Sincronizacion de procesos

Sobre los procesos cooperantes:

- Pueden compartir espacios de direcciones o datos a través de un archivo.
- Problema a considerar:
  - Como evitar la inconsistencia de los datos compartidos
  - Como acceder a espacios critico de código compartido.

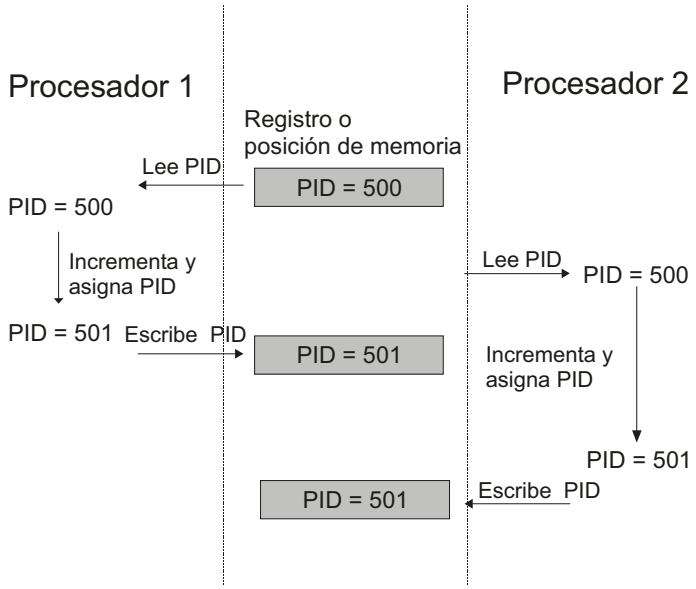
Alternativas de sincronización

- Semáforos
- Monitores
- Paso de mensajes

Sección crítica (definición)

- Sean un conjunto de procesos cooperantes. Cada proceso tiene un segmento de código en el cual puede modificar variables comunes, o un archivo, o una tabla.
- Llamamos sección crítica (SC) a ese segmento de código.

Para ilustrar este problema podemos ver el problema presentado por Carretero Pérez en la sección 5.2.1



Exclusión mutua

- Cuando un proceso esta ejecutando ese segmento de código crítico, ningún otro proceso puede ejecutarlo.
- La ejecución de la sección crítica es mutuamente exclusiva en el tiempo.

Solución al problema de la sección crítica

Condiciones para la solución:

- Exclusión mutua
- Progresión: que ordenadamente todos los procesos puedan ejecutarse y entrar en la SC.
- Espera limitada: una vez que requirió entrar a la SC, que pueda hacerlo después de un tiempo determinado.

La solución se implementa a través de diferentes algoritmos

Aplicación del protocolo

- Las instrucciones de maquina (load, test, load) se ejecutan atómicamente.
- La solución para múltiples procesos la da el Algoritmo del panadero (bakery algorithm).

Semáforos

Es una herramienta de sincronización

- Sirve para solucionar el problema de la sección crítica.
- Sirve para solucionar problemas de sincronización.

Implementación

- Es una variable entera sujeta a dos operaciones: wait y signal
- Estas operaciones se ejecutan de manera indivisible.
- Cuando un proceso modifica el valor del semáforo, otros procesos no pueden modificarlo simultáneamente.
- Se inicializa con valor no negativo
- Wait decrementa el valor del semáforo. Si el valor se hace negativo, el proceso se bloquea.
- Signal incrementa el valor del semáforo. Si el valor no es positivo, se desbloquea un proceso bloqueado por un wait.

Esquema de la SC con semáforos

```
Repeat
        Entry section
                critical section
        Exit section
                Remainder section
Until  false;
```

Ejemplo

Sean 2 procesos:

- A, que quiere ejecutar la sentencia X
- B, que quiere ejecutar la sentencia Y.
- Sem, es la variable común semáforo, inicializada en 0.

Pero:... Y se debe ejecutar después que X

Solución

- En proceso A:

```
        X;
Signal(sem);
```

- En proceso B:

```
Wait(sem);
Y;
```

Alternativas en la espera

- Busy waiting: gasta CPU, ejecuta un loop hasta poder entrar a SC.
- Autobloqueo: cuando el proceso ve que tiene que esperar, se bloquea.
  - Se pone en una cola asociada con el semáforo.
  - Se rearranca por un wakeup cuando se ejecuta el signal en los procesos en SC.

