Alumnos		
Apellido	Nombre	DNI
Francese	Diego	32.527.753
Paniagua	Roberto	32.496.671
Perez	Lucas	33.116.288
Petrini	Gisela	33.504.358

Índice

Desarrollo

Objetivo del Trabajo Práctico

El objetivo general del trabajo práctico se centra en la mejora y mantenibilidad del sistema operativo SODIUM, lo cual se realizará creando una biblioteca que manipule hilos a nivel de usuario.

Alcance

- Desarrollar la biblioteca de thread a nivel usuario (nivel 3).
- Hacer que el trabajo práctico 2 funcione. (apunte hilos_tips.doc)

Resumen del documento y explicación de su estructura

El presente documento pretende mostrar las ventajas de trabajar con hilos a nivel de usuario y las posibles implementaciones de una biblioteca que puede manipular estos hilos, mostrando las ventajas y desventajas que estas poseen.

Teoría

Introducción Teórica al tema a tratar

En la teoría explicaremos las diferentes clases de hilo (thread) que existen, como también sus características. También indicaremos cuales son las diferencias entre las llamadas al sistema bloqueantes y no bloqueantes.

Ademas indicaremos las funciones que poseerá la biblioteca Sodium Thread Library

Base Teórica

Procesos

En los sistemas operativos tradicionales, cada proceso tiene su propio espacio de direcciones y un único flujo (hilo) de control.

Frecuentemente hay situaciones en las que es deseable contar con múltiples hilos de control en el mismo espacio de direcciones, ejecutándose quasi-paralelamente, como si fueran procesos separados (excepto que comparten el mismo espacio de direcciones).

Hilos

El hilo es una secuencia de control que opera dentro del mismo espacio de direcciones (como otra secuencia de control independiente) del proceso, compartiendo ficheros abiertos y otros recursos.

Tipos de Hilos

- User level thread (ULT,Hilos de usuario)

  Su implementación está a cargo totalmente de la aplicación, a través de una biblioteca de hilos (es rápida ya que no hay llamada al sistema, con cambio de modo). El sistema operativo administra el proceso como una unidad, y la aplicación migra de un hilo a otro con su propia administración.

  El kernel no tiene conocimiento de la existencia de los hilos, no involucrar al kernel significa un intercambio rápido.

  Solo requiere un Stack y un contador de programa.

- Kernel level thread (KLT, Hilos de kernel)

  El sistema operativo conoce la existencia de los hilos. El kernel mantiene la información de contexto del proceso y de los hilos, la creación, programación y administración de los hilos es gestionada por el kernel en su propio espacio. El planificador no selecciona procesos para ser ejecutados sino hilos. El kernel planifica y gestiona estos hilos según sus propias necesidades y las necesidades de los servicios del mismo.

  Comparado con los procesos, el intercambio de hilos no requiere cambiar la información del acceso a memoria (ya que comparten el mismo espacio de direcciones) y los cambios de hilos son relativamente rápidos.

Para cada proceso el kernel tiene una tabla con una entrada por cada uno de los hilos del proceso, con los registros, estado, prioridades y otra información sobre cada hilo.

Los Process Level Thread (PLT) no seran tratados en el presente trabajo.

Algunas ventajas y desventajas de las distintas alternativas

- Ventaja del nivel-usuario: puede construirse encima del sistema operativo que no soporta *threads*. En el otro caso deben meterse en el *kernel*.
- Ventaja del nivel-usuario: puede usar su propio esquema de planificación. No es posible en el nivel-kernel.
- Ventaja del nivel-usuario: la conmutación de contexto de un *thread* en el nivel-usuario es más rápida que en el nivel-kernel. Es hecho por el *run time*, en el nivel-kernel requiere un *trap*.
- Ventaja del nivel-usuario: la escalabilidad es mayor. Las tablas en nivel-kernel son mantenidas dentro del *kernel*.
- Desventaja del nivel-usuario: trabajan en multiprogramación pura.
- Desventaja del nivel-usuario:los *system-calls* generan problemas de bloqueo.

Sincronización de hilos

Todos los hilos comparten el mismo espacio de direcciones y otros recursos como pueden ser archivos abiertos. Cualquier modificación de un recurso desde un hilo afecta al entorno del resto de los hilos del mismo proceso. Por lo tanto, es necesario sincronizar la actividad de los distintos hilos para que no interfieran unos con otros o corrompan estructuras de datos. Una ventaja de la programación multihilo es que los programas operan con mayor velocidad en sistemas de computadores con múltiples CPUs ya que los hilos del programa se prestan verdaderamente para la ejecución concurrente. En tal caso el programador necesita ser cuidadoso para evitar condiciones de carrera, y otros comportamientos no intuitivos. Los hilos generalmente requieren reunirse para procesar los datos en el orden correcto. Es posible que los hilos requieran de operaciones atómicas para impedir que los datos comunes sean cambiados o leídos mientras estén siendo modificados, para lo que usualmente se utilizan los semáforos. El descuido de esto puede generar interbloqueo.

Definiciones

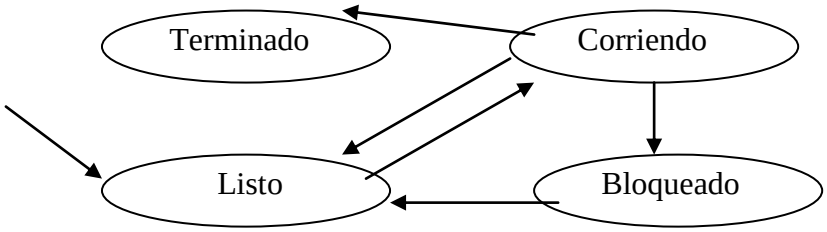
Multiprocesador: un equipo con más de un procesador.

Multiprocesamiento: la capacidad de un SO de tener varios programas (procesos en realidad) en ejecución.

Multitarea: es una forma especial de multiprocesamiento, donde la conmutación entre procesos se hace de manera tan rápido que da la sensación de que varios programas están corriendo en simultáneo.

Multithreaded: son programas (procesos) en los cuales hay varios hilos corriendo en paralelo (de manera real o ficticia como en el multiprocesamiento).

Estados de un proceso



Corriendo: est_a usando la CPU.
Bloqueado: no puede correr hasta que algo externo suceda (típicamente E/S lista).
Listo: el proceso no est_a bloqueado, pero no tiene CPU disponible como para correr.

Modos de realizar llamadas al sistema:

Bloqueante:

Hago el system call, para cuando recibo el control la E/S ya terminó. Mientras, me bloqueo.

No bloqueante:

Hago el system call, que retorna en seguida.

Puedo seguir haciendo otras cosas. Debo enterarme de alguna manera si mi

E/S terminó.

Cuando una aplicación emite una llamada a sistema bloqueante la aplicación se suspende ypasa del estado de ejecución (running) al de waiting, pasando a la cola de espera. Cuando se completa la E/S pasa a la cola de listos.

La razon para usar blocking es que no podemos determinar exactamente cuanto va a tardar una E/S.

Pero algunos procesos a nivel de usuario necesitan usar entrada/salida no bloqueante

(nonblocking I/O . E/S asincrónica). Tal es el caso del input por teclado y por mouse. O una aplicacion de video que lee marcos desde un disco, mientras va descomprimiendo y mostrando el output en la pantalla.

Para poder seguir procesando mientras se realiza I/O, se utilizan aplicaciones multihilo (multithread).

Otra alternativa es usar system calls asincrónicos. Un system call de este tipo retorna enseguida sin esperar que se complete la E/S. Mas tarde, a traves de un seteo de una variable en su espacio de direcciones, o por una interrupcion por software a la aplicación, se la avisa que se completó la E/S.

Veamos la diferencia entre nonblocking y llamadas a sistema asincrónicos: un read nonblocking retorna sin esperar que el dato este disponible mientras que el system call asincronico espera que se realice la transferencia pero no que se

Funciones sugeridas que debe tener la biblioteca a crear:

Creación de un hilo

- **th_t STL_spawn (STL_attr_t attr, void *(entry) (void *), void * arg)**
  Creación de un nuevo hilo con los atributos dados en *attr* (o una variable TH_ATTR_DEFAULT para atributos por defecto, lo cual implica que la prioridad, enlace y estado de cancelación serán heredados del hilo actual), inicialmente en la rutina provista en *entry,* el contador del despachador no es heredado del hilo actual si attr no es especificado (es inicializado a 0). Esta rutina de entrada es llamada como STL_*exit (entry (arg)),* dentro de la unidad del nuevo hilo el valor de retorno de entry es suministrado por un *STL_exit* implícito, así el nuevo hilo también puede salir simplemente retornando. Sin embargo el hilo puede salir en cualquier momento llamando a STL_*exit.*
  No hay límites internos dentro de la STL de cuantos hilos se pueden crear, excepto el límite impuesto por la memoria virtual disponible. STL internamente mantiene el seguimiento de los hilos en estructuras de datos dinámicas. La función retorna NULL en caso de algún error.

Esperar la terminación de otro hilo

- **th_t STL_join(STL_t tid, void **value)**
  Espera por la terminación del hilo especificado en *tid*. Primero suspende el hilo actual hasta que eh hilo *tid* termine. Luego el hilo suspendido es resumido y almacena el valor de la llamada a la función STL_*exit* (por parte de *tid*) en **value* y retorna al llamador. Un hilo puede ser esperado solo cuando tiene el atributo STL_ATTR_JOINEABLE seteado a TRUE (por defecto). Un hilo solo puede ser esperado una sola vez, luego de la llamada a STL_*join* el hilo tid es completamente removido del sistema.

Cancelar un hilo

- **int STL_cancel(STL_t tid)**
  Cancela un hilo especificado en *tid* de forma amigable. (Una forma más arbitraria es STL_*abort*). Como se realiza la cancelación, depende del estado de cancelación de *tid* (recordar que cada hilo tiene un atributo estado de cancelación) que el propio hilo puede configurar.
  Cuando el estado es STL_CANCEL_DISABLE una llamada a cancelación simplemente es marcada como pendiente. Cuando el estado es STL_CANCEL_ENABLE depende del tipo de cancelación que es realizada. Cuando el estado es STL_CANCEL_DEFERRED otra vez la llamada a cancelación es marcada como pendiente. Pero cuando el estado es STL_CANCEL_ASYNCHRONOUS el hilo es inmediatamente cancelado, antes que STL_*cancel* retorne.
  El efecto de una cancelación es igual a forzar implícitamente a la llamada STL_*exit (STL_CANCELED)* como un punto de cancelación.

- **int STL_abort(STL_t tid)**
  Esta es la forma cruel de cancelar un hilo *tid*. Cuando ya esté muerto y quiere hacer un join solo STL_join (tid, NULL) y esto lo elimina del Sistema. De otra forma se lo fuerza a no joinearse y a permitir cancelación asincrónica y luego se cancela vía *STL_cancel (tid)*

Suspender un hilo

- **int STL_suspend(STL_t tid)**
  Suspende el hilo *tid* hasta que el mismo sea reanudado con la función *STL_resume.* El hilo es movido a la cola de SUSPENDIDO. Suspender al hilo actual no está permitido para esta función. Esta función retorna TRUE si funciono correctamente y FALSE si ocurrió algún problema.
- **unsigned int STL_sleep(unsigned int sec)**
  Variante de la función *waitpid*. Suspende la ejecución del hilo actual hasta la cantidad de segundos *sec* se cumplan. Se garantiza que el hilo no se restablezca antes que pase el tiempo estipulado, debido a la naturaleza de la planificación non-preemptive de STL, puede ser restablecido después obviamente. La diferencia entre *sleep* y STL_*sleep* es que STL_*sleep* suspende solo la ejecución del hilo actual.

