Centro de
Investigación en
Computación

Pthreads

Hilos POSIX

POSIX
Threads

Course

Operating System (with focus on Security)

Instructor

Acosta Bermejo Raúl

Lecture notes

2024-B
7 de noviembre del 2024
Última actualizacioón

Instituto
Politécnico
Nacional

Table of contents (outline)

Tabla de contenido

1. Motivación
2. Introducción
3. Pthreads en Linux
4. Administración
5. Sincronización
6. Pthreads-safe
7. PTTT

Introducción

Pthreads (POSIX Threads)

POSIX es el acrónimo de **P**ortable **O**perating **S**ystem **I**nterface; la **X** viene de UNIX como seña de identidad de la API.

El término fue sugerido por Richard Stallman en respuesta a la demanda de la IEEE, que buscaba un nombre fácil de recordar.

Una traducción aproximada del acrónimo podría ser "Interfaz de sistema operativo portable".

Introducción

POSIX

Son una familia de estándares de llamadas al sistema operativo definidos por el IEEE y especificados formalmente en el **IEEE 1003**. Persiguen generalizar las interfaces de los sistemas operativos para que una misma aplicación pueda ejecutarse en distintas plataformas.

Especifica las interfaces de usuario y software al sistema operativo en 15 documentos diferentes:

- La **línea de comandos** (shell) estándar y las interfaces de *scripting* se basaron en Korn Shell. Otros programas a nivel de usuario (*user-level*), servicios y utilidades incluyen AWK, echo, ed y cientos de otras.
- Los servicios a nivel de programa requeridos incluyen definición de estándares básicos de **I/O**, (file, terminal, y servicios de red).
- También especifican una API para las bibliotecas de **threading**, que es muy utilizada en una gran variedad de sistemas operativos.

Introducción

Versiones y partes de POSIX

- **POSIX.1**, Core Services (implementa las llamadas del ANSI C estándar). Incluye:
  1) Creación y control de procesos, 2) Señales, 3) Excepciones de punto flotante, 4) Excepciones por violación de segmento, 5) Excepciones por instrucción ilegal, 6) Errores del bus., 7) Temporizadores, 8) Operaciones de ficheros y directorios (sobre cualquier fs montado), 9) Tuberías (*Pipes*), 10) Biblioteca C (Standard C), 11) Instrucciones de entrada/salida y de control de dispositivo (ioctl).
- **POSIX.1b**, extensiones para tiempo real:
  1) Planificación (*scheduling*) con prioridad, 2) Señales de tiempo real, 3) Temporizadores, 4) Semáforos, 5) Intercambio de mensajes (*message passing*), 6) Memoria compartida, 7) Entrada/salida síncrona y asíncrona, 8) Bloqueos de memoria.
- **POSIX.1c**, extensiones para hilos (*threads*):
  1) Creación, control y limpieza de hilos, 2) Planificación (*scheduling*), 3) Sincronización, 4) Manejo de señales.
- **POSIX.2**, Shell y Utilidades (IEEE Std 1003.2-1992)
  1) Intérprete de Comandos, 2) Programas de Utilidad
  Luego de 1997 el Grupo Austin realizó modificaciones a POSIX.
  Las especificaciones tienen el nombre de **Single Unix Specification** (Especificación Única de Unix)
- **POSIX:2001 o IEEE Std 1003.1-2001** equivale a la versión 3 de Single UNIX Specification.
  - Las base de definiciones, Tema 6.
  - Las interfaces y encabezamientos del sistema, Tema 6.
  - Los comandos y utilidades, Tema 6.
- **POSIX:2004 o IEEE Std 1003.1-2004** implica una pequeña actualización de POSIX:2001. Tiene dos correcciones técnicas de errores. Para más información sobre este estándar visitar: http://www.opengroup.org/onlinepubs/009695399/
- A partir de 2009 **POSIX:2008 o IEEE Std 1003.1-2008** representa la versión actual.
  - La base de definiciones, Tema 7,
  - Las interfaces encabezamientos del sistema, Tema 7.
  - Los comandos y utilidades, Tema 7.