Monitores

- Es una construcción de alto nivel para sincronización.
- Es mas fácil de controlar que los semáforos.
- Se implementan como biblioteca de programas.
- Es un modulo de soft con 1 o + procedimientos, una secuencia de inicialización y datos locales.
- Ejemplos: Una estructura de datos compartida, puedo ponerla dentro de un monitor

Características básicas

- Variables de datos locales solo se acceden a través de los procedimientos del monitor
- Un proceso entra al monitor invocando uno de sus procedimientos.
- Solo un proceso puede estar en el monitor en un momento dado (ofrece exclusión mutua)

Sintaxis de un monitor

```
Tipo nombre-monitor = monitor;
Declaración de variables
Procedure entry P1(...);
Begin...end;
....
Procedure entry Pn(...);
Begin...end;
Begin
Código inicialización
End.
```

Importante destacar!

- Asegura que solo un proceso a la vez puede estar activo dentro del monitor.
- El programador no necesita codificar explícitamente.
- Se agrega el constructor condition para sincronización.

Interacción por pase de mensajes

- Se implementa por las primitivas send y receive
- El proceso emisor (E, sender) envía información (mensaje) al receptor (R, receiver).

- Ventaja: se puede implementar en sistemas distribuidos, en multiprocesador, y mono con memoria compartida.

Características de sincronización

- Envío bloqueante, recepción bloqueante. El E y el R se bloquean hasta que se entrega el mensaje (Rendezvous).
- Envío no bloqueante, recepcion bloqueante. El E puede continuar, pero R se bloquea hasta que llega el mensaje.
- Envío no bloqueante, recepción no bloqueante. Nadie espera
- El send no bloqueante es la forma mas útil en programación concurrente. Cuidado con la generación de mensajes excesiva. El programador debe controlar la recepción.
- El Envío no bloqueante, recepción bloqueante es útil para procesos servidores que ofrezcan servicio o recursos a otros procesos.
- En cuanto al tipo de comunicación fue visto en el apunte anterior de procesos

Deadlocks o Interbloqueos o Bloqueos mutuos o Abrazo mortal

Una funcion del SO: DISTRIBUCION DE RECURSOS

- Recursos fisicos: CPU, memoria, dispositivos.
- Recursos logicos: archivos, semaforos y monitores
- Cada recurso puede tener instancias identicas (puede haber 2 impresoras del mismo tipo)
- Si son identicas, se puede asignar cualquier instancia del recurso
- **Definicion de clase: es el conjunto de instancias de un recurso**

Pautas de asignacion

- No se pueden asignar mas recursos que los que hay.
- El proceso debe solicitar un recurso antes de usarlo.
- El proceso debe liberar un recurso despues de usarlo.
- Secuencia: solicitud, uso, liberacion

System calls, estructuras y operaciones asociadas

- System calls:
  - Request-release device
  - Open file-Close file
  - Allocate –free memory
- Operaciones wait y signal
- Tablas para registro de uso de recursos

Deadlock (abrazo mortal o interbloqueo)

- Definicion: Un conjunto de procesos estan en deadlock cuando cada uno de ellos esta esperando por un recurso que esta siendo usado por otro proceso del conjunto
- Un estado de deadlock puede involucrar recursos de diferentes tipos.

Ejemplo

Supongamos un sistema con un scanner y una impresora.
Un proceso A esta usando el scanner y necesita la impresora.
  - Un proceso B esta usando la impresora y necesita el scanner.
  - Resultado: deadlock

Condiciones para que se cumpla deadlock

- Exclusion mutua
- Retencion y espera
- No apropiacion
- Espera circular
  - Se deben dar simultaneamente y mantenerse

- No son totalmente independientes: la espera circular depende de retencion y espera.

Exclusion mutua

- Un recurso solo puede usarlo un proceso a la vez: es no compartible.
- Si otro proceso requiere el recurso tiene que esperar a que el proceso que lo tiene lo libere

Retencion y esperar

- Debe haber un proceso que retenga por lo menos un recurso y que este esperando un recurso que tiene otro proceso retenido.

