# Necesidades:

- Una tabla, arreglo o estructura organizada y con acceso directo para llevar un control de los hilos. Los identificadores de los hilos son su posición en la estructura.
- Un thread tiene un identificador, una pila, una prioridad de ejecución y una dirección de inicio de la ejecución.
- Realizar un algoritmo que ordene los hilos según la planificación escogida.
- Estados permitidos por los hilos, por ejemplo:
    - Creación
    - Ejecución
    - Preparado
    - Bloqueado
    - Terminación
- Los threads de un proceso comparten su espacio de direcciones, pueden modificar variables globales, acceder a los descriptores de archivos abiertos o interferirse mutuamente de otras formas.
- Comparten un mismo **conjunto de recursos.**
- Se necesita tipar el identificador, así se obtendrá información de hilo
- Buscar las regiones criticas

# Funciones:

## my_thread_create

### pthread_create

*int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void *), void *arg);*

El *pthread_create()* rutina crea un nuevo hilo dentro de un proceso. El nuevo hilo se inicia en el *start_routine* la rutina de inicio que tiene un argumento arg de inicio. El nuevo hilo tiene atributos especificados con attr, o atributos predeterminados si attr es NULL.

Si la rutina *pthread_create()* tiene éxito se devuelve 0 y poner el nuevo thread id en hilo, de lo contrario un número de error se devolverá indicando el error.

Si los atributos de hilo objeto representado por attr se modifica más tarde, el hilo de nueva creación no se ve afectada. Si attr es NULL, se utilizan los atributos de hilo por defecto.

Con estas declaraciones y la inicialización:

```
pthread_t t;
void *foo(void *);
pthread_attr_t attr;
pthread_attr_init(&pta);
```

Los siguientes mecanismos de creación de tres de hilo son funcionalmente equivalentes:

```
rc = pthread_create(&t, NULL, foo, NULL);
rc = pthread_create(&t, &attr, foo, NULL);
```

### Parámetros

thread -> (Salida) de Pthread manejar al hilo creado.
attr -> (Entrada) Los atributos de hilo objeto que contiene los atributos que se asociará con el hilo recién creado. Si NULL se utilizan los atributos por defecto de hilo.
start_routine -> (Entrada) La función que se funciona como los nuevos hilos comienzan rutina.
arg -> (Entrada) Una dirección para el argumento de los hilos se inicia la rutina.

### Valor de retorno

0 -> *pthread_create()* se ha realizado correctamente.
valor -> *pthread_create()* no tuvo éxito. valor se establece para indicar la condición de error.

### Condiciones de error

Si *pthread_create()* no se ha realizado correctamente, el error devuelto por lo general indica uno de los siguientes errores. Bajo ciertas condiciones, el valor devuelto podría indicar un error distinto a los mencionados aquí.
[EINVAL] -> El valor especificado para el argumento no es correcto.
[EAGAIN] -> El sistema no tiene suficientes recursos para crear otro hilo o el número máximo de subprocesos de que se haya alcanzado este trabajo
[EBUSY] -> El sistema no puede permitir la creación del hilo en este proceso en este momento.

# my_thread_end

## pthread_exit

*void pthread_exit(void *status);*

El *pthread_exit()* rutina termina el hilo está ejecutando actualmente y hace de estado disponible para el hilo que se une con éxito, *pthread_join()*, con el hilo de terminación. En la adición de *pthread_exit()* ejecuta cualquier controlador de limpieza restantes en el orden inverso al que fueron empujados, *pthread_cleanup_push()*, después de lo cual todos los destructores específicas de hilo apropiado se llaman.

Una llamada implícita a *pthread_exit()* se hace si cualquier thread, que no sea el hilo en el que *main()* fue llamado primero, regresa de los la rutina de inicio especificados en *pthread_create()*. El valor de retorno ocupa el lugar del estado.

El proceso sale como si *exit()* se llama con un estado 0 si el hilo de terminación es el último hilo en el proceso.

El *pthread_exit()* rutina no puede regresar.

### Parámetros

status ->(Entrada) el código de salida del hilo.

### Authorities and locks

Ninguno.

### Valor de retorno
pthread_exit () no devuelve.

### Condiciones de error
Ninguno.

# my_thread_join

## pthread_join

*int pthread_join(pthread_t thread, void ** status);*

La función pthread_join () espera a que un hilo para terminar, se separa el hilo, luego devuelve el estado de las discusiones de salida.

En un llamada *pthread_join()* exitosa devolverá 0, y si el estado no es NULL entonces estado señalará el parámetro status de *pthread_exit()*. Si fracasa *pthread_join()* devolverá un número de error que indica el error.