Reanudar un hilo

- **int STL_resume(STL_t tid)**
  Esta función reanuda un hilo *tid* que ha sido suspendido previamente con *STL_suspend().* El hilo tiene que estar en la cola de SUSPENDIDO. Es movido a la cola de NUEVO, LISTO o ESPERANDO. (Dependiendo del estado en que se encontraba cuando fue suspendido por *STL_suspend ())*
  Variante de la función STL_sigwait, pero tiene un argumento adicional *ev_extra.* Cuando STL_sigwait suspende la ejecución del h. Esta función retorna TRUE si funciono correctamente y FALSE si ocurrió algún problema.

Despertar un hilo

- **int STL_sigwait_ev(const sigset_t *set, int *sigp, stl_event_t ev_extra)**
  Variante de la función STL_sigwait, pero tiene un argumento adicional *ev_extra.* Cuando STL_sigwait suspende la ejecución del hilo actual usualmente solo utiliza el evento señal seteado para despertar. Con esta función cualquier número de eventos extra pueden ser utilizados para despertar al hilo actual.

Ceder la ejecución

- **Int STL_yield (STL_t tid);**
  Esta función explícitamente cede el control de ejecución al hilo que lo planifico. Usualmente la ejecución es implícitamente transferida al planificador cuando un hilo espera por un evento. Pero cuando un hilo necesita muchas ráfagas de CPU,

puede ser razonable interrumpirlo explícitamente haciendo algunas llamadas a *STL_yield ()* para darle a otros hilos la chance de ejecutar también. Esta es obviamente la parte cooperative de esta STL.
Usualmente uno especifica *tid* como NULL para indicar al planificador que puede ser libremente decidir que hilo despachar próximo. Pero si uno quiere indicarle al planificador que un hilo en particular debe ser favorecido en el próximo paso de envío, uno puede especificar este hilo explícitamente. Esto trae el Viejo concepto de corutinas donde un hilo-rutina cambia a un hilo cooperativo particular. Si *tid* no es NULL y apunta a un hilo listo o Nuevo, se garantiza que este hilo reciba control de ejecución en el próximo paso de envío. Si *tid* esta en un estados diferente (que no sea STL_STATE_NEW o STL_STATE_READY) un error es generado y reportado.
Esta función retorna TRUE si funciono correctamente y solo FALSE si *tid* especifica un hilo invalid o un hilo que aun no esta listo.

- **STL_mutex_init(STL_mutex_t *mutex)**
  Dinámicamente inicializa una variable mutex de tipo *STL_mutex_t*. Alternativamente uno puede utilizar inicialización estática vía STL_*mutex_t mutex =STL_MUTEX_INI*

- **Int STL_mutex_acquire(STL_mutex_t *mutex, int tryonly, STL_event_t ev_extra)**
  Esta función adquiere un mutex definido como *mutex.* Si el mutex ya esta cerrado por otro hilo, la ejecución del hilo actual es suspendida hasta que el mutex sea abierto nuevamente o adicionalmente los eventos extra en *ev_extra* ocurran (cuando *ev_extra* no sea NULL). Cierres recursivos son soportados explícitamente. Un hilo es capaz de adquirir un mutex mas de una vez antes de qué el mutex sea liberado, aunque luego el mutex debe ser liberado la misma cantidad de veces hasta que el mutex pueda ser adquirible por otros hilos. Cuando *try* es TRUE esta función nunca suspende su ejecución, si retorna FALSE errno se setea a EBUSY.

- **Int STL_mutex_release(STL_mutex_t *mutex)**
  Esta función simplemente decrementa el contador de la recursión del cerrojo *mutex* y cuando es cero, libera al mutex definido como *mutex.*

Análisis del Caso

Opciones disponibles

Las opciones disponibles en el presente trabajo son que el dispatcher trabaje con lista enlazada de TCBs o que utilice Colas de Prioridad.

La ventaja que tiene la lista enlazada es que es fácil de implementar. La desventaja es que es indeficiente ya que debe recorrer n-1 hilos hasta encontrar el primero en el estado que se este buscando (Ejemplo, recorrer secuencialmente la lista de TCBs hasta encontrar el primer hilo "Listo" para ejecutarlo).

La cola de prioridad posee la ventaja de no necesitar un método para mantener ordenada la cola, ya que la inserción es en orden según la prioridad del hilo, y al poseer una cola por cada estado, cuando se busca un hilo de un estado determinado solo se toma el que este en el tope de la cola correspondiente a ese estado. La desventaja es que para prevenir starvation cada cierto tiempo hay que cambiar las prioridades de los hilos de la cola, y esto es overhead.

Elección y justificación de la elección

Nuestra elección es trabajar con colas de prioridad, ya que es la opción que posee el menor nivel de ineficiencia. Es menos indeficiente prevenir el starvation en la cola de prioridad que recorrer la lista por cada hilo.

Práctica

Introducción a la práctica a tratar

Se desarrollará una biblioteca que manipule threads a nivel de usuario. Dicha biblioteca trabará en nivel de privilegio 3.

Para el desarrollo de la biblioteca se utilizará se tendran en cuenta los consejos que la cátedra brindó para la creación de la biblioteca basándose en la biblioteca GNU Portable Threads (pth.h).

También se crearan dos binarios los cuales uno tendrá operaciones de entrada salida de forma bloqueante, y el otro tendrá estas operaciones de modo no bloqueante.

Resumen de modificaciones

- **Carpeta:** fs
  - **Archivo:** fat.c
- **Carpeta:** include/fs
  - **Archivo:** fat.h
- **Carpeta:** include/usr
  - **Archive:** libsodium.h
- **Carpeta:** include/kernel
  - **Archivo:** syscall.h
  - **Archivo:** system.h
- **Carpeta:** kernel
  - **Archivo:** syscall.c
  - **Archivo:** system.c
  - **Archivo:** system.c
- **Carpeta:** kernel/drivers
  - **Archivo:** floppy.c
- **Carpeta:** usr
  - **Archivo:** a_bin.c
  - **Archivo:** b_bin.c
  - **Archivo:** c_bin.c
  - **Archivo:** libsodium.c
- **Carpeta:** usr/shell
  - **Archivo:** shell3.c
  - **Archivo:** shell3.c

Documentación de las funciones y programas

Carpeta: fs
Archivo: fat.c

void leer_file2 (dword dwSector, unsigned int uiEsp, unsigned int uiDs)
Parametros que recibe: dword dwSector, unsigned int uiEsp, unsigned int uiDs
Devuelve: no devuelve nada
Que hace: lee sector por sector el archivo y le envia a nivel 3 los caracteres leidos dentro de ese sector

```
void leer_file2 (dword dwSector, unsigned int uiEsp, unsigned int uiDs)
{
	int iN;
	unsigned int caracter;
	if (uiModo == 1)
	{
		if(leerSector(bloque_read, dwSector+31) != 1)
		{
			vFnImprimir("No se pudo leer sector %d\n", dwSector);
			return;
		}
	}
	else
	{
		do
		{
			iN= leerSector_nonblock (bloque_read, dwSector+31);
			if (iN != 1)
			{
				while (!Int);
				while(!TCDMA(2));
				apagarMotor();
			}
		}while (iN != 1);
	}
	for ( iN=0;  iN < CANT_LINEAS*3;  iN++ )
	{
		caracter= (unsigned int) bloque_read[iN];
		asm volatile ("pushl %%ds\n\tmovl %0, %%ds\n\tmovl %2, (%1)\n\tpopl %%ds\n\t" :: "a" (uiDs),"b" (uiEsp),
"c" (caracter));
		uiEsp-= 4;
	}

}
```

void device_read_file2 (char *pcNombre, unsigned int uiEsp, unsigned int uiDs)
Parametros que recibe: char *pcNombre, unsigned int uiEsp, unsigned int uiDs
Devuelve: no devuelve nada
Que hace: busca el archivo q le pasas por parametro, lo encuentra, setea datos para poder leerlo y llama a leer_file2 para q lo lea hasta el final

```
void device_read_file2 (char *pcNombre, unsigned int uiEsp, unsigned int uiDs)
{
	fs_fat_dirent_t *pstuFsFatDirentDatosArchivo;
	uint32_t uCluster;
	uint32_t uTamanioArchivo;
	int iCantSectores, iSectoresLeidos=0;
	//provisorio hasta crear un comando mount para cargar el pstuFsFatHandleHandle
	header_get();
	//cargarFAT();
	carga_Rdir();
	pstuFsFatDirentDatosArchivo= (struct fs_fat_dirent_t*) pvFnKMalloc(sizeof(fs_fat_dirent_t), MEM_BAJA);
	if(fat_file_find2 (pcNombre, pstuFsFatDirentDatosArchivo, pstuFsFatHandleHandle) == 1)
	{
		uCluster = pstuFsFatDirentDatosArchivo->starting_cluster;
		uTamanioArchivo = pstuFsFatDirentDatosArchivo->filesize;
```

```
                iCantSectores = (uTamanioArchivo/512);
                if((uTamanioArchivo % 512) !=0)
                {
                        iCantSectores = iCantSectores +1;
                }
                while((iSectoresLeidos < iCantSectores))
                {
                        leer_file2 (uCluster, uiEsp, uiDs);
                        uCluster = BuscarNextSector(uCluster);
                        iSectoresLeidos++;
                }
        }
        else
        {
                vFnImprimir("\nArchivo %s no encontrado\n", pcNombre);
                return;
        }

        return;
}
```

char leer_caracter_file (dword dwSector, int iPosicion)
Parametros que recibe: dword dwSector, int iPosicion
Devuelve: devuelve caracter
Que hace: lee un carácter de archivo

```
char leer_caracter_file (dword dwSector, int iPosicion)
{
        if(leerSector (bloque_read, dwSector+31) != 1)
        {
                vFnImprimir ("No se pudo leer sector %d\n", dwSector);
                return;
        }
        return bloque_read[iPosicion];
}
```

