Clase Auxiliar V

Prof: L. Mateu
Aux: M. Leyton

26 de septiembre de 2006

1. POSIX Threads

Los POSIX Threads permiten trabajar con threads en C. Varios sistemas soportan gran parte del estandar POSIX para threads. Por ejemplo: Linux, Solaris y FreeBSD, entre otros.

A continuación se presenta una tabla de equivalencia para algunos de los métodos de nSystem.

nSystem	*POSIX Threads*
#include "nSystem.h"	#include <pthread.h>
nTask	sem_post
nEmitTask(...)	pthread_create(...)
nExitTask(...)	pthread_exit(...)
nWaitTask(...)	pthread_join(...)
nCurrentTask()	pthread_self()
nSem	sem_t
nMakeSem(...)	sem_init(...)
nWaitSem(...)	sem_wait(...)
nSignalSem(...)	sem_post(...)
nDestroySem	sem_destroy(...)

Se desea mostrar que POSIX Threads=>nSystem. Implemente los siguientes métodos de nSystem utilizando POSIX Threads.

```
typedef void* nTask;
nTask nEmitTask( void*(*proc)(void *), void *arg );
void nExitTask(int rc);
int nWaitTask(nTask task);
nTask nCurrentTask();

typedef void* nSem;
nSem nMakeSem(int count);
void nWaitSem(nSem sem);
int nWaitSemTime(nSem sem, int max_delay);
void nSignalSem(nSem sem);
void nDestroySem(nSem sem);

typedef void* nMonitor;
nMonitor nMakeMonitor();
void nDestroyMonitor(nMonitor mon);
void nEnterMonitor(nMonitor mon);
void nExitMonitor(nMonitor mon);

typedef void* nCondition;
nCondition nMakeCondition(nMonitor mon);
void nDestroyCondition(nCondition cond);
void nWaitCondition(nCondition cond);
void nSignalCondition(nCondition cond);

int nSend(nTask task, void *msg);
void *nReceive(nTask *ptask, int max_delay);
void nReply(nTask task, int rc);
```