No apropiacion

- El recurso debe ser liberado voluntariamente por el proceso que lo retiene cuando completa su tarea.

Espera circular

- Los proceso P_1 , P_2 ,P_3 ,...,P_n... son procesos esperando recursos.
- P_1 espera un recurso que retiene P_2
- P_2 espera un recurso que tiene P_3,
- ...
- P_n espera un recurso que tiene P_1

Grafos de asignacion de recursos

- Sirven para graficar el deadlock
- V, conjunto de vértices. Pueden ser procesos o recursos.
- Procesos: Representados por círculos.
- Recursos: Representados por rectángulos.
- Puntos: representan instancias de recursos.
- Un arco entre el proceso P_i y el recurso R_j indica que P_i solicito una instancia del recurso R_j (arco de requerimiento)
- Un arco entre el recurso R_j y el proceso P_i indica que el recurso R_j fue asignado al proceso P_i (arco de asignación)
- Si el grafo no contiene ciclos $\Rightarrow$ no hay deadlock
- Si el grafo contiene ciclos $\Rightarrow$ PUEDE SER que no hay deadlock

Métodos para el tratamiento del deadlock

Hay 3 métodos

Usar un protocolo que asegure que NUNCA se entrara en estado de deadlock

- Permitir el estado de deadlock y luego recuperar
- Ignorar el problema y esperar que nunca haya un deadlock

Prevención: el "nunca ocurrirá"

- Método que asegura que por lo menos una de las condiciones no pueda mantenerse.
- Analicemos cada condición

Prevención: La exclusión mutua

- Considerar que hay recursos compartibles y no compartibles.
- Mantener la exclusión mutua para los no compartibles

Prevención: Retención y espera

- Garantía: Si un proceso requiere un recurso, debe liberar otros.
- Alternativas:
  - Un proceso debe requerir y reservar todos los recursos a usar antes de comenzar la ejecución (precedencia de los system calls que hacer requerimiento antes de cualquier otro system call)
  - El proceso puede requerir procesos solo cuando no tiene ninguno.
- **Desventajas**
  - Baja utilización de recursos
  - Posibilidad de inanición de alguno de los procesos (starvation, o espera infinita)

Prevención: No apropiación

- Si un recurso no puede asignarse a un proceso y queda en wait, se liberan el resto de los recursos (apropiación). El proceso esperara ahora por todos sus recursos.
- Se aplica normalmente a recursos tales como ciclos de CPU y espacio de memoria que pueden ser salvados y restablecidos luego.

Prevención: Espera circular

- Imponer un ordenamiento de los recursos: un proceso puede requerir recursos en un orden numérico ascendente.
- Sea $F:R \rightarrow N$, N conjunto de los naturales.
- F asigna un numero único a cada recurso (los números pequeños para recursos muy usados).
- Un proceso, que ya tiene R_i puede requerir R_j si y solo si
- $F(R_j) > F(R_i)$

Ejemplo Prevención en Espera circular

- F(zip)=1; F(disco duro)=4, F(impresora)=7
- Un proceso que ya tiene asignado el disco, puede pedir la impresora.
- Si ya tiene la impresora, no puede solicitar el zip.

Como evitar el deadlock

- Existen algoritmos que previenen el deadlock.

- Trabajan sobre el requerimiento de los recursos.
- Impiden que se cumplan las 4 condiciones.
- Efectos secundarios: baja utilización de los recursos y de la performance del sistema.