char device_read_char_file (char *pcNombre, int iCaracter)
Parametros que recibe: char *pcNombre, int iCaracter
Devuelve: devuelve caracter
Que hace: Devuelve un caracter del archivo

```
char device_read_char_file (char *pcNombre, int iCaracter)
{
        fs_fat_dirent_t *pstuFsFatDirentDatosArchivo;
        char caracter;
        int iCaracterEnCluster, iN, iClusterBuscado= (iCaracter/512), iCantSectores, iSectoresLeidos=0;
        iCaracterEnCluster= iCaracter%512;
        uint32_t uCluster;
        uint32_t uTamanioArchivo;
        //provisorio hasta crear un comando mount para cargar el pstuFsFatHandleHandle
        header_get();
        cargarFAT();
        carga_Rdir();

        if (fat_file_find2 (pcNombre, pstuFsFatDirentDatosArchivo, pstuFsFatHandleHandle) == 1)
        {
                uCluster= pstuFsFatDirentDatosArchivo->starting_cluster;
                uTamanioArchivo= pstuFsFatDirentDatosArchivo->filesize;
                iCantSectores = (uTamanioArchivo/512);
                if((uTamanioArchivo % 512) !=0)
                        iCantSectores = iCantSectores +1;
                for (iN= 0; iN < iClusterBuscado; iN++)
                        uCluster = BuscarNextSector(uCluster);
                        caracter= leer_caracter_file (uCluster, iCaracterEnCluster);
        }
        else
        {
                vFnImprimir("\nArchivo %s no encontrado\n", pcNombre);
                return;
        }
        return caracter;
}
```

int fat_file_find2 (const char *pcNombre, fs_fat_dirent_t *pstFsFatDirentDatosArchivo , fs_fat_handle_t *pstuFsFatHandleHandle)
Parametros que recibe: const char *pcNombre, fs_fat_dirent_t *pstFsFatDirentDatosArchivo , fs_fat_handle_t *pstuFsFatHandleHandle
Devuelve: Posicion en el vector de procesos o -1 si no existe
Que hace: Busca un nombre de archivo en la fat

```
int fat_file_find2 (const char *pcNombre, fs_fat_dirent_t *pstFsFatDirentDatosArchivo , fs_fat_handle_t *pstuFsFatHandleHandle)
{

	char stNombreAux[MAX_PATH_LEN], stNombreArchivos[MAX_PATH_LEN], stNombreMay[MAX_PATH_LEN];
	uint8_t uX;
	int iJ, iDirLen, iN;
	uint8_t uRootDirFlag=1;
	dword dwDirSector, dwTamanioDir;
	iFnCopiaCadena (stNombreMay, pcNombre) ;
	str_to_upper (stNombreMay);
	dwDirSector= root_dir_start();
	dwTamanioDir= root_dir_size();
	fs_fat_dirent_t *pstuFsFatDirentAuxFat;
	do
	{
		fat_file_find_break:
			if ((iDirLen=str_dir_len(stNombreMay)) > 0)
			{
				strncpy(stNombreAux, pcNombre, iDirLen);
				pcNombre = pcNombre + iDirLen + 1;
			}
			else
				if(iFnCopiaCadena(stNombreAux, stNombreMay) <= 0)
					return -1;
		do
		{
			if (uiModo == 1)
				leerSector (buffer ,dwDirSector);
			else
			{
				do
				{
					iN= leerSector_nonblock (buffer ,dwDirSector);
					if (iN != 1)
					{
						while (!Int);
						while(!TCDMA(2));
						apagarMotor();
					}
				}while (iN != 1);
			}
			pstuFsFatDirentAuxFat = (fs_fat_dirent_t *) buffer;
			for( iJ=0 ; iJ < (SECTOR_SIZE / sizeof(fs_fat_dirent_t)) ; iJ++)
			{
				if( (pstuFsFatDirentAuxFat->filename[0]!=' ') &&(pstuFsFatDirentAuxFat->filename[0]!=FAT_EMPTY) && (pstuFsFatDirentAuxFat->filename[0]!=FAT_DELETED) && !(pstuFsFatDirentAuxFat->attributes & 0x0D) /*&& (pstuFsFatDirentAuxFat->attributes & 0x20)*/)
				{
					fat_adapta_name( pstuFsFatDirentAuxFat->filename , stNombreArchivos);
					if ( iFnCompararCadenas( &stNombreArchivos, &stNombreAux)==1 ) //Devuelve un 1 si la cadena1 es igual a la cadena2
					{
						if (iDirLen < 0) // es < 0 si le pase un archivo
						{
							memcpy( pstFsFatDirentDatosArchivo, pstuFsFatDirentAuxFat, sizeof(fs_fat_dirent_t));
							return 1;
						}
						dwDirSector = pstuFsFatDirentAuxFat->starting_cluster + 31;
						dwTamanioDir = 0xffff;
						uRootDirFlag=0;
						goto fat_file_find_break;
					}
				}
				pstuFsFatDirentAuxFat++;
			}
			if (uRootDirFlag==1)
```

```
                                        {
                                                dwDirSector++;
                                                if (dwDirSector > (root_dir_start() + root_dir_size() ))
                                                return 0;
                                        }
                                        else
                                        {
                                                if ((dwDirSector = pstuFsFatHandleHandle->get_next_cluster(
pstuFsFatHandleHandle->fat1, dwDirSector ))==LAST_SECTOR)
                                                return 0;
                                        }
                                } while (dwDirSector != LAST_SECTOR );
                        }
                        while ( iDirLen > 0 );
                        return 0;
}
```

Carpeta: include/fs
Archivo: fat.h

```
u8 Int;
unsigned int uiModo;
void leer_file2 (dword, unsigned int, unsigned int);
void device_read_file2 (char *, unsigned int, unsigned int);
char leer_caracter_file (dword, int);
char device_read_char_file (char *, int);
int fat_file_find2 (const char *, fs_fat_dirent_t *, fs_fat_handle_t *);
```

Carpeta: include/usr
Archive: libsodium.h

```
#define STL_PRIO_MAX 5
#define STL_PRIO_STD 0
#define STL_PRIO_MIN -5
#define STL_WALK_NEXT 1
#define STL_WALK_PREV 0
/* estados de thread */
#define STL_STATE_SCHEDULER 0           /* exclusivo para el planificador  */
#define STL_STATE_NEW 1                 /* creado pero no despachado       */
#define STL_STATE_READY 2               /* listo para ser despachado       */
#define STL_STATE_WAITING 3             /* suspendido a espera de evento   */
#define STL_STATE_DEAD 4                /* terminado, esperando join       */
#define BLOCK 0x10000
#define NONBLOCK 0x20000


unsigned int uiPosicion;                        //posicion en gdt de inicio del programa que usa trheads
unsigned int uiHilo;
unsigned int uiFuncion;
unsigned int uiEbp;
unsigned int uiEsp;
char *cStack;
unsigned int uiStackSize;

typedef unsigned int stl_mutex_t;

typedef struct stl_st *stl_t;
struct stl_st;

typedef struct mctx_st mctx_t;
struct mctx_st;

struct mctx_st
{
        unsigned int uiEax;
        unsigned int uiEbx;
        unsigned int uiEcx;
        unsigned int uiEdx;
        unsigned int uiEdi;
        unsigned int uiEsi;
        unsigned int uiEbp;
```

```
        unsigned int uiEsp;
};

typedef struct
{
    unsigned long int tv_sec;     /*!< Segundos */
    unsigned long int tv_usec;    /*!< Microsegundos */
} stl_time_t;

#define STL_TIME_NOW  (stl_time_t *)(0)
#define __gettimeofday(t) gettimeofday(t, NULL)
#define stl_time_set(t1,t2) \
  do { \
    if ((t2) == STL_TIME_NOW) \
      __gettimeofday((t1)); \
    else { \
      (t1)->tv_sec  = (t2)->tv_sec; \
      (t1)->tv_usec = (t2)->tv_usec; \
    } \
  } while (0)

struct stl_st
{
        unsigned long tid;
        unsigned long ptid;
        unsigned long pid;
        stl_t q_next;           /* proximo thread                    */
        stl_t q_prev;           /* thread anterior                   */
        int q_prio;             /* prioridad del hilo encolado       */
        unsigned int prioridad;
        unsigned int estado;
        mctx_t contexto;
        stl_time_t creacion;
        int funcion;
        unsigned int stack;
        unsigned int stacksize;
};

struct stl_pqueue_st
{
        stl_t q_head;
        int   q_num;
};
typedef struct stl_pqueue_st stl_pqueue_t;

stl_t stl_main;           /* thread principal                  */
stl_t stl_planif;         /* thread planificador     */
stl_t stl_actual;         /* thread actual       */
stl_pqueue_t stl_NQ;      /* cola de nuevos                    */
stl_pqueue_t stl_RQ;      /* cola de listos        */
stl_pqueue_t stl_WQ;      /* cola de esperando */
stl_pqueue_t stl_SQ;      /* cola de suspendidos           */
stl_pqueue_t stl_DQ;      /* cola de terminados        */


int stl_init (void);
int stl_planif_init (void);
stl_t stl_spawn (int, int);
void stl_setear_contexto (stl_t);
void stl_cambio_contexto (stl_t, stl_t);
void *stl_planificador (void);
void stl_exit (void *);
int stl_join (stl_t, void **);
int stl_yield_esp (stl_t);
int stl_yield (stl_t);
int stl_wait (unsigned int);
void stl_mutex_init (stl_mutex_t *);
int stl_mutex_acquire (stl_mutex_t *);
int stl_mutex_release (stl_mutex_t *);

stl_t stl_tcb_alloc (int, void *);
void stl_tcb_free (stl_t);
int stl_contar_threads (void);

void stl_pqueue_init (stl_pqueue_t *);
void stl_pqueue_insert (stl_pqueue_t *, int, stl_t);
```

```
void stl_pqueue_delete (stl_pqueue_t *, stl_t);
stl_t stl_pqueue_tail (stl_pqueue_t *);
void stl_pqueue_increase (stl_pqueue_t *);
stl_t stl_pqueue_delmax (stl_pqueue_t *);
int stl_pqueue_favorite (stl_pqueue_t *, stl_t);
stl_t stl_pqueue_walk (stl_pqueue_t *, stl_t, int);
int stl_pqueue_contains (stl_pqueue_t *, stl_t);
/* devuelve el numero de elementos de la cola: O(1) */
#define stl_pqueue_elements(q) \
   ((q) == NULL ? (-1) : (q)->q_num)
/* llega al primer thread de la cola; O(1) */
#define stl_pqueue_head(q) \
   ((q) == NULL ? NULL : (q)->q_head)
/* determina la prioridad requerida para favorecer a un thread; O(1) */
#define stl_pqueue_favorite_prio(q) \
   ((q)->q_head != NULL ? (q)->q_head->q_prio + 1 : STL_PRIO_MAX)

void stl_SaltarN0 (unsigned int, unsigned int, unsigned int, unsigned int);
void stl_ImprimirCadena (const char *, ...);
void stl_Entero (unsigned int);
void stl_Float (unsigned int, int);
void stl_Byte (unsigned int);
void stl_WordHexa (unsigned int);
void stl_ByteHexa (unsigned int);
void stl_Hexa (unsigned int);
void stl_ImprimirCaracter (unsigned int);
int stl_CopiaCaracter (char *, const char *);
int stl_CopiaCadena (char *, const char *);
int stl_kill (unsigned int);
unsigned long stl_solicitarCheckSum ();
unsigned int stl_ObtenerHora ();
```

Carpeta: include/kernel
Archivo: syscall.h

Declaraciones varias:

```
long lFnSysGettimeofday2(timeval *ptimervalTp, timezone *ptimezoneTzp);
```

