HOJA DE REFERENCIA – API DE NSYSTEM
NIVEL IMPLEMENTADOR

Listado de referencia de prototipos de funciones para nSystem 2005 (nSysimp.h)

- **Descriptor de tarea**

```
typedef struct Task /* Descriptor de una tarea */
{
  int status;       /* Estado de la tarea (READY, ZOMBIE ...) */
  char *taskname;   /* Util para hacer debugging */

  SP sp;            /* El stack pointer cuando esta suspendida */
  SP stack;         /* El stack */

  struct Task *next_task;   /* Se usa cuando esta en una cola */
  void *queue;              /* Debugging */

  /* Para el nExitTask y nWaitTask */
  int  rc;                  /* codigo de retorno de la tarea  */
  struct Task *wait_task;   /* La tarea que espera un nExitTask */

  /* Para nSend, nReceive y nReply */
  struct Queue *send_queue; /* cola de emisores en espera de esta tarea */
  union { void *msg; int rc; } send; /* sirve para intercambio de info */
  int wake_time;            /* Tiempo maximo de espera de un nReceive */
}
  *nTask;
```

- **Estados de una tarea**

```
#define READY      0  /* Elegible por el scheduler (incluye RUNNING) */
#define ZOMBIE     1  /* llamo nExitTask y espera nWaitTask (nExitTask) */
#define WAIT_TASK  2  /* espera el final de otra tarea (nWaitTask) */
#define WAIT_REPLY 3  /* hizo nSend y espera nReply (nSend) */
#define WAIT_SEND  4  /* hizo nReceive y espera nSend (nReceive) */
#define WAIT_SEND_TIMEOUT 5
                      /* hizo nReceive y espera nSend o timeout (nReceive) */
#define WAIT_READ  6  /* esta bloqueada en un read (nRead) */
#define WAIT_WRITE 7  /* esta bloqueada en un write (nWrite) */
#define WAIT_SEM   8  /* esta bloqueada en un semaforo (nWaitSem) */
#define WAIT_MON   9  /* esta bloqueada en un monitor (nEnterMonitor)*/
#define WAIT_COND 10  /* esta bloqueada en una condicion (nWaitCondition) */
#define WAIT_COND_TIMEOUT 11 /*esta bloqueado en un monitor con timeout */
#define WAIT_SLEEP 12 /* esta dormida en nSleep */
#define STATUS_END WAIT_SLEEP
```

- **Variables globales del planificador**

```
extern struct Queue *ready_queue;   /* Cola de tareas en espera de la CPU */

extern nTask current_task;  /* La tarea que tiene la CPU */

extern int current_slice;   /* Taman~o de una tajada de CPU */
```

- **Cambio de contexto**

```
/* Suspende la tarea actual y retoma la primera de la “ready_queue” */
void ResumeNextReadyTask(); /*Utilice Resume()*/
```

- **Manejo del Timer**

```
void TimeInit();

void ProgramTask(int timeout);

void CancelTask(nTask task);


/*
 * Programacion de Timers:
 *
 * Programa el timer de tipo "type" para que cause una interrupcion
 * dentro de "msecs" microsegundos. "handler" es el procedimiento que
 * se llama durante la interrupcion. Se llama con las interrupciones
 * deshabilitadas. Su retorno habilita las interrupciones.
 */

void SetAlarm(int type, int msecs, void (*timerhandler)());

#define REALTIMER 1    /* Cuenta el tiempo real */
#define VIRTUALTIMER 2 /* Cuenta el tiempo de uso de CPU */

/* El timer real se usa para los timeouts en los receive y el
 * time virtual para las tajadas de tiempo en el scheduler.
 */
```

- **Secciones críticas**

```
/* Para evitar errores, las secciones criticas se pueden anidar.
 * Esto significa que si se coloca una secciones critica dentro
 * de otra, las interrupciones se habilitaran solo al salir
 * de la secciones critica mas externa.
 */

void START_CRITICAL(); /* deshabilitacion de interrupciones */
void END_CRITICAL();   /* habilitacion de interrupciones */

  /* Cuidado: Puede haber un cambio de contexto en la seccion critica, por
   * lo que las interrupciones se habilitan en la tarea que se retoma.
   */
```

- **Manejo de colas (Con y sin prioridad)**

```
#include <nQueue.h>

/* Las siguientes funciones deben ser siempre invocadas con las
 * interrupciones deshabilitadas
 */

Queue MakeQueue();                      /* El constructor */

void PutTask(Queue queue, nTask task);  /* Agrega una tarea al final */

void PushTask(Queue queue, nTask task); /* Agrega una tarea al principio */

nTask GetTask(Queue queue);             /* Extrae la primera tarea */

int EmptyQueue(Queue queue);            /* Verdadero si la cola esta vacia */

int QueueLength(Queue queue);           /* Entrega el largo de la cola */

int QueryTask(Queue queue, nTask task); /* Verdadero si task esta en la cola */

void DeleteTaskQueue(Queue queue, nTask task ); /* Borra una tarea de la cola */

void DestroyQueue(Queue queue);         /* El destructor */



Squeue MakeSqueue();

void PutTaskSqueue(Squeue squeue, nTask task, int wake_time);

nTask GetTaskSqueue(Squeue squeue);

int GetNextTimeSqueue(Squeue squeue);

int EmptySqueue(Squeue squeue);

void DeleteTaskSqueue(Squeue squeue, nTask task);

void DestroySqueue(Squeue squeue);
```