Centro de
Investigación en
Computación

Threads

Introducción a los Hilos

Conceptos básicos

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

Introduction

Bibliografía

Los de cabecera:

1. El entorno de programación de UNIX, Brian Kernighan, Prentice Hall.
2. *Advanced Unix programming*, Marc Rochkind, Prentice Hall.
3. Unix programación avanzada, Francisco Marquez, Adison Wesley Iberoamericana. Didáctico y bueno para programar.

Muy completos, aunque no siempre muy didácticos:

- *Advanced Programming in the UNIX Environment*, W. Richard Stevens.
- *UNIX network Programming, APIs: Sockets and XTI*, W. Richard Stevens.

Introduction

Artículos

Una selección de artículos a leer son:

- *On Designing Lightweight Threads for Substrate Software*
  Matthew Haines, University of Wyoming
  - Proceedings of the USENIX 1997 Annual Technical Conference Anaheim, California, January 6-10, 1997.

Introduction

Estándares

Institute for Electrical and Electronic Engineers (***IEEE***)

Portable Operating System (**POSIX**) Estandares de SO desarrollados por IEEE.
Ejemplos de estandares:

1003.1 (1988) La interfaz de programación
1003.2 Shells y herramientas
1003.7 Administración del sistema

1003.1 Especifica la interfaz y no la implementación.
No hay diferencia entre System Call y Funciones de libreria.

International Organisation for Standardization (**ISO**)

American National Standars Institute (**ANSI**)
ANSI C (X3.159, 1989)

Introduction

Llamadas al sistema (formato y errores)

```
int nombre (parámetros);
```

En general al realizar *una Llamada al Sistema* el kernel regresa un valor positivo si no hay error y el valor -1 si hubo un error.

El tipo de error es almacenado en la variable `errno` y se puede imprimir con la función:

```
perror();
```

Threads

Hilos

Threads

Hilos

Los *Threads*, o *Lightweight Process* facilitan la comunicación entre procesos. El segmento de datos es el mismo.

Proceso

Datos
Código
Pila

Proceso

Datos				
Código	Código	...	Código	Código
Pila	Pila	...	Pila	Pila

Hilo Principal

Hilo Hijos

Threads

Hilos

Ventajas de los hilos sobre los procesos

- Ocupan menos espacio en memoria:
  - Tanto el segmento de Código como el de Pila de los hilos no contienen información inútil heredada del padre, son mas pequeños.
  - Los hilos usan el mismo segmento de Datos así que no se reserva memoria extra.
- Son más rápidos:
  - La creación toma menos tiempo ya que no se copia la información del segmento de Código y el de Datos.
  - La destrucción o terminación de los hilos también es más rápida liberando menos área de memoria.

Threads

Hilos

Antecedentes

- Existieron varias librerías para implemetar los hilos.
- Las primeras librerias se implementaron como aplicaciones, es decir, en el espacio de usuario.
  - La librería más conocida es la implementación **POSIX** conocida como ***Pthreads***.
  - Los Pthreas están incluidos en Linux y en varios de los SO modernos.

Threads

En el espacio de usuario

El kernel no sabe de su existencia. Ve un proceso con un hilo.

Ventajas

- El cambio de contexto entre hilos es más rápido que las invocaciones al kernel.
- Cada proceso puede tener su propio scheduler.
- Escalan mejor que hilos en el kernel. Tablas y pilas pueden ocupar mucho espacio.
- No pagan el precio de la verificación de la seguridad.
- Se usan en SO sin hilos.

Desventajas

Al ver procesos y no hilos:

- Las llamadas al sistema con bloqueo hechas por un hilo bloquean a los demás. Ejemplo el bloqueo por Fallo de página, Pipe vacio, etc.
- En general los sistemas bloquean al proceso. *Linux: Un handler por proceso.*
- Se complica la programación, probando variables y haciendo espera (`select`).
- El cambio de contexto no se hace por timer (bloquedas las interrupciones del timer). Sistemas Cooperativos.

Threads

En el espacio del kernel

El kernel tiene una tabla para los hilos con su PC, SP, etc.

Ventajas

- Bloqueo selectivo de hilos.
- Planificador preemptivo.
- No requieren nuevas llamadas al sistema para la espera ocupada.