Introducción

Compatibilidad POSIX

Los siguientes Sistemas Operativos son 100% compatibles con uno o varios estándares POSIX.

GNU/Linux,
A/UX,
AIX,
BSD/OS,
HP-UX,
INTEGRITY,
IRIX,
LynxOS

Mac OS X v10.5 en Procesadores Intel,
MINIX,
MPE/iX,
QNX (IEEE Std. 1003.13-2003 PSE52),
RTEMS (POSIX 1003.1-2003 Profile 52),
Solaris, OpenSolaris,
UnixWare,
velOSity,
VxWorks (IEEE Std. 1003.13-2003 PSE52)

Windows NT y posteriores: XP, Vista, 7

Pthreads en linux

Introducción

En Linux se han tenido dos implementaciones:

LinuxThreads Es la implementación original de Pthreads. Desde glibc 2.4, esta implementación ya no es soportada.

NPTL (*Native POSIX Threads Library*)

- Esta es la implementación moderna de Pthreads que en comparación con la anterior se apega mejor a los requerimientos de POSIX.1 y ofrece un mejor rendimiento cuando se crea un número grande de hilos.
- Está disponible desde la glibc 2.3.2, y requiere características que están presentes en Linux 2.6 kernel.

Pthreads en linux

Introducción

- Ambas implementaciones son llamadas 1:1 debido a que cada hilo tiene una representación directa y única en el kernel (planificador).
- Ambas implementaciones emplean la llamada al sistema de Linux denominada **clone**.
- En NPTL, las primitivas de sincronización de hilos (mutexes, thread joining, etc.) son implementadas usando la llamada al sistema de Linux denominada **futex**.
- A partir de glibc 2.3.2, el comando **getconf** se puede utilizar para determinar la implementación de hilos de la que se dispone:

```
$ getconf -a
GNU_LIBPTHREAD_VERSION  NPTL 2.17
```

Pthreads en linux

Introducción

- Debido a que la llamada al sistema **clone** es básica en implementación de los hilos es importante conocerla mejor ya que permite “jugar” con la implementación de los hilos.
- También hay que tomar en cuenta que esta función es única de Linux y no de otros sistemas operativos tipo UNIX como el de las Mac OS.
- Al igual que la función rfork de FreeBSD y la función sproc de IRIX, la función clone de Linux fue inspirada por la función rfork de Plan 9.