Carpeta: include/kernel
Archivo: system.h

Declaraciones varias:

```
timeval stuTimeval_tiempo;
unsigned int uiParametro1, uiParametro2, uiParametro3, uiParametro4;
// Guarda el tiempo del sistema, representado como la cantidad de milesimas de segundos desde una fecha determinada
unsigned long ulTiempo;
static unsigned stauFnLeerCmos (unsigned, char);
long vFnGetTimeInSecs ();
```

Carpeta: kernel
Archivo: syscall.c

long lFnSysGettimeofday2 (timeval *timervalTp, timezone *timezoneTzp)
Parametros que recibe: timeval *timervalTp, timezone *timezoneTzp
Devuelve: devuelve un long
Que hace: devuelve en stuTimeval_tiempo la fecha actual en seg y microseg dede el 1/1/2000

```
long lFnSysGettimeofday2 (timeval *timervalTp, timezone *timezoneTzp)
{
        timeval *timevalTime = (timeval *)(pstuPCB[uiParametro2].uiDirBase + (unsigned int)timervalTp);
        timezone *timezoneZona = (timezone *)(pstuPCB[uiParametro2].uiDirBase + (unsigned int)timezoneTzp);
        unsigned long ulMilisegundos= ulTiempo;
        timevalTime->tv_sec = ulMilisegundos/1000;
        timevalTime->tv_usec = ulMilisegundos - (timevalTime->tv_sec*1000);
        timevalTime->tv_sec += (iMinuteswest*60);
```

```
	timezoneZona->tz_minuteswest = iMinuteswest;
	timezoneZona->tz_dsttime = 0;
	stuTimeval_tiempo.tv_sec= timevalTime->tv_sec;
	stuTimeval_tiempo.tv_usec= timevalTime->tv_usec;
	return 0;
}
```

Carpeta: kernel
Archivo: system.c

Declaraciones varias:

```
unsigned int uiUltimoCaracter= 0, uiUltimoCaracter1= 0, uiUltimoCaracter2= 0, uiUltimoCaracter3= 0, uiCheckSum= 0, ds, uiCar,
digito, uiLinea;
char cCadena[4][512], caracter, cDigito[32];
```

Carpeta: kernel
Archivo: system.c

void vFnHandlerGenericoCallGate ()
Parametros que recibe: no recibe nada
Devuelve: no devuelve nada
Que hace: Handler para el manejo del callgate (interrupcion 0x80)

```
void vFnHandlerGenericoCallGate ()
{
	asm volatile ("movl %%ebx, %0" : "=m" (uiParametro4));
	asm volatile ("movl %%eax, %0" : "=m" (uiParametro3));
	asm volatile ("movl %%ecx, %0" : "=m" (uiParametro2));
	asm volatile ("movl %%esi, %0" : "=a" (uiParametro1));
	switch (uiParametro1)
	{
	……
```

Se agregan alguno casos mas:

```
	case 75: iFnCrearProceso ("A_BIN.BIN", 5, 3);  break;
		case 76: iFnCrearProceso ("B_BIN.BIN", 5, 3);  break;
		case 77:
			r= ulFnSysBrk (uiParametro3);
			asm volatile ("pushl %%ds\n\tmovl %0, %%ds\n\tmovl %2, (%1)\n\tpopl %%ds\n\t" :: "a"
(stuTSSTablaTareas[uiParametro2].ds),"b" (uiParametro4), "c" (r));
		break;
		case 78:
			stuTimeval_tiempo.tv_sec= stuTimeval_tiempo.tv_usec= 0;
			stuTimeval_tiempo.tv_sec= uiParametro2;
			stuTimeval_tiempo.tv_usec= uiParametro3;
		break;
		case 79:
			lFnSysGettimeofday2 (&stuTimeval_tiempo, NULL);
			asm volatile ("pushl %%ds\n\tmovl %0, %%ds\n\tmovl %2, (%1)\n\tpopl %%ds\n\t" :: "a"
(stuTSSTablaTareas[uiParametro2].ds),"b" (uiParametro4), "c" (stuTimeval_tiempo.tv_sec));
			uiParametro4+= 4;
			asm volatile ("pushl %%ds\n\tmovl %0, %%ds\n\tmovl %2, (%1)\n\tpopl %%ds\n\t" :: "a"
(stuTSSTablaTareas[uiParametro2].ds),"b" (uiParametro4), "c" (stuTimeval_tiempo.tv_usec));
		break;
		case 80: uiCheckSum= uiParametro2; break;
		case 81:
			cCadena[0][uiUltimoCaracter+1]= '\0';
			cCadena[1][uiUltimoCaracter1+1]= '\0';
			cCadena[2][uiUltimoCaracter2+1]= '\0';
			cCadena[3][uiUltimoCaracter3+1]= '\0';
		break;
		case 82:
			uiModo= uiParametro2 >> 16;
			uiParametro2= uiParametro2&0xFFFF;
			device_read_file2 (cCadena[uiParametro2], uiParametro4, uiParametro3);
		break;
		case 83:
			asm volatile ("pushl %%ds\n\tmovl %0, %%ds\n\tmovl %2, (%1)\n\tpopl %%ds\n\t" :: "a"
(uiParametro2),"b" (uiParametro4), "c" (uiCheckSum));
		break;
```

```
                        case 84:
                                uiAuxiliar= vFnGetTimeInSecs ();
                                asm volatile ("pushl %%ds\n\tmovl %0, %%ds\n\tmovl %2, (%1)\n\tpopl %%ds\n\t" :: "a"
(uiParametro2),"b" (uiParametro4), "c" (uiAuxiliar));
                        break;
                        case 85: iFnCrearProceso ("C_BIN.BIN", 5, 3);  break; }
}
long vFnGetTimeInSecs ()
{
        unsigned int uHora, uMinuto, uSegundo;
        long timeInSecs;
        static int staiCaracter;
        outb (0x0B, 0x70);
        if (inb (0x71)&0x04)
                staiCaracter= 0;
        else
                staiCaracter= 1;
        uHora= stauFnLeerCmos (4, staiCaracter);
        uMinuto= stauFnLeerCmos (2, staiCaracter);
        uSegundo= stauFnLeerCmos (0, staiCaracter);
        timeInSecs= uSegundo+uMinuto*60+uHora*60*60;
        return timeInSecs;
}

static unsigned stauFnLeerCmos (unsigned uRegistro, char cCaracter)
{
        unsigned uDigitoAlto, uDigitoBajo;
        outb (uRegistro, 0x70);
        uDigitoAlto= uDigitoBajo= inb (0x71);
        if (!cCaracter)
                return uDigitoBajo;
        uDigitoAlto >>= 4;
        uDigitoAlto &= 0x0F;
        uDigitoBajo &= 0x0F;
        return 10*uDigitoAlto+uDigitoBajo;
}
```

Carpeta: kernel/drivers
Archivo: floppy.c

u8 leerSector_nonblock (u32 Buffer, u16 NBloque)
Parametros que recibe: u32 Buffer, u16 NBloque
Devuelve: devuelve 1 o 0 dependiendo si se genero o no un error
Que hace: lee un sector del diskette

```
u8 leerSector_nonblock (u32 Buffer, u16 NBloque)
{
        CHS chs;
        int iN;
        while(!esperandoByteProc()&& !TCDMA(2));
        prenderMotor();
        setCanalDMA(2, Buffer, MDMADef | Tr_Esc | Canal2, 0x1FF);
        habilitarCanal(2);
        chs = LBAaCHS(NBloque);
        while(!envComDesplCabeza(chs.Cilindro, chs.Cabeza));
        Int = 0;

        for (iN= 0; iN < 300000; iN++);

        if (!Int)
                return 0;

        while(!envComLectura(chs.Cilindro, chs.Cabeza, chs.Sector));
        Int = 0;

        for (iN= 0; iN < 300000; iN++);

        if (!Int)
                return 0;

        while(!TCDMA(2));
        apagarMotor();
```

```
	if(operExitosa())
		return 1;

	while(!esperandoByteProc());
	leerEstadoFDC();
	vFnImprimirEstadoFDC();
	while(!ST0.Valido);
	vFnImprimirST0();
	while(!ST1.Valido);
	vFnImprimirST1();
	while(!ST2.Valido);
	vFnImprimirST2();
	while(!leerST3());
	vFnImprimirST3();
	return 0;
}
```

Carpeta: usr
Archivo: a_bin.c

Archivo correspodiente al TP2
Programa A correspondiente al TP2

```
#include <usr/leer_paralelo.h>
#include <usr/libsodium.h>
#include <kernel/libk/libk.h>

t_resultado resultado;
stl_t t[CANT_SIMULTANEOS];
t_resultado leer_paralelo ();
void *vFnleer_archivo (void *);
unsigned int ds;

int main ()
{
	asm volatile ("movl (%1), %0" : "=a" (uiPosicion) : "b" (0x2FC4));
	unsigned long ulValorDeComprobacion= stl_solicitarCheckSum ();
	unsigned int start_time=0, finish_time=0;
	int i, opt, retVal;
	bool bBadFiles=false;
	t_resultado res;

	/* Inicializamos la API multithreading */
	stl_ImprimirCadena ("\n#--------------------------------------------------------------------------------\n");

	start_time= stl_ObtenerHora ();

	/*A PARTIR DE AQUI TENEMOS CUATRO ARCHIVOS VALIDOS. FALTA DESARROLLAR EL METODO DE
LECTURA*/
	/*GUARDA QUE LOS NOMBRES DE LOS ARCHIVOS SE UBICAN A PARTIR DE argv[3]*/

	res= leer_paralelo ();

	finish_time= stl_ObtenerHora ();

	stl_ImprimirCadena ("\n# %d abin\tMAIN_INICIO\t%d\n", LEGAJO, start_time);
	stl_ImprimirCadena ("# %d abin\tRES=%d, SUMA=%d\n", LEGAJO, res.ok, res.suma);
	if (res.ok == true && res.suma == ulValorDeComprobacion)
	 for (i=0; i<CANT_SIMULTANEOS; i++)
	  stl_ImprimirCadena ("%d abin\tLECTURA%d\t%d\t%d\t%d.00\n", LEGAJO, i, res.lectura[i].inicio, res.lectura[i].fin,
res.lectura[i].fin-res.lectura[i].inicio);
	else
		stl_ImprimirCadena("# %d abin\tERROR DE EJECUCION!!!\n", LEGAJO);
	stl_ImprimirCadena ("# %d abin\tMAIN_FIN\t%d\t%d.00\n", LEGAJO, finish_time, finish_time-start_time);
	stl_kill (uiPosicion);
	while (1);
}

void *vFnleer_archivo (void *iParam)
{
	int id= (int) stl_actual->tid, iSuma= 0, iError= 0, iN, iRet, iDigito= 0;
	unsigned int lliNum= 0;
	char cBuffer[100];
```

```
		resultado.lectura[id-2].inicio= stl_ObtenerHora ();

		stl_SaltarN0 (82, (id-2)|BLOCK, ds, uiStackSize);
		cStack= uiStackSize;
		for (iN= uiStackSize; iN > (uiStackSize-(CANT_LINEAS)); iN--)
		{
			while ((*cStack)-48 != -38)
			{
				if ((*cStack)-48 != -48)
				{
					cBuffer[iDigito]= (*cStack);
					iDigito++;
				}
				cStack-=4;
			}
			cBuffer[iDigito]= '\0';
			iDigito= 0;
			cStack-=4;
			iSuma= (iSuma+((iFnCtoi (cBuffer))%32767))%32767;
		}
		if (!iError)
			resultado.suma+= iSuma;

		resultado.lectura[id-2].fin= stl_ObtenerHora ();

		stl_exit (3);
}

t_resultado leer_paralelo ()
{
		int iN;
		stl_init ();
		cStack= malloc (512);
		uiStackSize= cStack+512;
		asm volatile ("pushl %ds");
		asm volatile ("popl %0" : "=m" (ds));
		for (iN= 0; iN < CANT_SIMULTANEOS; iN++)
		{
			t[iN]= stl_spawn (vFnleer_archivo, 1);
			resultado.lectura[iN].id= (unsigned long) t[iN];
		}
		stl_join (t[0], NULL);
		stl_join (t[1], NULL);
		stl_join (t[2], NULL);
		stl_join (t[3], NULL);
		free (cStack);
		resultado.ok= true;
		return resultado;
}
```

