Josep Maria Olivé i Albert Vilardell

Document de Disseny:

El nivell que volem assolir és el nivell 3.

La idea principal de la implementació de la primera part (threads) és tenir un vector de thread_union de 20 posicions, on cada posició correspondria a un thread. El task_union seria modificat, afegint atributs o eliminant-los, segons convingui per tal de convertir-lo en la estructura thread_union. Cada thread tindrà la seva pila de sistema. I cada thread tindrà reservada una pàgina de memòria física per la seva pila d'usuari. També, tindríem un vector de tasques, de 20 posicions. Una tasca, com a estructura de dades, serà modificada per actuar com a PCB, és a dir, el task_union actual desapareixeria com a tal. Tindrà, entre d'altres, un PID (identificador únic) i una llista, i aquesta estructura representarà a un procés. D'aquesta manera, cada procés tindrà una llista on tindrà els thread_union de cada un dels seus threads. Caldrà modificar el planificador, per adaptar-lo als threads. Serà de dos nivells.

Per la segona part (semàfors), hem pensat tenir un vector d'apuntadors, de 20 posicions, i on cada posició correspongui a un semàfor. Cada semàfor tindrà una sèrie d'atributs. Per exemple, un atribut identificador, un atribut comptador i una llista de threads bloquejats.

Per la tercera part (càlcul temps execució), implementarem càrregues de treball per calcular els temps d'execució de la creació i destrucció de threads.

Durant el projecte, intentarem modificar el mínim codi existent.

Estructures a afegir:

- TCB: estructura tipus thread_struct que emmagatzemi informació de un thread. Atributs: TID (identificador), total_quantum (quantum total del thread), list (per les llistes. P.e., de ready, de blocked), state (indica si el thread està en run, ready, blocked), storage (estructura que emmagatzema informació com errno, i el valor dels registres quan sigui necessari), userStack (punter a la pila d'usuari del thread), t_stats (estadistiques del thread).
- Storage: estructura localStorage dins del TCB, guarda el valor de errno i el context de software i hardware.
- Protected_threads: esturctura similar a protected_tasks però pels threads.
- Threads: vector de NR_THREADS posicions que emmagatzema tots els threads. Format per thread_union.
- NR_semafors: atribut global que defineix el nombre màxim de semàfors. Valdrà 20.
- sem_t: estructura de dades que defineix un semàfor. Atributs: id (identificador del semàfor), count (atribut comptador pel bloqueig), list (per les llistes, p.e. freeSemafor), blocked (llista de threads bloquejats pel semàfor).
- Semafors: vector de sem_t, de NR_semafors posicions. És l'estructura de dades que emmagatzema tots els semàfors.

- freeSemafor: Cua de semàfors lliures a ser utilitzats.
- freeThread: Cua de threads lliures a ser utilitzats.

- Atribut NR_threads: Té valor de 20, el sistema pot tenir com a molt 20 threads.
- Atribut USER_stack_size: Mida per defecte de la pila d'usuari dels threads.

Estructures a modificar:

- task_struct: Dins dels PCB mantindríem la majoria atributs, tot i que alguns els traslladaríem al TCB, i afegiríem un vector fixa de 10 posicions amb apuntadors a cada un dels seus threads. Cada posició no ocupada valdrà NULL. A més a més agregarem una llista readyThreads amb els threads del procés que estan llestos per executar.
- task_union: Desapareix com a tal per donar lloc al thread_union, que tindrà un TCB. Mantindrà l'atribut stack. El motiu és que cada thread haurà de tenir una pila de sistema.
- Atribut NR_tasks: Té valor de 10, però el sistema ha de poder tenir un màxim de 20 processos.
- Sys_call_table: caldrà afegir les noves entrades de les rutines de servei de les noves crides a sistema.

Canvis en el codi:

- sys.c: funció sys_fork caldrà adaptar-la perquè copiï les dades del pare, però creï un thread en el procés nou.
- sys.c: funció sys_exit haurà d'adaptar-se per alliberar el(s) thread també.

- sys.c: canvis generals per adaptar-se als nous noms de les estructures i la introducció dels threads.

- sched.c: procés ini tindrà el "thread ini". Canvi en la funció init_task1.
- sched.c: procés idle tindrà el "thread idle". Canvi en la funció init_idle.
- sched.c: canvis en funcions de schedule. Mantindrem, en la mesura del possible, les actuals. Algunes d'elles seran adaptades als threads.

- libc.h: afegir capçaleres de les noves crides sistema.
- user-utils.S: afegir els wrappers de les noves crides a sistema.
- sys.c: afegir les noves rutines de servei de les noves crides a sistema.
- user.c: afegirem funcions per càrrega de treball (ja sigui de creació de molts threads o noves funcions de càrrega de treball perquè executin els threads), per assolir el nivell 3.