Como evitar el deadlock: Algoritmos

- Algoritmo que determina el estado seguro de un sistema.
- Algoritmo del Banquero (para mas de una instancia por recurso)

Detección del deadlock

Si no contamos con prevención debemos contar con:

- Un algoritmo que examine si ocurrió un deadlock
- Un algoritmo para recuperación del deadlock

Recuperacion desde el deadlock

- El operador lo puede resolver manualmente.
- Esperar que el sistema se recupere automaticamente desde el deadock.
  - **Abortando todos o algunos procesos o hasta que el ciclo desaparezca**
  - **Sacandole recursos a procesos**

Recuperacion por apropiacion de recursos

Considerar 3 puntos

- ***Selección de procesos victima***: a cuales les saco los recursos y que recursos?
- ***Rollback***: debemos volver hacia una instancia segura al proceso que le sacamos los recursos? Difícil. Lo aborto y reinicio? Es mas probable
- ***Starvation***: asegurarnos de no sacarle los recursos siempre al mismo proceso

Práctica 5 Comunicación y sincronización de procesos

Tuberías

Una tubería es un mecanismo de comunicación y sincronización. Desde el punto de vista de su utilización, es como un pseudoarchivo mantenido por el sistema operativo. Conceptualmente, cada proceso ve la tubería como un conducto con dos extremos, uno de los cuales se utiliza para escribir o insertar datos y el otro para extraer o leer datos de la tubería. La escritura se realiza mediante el servicio que se utiliza para escribir datos en un archivo. De igual forma, la lectura se lleva a cabo mediante el servicio que se emplea para leer de un archivo.

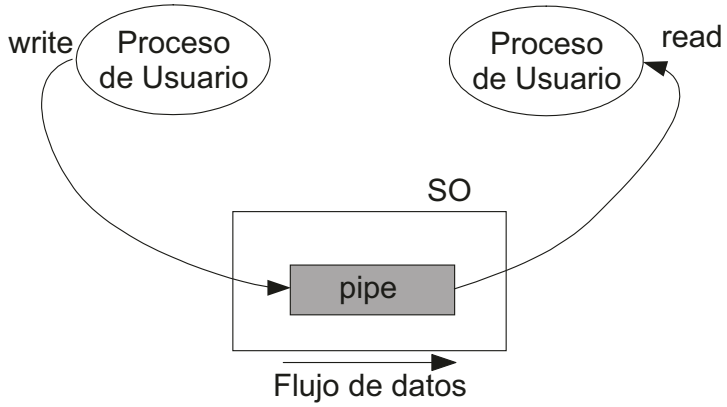


Figura 5.8 Carretero Pérez

En POSIX existen tuberías sin nombre, o simplemente "pipes", y tuberías con nombre, o FIFOS. Un pipe no tiene nombre y, por tanto, sólo puede ser utilizado entre los procesos que lo hereden a través de la llamada *fork()*.

A continuación veremos un ejemplo de Carretero Pérez de ejecución de mandatos con tuberías. Podemos colocar el comando:

$ ls | wc

que genera un listado de archivos y lo comunica a través de un *pipe* con *wc* que es el comando **w**ord **c**ount, para contar palabras, para mayor información coloque el comando:

$ man wc

El carácter "|" le indica al intérprete *bash* que debe crear una tubería entre los dos procesos. A continuación se presenta el programa 5.9 de Carretero Pérez que permite la ejecución de este comando, mediante llamadas a sistema.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

void main(void)
{
  int fd[2];
  pid_t pid;

  if (pipe(fd) < 0)  {
    perror("Error al crear la tuberia");
    exit(-1);
 }
 pid = fork();
 switch(pid)  {
    case -1:   /* error */
      perror("Error en el fork");
      exit(-1);
      case 0:    /* proceso hijo ejecuta ``ls'' */
       close(fd[0]); /* cierra el pipe de lectura */
       close(STDOUT_FILENO); /* cierra la salida estandar */
       dup(fd[1]);
       close(fd[1]);
       execlp(``ls'',``ls'',NULL);
       perror(``execlp'');
       exit(-1);
   default:  /* proceso padre ejecuta ``wc'' */
       close(fd[1]); /* cierra el pipe de escritura */
       close(STDIN_FILENO); /* cierra la entrada estandar */
       dup(fd[0]);
       close(fd[0]);
       execlp(``wc'',``wc'',NULL);
       perror(``execlp'');
      }
}
```

Se compila