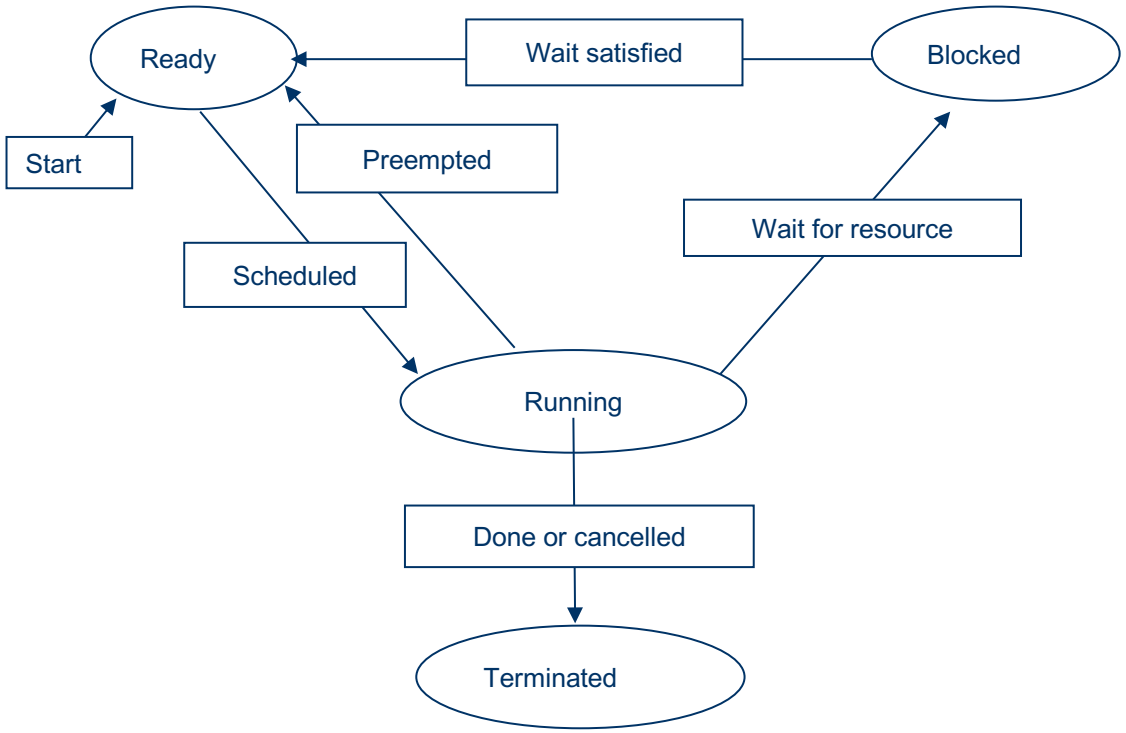
Tareas

- Optativa: Investigar como funciona la función rfork de Plan 9.
- Obligatoria: Lo mismo para clone (buscar el código).

Pthreads en linux

Diagrama de estados

Es un conjunto de funciones que permiten la creación y ejecución de múltiples hilos en C/C++ para realizar múltiples tareas de manera simultanea.



Pthreads en linux

Funciones y datos

Todas las funciones de hilos POSIX thread tienen la forma:

pthread[_object] _operation

La mayoría de las funciones de la librería POSIX de hilos regresa cero en caso de éxito y un valor diferente de cero en caso de error.

Compilación

- Se debe usar la biblioteca, `#include <pthread.h>` en los programas.
- Se debe compilar y ligar con la biblioteca pthread:

```
$ gcc -o prog prog.c -lpthread
```

PThreads

API POSIX (primitivas)

Las funciones POSIX de threads se pueden clasificar en los siguientes grupos :

- Thread Routines
- Attribute Object Routines
- Mutex Routines
- Condition Variable Routines
- Read/Write Lock Routines
- Per-Thread Context Routines
- Cleanup Routines

Pthreads en linux

API

Grupo	Prefijo de Rutina	Función
1	Pthread_	Hilos y rutinas diversas
2	Pthread_attr_	Atributos de los hilos
3	Pthread_mutex	Mutex (semáforos)
4	Pthread_mutexattr_	Atributos de los semáforos
5	Pthread_cond_	Variables condicionales
6	Pthread_condattr_	Atributos de las Var. Cond.
7	Pthread_key_	Llaves de los hilos
8	Pthread_rwlock_	Operaciones de lectura/escritura
9	Pthread_barrier_	Barreras

Ejemplos del API

1

pthread_create
pthread_cancel
pthread_detach
pthread_equal
pthread_exit
pthread_kill
pthread_join
pthread_self
pthread_yield

2

pthread_attr_destroy
pthread_attr_init
pthread_attr_getdetachstate
pthread_attr_setdetachstate
pthread_attr_getguardsize
pthread_attr_setguardsize
pthread_attr_getstack
pthread_attr_setstack
pthread_attr_getinheritsched
pthread_attr_setinheritsched
pthread_attr_getschedparam
pthread_attr_setschedparam
pthread_attr_getschedpolicy
pthread_attr_setschedpolicy
pthread_attr_getscope
pthread_attr_setscope

3

pthread_mutex_init()
pthread_mutex_destroy()
pthread_mutex_lock()
pthread_mutex_trylock()
pthread_mutex_unlock()