### Parámetros
thread -> (Entrada) de Pthread manejar al subproceso de destino
estado -> (Salida) Dirección de la variable para recibir el código de salida del hilo.

### Authorities and locks
Ninguno.

### Valor de retorno
0 -> *pthread_join()* se ha realizado correctamente.
valor -> *pthread_join()* no tuvo éxito. valor se establece para indicar la condición de error.

### Condiciones de error
Si *pthread_join()* no se ha realizado correctamente, el error devuelto por lo general indica uno de los siguientes errores. Bajo ciertas condiciones, el valor devuelto podría indicar un error distinto a los mencionados aquí.
[EINVAL] -> El valor especificado para el argumento no es correcto.
[ESRCH] -> El hilo especificado no se pudo encontrar.

# my_thread_detach

## pthread_detach

*int pthread_detach(pthread_t thread);*

La función *pthread_detach()* indica que los recursos del sistema para el subproceso especificado deben ser reclamados cuando termina el hilo. Si el hilo ya se terminó, los recursos se recuperan inmediatamente. Esta rutina no causa que el hilo termine el final. Después *pthread_detach()* ha sido emitida, no es válida para tratar de *pthread_join()* con el subproceso de destino.

Finalmente, debe llamar *pthread_join()* o *pthread_detach()* para cada subproceso que se crea acoplable (con un detachstate de *PTHREAD_CREATE_JOINABLE*) para que el sistema pueda recuperar todos los recursos asociados con el hilo. El incumplimiento de unirse o separarse hilos acoplables se traducirá en pérdidas de memoria y otros recursos hasta que termine el proceso.

Si el hilo no representa un hilo undetached válida, *pthread_detach()* devolverá *ESRCH*.

### Parámetros
thread -> (Entrada) de Pthread manejar al subproceso de destino

### Autoridades y locks
Ninguno.

### Valor de retorno
0 -> pthread_detach () se ha realizado correctamente.
valor -> pthread_detach () no tuvo éxito. valor se establece para indicar la condición de error.

### Condiciones de error
Si *pthread_detach()* no se ha realizado correctamente, el error devuelto por lo general indica uno de los siguientes errores. Bajo ciertas condiciones, el valor devuelto podría indicar un error distinto a los mencionados aquí.
[EINVAL] -> El valor especificado para el argumento no es correcto.
[ESRCH] -> No se pudo encontrar el punto que coincide con el valor especificado

# my_thread_yield

## pthread_yield / sched_yield

*int pthread_yield(void); / int sched_yield(void);*

La función *pthread_yield()* produce el procesador a un subproceso ejecutable en cola en la misma prioridad que el hilo actual, si es que existe. Si hay varios, el hilo que ha estado esperando más tiempo se ejecutará. El hilo que invoca *pthread_yield()* sigue siendo ejecutable y se volverá a la cola al final de la lista de temas en espera de ejecución en el nivel de prioridad dado.

Si no hay hilos esperando para correr a la misma prioridad, *pthread_yield()* devuelve inmediatamente.

La función *sched_yield()* es idéntica a *pthread_yield()*.

### Parámetros
Ninguno.

### Autoridades y locks
Ninguno.

### Valor de retorno
0 -> *sched_yield()* se ha realizado correctamente.
valor -> *sched_yield()* no tuvo éxito. valor se establece para indicar la condición de error.

### Condiciones de error
El *sched_yield()* API actualmente no devuelve un error.

# my_mutex_init

## pthread_mutex_init

*int pthread_mutex_init(pthread_mutex_t * mutex, const pthread_mutex_attr *attr);*

La función *pthread_mutex_init()* inicializa un mutex con los atributos especificados para su uso. El nuevo mutex puede utilizarse inmediatamente para serializar los recursos críticos. Si se especifica attr como NULL, todos los atributos son los predeterminados atributos mutex son para el mutex de nueva creación.

Si la rutina *pthread_mutex_init()* tiene éxito se devuelve 0 y poner la nueva Identificación del mutex en mutex, de lo contrario un número de error se devolverá indicando el error.

Con estas declaraciones y la inicialización:

```
pthread_mutex_t      mutex2;
pthread_mutex_t      mutex3;
pthread_mutexattr_t  mta;
pthread_mutexattr_init(&mta);
```

Los siguientes mecanismos de inicialización tres mutex son funcionalmente equivalentes.

```
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_init(&mutex2, NULL);
pthread_mutex_init(&mutex3, &mta);
```

Las tres exclusiones mutuas son creados con los atributos mutex default.