Carpeta: usr
Archivo: b_bin.c

Archivo correspodiente al TP2
Programa B correspondiente al TP2

```
#include <usr/leer_paralelo.h>
#include <usr/libsodium.h>
#include <kernel/libk/libk.h>

t_resultado resultado;
stl_t t[CANT_SIMULTANEOS];
t_resultado leer_paralelo ();
void *vFnleer_archivo (void *);
unsigned int ds;

int main ()
{
		asm volatile ("movl (%1), %0" : "=a" (uiPosicion) : "b" (0x2FC4));
		unsigned long ulValorDeComprobacion= stl_solicitarCheckSum ();
		unsigned int start_time=0, finish_time=0;
		int i, opt, retVal;
```

```
	bool bBadFiles=false;
	t_resultado res;

	/* Inicializamos la API multithreading */
	stl_ImprimirCadena ("\n#--------------------------------------------------------------------------------\n");

	start_time= stl_ObtenerHora ();

	/*A PARTIR DE AQUI TENEMOS CUATRO ARCHIVOS VALIDOS. FALTA DESARROLLAR EL METODO DE
LECTURA*/
	/*GUARDA QUE LOS NOMBRES DE LOS ARCHIVOS SE UBICAN A PARTIR DE argv[3]*/

	res= leer_paralelo ();

	finish_time= stl_ObtenerHora ();

	stl_ImprimirCadena ("\n# %d bbin\tMAIN_INICIO\t%d\n", LEGAJO, start_time);
	stl_ImprimirCadena ("# %d bbin\tRES=%d, SUMA=%d\n", LEGAJO, res.ok, res.suma);
	if (res.ok == true && res.suma == ulValorDeComprobacion)
	 for (i=0; i<CANT_SIMULTANEOS; i++)
	  stl_ImprimirCadena ("%d bbin\tLECTURA%d\t%d\t%d\t%d.00\n", LEGAJO, i, res.lectura[i].inicio, res.lectura[i].fin,
res.lectura[i].fin-res.lectura[i].inicio);
	else
		stl_ImprimirCadena("# %d bbin\tERROR DE EJECUCION!!!\n", LEGAJO);
	stl_ImprimirCadena ("# %d bbin\tMAIN_FIN\t%d\t%d.00\n", LEGAJO, finish_time, finish_time-start_time);
	stl_kill (uiPosicion);
	while (1);
}

void *vFnleer_archivo (void *iParam)
{
	int id= (int) stl_actual->tid, iSuma= 0, iError= 0, iN, iRet, iDigito= 0;
	unsigned int lliNum= 0;
	char cBuffer[100];

	resultado.lectura[id-2].inicio= stl_ObtenerHora ();

	stl_SaltarN0 (82, (id-2)|NONBLOCK, ds, uiStackSize);
	cStack= uiStackSize;
	for (iN= uiStackSize; iN > (uiStackSize-(CANT_LINEAS)); iN--)
	{
		while ((*cStack)-48 != -38)
		{
			if ((*cStack)-48 != -48)
			{
				cBuffer[iDigito]= (*cStack);
				iDigito++;
			}
			cStack-=4;
		}
		cBuffer[iDigito]= '\0';
		iDigito= 0;
		cStack-=4;
		iSuma= (iSuma+((iFnCtoi (cBuffer))%32767))%32767;
	}
	if (!iError)
		resultado.suma+= iSuma;

	resultado.lectura[id-2].fin= stl_ObtenerHora ();

	stl_exit (3);
}

t_resultado leer_paralelo ()
{
	int iN;
	stl_init ();
	cStack= malloc (512);
	uiStackSize= cStack+512;
	asm volatile ("pushl %ds");
	asm volatile ("popl %0" : "=m" (ds));
	for (iN= 0; iN < CANT_SIMULTANEOS; iN++)
	{
		t[iN]= stl_spawn (vFnleer_archivo, 1);
		resultado.lectura[iN].id= (unsigned long) t[iN];
	}
```

```
	stl_join (t[0], NULL);
	stl_join (t[1], NULL);
	stl_join (t[2], NULL);
	stl_join (t[3], NULL);
	free (cStack);
	resultado.ok= true;
	return resultado;
}
```

Carpeta: usr
Archivo: c_bin.c

Archivo extra correspodiente al TP2
Programa C usado para testear funcionalidades

```
#include <usr/leer_paralelo.h>
#include <usr/libsodium.h>

static void *func1 (void);
static void *func2 (void);
static void *func3 (void);
static void *func4 (void);
static void *func5 (void);
int mi_fun (int);
stl_t h1, h2, h3, h4, h5;
stl_mutex_t mut;

int main ()
{
	asm volatile ("movl (%1), %0" : "=a" (uiPosicion) : "b" (0x2FC4));
	int suma;
	suma= 10;

	stl_ImprimirCadena ("\nINICIO PROGRAMA\n");
	stl_init ();
	stl_mutex_init (&mut);
	stl_ImprimirCadena ("BIBLIOTECA INICIALIZADA\n");
	h1= stl_spawn (func1, 1);
	h2= stl_spawn (func2, 1);
	h3= stl_spawn (func3, 1);
	h4= stl_spawn (func4, 1);
	h5= stl_spawn (func5, 1);

	stl_join (h1, NULL);
	stl_join (h2, NULL);
	stl_join (h3, NULL);
	stl_join (h4, NULL);
	stl_join (h5, NULL);
	stl_ImprimirCadena ("\n%d\n", suma);
	stl_ImprimirCadena ("YO DEBERIA APARECER\n");
	stl_kill (uiPosicion);
	while (1);
}

static void *func1 (void)
{
	int i= 0;
	i= h1->tid;
	stl_ImprimirCadena ("SOY %d, MI PADRE ES %d\n", h1->tid, h1->ptid);
	stl_ImprimirCadena ("SOY %d\n", i);
	stl_exit (3);
}

static void *func2 (void)
{
	stl_ImprimirCadena ("SOY %d, MI PADRE ES %d, ME CREARON %d %d\n", h2->tid, h2->ptid, h2->creacion.tv_sec,
h2->creacion.tv_usec);
	stl_exit (3);
}

static void *func3 (void)
{
	int i= 35;
```

```
		stl_ImprimirCadena ("SOY %d, MI PADRE ES %d, MI i ES %d\n", h3->tid, h3->ptid, i);
		stl_ImprimirCadena ("MUT ANTES: %d\n", mut);
		stl_mutex_acquire (&mut);
			i++;
			stl_ImprimirCadena ("MUT DURANTE: %d\n", mut);
		stl_mutex_release (&mut);
		stl_ImprimirCadena ("MUT DESPUES: %d\n", mut);
		stl_exit (3);
}

static void *func4 (void)
{
		int i= 23, j= 11;
		stl_ImprimirCadena ("SOY %d, MI PADRE ES %d, MI i ES %d, MI j ES %d\n", h4->tid, h4->ptid, i, j);
		j= mi_fun (i);
		stl_ImprimirCadena ("VOLVI, SOY %d, MI i ES %d, MI j ES %d\n", h4->tid, i, j);
		stl_exit (3);
}

static void *func5 (void)
{
		stl_ImprimirCadena ("SOY %d, MI PADRE ES %d\n", h5->tid, h5->ptid);
		stl_exit (3);
}

int mi_fun (int a)
{
		return (a+14);
}
```

Carpeta: usr
Archivo: libsodium.c

int gettimeofday (timeval *ptimevalTp, timezone *ptimezoneTzp)
Parametros que recibe: timeval *ptimevalTp, timezone *ptimezoneTzp
Devuelve: si esta todo bien devuelve 1
Que hace: recibe la direccion de una estructura timeval, y hace un callgate al gettimeofday de nivel 0 para que esa estructura se llene con la fecha actual en seg y miliseg

```
int gettimeofday (timeval *ptimevalTp, timezone *ptimezoneTzp)
{
		unsigned int p4;
		stl_SaltarN0 (78, ptimevalTp->tv_sec, ptimevalTp->tv_usec, 0);
		asm volatile ("movl %%esp, %0" : "=m" (p4));
		stl_SaltarN0 (79, uiPosicion, 0, p4);
		asm volatile ("movl (%1), %0" : "=a" (ptimevalTp->tv_sec) : "b" (p4));
		p4+= 4;
		asm volatile ("movl (%1), %0" : "=a" (ptimevalTp->tv_usec) : "b" (p4));
		return 1;
}
```

unsigned long brk (unsigned long limite)
Parametros que recibe: unsigned long limite
Devuelve: la direccion de comienzo
Que hace: hace un callgate que termina llegando al brk de nivel 0, que reserva un espacio de memoria (limite) y devuelve la direccion de comienzo

```
unsigned long brk (unsigned long limite)
{
		long liRetorno;
		//SYS_CALL_1 (liRetorno, errno, __NR_brk, limite);
		unsigned int p1= 77, p2, p3, p4;
		p2= uiPosicion;
		p3= (unsigned int)limite;
		asm volatile ("movl %%esp, %0" : "=m" (p4));
		stl_SaltarN0 (p1, p2, p3, p4);
		asm volatile ("movl (%1), %0" : "=a" (liRetorno) : "b" (p4));
		return liRetorno;
}
```

```
int stl_inicializado= -1;
int es_exit= 0;
unsigned int esp, ebp;
stl_t t1, t2;
```