4

pthread_mutexattr_gettype
pthread_mutexattr_settype
pthread_mutexattr_getprotocol
pthread_mutexattr_setprotocol

pthread_setconcurrency()
pthread_getconcurrency()
pthread_setschedparam

sched_get_priority_max
sched_set_priority_min

5

pthread_cond_destroy( )
pthread_cond_init( )
pthread_cond_broadcast( )
pthread_cond_signal( )
pthread_cond_timedwait()
pthread_cond_wait()

Referencias

https://computing.llnl.gov/tutorials/pthreads/

Pthreads en Linux

Funciones y datos

1. Administración de Procesos
   - Creación, terminación
   - Atributos (pthreads & mutex):
     - detach state,
     - stack size,
     - stack addr,
     - cancel state,
     - cancel type,
     - get/set sched policy and param,
     - Inheritedsched
     - Priority aware mutexes,
     - get/set protocol,
     - prioceiling
2. Mecanismos de Sincronización
   - Join
   - Mutex (blocking)
   - Variables condicionales

Administración

pthread_create (1)

Semántica

Crea un hilo de tal forma que : 1) Cada uno tiene su propio código+pila (variables locales), 2) Comparten los datos (variables globales).
Después de creado se hace ejecutable (ready). Un hilo puede crear otros hilos y en estos no hay jerarquía o dependencia.

Parámetros

pthread_t* id

Regresa el identificador del hilo para poderlo manipular.

const pthread_attr_t *attributes

Define los atributos de creación del hilo. Hay varios posibles, como por ejemplo la prioridad (se ejecuta primero y con más tiempo). La estructura es inicializada con otras funciones como pthread_attr_init. Si es **NULL**, el hilo se crea con sus atributos por defecto.

void *(*thread_function)()

Un apuntador a la función (código) que ejecuta el hilo.

void * arguments

Un apuntador a una variable o estructura de datos que se pasa como parámetro del hilo.

Administración

pthread_create (2)

Valores de retorno

int Si no hay error regresa 0, en caso contrario regresa:

1.[EAGAIN] El sistema:
- No tiene los recursos necesarios para crear otro hilo, o
- Impone un límite en el número total de hilos o procesos. Este caso ocurre cuando se alcanza el límite:
  1. De recursos por software (RLIMIT_NPROC, usando setrlimit). Se limita el número de procesos de un usuario, o
  2. Del número de hilos que el kernel permite (*/proc/sys/kernel/threads-max*, PTHREAD_THREADS_MAX).

2.[EINVAL] El valor especificado en *attr* es invalido.

3.[EPERM] El que invoca la función no tiene los permisos apropiados para configurar los parámetros del planificador solicitados o la política del planificador seleccionada.

Administración

pthread_create

Herencia

El nuevo hilo hereda:

1. Una copia de la máscara de señales del creador (pthread_sigmask).
2. El conjunto de señales pendientes para el nuevo hilo esta vacío (sigpending).
3. El nuevo hilo no hereda una pila de señales alterna (sigaltstack).
4. El nuevo hilo hereda el ambiente de punto-flotante el invocador (fenv).

```
Pthread_t pthread self(void)
```

Semántica

Regresa el identificador del hilo actual.

Administración

Terminación

Un proceso termina cuando:

1. Llama directamente a exit.
2. Uno de sus hilos llama a exit.
3. Regresa de la función main()
4. Recibe la señal de terminación.

En cualquiera de estos casos, TODOS los hilos de un proceso terminan.

Un hilo termina cuando:

1. Regresa de la función que lo define.
2. Se invoca la función pthread_exit. Los demás hilos no terminan.
3. El hilo es cancelado por otro mediante pthread_cancel.

Administración

pthread_cancel

```
int pthread_cancel(pthread_t tid)
```

Semántica

Manda una solicitud de cancelación al hilo con identificador *tid*.

