
register message *m_ptr; /* pointer to request message */
{
/* Handle sys_fork(). PR_ENDPT has forked. The child is PR_SLOT. */
#if (CHIP == INTEL)
    reg_t old_ldt_sel;
#endif
    register struct proc *rpc; /* child process pointer */
    struct proc *rpp; /* parent process pointer */
    int i, gen;
    int p_proc;

    if(!isokendpt(m_ptr->PR_ENDPT, &p_proc))
        return EINVAL;
    rpp = proc_addr(p_proc);
    rpc = proc_addr(m_ptr->PR_SLOT);
    if (isempty(rpp) || ! isempty(rpc)) return(EINVAL);

    /* Copy parent 'proc' struct to child. And reinitialize some fields. */
    gen = _ENDPOINT_G(rpc->p_endpoint);
#if (CHIP == INTEL)
    old_ldt_sel = rpc->p_ldt_sel; /* backup local descriptors */
    *rpc = *rpp; /* copy 'proc' struct */
    rpc->p_ldt_sel = old_ldt_sel; /* restore descriptors */
#else
    *rpc = *rpp; /* copy 'proc' struct */
#endif
    if(++gen >= _ENDPOINT_MAX_GENERATION) /* increase generation */
        gen = 1; /* generation number wraparound */
    rpc->p_nr = m_ptr->PR_SLOT; /* this was obliterated by copy */
    rpc->p_endpoint = _ENDPOINT(gen, rpc->p_nr); /* new endpoint of slot */

    /* Only one in group should have SIGNALED, child doesn't inherit tracing. */
    rpc->p_rts_flags |= NO_MAP; /* inhibit process from running */
    rpc->p_rts_flags &= ~(SIGNALED | SIG_PENDING | P_STOP);
    sigemptyset(&rpc->p_pending);

```

```
rpc->p_reg.retreg = 0;      /* child sees pid = 0 to know it is child */
rpc->p_user_time = 0;      /* set all the accounting times to 0 */
rpc->p_sys_time = 0;

/* Parent and child have to share the quantum that the forked process had,
 * so that queued processes do not have to wait longer because of the fork.
 * If the time left is odd, the child gets an extra tick.
 */
rpc->p_ticks_left = (rpc->p_ticks_left + 1) / 2;
rpp->p_ticks_left = rpp->p_ticks_left / 2;

/* If the parent is a privileged process, take away the privileges from the
 * child process and inhibit it from running by setting the NO_PRIV flag.
 * The caller should explicitly set the new privileges before executing.
 */
if (priv(rpp)->s_flags & SYS_PROC) {
    rpc->p_priv = priv_addr(USER_PRIV_ID);
    rpc->p_rts_flags |= NO_PRIV;
}

/* Calculate endpoint identifier, so caller knows what it is. */
m_ptr->PR_ENDPT = rpc->p_endpoint;

/* CODIGO 3.2 */

orden[rpc->p_nr + NR_TASKS] = antiguedad ++;

/* CODIGO 3.2 */

return(OK);
}

#endif /* USE_FORK */
```