int stl_init (void)
Parametros que recibe: recibe void
Devuelve: devuelve 1 o -1 dependiendo de si pudo o no pudo
Que hace: inicializa la biblioteca permitiendo la creación de nuevos threads por parte de procesos pesados

```
int stl_init (void)
{

	asm volatile ("movl %%ebp, %0" : "=m" (uiEbp));
		asm volatile ("movl (%1), %0" : "=a" (ebp) : "b" (uiEbp));
	uiEbp+= 4;
	asm volatile ("movl (%1), %0" : "=a" (uiFuncion) : "b" (uiEbp));
		esp= uiEbp+ 4;
		if (stl_inicializado == 1)
			return -1;
		else
		stl_inicializado= 1;
		uiHilo= 0;
		if (!stl_planif_init())
		{
			stl_ImprimirCadena ("ERROR EN stl_planif_init()\n");
		return -1;
		}
		if ((stl_planif= stl_spawn (stl_planificador, STL_PRIO_MAX)) == NULL)
		{
			stl_ImprimirCadena ("ERROR AL CREAR PLANIFICADOR\n");
		return -1;
		}
		if ((stl_main= stl_spawn (uiFuncion, STL_PRIO_STD)) == NULL)
		{
			stl_ImprimirCadena ("ERROR AL CREAR EL MAIN THREAD\n");
			return -1;
		}
		stl_actual= stl_planif;
		stl_main->contexto.uiEsp= esp;
		stl_main->contexto.uiEbp= ebp;
		stl_cambio_contexto (stl_planif, stl_main);
}
```

int stl_planif_init (void)
Parametros que recibe: recibe void
Devuelve: devuelve 1 si pudo hacerlo
Que hace: inicializa el planificador

```
int stl_planif_init (void)
{
		stl_planif= NULL;
		stl_actual= NULL;
		stl_main= NULL;
		/* inicializa colas de threads */
		stl_pqueue_init (&stl_NQ);
		stl_pqueue_init (&stl_RQ);
		stl_pqueue_init (&stl_WQ);
		stl_pqueue_init (&stl_SQ);
		stl_pqueue_init (&stl_DQ);
		return 1;
}
```

stl_t stl_spawn (int funcion, int prioridad)
Parametros que recibe: int funcion, int prioridad
Devuelve: devuelve el hilo o devuelve null
Que hace: Creación de un nuevo hilo con los atributos dados

```
stl_t stl_spawn (int funcion, int prioridad)
{
```

```
	stl_t t;
	stl_time_t ts;
	ts.tv_sec= ts.tv_usec= 0;

	if ((t = stl_tcb_alloc (uiHilo, NULL)) == NULL)
	{
		stl_ImprimirCadena ("No se pudo reservar memoria para el thread\n");
		return NULL;
	}
	t->tid= uiHilo;
	t->funcion= funcion;
	t->ptid= 0;
	t->pid= uiPosicion;
	stl_time_set (&ts, STL_TIME_NOW);
	stl_time_set (&t->creacion, &ts);
	t->prioridad= prioridad;
	if (uiHilo)
	{
	t->estado= STL_STATE_NEW;
	stl_pqueue_insert (&stl_NQ, t->prioridad, t);
	}
	if (uiHilo != 1)
		stl_setear_contexto (t);
	switch (uiHilo)
	{
		case 0: t->ptid= 0; break;
		case 1: t->ptid= 0; break;
		default: t->ptid= stl_actual->tid; break;
	}
	uiHilo++;
	return t;
}
```

void stl_setear_contexto (stl_t t1)
Parametros que recibe: stl_t t1
Devuelve: no devuelve nada
Que hace: setea el contexto

```
void stl_setear_contexto (stl_t t1)
{
	t1->contexto.uiEax= 0;
	t1->contexto.uiEbx= 0;
	t1->contexto.uiEcx= 0;
	t1->contexto.uiEdx= 0;
	t1->contexto.uiEdi= 0;
	t1->contexto.uiEsi= 0;
}
```

void stl_cambio_contexto (stl_t T1, stl_t T2)
Parametros que recibe: stl_t T1, stl_t T2
Devuelve: no devuelve nada
Que hace: realiza el cambio de contexto

```
void stl_cambio_contexto (stl_t T1, stl_t T2)
{
	t1= T1;
	t2= T2;
	if (!t2->tid && !t2->funcion)
	{
		asm volatile ("movl %%ebp, %0" : "=m" (uiEbp));
		asm volatile ("movl (%1), %0" : "=a" (t2->contexto.uiEbp) : "b" (uiEbp));
	uiEbp+= 4;
		asm volatile ("movl (%1), %0" : "=a" (t2->funcion) : "b" (uiEbp));
		t2->contexto.uiEsp= uiEbp+ 4;
	}
	asm volatile ("movl %%eax, %0" : "=m" (t2->contexto.uiEax));
	asm volatile ("movl %%ebx, %0" : "=m" (t2->contexto.uiEbx));
	asm volatile ("movl %%ecx, %0" : "=m" (t2->contexto.uiEcx));
	asm volatile ("movl %%edx, %0" : "=m" (t2->contexto.uiEdx));
	asm volatile ("movl %%edi, %0" : "=m" (t2->contexto.uiEdi));
	asm volatile ("movl %%esi, %0" : "=m" (t2->contexto.uiEsi));
	if (!es_exit)
		stl_actual= t1;
	else
```

```
                es_exit= 0;
        asm volatile ("movl %0, %%eax" :: "m" (t1->contexto.uiEax));
        asm volatile ("movl %0, %%ebx" :: "m" (t1->contexto.uiEbx));
        asm volatile ("movl %0, %%ecx" :: "m" (t1->contexto.uiEcx));
        asm volatile ("movl %0, %%edx" :: "m" (t1->contexto.uiEdx));
        asm volatile ("movl %0, %%edi" :: "m" (t1->contexto.uiEdi));
        asm volatile ("movl %0, %%esi" :: "m" (t1->contexto.uiEsi));
        asm volatile ("movl %0, %%esp" :: "m" (t1->contexto.uiEsp));
        asm volatile ("movl %0, %%ebp" :: "m" (t1->contexto.uiEbp));
        asm volatile ("jmp %0" :: "m" (t1->funcion));
}
```

void *stl_planificador (void)
Parametros que recibe: recibe void
Devuelve: no devuelve nada
Que hace: es el planificador de la cola de threads

```
void *stl_planificador (void)
{
        stl_t t;
        stl_planif->estado = STL_STATE_SCHEDULER;
        stl_planif->funcion= 0;
        for (;;)
        {
        /* mueve threads de la cola de nuevos a la de listos y opcionalmente les da maxima prioridad para que
        empiecen inmediatamente */
                while ((t = stl_pqueue_tail (&stl_NQ)) != NULL)
                {
                stl_pqueue_delete (&stl_NQ, t);
                t->estado= STL_STATE_READY;
                      stl_pqueue_insert (&stl_RQ, 0, t);
        }
        /* Encuentra el primer thread en la cola de listos */
                stl_actual= stl_pqueue_delmax (&stl_RQ);
        if (stl_actual == NULL)
        {
                                stl_ImprimirCadena ("STL PLANIFICADOR ERROR: no mas threads que planificar?");
                                stl_kill (uiPosicion);
                                while (1);
        }
        stl_cambio_contexto (stl_actual, stl_planif);
        /* si el thread anterior esta marcado como muerto lo elimina  */
        if (stl_actual->estado == STL_STATE_DEAD)
                {
        stl_pqueue_insert (&stl_DQ, STL_PRIO_STD, stl_actual);
                stl_actual= NULL;
        }
        /* Si el thread tiene que esperar un evento se lo mueve a la cola de espera */
        if (stl_actual != NULL && stl_actual->estado == STL_STATE_WAITING)
                {
                stl_pqueue_insert (&stl_WQ, stl_actual->prioridad, stl_actual);
                stl_actual= NULL;
        }
        /* migra threads antiguos en la cola de listos con mayor prioridad, para evitar starvation e inserta a lo ultimo el thread
que termino de ejecutar */
                stl_pqueue_increase (&stl_RQ);
                if (stl_actual != NULL)
                        stl_pqueue_insert (&stl_RQ, stl_actual->prioridad, stl_actual);
        }
}
```

stl_t stl_tcb_alloc (int parametro, void *stackaddr)
Parametros que recibe: int parametro, void *stackaddr
Devuelve: devuelve la nueva tcb
Que hace: reserva espacio para una tcb

```
stl_t stl_tcb_alloc (int parametro, void *stackaddr)
{
        stl_t t;
        unsigned int stack= 0;
        if ((t = (stl_t) malloc (sizeof (struct stl_st))) == NULL)
                return NULL;
        if (parametro != 1)
                t->stacksize= 3*1024;
```

```
		else
			t->stacksize= 0;
		t->stack= NULL;
		if (parametro != 1)
		{
			if ((t->stack= (unsigned int) malloc (t->stacksize)) == NULL)
				return NULL;
			else
			{
				stack= t->stack+t->stacksize-0x10;
				t->contexto.uiEbp= stack;
				t->contexto.uiEsp= stack;
			}
		}
		return t;
}
```

void stl_exit(void *value)
Parametros que recibe: void *value
Devuelve: no devuelve nada
Que hace: termina el thread quitandolo de ejecución y poniendolo en una cola pendiente de limpieza

```
void stl_exit(void *value)
{
		if (stl_actual != stl_main)
		{
		stl_actual->estado = STL_STATE_DEAD;
			es_exit= 1;
		stl_cambio_contexto (stl_planif, stl_actual);
		}
		else
		{
		stl_ImprimirCadena ("\nPrograma Finalizado\n");
			stl_kill (uiPosicion);
		}
}
```

int stl_join (stl_t tid, void **value)
Parametros que recibe: stl_t tid, void **value
Devuelve: devuelve 1 o -1 dependiendo de si pudo o no
Que hace: espera la terminacion de un determinado thread

```
int stl_join (stl_t tid, void **value)
{
		asm volatile ("movl %%ebp, %0" : "=m" (uiEbp));
		asm volatile ("movl (%1), %0" : "=a" (stl_actual->contexto.uiEbp) : "b" (uiEbp));
	uiEbp+= 4;
		asm volatile ("movl (%1), %0" : "=a" (uiFuncion) : "b" (uiEbp));
		stl_actual->contexto.uiEsp= uiEbp+ 4;
		if (tid == stl_actual)
		return -1;
		if (stl_contar_threads() == 1)
		return -1;
		if (tid == NULL)
		tid = stl_pqueue_head (&stl_DQ);
		if (tid == NULL || (tid != NULL && tid->estado != STL_STATE_DEAD))
		stl_wait (tid->tid);
		if (tid == NULL)
		tid = stl_pqueue_head (&stl_DQ);
		if (tid == NULL || (tid != NULL && tid->estado != STL_STATE_DEAD))
		return -1;
		stl_pqueue_delete (&stl_DQ, tid);
		stl_tcb_free(tid);
		return 1;
}
```

int stl_wait (unsigned int tid)
Parametros que recibe: unsigned int tid
Devuelve: devuelve 1 si pudo
Que hace: espera uno o mas eventos

```
int stl_wait (unsigned int tid)
{
```

```
        int nonpending= 1;
        /* cambia el estado del hilo actual a esperando y transfiere el control al planificador */
        stl_pqueue_insert (&stl_RQ, STL_PRIO_MIN, stl_actual);
    stl_actual->estado= STL_STATE_READY;
        stl_yield (NULL);
        return nonpending;
}
```

