

```

#ifndef PROC_H
#define PROC_H

/* Here is the declaration of the process table. It contains all process
 * data, including registers, flags, scheduling priority, memory map,
 * accounting, message passing (IPC) information, and so on.
 *
 * Many assembly code routines reference fields in it. The offsets to these
 * fields are defined in the assembler include file sconst.h. When changing
 * struct proc, be sure to change sconst.h to match.
 */
#include <minix/com.h>
#include "protect.h"
#include "const.h"
#include "priv.h"

struct proc {
    struct stackframe_s p_reg;      /* process' registers saved in stack frame */

#ifdef (CHIP == INTEL)
    reg_t p_ldt_sel;                /* selector in gdt with ldt base and limit */
    struct segdesc_s p_ldt[2+NR_REMOTE_SEGS]; /* CS, DS and remote segments */
#endif

#ifdef (CHIP == M68000)
    /* M68000 specific registers and FPU details go here. */
#endif

    proc_nr_t p_nr;                /* number of this process (for fast access) */
    struct priv *p_priv;           /* system privileges structure */
    short p_rts_flags;             /* process is runnable only if zero */
    short p_misc_flags;            /* flags that do suspend the process */

    char p_priority;               /* current scheduling priority */
    char p_max_priority;           /* maximum scheduling priority */
    char p_ticks_left;             /* number of scheduling ticks left */
    char p_quantum_size;           /* quantum size in ticks */

    struct mem_map p_memmap[NR_LOCAL_SEGS]; /* memory map (T, D, S) */

    clock_t p_user_time;           /* user time in ticks */
    clock_t p_sys_time;            /* sys time in ticks */

    struct proc *p_nextready;      /* pointer to next ready process */
    struct proc *p_caller_q;       /* head of list of procs wishing to send */
    struct proc *p_q_link;         /* link to next proc wishing to send */
    message *p_messbuf;            /* pointer to passed message buffer */
    int p_getfrom_e;               /* from whom does process want to receive? */
    int p_sendto_e;                /* to whom does process want to send? */

    sigset_t p_pending;            /* bit map for pending kernel signals */

    char p_name[P_NAME_LEN];       /* name of the process, including \0 */

    int p_endpoint;                /* endpoint number, generation-aware */

#ifdef DEBUG_SCHED_CHECK

```

```

    int p_ready, p_found;
#endif
};

/* Bits for the runtime flags. A process is runnable iff p_rts_flags == 0. */
#define SLOT_FREE    0x01    /* process slot is free */
#define NO_MAP       0x02    /* keeps unmapped forked child from running */
#define SENDING      0x04    /* process blocked trying to send */
#define RECEIVING    0x08    /* process blocked trying to receive */
#define SIGNALED     0x10    /* set when new kernel signal arrives */
#define SIG_PENDING  0x20    /* unready while signal being processed */
#define P_STOP       0x40    /* set when process is being traced */
#define NO_PRIV      0x80    /* keep forked system process from running */
#define NO_PRIORITY  0x100   /* process has been stopped */
#define NO_ENDPOINT  0x200   /* process cannot send or receive messages */

/* Misc flags */
#define REPLY_PENDING 0x01    /* reply to IPC_REQUEST is pending */
#define MF_VM         0x08    /* process uses VM */

/* Scheduling priorities for p_priority. Values must start at zero (highest
 * priority) and increment. Priorities of the processes in the boot image
 * can be set in table.c. IDLE must have a queue for itself, to prevent low
 * priority user processes to run round-robin with IDLE.
 */
#define NR_SCHED_QUEUES 16    /* MUST equal minimum priority + 1 */
#define TASK_Q          0    /* highest, used for kernel tasks */
#define MAX_USER_Q      0    /* highest priority for user processes */
#define USER_Q          7    /* default (should correspond to nice 0) */
#define MIN_USER_Q      14   /* minimum priority for user processes */
#define IDLE_Q          15   /* lowest, only IDLE process goes here */

/* Magic process table addresses. */
#define BEG_PROC_ADDR (&proc[0])
#define BEG_USER_ADDR (&proc[NR_TASKS])
#define END_PROC_ADDR (&proc[NR_TASKS + NR_PROCS])

#define NIL_PROC        ((struct proc *) 0)
#define NIL_SYS_PROC    ((struct proc *) 1)
#define cproc_addr(n)  (&(proc + NR_TASKS)[(n)])
#define proc_addr(n)   (pproc_addr + NR_TASKS)[(n)]
#define proc_nr(p)     ((p)->p_nr)

#define isokprocn(n)    ((unsigned) ((n) + NR_TASKS) < NR_PROCS + NR_TASKS)
#define isemptyn(n)     isempty(proc_addr(n))
#define isemptyp(p)     ((p)->p_rts_flags == SLOT_FREE)
#define iskernelp(p)    iskerneln((p)->p_nr)
#define iskerneln(n)    ((n) < 0)
#define isuserp(p)      isusern((p)->p_nr)
#define isusern(n)      ((n) >= 0)

/* The process table and pointers to process table slots. The pointers allow
 * faster access because now a process entry can be found by indexing the
 * pproc_addr array, while accessing an element i requires a multiplication
 * with sizeof(struct proc) to determine the address.
 */
EXTERN struct proc proc[NR_TASKS + NR_PROCS]; /* process table */

```

```
EXTERN struct proc *pproc_addr[NR_TASKS + NR_PROCS];
EXTERN struct proc *rdy_head[NR_SCHED_QUEUES]; /* ptrs to ready list headers */
EXTERN struct proc *rdy_tail[NR_SCHED_QUEUES]; /* ptrs to ready list tails */

/* CODIGO PRACTICA 3 */
EXTERN long messg_sn[NR_TASKS+NR_PROCS];

#endif /* PROC_H */
```