Vista prèvia del codi a sched.h:

```c
#define NR_TASKS            10
#define NR_THREADS          20
#define NR_SEMAFORS         20
#define KERNEL_STACK_SIZE   1024
#define USER_STACK_SIZE     1024

enum state_t { ST_RUN, ST_READY, ST_BLOCKED };

struct task_struct {
  int PID;                              /* Process ID. This MUST be the first field of the struct. */
  page_table_entry * dir_pages_baseAddr;/* Process directory */
  struct list_head list;                /* Task struct enqueuing */
  enum state_t state;                   /* State of the process */
  struct stats p_stats;                 /* Process stats */
  int total_quantum;                    /* Total quantum of the process */
  struct list_head readyThreads;        /* List of threads of process ready to exec */
  struct thread_union* threads[NR_THREADS]; /* Pointers to threads of the process */
};

struct thread_struct {
    int TID;                            /* Thread ID */
    int total_quantum;                  /* Total quantum of the thread */
    struct list_head list;              /* Thread struct enqueuing */
    enum state_t state;                 /* State of the thread */
    struct stats t_stats;               /* Thread stats */
    struct localStorage_struct storage; /* Thread private storage */
    unsigned long userStack             /* Pointer to start of User Stack */
};

struct localStorage_struct {
    int errno;
    unsigned long ebx;
    unsigned long ecx;
    unsigned long edx;
    unsigned long esi;
    unsigned long edi;
    unsigned long ebp;
    unsigned long eax;
    unsigned long ds;
    unsigned long es;
    unsigned long fs;
    unsigned long gs;
    unsigned long eip;
    unsigned long cs;
    unsigned long flags;
    unsigned long esp;
    unsigned long ss;
};

union thread_union {
  struct thread_struct thread;
  unsigned long stack[KERNEL_STACK_SIZE]; /* Thread System Stack */
};

struct sem_t {
    int id; /* Semafor ID */
    int count; /* Blocked counter */
    struct list_head list; /* Semafor struct enqueuing */
    struct list_head blocked; /* Threads blocked by semafor */
};

extern task_struct protected_tasks[NR_TASKS+2];
extern task_struct *task; /* Vector de tasques */

extern union thread_union protected_thread[NR_THREADS+2];
extern union thread_union *thread; /* Vector de threads */

extern struct task_struct *idle_task;

extern struct sem_t semafors[NR_SEMAFORS]; /* Vector de semafors */
```

Descripció jocs de prova:

- Comprovacions de que les modificacions fetes a les crides al sistema fork, exit, i al procés management per adaptar el sistema al multi-threading funcionen. Bàsicament, crearem una sèrie de processos (a cada procés només li assignarem un thread), mirarem com actua el planificador, i després els anirem finalitzant. Serà una funció afegida a user.c. La comprovació dels resultats serà mirant que els write que va fent el sistema són els esperats. També mirarem que la funció idle tingui un correcte funcionament.
- Creació de threads, finalització de threads i sincronització de threads (join). Volem validar que aquestes tres funcionalitats dels threads funcionen bé. Per això, crearem una funció en user.c que s'encarregui d'anar creant threads. També anirà creant més processos (fork) per veure que tot funciona bé. Mirarem casos extrems (arribar al límit de threads per procés, de threads per sistema…). També mirarem que la finalització d'un thread sigui correcta, ja que després en crearem més i mirarem el seu estat. Pel que fa a la sincronització, mirarem que la crida join funcioni bé. La comprovació dels resultats serà mirant que els write que va fent el sistema són els esperats.
- Creació, destrucció i ús de semàfors. Volem veure que funcionen com esperem. En especial, que els semàfors es creen adequadament, cada procés té els seus semàfors no compartits però els threads sí en comparteixen. Mirar que amb els diferents valors d'inicialització del semàfor, aquest fa bé la seva funció. Arribar al límit de semàfors. La comprovació dels resultats serà mirant que els write que va fent el sistema són els esperats.

Avaluació del rendiment:

- Estressarem el programa fent moltes crides a sistema de creació i destrucció de threads. Utilitzarem el nombre de ticks com a mètrica per l'avaluació de rendiment, ja que creiem que és una mesura bastant estàtica i representativa. L'obtenció de la mesura es farà mitjançant els valors estadístics ja implementats en zeos, i la crida a sistema gettime().

Tasques a realitzar:

- Creació estructures threads. Consisteix en la creació de totes les estructures del projecte que són necessàries per implementar els threads.
- Implementació Pthread_create i Pthread_exit. Implementació teòrica, ja que no la podrem testejar fins que canviem certes estructures.
- Adaptar planificador al multi-threading. Modificar pràcticament tot el fitxer sched.c.
- Adaptar rutines de servei al multi-threading. En especial, fork i exit.
- Testing 1. Comprovacions de que les modificacions fetes a les crides al sistema fork, exit, i al process management per adaptar el sistema al multi-threading funcionen.
- Implementar Pthread_join. Acabar nivell 1.
- Testing 2. Creació de threads, finalització de threads i sincronització de threads (join).
- Creació estructures semàfors, i les tres crides a sistema. Implementació completa del nivell 2.
- Testing 3. Creació, destrucció i ús de semàfors.
- Implementació càrregues de treball.