CC41B : Sistemas Operativos

Prof.: Luis Mateu.
Autor: Jaime Catal'an.

El Nano System : nSystem

`nSystem` es un sistema de procesos livianos para Unix con fines pedag'ogicos. El sistema consiste de unos cientos de l'ineas de c'odigo en C (la mitad son comentarios) que implementan :

- Creaci'on y destrucci'on de procesos livianos (tareas). Estos procesos comparten un mismo espacio de direcciones (est'an dentro de un solo proceso UNIX).
- Paso de Mensajes para la sincronizaci'on de tareas.
- Entrada/Salida no bloqueante para el proceso UNIX.
- Un scheduler muy simple. Permite implementar administraci'on preemptive y non-preemtive.

Archivos fuentes

En `cipres:~cc41b/nSystem/` Ud. encontrar'a los fuentes de nSystem. Los archivos contienen :

- `nSystem.h` : Encabezados que debe incluir toda aplicaci'on que usa nSystem. Estudiar cuidadosamente.
- `nSysimp.h` : Encabezados con las estructuras de datos que usa nSystem internamente.
- `nProcess.c` : Implementaci'on de nSystem.
- `nMsg.c` : Implementaci'on mensajes entre tareas.
- `nSem.c` : Implementaci'on de sem'aforos.
- `nMonitor.c` : Implementaci'on de monitores.
- `nIO.c` : E/S bloqueante para la tarea, pero no para el resto.
- `nMain.c` : El main del programa C.
- `nOther.c` : Procedimientos varios. (ej. nPrintf, nMalloc, etc.)
- `nQueue.c` : Manejo de colas.
- `nDep.c` : Rutinas varias que hacen poco, pero son fuertemente dependientes de Unix.
- `nStack-sparc.s` : Escrito en assembler, contiene procedimientos para cambiar la pila en la arquitectura sparc.
- `nStack-i386.s` : Escrito en assembler, contiene procedimientos para cambiar la pila en la arquitectura intel 386.

Estado actual de nSystem

A continuaci'on se detallan los procedimientos p'ublicos de nSystem.

- `int nMain(/* int argc, char *argv[] */)` : Este procedimiento es provisto por el programador y es invocado al comenzar la ejecuci'on. El retorno de este procemiento termina prematuramente todas las tareas pendientes. (No siempre sera necesario colocar argc y argv, por ello los argumentos estan comentados).

Creaci'on y Muerte de tareas :

- `nTask nEmitTask( int (*proc)(), parametro1 ... parametro14 )` : Emite una tarea que ejecuta el procedimiento `proc`. Acepta un m'aximo de 14 par'ametros (enteros o punteros).
- `void nExitTask(int rc)` : Termina la ejecuci'on de una tarea, `rc` es el c'odigo de retorno de la tarea.
- `int nWaitTask(nTask task)` : Espera a que una tarea termine, entrega el c'odigo de retorno dado a `nExitTask`.
- `void nExitSystem(int rc)` : Termina la ejecuci'on de todas las tareas (shutdown del proceso Unix), `rc` es el c'odigo de retorno del proceso UNIX.

Par'ametros para las tareas :

- `void nSetStackSize(int new_size)` : Define el tama'no del stack de las tareas que se emitan a continuaci'on.
- `void nSetTimeSlice(int slice)` : Tama'no de la tajada de tiempo para la administraci'on *Round–Robin* (preemptive) (`slice` esta en miliseg). Degenera en *FCFS* (non–preemptive) si `slice` es cero, lo que es muy 'util para depurar programas. El valor por omisi'on de la tajada de tiempo es cero.
- `void nSetTaskName( /* char *format, <args>... */ )` : Asigna el nombre de la tarea que la invoca. El formato y los par'ametros que recibe son an'alogos a los de `printf`.

Paso de Mensajes :

- `int nSend(nTask task, void *msg)` : Env'ia el mensaje `msg` a la tarea `task`. Un mensaje consiste en un puntero a un 'area de datos de cualquier tama'no. El emisor se queda bloqueado hasta que se le haga `nReply`. `nSend` retorna un entero especificado en `nReply`.
- `void *nReceive(nTask *ptask, int max_delay)` : Recibe un mensaje proveniente de cualquier tarea. La identificaci'on del emisor queda en `*ptask`. El mensaje retornado por `nReceive` es un puntero a un 'area que ha sido posiblemente creada en la pila del emisor. Dado que el emisor contin'ua bloqueado hasta que se haga `nReply`, el receptor puede accesar libremente esta 'area sin peligro de que sea destruida por el emisor.

  La tarea que la invoca queda bloqueada por `max_delay` miliseg, esperando un mensaje. Si el per'iodo finaliza sin que llegue alguno, se retorna `NULL` y `*ptask` queda `NULL`. Si `max_delay` es 0, la tarea no se bloquea (nReceive retorna de inmediato). Si `max_delay` es -1, la tarea espera indefinidamente la llegada de un mensaje.
- `void nReply(nTask task, int rc)` : Responde un mensaje enviado por `task` usando `nSend`. Desde ese instante, el receptor no puede accesar la informaci'on contenida en el mensaje que hab'ia sido enviado, ya que el emisor podr'ia destruirlo. El valor `rc` es el c'odigo de retorno para el emisor. nReply no se bloquea.

Entrada y Salida :

Estas funciones son equivalentes a open, close, read y write en UNIX. Sus par'ametros son an'alogos a los de las de UNIX. Las "nano" funciones son no bloqueantes para el proceso UNIX, s'olo bloquean la tarea que las invoca.

- `int nOpen(char *path, int flags, int mode)` : Abre un archivo.
- `int nClose(int fd)` : Cierra un archivo.
- `int nRead(int fd, char *buf, int nbyte)` : Lee de un archivo.
- `int nWrite(int fd, char *buf, int nbyte)` : Escribe en un archivo.

Servicios Miscel'aneos :

- `nTask nCurrentTask(void)` : Entrega el identificador de la tarea que la invoca.
- `int nGetTime(void)` : Entrega la "hora" en miliseg.
- `void *nMalloc(int size)` : Es un `malloc` ininterrumpible.
- `void nFree(void *ptr)` : Es un `free` ininterrumpible.
- `void nFatalError(/* char *procname, char *format, ...*/ )` : Escribe salida formateada en la salida est'andar de errores y termina la ejecuci'on (del proceso Unix). El formato y los par'ametros que recibe son an'alogos a los de `printf`.
- `void nPrintf(/* char *format, ... */)` : Es un `printf` solo bloqueante para la tarea que lo invoca.
- `void nFprintf(/* int fd, char *format, ... */)` : Es "como" un `fprintf`, solo bloqueante para la tarea que lo invoca, pero recibe un fd, no un FILE *.
- `void nSetNonBlocking()` : Coloca la E/S est'andar en modo no bloqueante. De esta forma, al leer la entrada esta'ndar s'olo se bloquear'a la tarea lectora y no el resto de las tareas.