1. Cuando y si el hilo reacciona a la solicitud de cancelación depende de dos atributos que son controlados por el hilo mismo: ***state*** y ***type***.
2. El estado cancelable de un hilo se configura con pthread_setcalcelstate, el cual puede ser *enabled* (valor por default de un nuevo hilo) o *disabled*.
3. Si un hilo tiene desabilitada las cancelaciones, una solicitud permanece encolada hasta que el hilo habilita las cancelaciones.
4. Si un hilo habilita las cancelaciones, entonces el *type* determina cuando ocurrirá la cancelación, la cual pueder ser *asynchronous* o *deferred* (valor por default de un nuevo hilo).
   - Asynchronous significa que el hilo puede ser cancelado en cualquier momento (normalmente de forma inmediata, pero el sistema no garantiza esto).
   - Deferred significa que la cancelación será retrasada hasta que el hilo invoque a una función que sea ***punto de cancelación***. POSIX especifica la lista de funciones que son o pueden ser puntos de cancelación.

Administración

Ejemplo 1: pthread_create

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5
void *PrintHello(void *threadid)
{
 long tid;
 tid = (long)threadid;
 printf("Hello World! It's me, thread #%ld!\n", tid);
 pthread_exit(NULL);
}

int main (int argc, char *argv[])
{
  pthread_t threads[NUM_THREADS];
  int rc;
  long t;
  for(t=0; t<NUM_THREADS; t++){
    printf("In main: creating thread %ld\n", t);
    rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
    if (rc){
      printf("ERROR; return code from pthread_create() is %d\n", rc);
      exit(-1);
    }
  }
  pthread_exit(NULL);
}
```


Administración

Ejemplo 1: pthread_create, salida

```
In main: creating thread 0
In main: creating thread 1
Hello World! It's me, thread #0!
In main: creating thread 2
Hello World! It's me, thread #1!
Hello World! It's me, thread #2!
In main: creating thread 3
In main: creating thread 4
Hello World! It's me, thread #3!
Hello World! It's me, thread #4!
```

Administración

Ejemplo 2: parámetros

```
struct thread_data
{
  int thread_id;
  int sum;
  char *message;
};

struct thread_data thread_data_array[NUM_THREADS];

void *PrintHello(void *threadarg)
{
  struct thread_data *my_data;
  ...
  my_data = (struct thread_data *) threadarg;
  taskid = my_data->thread_id;
  sum = my_data->sum;
  hello_msg = my_data->message;
   ...
}

int main (int argc, char *argv[])
{
   ...
  thread_data_array[t].thread_id = t; thread_data_array[t].sum = sum;
  thread_data_array[t].message = messages[t];
   rc =  pthread_create(&threads[t], NULL, PrintHello, (void *) &thread_data_array[t]);
   ...
}
```

Administración

Ejemplo 2: parámetros, salida

```
Creating thread 0
Creating thread 1
Creating thread 2
Creating thread 3
Creating thread 4
Creating thread 5
Creating thread 6
Creating thread 7
Thread 0: English: Hello World! Sum=0
Thread 1: French: Bonjour, le monde! Sum=1
Thread 2: Spanish: Hola al mundo Sum=3
Thread 3: Klingon: Nuq neH! Sum=6
Thread 4: German: Guten Tag, Welt! Sum=10
Thread 5: Russian: Zdravstvytye, mir! Sum=15
Thread 6: Japan: Sekai e konnichiwa! Sum=21
Thread 7: Latin: Orbis, te saluto! Sum=28
```

Sincronización

Join

Es el mecanismo más básico de sincronización para que un padre espere a que uno de sus hijos termine (proceso/hilo).

Se usa la función **join** que existe en varios lenguajes o librerias. En Pthread se usa:

