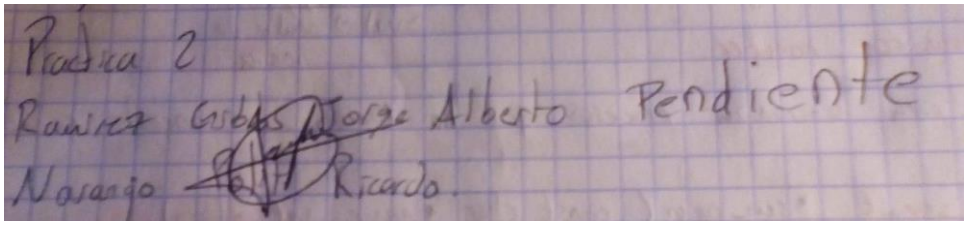
Instituto Politécnico Nacional

Escuela Superior de Cómputo

Práctica #2
Semáforos



Fecha de entrega: 20/09/2018

Maestro: Rangel González Josué

Materia: “Aplicaciones para comunicaciones de red”

Grupo: 3CV8

Equipo:

Naranjo Polit Ricardo
Ramirez Gibbs Jorge Alberto

Introducción

Un proceso es el resultado de una técnica que emplean los sistemas operativos para permitir la ejecución simultánea de distintas aplicaciones. Esta técnica consiste en dejar correr a una aplicación por un tiempo, digamos 10 ms. Cuando se agota el tiempo, el kernel del SO retoma el control y se lo entrega a otra aplicación. Al mismo tiempo que el SO cambia el control de una aplicación a otra, también intercambia, por cada aplicación, información adicional tal como:

- Qué archivos están abiertos.
- La memoria de la aplicación.
- La pila de ejecución.

O sea, al salto se le suma el intercambio de información de contexto. Y a la unidad compuesta por toda esa información contextual se la conoce como *proceso*.

El concepto de thread implica solamente saltar, pero mantener el contexto. Es decir que distintas partes de la aplicación se ejecutan concurrentemente, compartiendo memoria y archivos abiertos (la pila de ejecución no se intercambia).

Cuando Linux era jovencito, no tenía threads. Los demás sistemas operativos se pavoneaban exponiendo complejos mecanismos de threads. En estos sistemas (como también es hoy en Windows), los threads son conceptos de primer nivel en el sistema. Es decir que están implementados como algo especial y fijo.

Mientras tanto Linux crecía y crecía, pero solamente tenía procesos. Las primeras versiones de Java tenían que, en Linux, simular la existencia de los threads. Linus Torvalds, el creador de Linux, siempre pensó que los threads eran algo muy parecido a los procesos, tan parecidos que debían ser lo mismo. Mientras que un proceso nace de un fork "completo", los threads vendrían a ser el resultado de un fork parcial, en el que, en vez de duplicar todo el contexto, el proceso hijo mantenga parte de él compartido con el padre. Entonces se creó una llamada al sistema que permite decirla al sistema cuáles partes del contexto se desean duplicar, esta llamada es **clone()**.

Sin embargo, la forma estándar de manipular threads en UNIX (conocida como pthreads) no era del todo compatible con esta conceptualización. Por ejemplo, espera que cada thread vea el mismo "identificador de proceso", y con el mecanismo de Linux eso no era posible, ya que cada thread era un proceso distinto.

A través de los años Linux fue añadiendo funciones para dar mejor soporte a una API pthreads. La implementación actual de pthreads (conocida como NPTL) llegó actualmente a ofrecer el manejo exacto que ofrecen otros sistemas.

Veamos cómo es esa API...

La función principal es obviamente la que permite crear un thread. Esta función es pthread_create:

```
int pthread_create(pthread_t * thread, pthread_attr_t * attr, void *(*funcion)(void *), void *arg);
```

Los parámetros son los siguientes:

thread
Un puntero en donde la función guardará el identificador del nuevo thread creado.

attr
Atributos del thread, puede ser NULL.

funcion
Acá empieza la ejecución del nuevo thread. Es un puntero a una función declarada de esta manera: void mi_thread(void *arg);.

arg
Parámetro a pasar a la función del nuevo thread

Esta función devuelve el control inmediatamente... mientras tanto, el nuevo thread inicia su recorrido por la función apuntada por funcion. ¿Hasta cuándo? Como mucho hasta que termina ese método, cuando sale, termina el thread. Si un thread necesita esperar a que otro termina (por ejemplo el thread padre esperar a que termine el hijo) puede usar la función pthread_join(). ¿Por qué se llama así? Bueno, crear un proceso es como una bifuración, se abren dos caminos... que uno espere a otro es lo contrario, una unificación (join en inglés).

