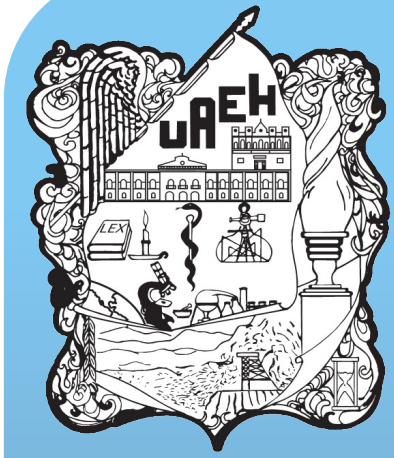

Universidad Autónoma del Estado de Hidalgo

Instituto de Ciencias Básicas e Ingeniería

Área Académica de Computación y Electrónica

Licenciatura en Sistemas Computacionales

Sistemas Operativos

Docente: M.G.A. Theira Irasema Samperio Monroy

Tema:
Mecanismos de Exclusión Mutua

Resumen:

Se describen los diferentes mecanismos utilizados por un sistema operativo para garantizar la exclusión mutua en la sincronización de procesos.

Palabras Clave: Exclusión mutua, Sincronización de procesos, Sistemas operativos.

Tema:
Mecanismos de Exclusión Mutua

Abstract:

This paper describes some of the main algorithms implemented by the operating system to guarantee mutual exclusion in synchronization process.

Keywords: Mutual exclusion, Synchronization process, Operating system.

Introducción

En sistemas de multiprogramación con un único procesador, los procesos se intercalan en el tiempo para dar la apariencia de ejecución simultánea.
Uno de los grandes problemas que nos podemos encontrar es que el hecho de compartir recursos está lleno de riesgos [1].
Por ejemplo, si dos procesos hacen uso al mismo tiempo de una variable global y ambos llevan a cabo tanto operaciones de lectura como de escritura sobre dicha variable, el orden en que se ejecuten estas lecturas y escrituras es crítico, puesto que se verá afectado el valor de la variable.

Mecanismos para garantizar la exclusión mutua

1. Semáforos
2. Monitores
3. Algoritmo de Decker
4. Algoritmo de Peterson

Semáforos

Método clásico para restringir o permitir el acceso a recursos compartidos en un entorno de multiprocesamiento.

Un tipo simple de semáforo es el binario, que puede tomar solamente los valores 0 y 1. Se inicializan en 1 y son usados cuando sólo un proceso puede acceder a un recurso a la vez.

En el siguiente ejemplo se crean y ejecutan n procesos que intentarán entrar en su sección crítica cada vez que puedan, y lo lograrán siempre de a uno por vez, gracias al uso del semáforo s inicializado en 1 [2].

```
const int n;          /* número de procesos */
variable semaforo s;        /* declaración de la variable semáforo de valor entero*/
Inicia (s,1)          /* Inicializa un semáforo con nombre s con valor 1 */
void P (int i) {
     while (cierto) {
          P(s)        /* En semáforos binarios, lo correcto es poner un P(s) antes de
                         entrar en la sección crítica, para restringir el uso de esta región
                         del código*/
 /* SECCIÓN CRÍTICA */
          V(s)             /* Tras la sección crítica, volvemos a poner el semáforo a 1
                              para que otro proceso pueda usarla */
/* RESTO DEL CÓDIGO */
 }
}
int main( ){
        Comenzar-procesos(P(1), P(2),...,P(n));
}
```

Monitores

Dentro de un sistema operativo, un monitor es un programa que observa y administra los procesos dentro del CPU.

Un monitor contiene el código relativo a un recurso compartido en un solo módulo de programa.

La implementación del monitor garantiza la exclusión mutua , mediante semáforos o algún otro mecanismo, o implícitamente en los lenguajes concurrentes.

Se pueden implementar en memoria u otro recursos compartidos [3].