```
int pthread_join(pthread_t id, void** status);
```

Semántica

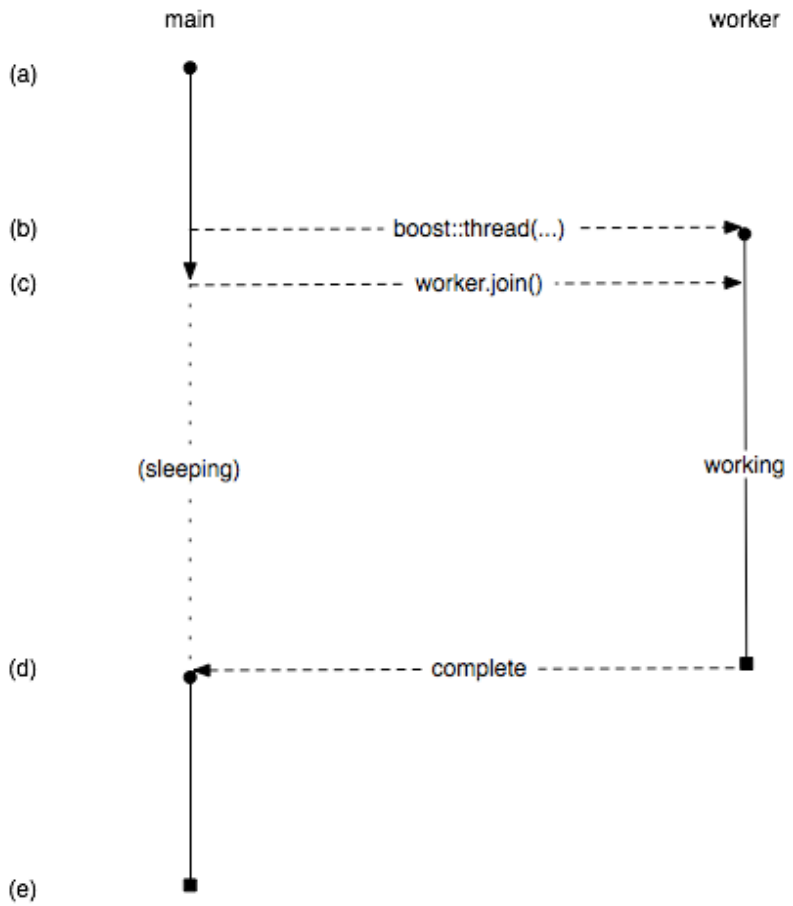
La subrutina pthread_join() bloquea al hilo que la invoca hasta que el hilo especificado (id) termina.

Parámetros

- id es el identificador del hilo al que se desea unir (join).
- status contiene el valor que regresa el hilo que termina mediante la función pthread exit().

Sincronización

Join



Sincronización

Join

Valores de regreso

Regresa cero en caso de éxito, y sino un positivo indicando el tipo de error.

Errores

La función fallará si:

[EINVAL] La implementación detectó que el valor especificado por el hilo (tid) no hace referencia a un hilo *joinable*.

[ESRCH] El hilo cuyo identificador se dio, no pudo ser encontrado.

La funcción podría fallar si:

[EDEADLK] Un deadlock fue detectado o el identificador del hilo es el mismo del que realizó la invocación.

[EINTR] Nunca se regresa un código de error de este tipo.

Sincronización

Detach

Indica que no se está interesado en el valor que regresa un hilo.

- Por default se guarda el valor para todos los hilos.
- Una vez terminado el hilo, se borran todos los datos del hilo y así se tiene más espacio para crear otros hilos.
- Sólo los hilos creados como *joinable* se puede usar la función **join**.
- Si un hilo es creado como *detached*, nunca puede esperarse a que termine su ejecución con **join**.

Sincronización

Ejemplo 3: Join

```
void *BusyWork(void *t)
{
  int i;
  long tid;
  double result=0.0;
  tid = (long)t;
  printf("Thread %ld starting...\n",tid);
  for (i=0; i<1000000; i++) {
     result = result + sin(i) * tan(i);
   }
   printf("Thread %ld done. Result = %e\n",tid, result);
   pthread_exit((void*) t);
}

int main (int argc, char *argv[])
{
  pthread_t thread[NUM_THREADS];
  pthread_attr_t attr;
  int rc;
  long t;
  void *status; /* Initialize and set thread detached attribute */
```

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define NUM_THREADS 4
```