### Parámetros
mutex -> (Entrada) La dirección de la variable para contener un objeto mutex.
attr -> (Entrada) La dirección de la variable que contiene el mutex atributos de los objetos.

### Autoridades y locks
Ninguno.

### Valor de retorno
0 -> pthread_mutex_init () se ha realizado correctamente.
valor -> pthread_mutex_init () no tuvo éxito. valor se establece para indicar la condición de error.

### Condiciones de error
Si pthread_mutex_init () no se ha realizado correctamente, el error devuelto por lo general indica uno de los siguientes errores. Bajo ciertas condiciones, el valor devuelto podría indicar un error distinto a los mencionados aquí.
[EINVAL] -> El valor especificado para el argumento no es correcto.
[ENOMEM] -> El sistema no puede asignar los recursos necesarios para crear el mutex.

# my_mutex_destroy

## pthread_mutex_destroy

*int pthread_mutex_destroy(pthread_mutex_t *mutex);*

La función *pthread_mutex_destroy()* destruye el mutex llamado. El mutex destruido ya no puede ser utilizado.

Si *pthread_mutex_destroy()* se llama en un mutex que está bloqueado por otro hilo, la petición fallará con un error *EBUSY*. Si el subproceso de la llamada tiene el mutex bloqueado, otros subprocesos en espera para la exclusión mutua a través de una llamada a pthread_mutex_lock() en el momento de la llamada a *pthread_mutex_destroy()* fallará con el error *EDESTROYED*.

### Parámetros

mutex -> (Entrada) Dirección del mutex que será destruido

### Autoridades y locks

Ninguno.

### Valor de retorno

0 -> pthread_mutex_destroy () se ha realizado correctamente.
valor -> pthread_mutex_destroy () no tuvo éxito. valor se establece para indicar la condición de error.

### Condiciones de error

Si pthread_mutex_destroy () no se ha realizado correctamente, el error devuelto por lo general indica uno de los siguientes errores. Bajo ciertas condiciones, el valor devuelto podría indicar un error distinto a los mencionados aquí.
[EBUSY] -> El mutex es actualmente propiedad de otro hilo.
[EINVAL] -> El valor especificado para el argumento no es correcto.

# my_mutex_lock

## pthread_mutex_lock

*int pthread_mutex_lock(pthread_mutex_t *mutex);*

La función *pthread_mutex_lock()* adquiere la propiedad del mutex especificado. Si el mutex está actualmente bloqueado por otro subproceso, la llamada a *pthread_mutex_lock()* se bloqueará hasta que el hilo renuncia a la propiedad a través de una llamada a *pthread_mutex_unlock()*.

Si una señal se entrega a un hilo, mientras que el hilo está esperando un mutex, cuando el manejador devuelve, la espera se reanudará. *pthread_mutex_lock()* no devolverá EINTR como algunas otras llamadas a la función de bloqueo.

La destrucción de un mutex retenido es una forma común para serializar la destrucción de los objetos que están protegidos por ese mutex, y está permitido. La llamada a

*pthread_mutex_lock()* puede fallar con el error *EDESTROYED* si el mutex es destruido por el hilo que actualmente se sostenía.

### Tipos de exclusión mutua

Un mutex normal no puede ser bloqueado en varias ocasiones por el propietario. Los intentos de un hilo Para volver a bloquear un mutex que ya posea, o para bloquear un mutex que se celebró por otro subproceso cuando el thread finaliza resultado en una condición de interbloqueo.

Un mutex recursivo se puede bloquear en varias ocasiones por el propietario. La exclusión mutua no se convierta desbloqueado hasta que el dueño ha llamado pthread_mutex_unlock () para cada solicitud de bloqueo exitoso que tiene en circulación en el mutex.

Un *errorcheck* mutex comprueba para condiciones de estancamiento que se producen cuando un hilo re-bloquea un mutex que ya posea. Si un hilo intenta volver a bloquear un mutex que ya tiene, los solicitud de bloqueo se produce el error *EDEADLK*

### Parámetros

mutex -> (Entrada) La dirección de la exclusión mutua para bloquear

### Autoridades y locks

Ninguno.

### Valor de retorno

0 -> *pthread_mutex_lock()* se ha realizado correctamente.
valor -> *pthread_mutex_lock()* no tuvo éxito. valor se establece para indicar la condición de error.