Desventajas

La mayoría de las primitivas son *Llamadas al Sistema* que son más costosas, ejemplo :

- Las llamadas que bloquean hilos (semáforos).
- Creación y destrucción de hilos. Se usa reciclaje de hilos.
- Llamadas que realizan la verificación de la seguridad.
- Muchos procedimientos de biblioteca son no-reentrantes.

Tarea

Leer las Activaciones del Planificador. Anderson 91, Tanenbaum pag 182-184.

Pthreads

Hilos POSIX

En Pthreads, las funciones inician con pthread_, por ejemplo la primitiva de creación :

```
int pthread_create(pthread_t *tid,
                   const pthread_attr_t *attr,
                   void *(*func) (void*),
                   void *arg);
```

```
#include <pthread.h>
```

Cada thread tiene su propio ID (`tid`), registros (PC, SP), mascara de señales, etc.

Varias de sus características, como tamaño de la pila, se especifican como atributos (`attr`).

El código del thread es un apuntador a una función (`func`) con sus parametros (`arg`).

Implementación: Hilos en el espacio de Usuario o Kernel.

Threads

Ejemplo

```
#include <stdio.h>
#include <pthread.h>

void thread(void);

int main (int argc, char **argv)
{
 int pid;
 pthread_t tid;

   pid = pthread_create(&tid, NULL, (void *)&thread, NULL);
   if( pid < 0 ){
     printf("Error en la creacion del Thread\n");
     return 1;
   }

   while(1){
     printf("\tP=%d\n", getpid() );
   }

}

void thread(void)
{
  while(1){
    printf("\t\t\t H=%d\n", getpid());
  }
}
```

En Linux

```
$ gcc -pthread -o p p.c
$ man pthread_create
$ man pthread_attr_init
```

Threads

En Java

Los Threads en Java se implementan con los servicios de concurrencia que ofrezca el Sistema Operativo donde se ejecute la JVM.

Significa que un programa con Threads se comporta de forma diferente según la implementación del SO, es decir, del Scheduler.

En un SO con un Scheduler cooperativo (Win 3.1, MacOS 9) se debe utilizar explicitamente la instrucción `yield` para que el sistema evolucione.
También se usa `Thread.sleep(5)` o cambiar la prioridad con `setPriority(Thread.NORM_PRIORITY-1);`

En Java existen, por default, varios Threads que controlan los recursos más comunes como el desplegado gráfico. Esto significa que el programador debe de considerar los problemas de concurrencia.

Threads

En Java

En Java hay dos formas de crear Threads:

```
class Hilo extends Thread
{
    public void run()
    {
     // Código del Thread
    }
}
```

La ejecución inicia cuando una instancia del hilo llama a **`start()`**.

Labo

Prácticas

Labo

Prácticas

1. Hacer un programa que cree dos proceso y después de la creación todos los procesos impriman, en forma iterativa y en columnas, su Id. Ver como se realiza el cambio de contexto, Cada cuando se realiza? Se puede medir ese tiempo? Se puede asumir que el orden de creación es el orden de ejecución?, Se pueden suspender, de alguna forma , los procesos?, Si se mata uno que hacer para que se recorran las columnas? Analizar los procesos con comandos como *ps* (ver Windows).

2. Crear un programa que encuentre el número máximo de procesos que puede crear un usuario. Recuerde que cuando inicia una sesión en sistemas Unix ya se existen varios procesos. Cuantos se pueden creer en el sistema? Es constante o se calcula?

3. Hacer un programa que verifique que cuando se crea un proceso, este último hereda una copia de los datos del padre (variables del ambiente, apuntadores a archivos, variables globales, variables locales).

Labo

Prácticas

5. Sin utilizar (aunque pudieran ya conocerse) las técnicas que se verán en las unidades 3 y 4, implemente el *Problema de los Filósofos*. y el de *Los Lectores Escritores*. Como modelaron las actividades (comer, tener hambre, filosofar)?, y los tenedores? Una vez implementado, que tan fácil se puede modificar para agregar más filósofos? Como se sabe si implemento bien (liveness, deadlock)?

6. Repetir las prácticas de Procesos con Threads.

7. Programar unos de los problemas anteriores en Java

8. Utilizar los mecanismos de sincronización y comunicación
   Semáforos, Monitores, Colas, Segmentos de Memoria, etc.

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