int stl_yield (stl_t to)
Parametros que recibe: stl_t to
Devuelve: devuelve 1 o -1 dependiendo de si pudo o no
Que hace: le devuelve el control al planificador para un context

```
int stl_yield (stl_t to)
{
        stl_pqueue_t *q = NULL;
        /* el thread tiene que ser nuevo o listo o se ignora el pedido */
        if (to != NULL)
        {
        switch (to->estado)
                {
                case STL_STATE_NEW:    q = &stl_NQ; break;
                case STL_STATE_READY:  q = &stl_RQ; break;
                default:               q = NULL;
        }
        if (q == NULL || !stl_pqueue_contains(q, to))
                return -1;
        }
        /* se le da a un thread la maxima prioridad en su cola */
        if (to != NULL && q != NULL)
        stl_pqueue_favorite (q, to);
        /* cambio al planificador */
        stl_actual->funcion= uiFuncion;
        stl_cambio_contexto (stl_planif, stl_actual);
        return 1;
}
```

void stl_tcb_free (stl_t t)
Parametros que recibe: stl_t t
Devuelve: no devuelve nada
Que hace: libera un tcb

```
void stl_tcb_free (stl_t t)
{
        if (t == NULL)
        return;
        if (t->stacksize > 0)
                free (t->stack);
        free(t);
        return;
}
```

void stl_mutex_init (stl_mutex_t *mutex)
Parametros que recibe: stl_mutex_t *mutex
Devuelve: no devuelve nada
Que hace: inicializa mutex

```
void stl_mutex_init (stl_mutex_t *mutex)
{
        *mutex= -1;
        return;
}
```

int stl_mutex_acquire (stl_mutex_t *mutex)
Parametros que recibe: stl_mutex_t *mutex
Devuelve: devuelve 1 o -1 dependiendo de si pudo o no
Que hace: obtiene mutex

```
int stl_mutex_acquire (stl_mutex_t *mutex)
{
        while (*mutex != -1);
        *mutex= stl_actual->tid;
```

```
	return 1;
}
```

int stl_mutex_release (stl_mutex_t *mutex)
Parametros que recibe: stl_mutex_t *mutex
Devuelve: devuelve 1 o -1 dependiendo de si pudo o no
Que hace: libera mutex

```
int stl_mutex_release (stl_mutex_t *mutex)
{
	if (*mutex == -1)
		return -1;
	*mutex= -1;
	return 1;
}

/*----------------------COMIENZO DE LA CODIFICACION DE FUNCIONES ADICIONALES ----------------------------*/
```

void stl_SaltarN0 (unsigned int uiParametro1, unsigned int uiParametro2, unsigned int uiParametro3, unsigned int uiParametro4)
Parametros que recibe: unsigned int uiParametro1, unsigned int uiParametro2, unsigned int uiParametro3, unsigned int uiParametro4
Devuelve: no devuelve nada
Que hace: cambia a nivel 0

```
void stl_SaltarN0 (unsigned int uiParametro1, unsigned int uiParametro2, unsigned int uiParametro3, unsigned int uiParametro4)
{
	unsigned int uiesp;
	unsigned int uiEstructuraAuxiliar[2];
	uiEstructuraAuxiliar[1]= ((0xA00)*8)|3;	//ref.GDT
	asm volatile ("pushl %0" :: "m" (uiParametro4));
	asm volatile ("pushl %0" :: "m" (uiParametro3));
	asm volatile ("pushl %0" :: "m" (uiParametro2));
	asm volatile ("pushl %0" :: "m" (uiParametro1));
	asm volatile ("lcall *%0": :"m" (*uiEstructuraAuxiliar));;
}
```

void stl_ImprimirCadena (const char *cnstCadena, ...)
Parametros que recibe: const char *cnstCadena
Devuelve: no devuelve nada
Que hace: Imprime una cadena por pantalla, recibiendo argumentos variables

```
void stl_ImprimirCadena (const char *cnstCadena, ...)
{
	va_list (lista_argumentos);
	va_start (lista_argumentos, cnstCadena);
	int iCifras;
	int iCifrasDecimales; //Cantidad de decimales para imprimir flotantes
	while (*cnstCadena)
	{
		if (*cnstCadena == '%')
		{
			cnstCadena++;
			iCifras = 0; //Se parsea la cantidad de digitos pedida
			while ( *cnstCadena >= '0' && *cnstCadena <= '9')
			{
				iCifras = iCifras * 10 + *(cnstCadena) - '0'; //Actualmente se ignora
				cnstCadena++;
			}
			if( *(cnstCadena) == '.' ) //Se parsea la precision decimal
			{
				cnstCadena++;
		iCifrasDecimales = 0;
				while (*cnstCadena >= '0' && *cnstCadena <= '9')
				{
					iCifrasDecimales = iCifrasDecimales*10 + *(cnstCadena) - '0';
				cnstCadena++;
				}
			}
			else
		iCifrasDecimales = 6;	//Cantidad de decimales por defecto
			switch (*cnstCadena)
```

```
                                {
                                        case 'd': stl_Entero ((unsigned int) (va_arg(lista_argumentos, int))); break;
                                        case 'f': stl_Float ((unsigned int) (va_arg(lista_argumentos, double)), iCifrasDecimales);
break;
                                        case 'w': stl_Entero ((unsigned int) (va_arg(lista_argumentos, word))); break;
                                        case 'b': stl_Byte ((unsigned int) (va_arg(lista_argumentos, byte))); break;
                                        case 'x':
                                                cnstCadena++;
                                                if (*cnstCadena == 'w')
                                                        stl_WordHexa ((unsigned int) (va_arg (lista_argumentos, word)));
                                                else if (*cnstCadena == 'b')
                                                        stl_ByteHexa ((unsigned int) (va_arg (lista_argumentos, byte)));
                                                else
                                                {
                                                        cnstCadena--;
                                                        stl_Hexa ((unsigned int) (va_arg (lista_argumentos, int)));
                                                }
                                        break;
                                        case 's': stl_ImprimirCadena (va_arg (lista_argumentos, char *)); break;
                                }
                        }
                        else
                                stl_ImprimirCaracter ((unsigned int) (*cnstCadena));
                        cnstCadena++;
                }
                va_end (lista_argumentos);
}

void stl_Entero (unsigned int entero)
{
        stl_SaltarN0 (20, entero, 0, 0);
}

void stl_Float (unsigned int flotante, int iCifrasDecimales)
{
        stl_SaltarN0 (62, flotante, iCifrasDecimales, iCifrasDecimales);
}

void stl_Byte (unsigned int uiByte)
{
        stl_SaltarN0 (21, uiByte, 0, 0);
}

void stl_WordHexa (unsigned int uiWH)
{
        stl_SaltarN0 (22, uiWH, 0, 0);
}

void stl_ByteHexa (unsigned int uiBH)
{
        stl_SaltarN0 (23, uiBH, 0, 0);
}

void stl_Hexa (unsigned int uiH)
{
        stl_SaltarN0 (24, uiH, 0, 0);
}

void stl_ImprimirCaracter (unsigned int iCaracter)
{
        stl_SaltarN0 (13, iCaracter, 0, 0);
}
```

int stl_kill (unsigned int uiPos)
Parametros que recibe: unsigned int uiPos
Devuelve: devuelve 1 si fue exitoso
Que hace: hace kill de thread

```
int stl_kill (unsigned int uiPos)
{
        stl_SaltarN0 (5, uiPos, 0, 0);
        return 1;
}
```

void stl_pqueue_init (stl_pqueue_t *q)
Parametros que recibe: stl_pqueue_t *q
Devuelve: no devuelve nada
Que hace: inicializa la cola de prioridad

```
void stl_pqueue_init (stl_pqueue_t *q)
{
	if (q != NULL)
	{
	q->q_head = NULL;
	q->q_num  = 0;
	}
	return;
}
```

void stl_pqueue_insert (stl_pqueue_t *q, int prio, stl_t t)
Parametros que recibe: stl_pqueue_t *q, int prio, stl_t t
Devuelve: no devuelve nada
Que hace: inserta el thread de la cola de prioridad; O(n)

```
void stl_pqueue_insert (stl_pqueue_t *q, int prio, stl_t t)
{
	stl_t c;
	int p;
	if (q == NULL)
	return;
	if (q->q_head == NULL || q->q_num == 0)
	{
	/* agrega un primer elemento */
	t->q_prev = t;
	t->q_next = t;
	t->q_prio = prio;
	q->q_head = t;
	}
	else if (q->q_head->q_prio < prio)
	{
	/* agrega un nuevo tope de cola */
	t->q_prev = q->q_head->q_prev;
	t->q_next = q->q_head;
	t->q_prev->q_next = t;
	t->q_next->q_prev = t;
	t->q_prio = prio;
	t->q_next->q_prio = prio - t->q_next->q_prio;
	q->q_head = t;
	}
	else
	{
	/* inserta luego de elementos con igual o mayor prioridad */
	c = q->q_head;
	p = c->q_prio;
	while ((p - c->q_next->q_prio) >= prio && c->q_next != q->q_head)
		{
		c = c->q_next;
		p -= c->q_prio;
	}
	t->q_prev = c;
	t->q_next = c->q_next;
	t->q_prev->q_next = t;
	t->q_next->q_prev = t;
	t->q_prio = p - prio;
	if (t->q_next != q->q_head)
		t->q_next->q_prio -= t->q_prio;
	}
	q->q_num++;
	return;
}
```

void stl_pqueue_delete (stl_pqueue_t *q, stl_t t)
Parametros que recibe: stl_pqueue_t *q, stl_t t
Devuelve: no devuelve nada
Que hace: elimina el thread de la cola de prioridad; O(n)

```
void stl_pqueue_delete (stl_pqueue_t *q, stl_t t)
```

```
{
        if (q == NULL)
        return;
        if (q->q_head == NULL)
        return;
        else if (q->q_head == t)
        {
        if (t->q_next == t)
                {
                /* elimina el ultimo elemento y vacia la cola */
                t->q_next = NULL;
                t->q_prev = NULL;
                t->q_prio = 0;
                q->q_head = NULL;
                q->q_num  = 0;
        }
        else
                {
                /* elimina el tope de la cola */
                t->q_prev->q_next = t->q_next;
                t->q_next->q_prev = t->q_prev;
                t->q_next->q_prio = t->q_prio - t->q_next->q_prio;
                t->q_prio = 0;
                q->q_head = t->q_next;
                q->q_num--;
        }
        }
        else
        {
        t->q_prev->q_next = t->q_next;
        t->q_next->q_prev = t->q_prev;
        if (t->q_next != q->q_head)
                t->q_next->q_prio += t->q_prio;
        t->q_prio = 0;
        q->q_num--;
        }
        return;
}
```

stl_t stl_pqueue_tail (stl_pqueue_t *q)
Parametros que recibe: stl_pqueue_t *q
Devuelve: devuelve el último thread de la cola
Que hace: llega hasta el último thread de la cola

```
stl_t stl_pqueue_tail (stl_pqueue_t *q)
{
        if (q == NULL)
        return NULL;
        if (q->q_head == NULL)
        return NULL;
        return q->q_head->q_prev;
}
```

void stl_pqueue_increase (stl_pqueue_t *q)
Parametros que recibe: stl_pqueue_t *q
Devuelve: no devuelve nada
Que hace: incrementa la prioridad de todos los threads de la cola

```
void stl_pqueue_increase (stl_pqueue_t *q)
{
        if (q == NULL)
        return;
        if (q->q_head == NULL)
        return;
        q->q_head->q_prio += 1;
        return;
}
```