### Condiciones de error

Si *pthread_mutex_lock()* no se ha realizado correctamente, el error devuelto por lo general indica uno de los siguientes errores. Bajo ciertas condiciones, el valor devuelto podría indicar un error distinto a los mencionados aquí.
[EINVAL] -> El valor especificado para el argumento no es correcto.
[EDESTROYED] -> Mientras se espera el bloqueo mutex que deben cumplir, el mutex fue destruido.
[EOWNERTERM] -> Un hilo terminado en la celebración de la exclusión mutua, y el mutex es un tipo mutex ownerterm.
[EDEADLK] -> Un hilo trató de volver a bloquear un mutex que ya posea, y el mutex es un tipo mutex errorcheck.
[ERECURSE] -> El mutex recursivo no puede ser bloqueado de forma recursiva de nuevo.

# my_mutex_unlock

## pthread_mutex_unlock

*int pthread_mutex_unlock(pthread_mutex_t *mutex);*

Si el subproceso actual es el propietario de la exclusión mutua especificada por mutex, entonces la rutina *pthread_mutex_unlock()* deberá desbloquear el mutex. Si hay algunos hilos bloqueados esperando el mutex, los planificador determinará qué flujo obtiene la cerradura de la

exclusión mutua, de lo contrario el mutex está disponible para el siguiente subproceso que llama a la rutina *pthread_mutex_lock()*, o *pthread_mutex_trylock()*.

### Parámetros
mutex -> (Entrada) Dirección de la exclusión mutua para desbloquear

### Autoridades y Locks
Para completar con éxito, el bloqueo mutex debe celebrarse antes de llamar *pthread_mutex_unlock()*.

### Valor de retorno
0 -> pthread_mutex_unlock () se ha realizado correctamente.
valor -> pthread_mutex_unlock () no tuvo éxito. valor se establece para indicar la condición de error.

### Condiciones de error
Si pthread_mutex_unlock () no se ha realizado correctamente, el error devuelto por lo general indica uno de los siguientes errores. Bajo ciertas condiciones, el valor devuelto podría indicar un error distinto a los mencionados aquí.
[EINVAL] -> El valor especificado para el argumento no es correcto.
[EPERM] -> La exclusión mutua no está actualmente en manos de la persona que llama

# my_mutex_trylock

## pthread_mutex_trylock

*int pthread_mutex_trylock(pthread_mutex_t *mutex);*
El *pthread_mutex_trylock()* rutina deberá bloquear el mutex especificado y retorno 0, de lo contrario un número de error se devolverá indicando el error. En todos los casos la rutina *pthread_mutex_trylock()* no bloqueará el hilo actual en ejecución.

Un fallo de *EDEADLK* indica que el mutex ya se lleva a cabo por el subproceso de llamada.

### Parámetros
mutex -> (Entrada) Dirección de la exclusión mutua para bloquear

### Autoridades y Locks
Ninguno.

### Valor de retorno
0 -> pthread_mutex_trylock () se ha realizado correctamente.
valor -> pthread_mutex_trylock () no tuvo éxito. valor se establece para indicar la condición de error.

### Condiciones de error
Si pthread_mutex_trylock () no se ha realizado correctamente, el error devuelto por lo general indica uno de los siguientes errores. Bajo ciertas condiciones, el valor devuelto podría indicar un error distinto a los mencionados aquí.
[EINVAL] -> El valor especificado para el argumento no es correcto.

[EBUSY] -> El mutex está actualmente bloqueado por otro hilo. Un hilo terminado en la celebración de la exclusión mutua, y el mutex es un tipo mutex ownerterm. Un hilo trató de volver a bloquear un mutex que ya posea, y el mutex es un tipo mutex errorcheck.
[ERECURSE] -> El mutex recursivo no puede ser bloqueado de forma recursiva de nuevo.

# Preguntas

1. Como manejar la memoria? Una introducción o explicación de lo que busca con el manejo de memoria.
2. Si puede recapitular lo de los mutex y explicar cómo espera q os implementamos

# Diseño de un paquete de hilos

Los objetivos de diseño que deben prevalecer son :

- Planificación preemptiva: Políticas de planificación como Round-Robin y señales o eventos asíncronos junto con prioridades, sólo pueden ser soportados mediante un diseño de kernel preemptivo.
- Cambios de contexto rápidos: El cambio de contexto debe ser la única causa por la cual el control es transferido de un hilo a otro, de forma que se debe intentar reducir la sobrecarga del cambio de contexto.
- Impedir el crecimiento de pila ilimitado: Si se produce un evento asíncrono mientras se ejecuta un gestor de interrupciones, otro gestor puede ser introducido en la pila y así hasta el infinito. Este caso se debe evitar mediante alguna solución particular.