Semáforos System V

Los semáforos son una herramienta básica de los sistemas operativos multitarea, permiten gestionar el acceso sincronizado (exclusión mutua) a la región crítica por parte de múltiples procesos. Desde la época en que fueron desarrollados por Dijkstra han evolucionado desde ser una variable con dos funciones asociadas hasta lo que son hoy; un mecanismo de conjuntos de semáforos conformado por estructuras de datos, system calls y código de usuario.

Cuando por la lógica de trabajo se utilizan dos o más recursos compartidos, pueden presentarse problemas de bloqueos cruzados, también llamados Deadlock (abrazo mortal). Para evitar este conflicto los semáforos se implementan mediante conjuntos, en lugar de entidades independientes. Donde a cada recurso compartido se le asigna un semáforo. Es obvio que puede implementarse si es suficiente un conjunto con un único semáforo. También es imprescindible que las funciones asociadas (system calls) sean “atómicas”. Esta característica les permite llevar a cabo TODA la tarea en el conjunto sin que en medio haya detenciones por medio del planificador de la CPU. Si no fuera así, por más que las operaciones se realicen por conjuntos la gestión sería inconsistente.

Los semáforos pueden ser inicializados con cualquier valor entero positivo (este valor indicará la cantidad de recursos disponibles). A veces es necesario utilizar un semáforo que

como valor máximo admita un uno (1), a estos se los denomina “semáforos binarios”.

El tipo de semáforos que se utilizará en este trabajo es la implementación IPC UNIX System Vdesarrollada por la empresa de comunicaciones AT&T. Las herramientas IPC(Inter Process Communication) además de los semáforos son las colas de mensajes y las memorias compartidas. Estas herramientas son muy útiles a la hora de desarrollar aplicaciones que respondan al modelo CLIENTE-SERVIDOR.

Descripción del problema

Escriba un programa que tenga 4 productores y 3 consumidores. Existen 2 secciones críticas, cada una está dividida en 5, la primera sección guarda números de teléfono que son únicos por cada productor, en la segunda zona critica se guardan mensajes y solo se accede a esta zona al producir un número en la zona de teléfonos.

Descripción de la solución

Se ocupan 7 hilos; 4 para los productores y 3 para los consumidores. La solución propuesta se realiza con la técnica de semáforos cruzados, la cual necesita dos semáforos por cada división de la zona crítica y otros dos semáforos por cada zona critica, lo cual nos da un total de 24 semáforos.

Los semáforos de los productores de las zonas críticas contendrán el valor 5 puesto que son capaces de contener 5 divisiones, los semáforos de cada división de los productores tendrán valor de 1 y los otros semáforos que corresponden a los consumidores se inician en 0.

```
//inicia el semaforo
semctl(semid,0,SETVAL,5);
semctl(semid,1,SETVAL,0);
semctl(semid,2,SETVAL,5);
semctl(semid,3,SETVAL,0);
for(int i = 4; i < 9; i++)
  semctl(semid,i,SETVAL,1);
for(int i = 9; i < 14; i++)
  semctl(semid,i,SETVAL,0);
for(int i = 14; i < 19; i++)
  semctl(semid,i,SETVAL,1);
for(int i = 19; i < 24; i++)
  semctl(semid,i,SETVAL,0);
```

Los productores primero tratarán de cerrar el semáforo de la primera sección critica, y después tratarán de entrar en alguna división de ésta, si logran entrar escribirán su número de teléfono, al terminar abrirán el semáforo del consumidor que corresponde a la división y al semáforo de la primera sección critica de los consumidores.

```
while(cierre(0)==-1){}
for(int j = 0; j < 5; j++){
  if((semctl(semid,j+4,GETVAL,1))!=0){
    cierre(j+4);
    if(i==iteraciones){
      secCritTel[j]=-1;
      printf("Produce: -1 %d\n",*lugar+1);
    }else{
      secCritTel[j] = 0;
      for(int x = 1; x < 1000000; x *= 10)
        secCritTel[j]+=(*lugar+1)*x;
      printf("Produce: %d\n",secCritTel[j]);
    }
    apertura(j+9);
    break;
  }
}
apertura(1);
```