2. API nSystem (parcial), basdo en POSIX

```
#include <malloc.h>
#include <pthread.h>
#include <semaphore.h>
#include "ports.h"
#include <time.h>

/*************************************************************
 * Estructura de un Task
 *************************************************************/
typedef struct nTask{
    pthread_t thread;       /* Thread posix               */
    Port *port;             /* Port para la mensajeria    */
    void*(*proc)(void *);   /* Main del thread            */
    void *arg;                /* Argumentos del Main        */
} *nTask;

pthread_key_t current_task; /* key aputando al task actual */

int initNSystem(){
    pthread_key_create(&current_task,NULL);
}

/*************************************************************
 * Creacion y manejo de tareas
 *************************************************************/

nTask nEmitTaskDummy(void *arg){

    nTask this_task=(nTask)arg;

    //Registramos el descriptor en el curren_task
    pthread_setspecific(current_task, (void*) this_task);

    //Llamamos al main del thread con su argumento
    this_task->proc(this_task->arg);
}

/* Esta version de nEmitTask solamente acepta un argumento
 * tipo (void*) por simplicidad. */
nTask nEmitTask( void*(*proc)(void *), void *arg ){

    nTask new_task=(nTask)malloc(sizeof(*new_task));

    new_task->port=makePort();
    new_task->arg=arg;
    new_task->proc=proc;

    pthread_create(&(new_task->thread), NULL, &nEmitTaskDummy, (void*)new_task);
    return new_task;
}

nTask nCurrentTask(){
    /* obtenemos el descriptor de esta tarea */
    return (nTask)pthread_getspecific(current_task);
}

void nExitTask(int rc){
```

```
    nTask this_task = nCurrentTask();
    free(this_task->port);
    free(this_task);
    pthread_exit((int *)rc);
}

int nWaitTask(nTask task){
    void *retval;
    pthread_join(task->thread,&retval);
    int *int_retval=retval;
    return (int)int_retval;
}

/*************************************************************
 * Semaforos
 *************************************************************/
typedef struct nSem{
    sem_t psem;
} *nSem;

nSem nMakeSem(int count){

    nSem new_sem= (nSem)malloc(sizeof(*new_sem));
    sem_init(&(new_sem->psem), 0, count);
    return new_sem;
}

void nWaitSem(nSem sem){
    sem_wait(&(sem->psem));
}

/* Este metodo no existe en nSystem, pero es util para
 * implementar mensajes con timeout en base semaforos.
 */
int nWaitSemTime(nSem sem, int max_delay){
    struct timespec t;
    t.tv_sec=max_delay;
    t.tv_nsec=0;

    return sem_timedwait(&(sem->psem),t);
}

void nSignalSem(nSem sem){
    sem_post(&(sem->psem));
}

void nDestroySem(nSem sem){
    sem_destroy(&(sem->psem));
    free(sem);
}

/*************************************************************
 * Mensajeria Basada en Ports
 *************************************************************/

int nSend(nTask task, void *msg){
    portSend(task->port, msg);
}
void *nReceive(nTask *ptask, int max_delay){
```

```
    nTask this_task=nCurrentTask();
    return portReceive(this_task->port, max_delay);
}

/* Este metodo queda propuesto, ya que se deben implementar
 * unos Ports mas poderosos.
 */
void nReply(nTask task, int rc){

}

/*************************************************************
 * Monitores
 *************************************************************/
typedef struct nMonitor{
    pthread_mutex_t mutex;
} *nMonitor;

nMonitor nMakeMonitor(){
    nMonitor new_monitor = (nMonitor)malloc(sizeof(*new_monitor));
    pthread_mutex_init(&(new_monitor->mutex),NULL);
}

void nDestroyMonitor(nMonitor mon){
    pthread_mutex_destroy(&(mon->mutex));
    free(mon);
}

void nEnterMonitor(nMonitor mon){
    pthread_mutex_lock(&(mon->mutex));
}

void nExitMonitor(nMonitor mon){
    pthread_mutex_unlock(&(mon->mutex));
}

typedef struct nCondition{
    nMonitor mon;
    pthread_cond_t cv;
} *nCondition;

nCondition nMakeCondition(nMonitor mon){
    nCondition new_condition = (nCondition)malloc(sizeof(*new_condition));
    pthread_cond_init(&(new_condition->cv), NULL);
    new_condition->mon=mon;
}

void nDestroyCondition(nCondition cond){
    pthread_cond_destroy(&(cond->cv));
    nDestroyMonitor(cond->mon);
    free(cond);
}

void nWaitCondition(nCondition cond){
    pthread_cond_wait(&(cond->cv), &(cond->mon->mutex));
}

void nSignalCondition(nCondition cond){
```

```
    pthread_cond_signal(&(cond->cv));
}
```

3. Ports vistos en clase auxiliar IV

```
#include <malloc.h>
#include "nsyspos.h"

typedef struct Port{
    nSem hayMsg; //Bloque lector si no hay mensajes
    nSem leidoMsg; //Bloquea emisor hasta un receive
    nSem vacioMsg; //Bloquea emisor si ya hay un mensaje
    void *msg;
} Port;

Port* makePort(){
    Port *new_port=
        (Port*) malloc(sizeof(*new_port));

    new_port->hayMsg=nMakeSem(0);
    new_port->leidoMsg=nMakeSem(0);
    new_port->vacioMsg=nMakeSem(1);
    new_port->msg=NULL;
}

void portSend(Port *p, void *msg){

    nWaitSem(p->vacioMsg);

    p->msg=msg;

    nSignalSem(p->hayMsg);
    nWaitSem(p->leidoMsg);
}

void *portReceive(Port *p, int max_delay){

    void *retval;

    if(!nWaitSemTime(p->hayMsg,max_delay))
        return NULL;

    retval=p->msg;

    nSignalSem(p->leidoMsg);
    nSignalSem(p->vacioMsg);

    return retval;
}

void destroyPort(Port *p){
    free(p);
}
```