Sincronización

Ejemplo 3: Join

```
  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
  for(t=0; t<NUM_THREADS; t++) {
    printf("Main: creating thread %ld\n", t);
    rc = pthread_create(&thread[t], &attr, BusyWork, (void *)t);
    if (rc) {
      printf("ERROR; return code from pthread_create() is %d\n", rc);
      exit(-1);
    }
  }

  /* Free attribute and wait for the other threads */
  pthread_attr_destroy(&attr);
  for(t=0; t<NUM_THREADS; t++) {
      rc = pthread_join(thread[t], &status);
      if (rc) {
        printf("ERROR; return code from pthread_join() is %d\n", rc);
        exit(-1);
       }
       printf("Main: completed join with thread %ld having a status of %ld\n",t,(long)status);
    }
    printf("Main: program completed. Exiting.\n");
    pthread_exit(NULL);
}
```

Sincronización

Ejemplo 3: Join, salida

Salida del programa

```
Main: creating thread 0
Main: creating thread 1
Thread 0 starting...
Main: creating thread 2
Thread 1 starting...
Main: creating thread 3
Thread 2 starting...
Thread 3 starting...
Thread 1 done. Result = -3.153838e+06
Thread 0 done. Result = -3.153838e+06
Main: completed join with thread 0 having a status of 0
Main: completed join with thread 1 having a status of 1
Thread 3 done. Result = -3.153838e+06
Thread 2 done. Result = -3.153838e+06
Main: completed join with thread 2 having a status of 2
Main: completed join with thread 3 having a status of 3
Main: program completed. Exiting.
```

Sincronización

Yield

Provoca que el hilo que invoca a pthread_yiled() ceda el control al planificador, es decir, este es colocado en la cola de planificador y otro hilo es ejecutado.

Esta función no tiene parámetros y regresa:

- Cero si no hay error y
- En caso contrario un valor positivo

  En Linux no hay errores conocidos.

Sincronización

Suspend / Resume

- When a thread executes a thread suspend to suspend the execution of itself or another thread, the indicated thread will be suspended ***until*** the execution of a thread resume that releases the indicated thread.
- Both thread yield and thread suspend cause the execution of a thread to be suspended. What is the difference?
  - With thread yield, the yielding thread is put back to the ***ready*** queue and will run when its turn comes. Thus, a yielding thread is runnable if the CPU becomes free in the future, although it is suspended.
  - With a thread suspend, the suspended thread is ***not*** in the ready queue, and, as a result, the scheduler will not be able to pick it up and let it run when the CPU becomes free. Instead, the execution of a suspended thread can be resumed only by a specific thread resume call.
- Thread suspend/resume can:
  - Be very useful. For example, this may be more efficient that creating a new thread to handle the task and then terminating the thread.
  - Post some problem. May lead to a system deadlock, the use of thread suspend and resume is usually not recommended.
- Some systems such as the Pthread do not support thread suspend and resume.

Threads

Tiempos de ejecución

```
#include <sys/times.h>

   clock_t times(struct tms *buffer);

strcut tms
{
   clock_t tms_utime;   Tmpo Modo Usuario (TMU)
   clock_t tms_stime;   Tmpo Modo Supervisor (TMS)
   clock_t tms_cutime;  TMU de los hijos
   clock_t tms_cstime;  TMS de los hijos
}
```

Regresa 4 tiempos de ejecución que no consideran el tiempo dedicado por los procesos del sistema a los procesos del usuario.

clock_t Contabiliza los pulsos de reloj. Se tienen CLK_TCK pulsos por segundo.