Al terminar de escribir en la primera sección, tratará de escribir en la segunda y repetir el proceso anterior, pero con la nueva sección.

```
cierre(2);
for(int j = 0; j < 5; j++){
  if((semctl(semid,j+14,GETVAL,1))!=0){
    cierre(j+14);
    if(i==iteraciones){
      secCritMen[j]=-1;
      printf("Produce: Mensaje %d %d\n",secCritMen[j],*lugar+1);
    }else{
      secCritMen[j] = (*lugar+1);
      printf("Produce: Mensaje %d\n",secCritMen[j]);
    }
    apertura(j+19);
    break;
  }
}
apertura(3);
```

Una cosa a notar es que cuando se llegan el límite de producciones a realizar el productor escribirá un valor específico para avisar a los consumidores que dejará de escribir en las secciones críticas.

Los consumidores se encuentran dentro de un ciclo que terminará cuando el conteo de señales terminales de los productores llegue al número de productores en cuestión, en otras palabras, terminará cuando todos los productores hayan avisado que no producirán más.

El consumidor primero comprueba que pueda el semáforo tenga un valor positivo para poder ingresar a la primera sección critica, después ingresa a dicha sección y cierra la sección, posteriormente se buscará que alguna división contenga producciones a ser consumidas, cuando encuentre una división a consumir, la consume y sale de la sección

crítica y desbloquea el semáforo de los productores para que puedan volver a escribir en esa división y sección respectivamente.

```
while(cont<8){// Consumos en la seccion critica
  if((semctl(semid,1,GETVAL,1))!=0){
    cierre(1);
    for(int j = 0; j < 5; j++){
      if((semctl(semid,j+9,GETVAL,1))!=0){
      cierre(j+9);
      printf("Consume: %d\n",secCritTel[j]);
      if(secCritTel[j]==-1){
        cont++;
      }
      apertura(j+4);
      break;
    }
  }
  apertura(0);
```

Posteriormente se realiza la comprobación de la segunda sección critica a consumir y de sus divisiones, utilizando el mismo método que la primera sección.

```
if((semctl(semid,3,GETVAL,1))!=0){
  cierre(3);
  for(int j = 0; j < 5; j++){
    if((semctl(semid,j+19,GETVAL,1))!=0){
    cierre(j+19);
    printf("Consume: Mensaje %d\n",secCritMen[j]);
    if(secCritMen[j]==-1){
      cont++;
    }
    apertura(j+14);
    break;
  }
}
apertura(2);
}
```

Es necesario notar que, en las secciones anteriores de código, el consumidor está comprobando la existencia de la señalización de que termina algún productor para incrementar el contador de los productores que han terminado y así poder salir del ciclo.

Conclusiones

Ricardo Naranjo Polit:

En esta práctica vimos que el número de semáforos necesarios para la administración de las secciones críticas en hilos puede llegar a ser alto puesto que

depende del número de las secciones y del método utilizado para administrarlas, en este caso el método utilizado fue el cruzado.

Es necesario tener mucho cuidado al entender la lógica de los semáforos puesto que puede ser complicada de entender, uno de los problemas que se están teniendo al realizar la práctica fue que los consumidores se quedaban esperando que se desbloquearan sus semáforos aun cuando todos los productores terminaron su producción, lo cual generaba una especie de deadlock, por lo cual fue necesario poner una condición para evitar este caso de situaciones.

Ramírez Gibbs Jorge Alberto

Es necesario considerar varias técnicas de uso de semáforos antes de implementarlos en un programa ya que creo puede determinar la complejidad , la carga de trabajo de la implementación y los recursos que se utilicen.
En este caso tuvimos que emplear más semáforos que en ocasiones anteriores y eso hizo un poco más tediosa la programación debido a que podemos tener errores en la ejecución si no tenemos en mente el funcionamiento de todo el sistema.

La manera de señalizar a los productores y a los consumidores, así como el cierre y apertura de la zona crítica son cruciales para que no caigamos en inconsistencia de datos, ni deadlocks.