Algoritmo de Decker

Solución que garantiza la exclusión mutua al gestionar elegantemente este proceso.
Da la posibilidad de que el otro proceso entre en su sección crítica una segunda vez si lo hace antes que el proceso que acaba de dejar el bucle de espera, asigne el valor cierto a su intención de entrar. Pero cuando de nuevo tiene el procesador, realiza la asignación y es su turno, así que se ejecuta.
En consecuencia, no produce aplazamiento indefinido [2].

Algoritmo de Decker

```
program algoritmo_Decker;
  var  proceso_favorecido: (primero, segundo);
       p1deseaentrar; p2deseaentrar: boolean;
  procedure proceso_uno;
     while true do
       begin
         tareas_previas_uno;
         p1deseanetrar:=true;
         while p2deseaentrar do
           if proceso_favorecido=segundo then
              begin
                   p1deseaentrar:=false;
                   while proceso_favorecido=segundo do;
                   p1deseaentrar:=true;
              end;
          seccion_critica_uno;
          proceso_favorecido:=segundo;
          p1deseaentrar:=false;
          otras_tareas_uno;
end;
```

(1)

```
procedure proceso_dos;
    while true do
        begin
            tareas_previas_dos;
            p2deseaentrar:=true;
            while p1deseaentrar do
                if proceso_favorecido=primero then
                    begin
                        p2deseaentrar:=false;
                        while proceso_favorecido=primero do;
                        p2deseaentrar:=true;
                    end;
            seccion_critica_dos;
            proceso_favorecido:=primero;
            p2deseaentrar:=false;
            otras_tareas_dos;
end;

begin
    p1deseaentrar:=false;
    p2deseaentrar:=false;
    proceso_favorecido:=primero;
    parbegin
        proceso_uno;
        proceso_dos;
    parend
end.
```

(1)

Algoritmo de Peterson

Simplifica el algoritmo de Decker.

El protocolo de entrada es más elegante con las mismas garantías de exclusión mutua, imposibilidad de bloqueo mutuo y de aplazamiento indefinido.

Algoritmo de Peterson

```
program algoritmo_Peterson;
  var  proceso_favorecido: (primero, segundo);
         p1deseaentrar; p2deseaentrar: boolean;
  procedure proceso_uno;
      while true do
        begin
           tareas_previas_uno;
           p1deseanetrar:=true;
           proceso_favorecido:=segundo;
           while p2deseaentrar
                 and proceso_favorecido=segundo do;
            seccion_critica_uno;
            proceso_favorecido:=segundo;
            p1deseaentrar:=false;
            otras_tareas_uno;
end;
```

(2)

```
procedure proceso_dos;
    while true do
        begin
            tareas_previas_dos;
            p2deseaentrar:=true;
            proceso_favorecido:=primero;
            while p1deseaentrar
                and proceso_favorecido=primero do;
            seccion_critica_dos;
            p2deseaentrar:=false;
            otras_tareas_dos;
end;

begin
    p1deseaentrar:=false;
    p2deseaentrar:=false;
    proceso_favorecido:=primero;
    parbegin
        proceso_uno;
        proceso_dos;
    parend
end.
```

(2)

Referencias (imágenes)

(1) González N., Pablo (s.f.) *"4.2.1 Soluciones de software al problema de exclusión mutua. Algoritmo de Decker". Consultado el 15 de febrero de 2014 en http://lsi.vc.ehu.es/pablogn/docencia/manuales/SO/TemasSOuJaen/CONCURRENCIA/2UtilizandoMemoriaCompartidaI.htm*

(2) González N., Pablo (s.f.) *"4.2.1 Soluciones de software al problema de exclusión mutua. Algortimo de Peterson". Consultado el 16 de febrero de 2014 en http://lsi.vc.ehu.es/pablogn/docencia/manuales/SO/TemasSOuJaen/CONCURRENCIA/2UtilizandoMemoriaCompartidaI.htm*

Referencias

[1] Deitel, H. M. (2004). Sistemas Operativos. Addison Wesley.

[2] Pérez, J. (2001). Sistemas Operativos. Una visión aplicada. México: Mc Graw-Hill.

[3] Stallings, W. (2003). Sistemas Operativos. Prentice Hall.