El comando time regresa estos tiempos. Verificar lo anterior haciendo un programa

$ time programa

real 0m3.28s user 0m2.28s sys 0m1.00s

Thread-safe functions

Re-entrantes

Son funciones que pueden ser ejecutadas por varios hilos al mismo tiempo de tal forma que el resultado que entrega la función es el mismo (con los mismos datos de entrada) sin importar el hilo que la ejecute.

POSIX.1-2001 y POSIX.1-2008 especifica que todas las funciones especificadas en el estandar deben ser ***thread-safe***, excepto por las funciones siguientes:

asctime() basename() catgets() crypt() ctermid() if passed a non-NULL argument ctime() dbm_clearerr() dbm_close() dbm_delete() dbm_error() dbm_fetch() dbm_firstkey() dbm_nextkey() dbm_open() dbm_store() dirname() dlerror() drand48() ecvt() [POSIX.1-2001 only (function removed in POSIX.1-2008)] encrypt() endgrent() endpwent() endutxent() fcvt() [POSIX.1-2001 only (function removed in POSIX.1-2008)] ftw() gcvt() [POSIX.1-2001 only (function removed in POSIX.1-2008)] getc_unlocked() getchar_unlocked() getdate() getenv() getgrent() getgrgid() getgrnam() gethostbyaddr() [POSIX.1-2001 only (function removed in POSIX.1-2008)] gethostbyname() [POSIX.1-2001 only (function removed in POSIX.1-2008)] gethostent() getlogin() getnetbyaddr() getnetbyname() getnetent() getopt() getprotobyname() getprotobynumber() getprotoent() getpwent() getpwnam() getpwuid() getservbyname() getservbyport() getservent() getutxent() getutxid() getutxline() gmtime() hcreate() hdestroy() hsearch() inet_ntoa() l64a() lgamma() lgammaf() lgammal() localeconv() localtime() lrand48() mrand48() nftw() nl_langinfo() ptsname() putc_unlocked() putchar_unlocked() putenv() pututxline() rand() readdir() setenv() setgrent() setkey() setpwent() setutxent() strerror() strsignal() [Added in POSIX.1-2008] strtok() system() [Added in POSIX.1-2008] tmpnam() if passed a non-NULL argument ttyname() unsetenv() wcrtomb() if its final argument is NULL wcsrtombs() if its final argument is NULL wcstombs() wctomb()

Se garantiza esta característica mediante el uso de: 1) semáforos, o 2) Ambientes de ejecución diferentes (pilas y/o variables).

POSIX Thread Trace Toolkit

PTTT

- Es una aplicación muti-hilos que puede ser depurada (trace) sin ser recompilada.
- Una traza es analizada una vez que la aplicación es detenida, es decir, el **análisis es post-mortem**.
- Se tienen 3 formatos de trazas:
  - Formato texto para ser leído por personas (*human readable*).
  - Formato texto para ser leído por una computadora. Este formato se parsea fácilmente por otros programas para extraer información útil.
  - Formato gráfico que se basa en una herramienta de visualización interactiva (disponible en Debian).

POSIX Thread Trace Toolkit

PTTT

Ofrece las siguientes características:

- **trace cut**: it is possible to only work on a part of the trace, selected with temporal or numeric criteria.
- **trace split**: it is possible to split the trace into several files to get either one file per process or one file per thread.
- **trace filtering**: it is possible to filter the trace on various criteria, such as event names, object names, kind of objects or pid.
- **log levels**: it is possible to dynamically switch from light to richer or full trace.
- **continuous recording**: the tool can keep only last traces of the execution.
- handle **large volume** of traces.
- handle **bad situations** (hang, crash, kill).
- **thread contention analysis** (most contented threads and NPTL objects).

The end

Contacto

Raúl Acosta Bermejo

http:www.cic.ipn.mx
http://www.ciseg.cic.ipn.mx/

racostab@ipn.mx
racosta@cic.ipn.mx

57-29-60-00
Ext. 56652