```
$ gcc   -o       programa       programa.c
```

Y se ejecuta

```
$./programa
```

Otro ejemplo: Se trata simplemente de procesos en concurrencia sobre los 100 primeros caracteres de un archivo (se ejecutarán varios ejemplares del mismo programa). Para resolver el problema de la exclusión mutua sobre el acceso, se coloca un bloqueo bloqueante, utilizamos la función *lockf()*, cuyo primer parámetro es un descriptor de archivo abierto, ya sea para lectura o para lectura/escritura. El siguiente parámetro de operación es alguno de los siguientes:

- F_ULOCK: eliminación de bloqueos
- F_LOCK: bloqueo exclusivo en modo bloqueante
- F_TLOCK: bloqueo exclusivo en modo no bloqueante
- F_TEST: comprobación de existencia de un bloqueo

El tercer parámetro es el **tamaño**, que permite especificar la extensión del bloqueo. La cual se expresa con respecto a la posición actual (el valor puede ser negativo para bloquear una zona anterior a la posición actual). Un tamaño nulo permite bloquear hasta el final del archivo (cualquiera que pueda ser en el futuro).

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/file.h>

main()
{
      int d;
      char buffer[100];
      d=open("archivo", O_RDWR | O_CREAT,S_IRWXU, 0);
      write(d,buffer,100);
      if(lockf(d, F_LOCK, 100) == -1)
            perror("lockf");
      else
            printf("Proceso %d: bloqueo colocado\n", getpid());
      sleep(5);
      printf("Proceso %d: bloqueo eliminado\n", getpid());
      lockf(d, F_ULOCK, 100);
}
```

La llamada a *open()* abre el archivo en modo lectura/escritura, si "archivo" no existe lo crea. Luego la llamada a *write()* le escribe 100 caracteres, pero como *buffer* no ha sido inicializado le escribe basura, lo cual no nos importa en este caso. Se compila con:

```
$ gcc  -o      bloqueo        bloqueo.c
```

Y se ejecutan varias copias:

```
$ ./bloqueo &  ./bloqueo &    ./bloqueo &
```

enviándolas al *background*, veremos que el primer proceso coloca el bloqueo sobre el archivo, espera 5 segundos y luego lo saca, sólo entonces el segundo proceso puede colocar

el bloqueo, esperar 5 segundos y luego sacarlo, para que el tercer proceso pueda hacer lo mismo.

Autoevaluación

1) ¿Cuál no es un mecanismo de sincronización válido entre procesos Unix?
   a) Cerrojos implementados sobre archivos.
   b) Un PIPE.
   c) Instrucciones test-and-set.
   d) Las señales.
2) ¿Qué NO es cierto respecto a las propiedades que se deben cumplir en el problema de la sección crítica?
   a) No deben importar las velocidades relativas de los procesos.
   b) Exclusión mutua.
   c) Inanición
   d) No debe haber espera activa.
3) Se desea usar un semáforo S para asegurar que el inicio de una determinada actividad del proceso P2 comienza después de que finalice una actividad del proceso P1. ¿Qué primitiva de semáforo debe usar cada proceso y cuál debe ser el valor inicial del semáforo?
   a) P1 wait(S) y P2 signal(S); S=0.
   b) P1 signal(S) y P2 wait(S); S=0.
   c) P1 wait(S) y P2 signal(S); S=1.
   d) P1 signal(S) y P2 wait(S); S=1.
4) ¿Qué es falso acerca de los PIPES de Unix?
   a) Permiten comunicar a un proceso con el Sistema Operativo.
   b) Se puede utilizar para sincronizar procesos.
   c) Se puede utilizar para comunicar procesos.
   d) Tiene un buffer para amortiguar las velocidades en la comunicación entre procesos.
5) ¿Qué es cierto acerca del algoritmo de la panadería de Lamport?
   a) Resuelve el problema de los lectores-escritores.
   b) Resuelve el problema del productor-consumidor.
   c) Resuelve el problema de la sección crítica para 2 procesos.
   d) Resuelve el problema de la sección crítica para n procesos.
6) Sea un programa concurrente con tres procesos iguales cuyo código consiste tan solo en incrementar en uno una variable V compartida entre ellos. ¿Cuál de las siguientes opciones es correcta respecto al posible valor resultante de V después de la ejecución concurrente de los tres procesos si V vale 0 inicialmente?
   a) V tiene valor 1,5.
   b) V tiene valor 4.
   c) V tiene valor 0.
   d) V tiene valor 2.
7)