stl_t stl_pqueue_delmax (stl_pqueue_t *q)
Parametros que recibe: stl_pqueue_t *q
Devuelve: devuelve null o el nuevo tope de la cola
Que hace: remueve el thread de maxima prioridad de la cola; O(1)

```
stl_t stl_pqueue_delmax (stl_pqueue_t *q)
{
	stl_t t;
	if (q == NULL)
	return NULL;
	if (q->q_head == NULL)
	t = NULL;
	else if (q->q_head->q_next == q->q_head)
	{
	/* remueve el ultimo elemento y vacia la cola */
	t = q->q_head;
	t->q_next = NULL;
	t->q_prev = NULL;
	t->q_prio = 0;
	q->q_head = NULL;
	q->q_num  = 0;
	}
	else
	{
	/* remueve el tope de la cola */
	t = q->q_head;
	t->q_prev->q_next = t->q_next;
	t->q_next->q_prev = t->q_prev;
	t->q_next->q_prio = t->q_prio - t->q_next->q_prio;
	t->q_prio = 0;
	q->q_head = t->q_next;
	q->q_num--;
	}
	return t;
}
```

int stl_pqueue_favorite (stl_pqueue_t *q, stl_t t)
Parametros que recibe: stl_pqueue_t *q, stl_t t
Devuelve: devuelve un 1 o un -1 dependiendo de si pudo o no
Que hace: mueve un thread de la cola su tope; O(n)

```
int stl_pqueue_favorite (stl_pqueue_t *q, stl_t t)
{
	if (q == NULL)
	return -1;
	if (q->q_head == NULL || q->q_num == 0)
	return -1;
	/* el elemento ya esta en el tope */
	if (q->q_num == 1)
	return 1;
	/* mueve al tope */
	stl_pqueue_delete(q, t);
	stl_pqueue_insert(q, stl_pqueue_favorite_prio(q), t);
	return 1;
}
```

stl_t stl_pqueue_walk (stl_pqueue_t *q, stl_t t, int direction)
Parametros que recibe: stl_pqueue_t *q, stl_t t, int direction
Devuelve: devuelve una estructura stl_t de un thread
Que hace: va al thread anterior o siguiente de la cola; O(1)

```
stl_t stl_pqueue_walk (stl_pqueue_t *q, stl_t t, int direction)
{
	stl_t tn;
	if (q == NULL || t == NULL)
	return NULL;
	tn = NULL;
	if (direction == STL_WALK_PREV)
	{
	if (t != q->q_head)
		tn = t->q_prev;
	}
	else if (direction == STL_WALK_NEXT)
	{
	tn = t->q_next;
	if (tn == q->q_head)
		tn = NULL;
	}
	return tn;
```

```
}
```

int stl_pqueue_contains (stl_pqueue_t *q, stl_t t)
Parametros que recibe: stl_pqueue_t *q, stl_t t
Devuelve: devuelve un 1 o un -1 dependiendo de si lo encontro o no
Que hace: analiza si un thread pertenece o no a una cola O(n)

```
int stl_pqueue_contains (stl_pqueue_t *q, stl_t t)
{
        stl_t tc;
        int found;
        found = -1;
        for (tc = stl_pqueue_head(q); tc != NULL;
    tc = stl_pqueue_walk(q, tc, STL_WALK_NEXT))
        {
        if (tc == t)
                {
                found = 1;
                break;
        }
        }
        return found;
}
```

int stl_contar_threads (void)
Parametros que recibe: recibe void
Devuelve: devuelve un int con la cantidad de threads
Que hace: cuenta la cantidad d threads por cola de prioridad y los suma

```
int stl_contar_threads (void)
{
   int rc= 0;
   rc+= stl_pqueue_elements (&stl_NQ);
   rc+= stl_pqueue_elements (&stl_RQ);
   rc+= stl_pqueue_elements (&stl_WQ);
   rc+= stl_pqueue_elements (&stl_SQ);
   rc+= stl_pqueue_elements (&stl_DQ);
   rc+= 1;
   return rc;
}
```

unsigned long stl_solicitarCheckSum ()
Parametros que recibe: no recibe nada
Devuelve: devuelve un unsigned long con el checksum
Que hace: solicita y devuelve el checksum

```
unsigned long stl_solicitarCheckSum ()
{
        unsigned long ulRes;
        unsigned int esp, ds;
        asm volatile ("pushl %ds");
        asm volatile ("popl %0" : "=m" (ds));
        asm volatile ("movl %%esp, %0" :"=m" (esp));
        stl_SaltarN0 (83, ds, 0, esp);
        asm volatile ("movl (%1), %0" : "=a" (ulRes) : "b" (esp));
        return ulRes;
}
```

unsigned int stl_ObtenerHora ()
Parametros que recibe: no recibe nada
Devuelve: devuelve un unsigned int con la hora
Que hace: esta funcion se encargar de obtener la hora

```
unsigned int stl_ObtenerHora ()
{
        unsigned int uiHora, ds, esp;
        asm volatile ("pushl %ds");
        asm volatile ("popl %0" : "=m" (ds));
        asm volatile ("movl %%esp, %0" :"=m" (esp));
        stl_SaltarN0 (84, ds, 0, esp);
        asm volatile ("movl (%1), %0" : "=a" (uiHora) : "b" (esp));
        return uiHora;
```

```
}
```

Carpeta: usr/shell
Archivo: shell3.c

Declaraciones varias:

```
void vFnInstanciarAbin (int);
void vFnInstanciarBbin (int);
```

Carpeta: usr/shell
Archivo: shell3.c

void vFnShell()
Parametros que recibe: no recibe nada
Devuelve: no devuelve nada
Que hace: se fija que lo que ingresaste como comando sea un comando valido. Si lo es, llama a alguna funcion que lo ejecute, sino, te avisa que no es un comando

```
void vFnShell()
{
……
```

Se agregan algunas condiciones más:

```
else if (iComandoPos= (iFnEsCmdShell ("abin")) >= 0)
        {
                char stArg[5][16];
                int iArg1= 0, iN, iM;
                if ((iFnGetArg (iComandoPos, 1, stArg[0], 16) == 1) && (iFnGetArg (iComandoPos, 2, stArg[1], 16) == 1)
&& (iFnGetArg (iComandoPos, 3, stArg[2], 16) == 1) && (iFnGetArg (iComandoPos, 4, stArg[3], 16) == 1) && (iFnGetArg
(iComandoPos, 5, stArg[4], 16) == 1))
                {
                        iArg1 = iFnCtoi (stArg[0]);
                        vFnEnviarDigito (iArg1);
                        for (iM= 1; iM < 5; iM++)
                                for (iN= 0; iN < 16; iN++)
                                        vFnEnviarCaracter ((unsigned int) stArg[iM][iN], iN, iM-1);
                        vFnInstanciarAbin (iComandoPos);
                }
                else
                {
                        //CANT_SIMULTANEOS + 1 (NOMBRE DEL EJECUTABLE) + 1 (CHECKSUM)
                        vFnImprimirCadena ("Error: Se espera un valor de comprobacion y 4 nombres de archivo.\n");
                        vFnImprimirCadena ("Forma de uso: abin valor_de_comprobacion lista_de_archivos\n");
                }
        }
        else if (iComandoPos= (iFnEsCmdShell ("bbin")) >= 0)
        {
                char stArg[5][16];
                int iArg1= 0, iN, iM;
                if ((iFnGetArg (iComandoPos, 1, stArg[0], 16) == 1) && (iFnGetArg (iComandoPos, 2, stArg[1], 16) == 1)
&& (iFnGetArg (iComandoPos, 3, stArg[2], 16) == 1) && (iFnGetArg (iComandoPos, 4, stArg[3], 16) == 1) && (iFnGetArg
(iComandoPos, 5, stArg[4], 16) == 1))
                {
                        iArg1 = iFnCtoi (stArg[0]);
                        vFnEnviarDigito (iArg1);
                        for (iM= 1; iM < 5; iM++)
                                for (iN= 0; iN < 16; iN++)
                                        vFnEnviarCaracter ((unsigned int) stArg[iM][iN], iN, iM-1);
                        vFnInstanciarBbin (iComandoPos);
                }
                else
                {
                        //CANT_SIMULTANEOS + 1 (NOMBRE DEL EJECUTABLE) + 1 (CHECKSUM)
                        vFnImprimirCadena ("Error: Se espera un valor de comprobacion y 4 nombres de archivo.\n");
                        vFnImprimirCadena ("Forma de uso: bbin valor_de_comprobacion lista_de_archivos\n");
                }
        }
        else if (iComandoPos= (iFnEsCmdShell ("cbin")) >= 0)
                        vFnInstanciarCbin (iComandoPos);
```

```
……
}
```

void vFnInstanciarAbin (int iComandoPos)
Parametros que recibe: int iComandoPos
Devuelve: no devuelve nada
Que hace: instancia a a_bin

```
void vFnInstanciarAbin (int iComandoPos)
{
        vFnSaltarN0 (75, 0, 0, 0);
}
```

void vFnInstanciarBbin (int iComandoPos)
Parametros que recibe: int iComandoPos
Devuelve: no devuelve nada
Que hace: instancia a b_bin

```
void vFnInstanciarBbin (int iComandoPos)
{
        vFnSaltarN0 (76, 0, 0, 0);
}
```

void vFnInstanciarCbin (int iComandoPos)
Parametros que recibe: int iComandoPos
Devuelve: no devuelve nada
Que hace: instancia a c_bin

```
void vFnInstanciarCbin (int iComandoPos)
{
        vFnSaltarN0 (85, 0, 0, 0);
}
```

Conclusión

En la práctica realizada hemos notado que los archivos con llamadas al sistema bloqueantes han sido más eficientes que los que poseen llamadas no bloqueantes, en contraposición a lo que sucedia en el Trabajo Práctico 2 hecho en Linux. Esto debe a que la disketera es un dispositivo mucho mas lento que el disco rígido, por lo tanto el dato buscado no siempre va a estar disponible inmediatamente, las syscalls no bloqueantes requieren que esto sea así o de lo contrario cancelan la lectura y la vuelven a iniciar.

Al no estar el dato disponible en la mayoria de los casos, la syscall se debe llamar más veces, lo que baja el rendimiento de la lectura, por eso, en este caso una syscall que bloquea al hilo hasta que el dato esté disponible es más eficiente.

También creamos un binario bin_c, con el cual hemos utilizado para realizar una serie de pruebas sobre los hilos. La creación de este ejecutable nos ha hecho comprender más acerca de las ventajas que se poseen al trabajar con hilos a nivel de usuario.

Bibliografia

GNU Pth - The GNU Portable Threads
Copyright © 1999-2006 Ralf S. Engelschall <rse@gnu.org>

Teoria Hilos V2.pps
Apunte teórico brindado por la cátedra

Biblioteca de threads V3.pps
Apunte teórico brindado por la cátedra

Hilos_tips.doc
Apunte teórico brindado por la cátedra

Comando man del linux