
Ingeniería en Sistemas Computacionales

Fundamentos de Sistemas Operativos

Actividad 13

IS730547 – Santiago Cordova Berrelleza

IS727272 - Marco Ricardo Cordero Hernández

Jal., 14 de junio de 2023

**1.-** Considerando las definiciones de semáforos tanto enteros como binarios, uno de los requisitos para que estos funcionen es que la ejecución de las llamadas waitsem y signalsem se haga de manera atómica. Muestra con un ejemplo por qué pueden fallar si no se asegura la atomicidad en la ejecución de ambas funciones. Por ejemplo: dos procesos/hilos ejecutan waitsem sin garantizar la atomicidad de waitsem; o un proceso ejecuta waitsem y otro signalsem sin garantizar la atomicidad de waitsem y seignalsem; o dos procesos ejecutan signalsem sin garantizar la atomicidad la función signalsem.

**R**: A continuación se muestra una ejecución de dos procesos que hacen uso de semáforos, en donde los cuales realizan una operación wait *no atómica* dentro de un semáforo de enteros

| P | Q | Estado P | Estado Q | Contador |
|---|---|---|---|---|
| | | Listo | Listo | 1 |
| s.contador—; | | En ejecución | | 0 |
| | s.contador—; | | En ejecución | -1 |
| if s.contador < 0 | | En ejecución | | |
| | if s.contador < 0 | | En ejecución | |
| Bloquear proceso | | Bloqueado | | |
| | Bloquear proceso | | Bloqueado | |

Con la ejecución anterior se ha demostrado la necesidad de operaciones atómicas, ya que en caso de no asegurarse esto, se impondrá un deadlock ante los procesos que se encuentren actualmente en el semáforo, deteniendo la ejecución del programa de forma indeterminada. ■

**2.-** Compara las definiciones de semáforos enteros donde los semáforos enteros pueden tomar valores negativos y aquellos donde no toman valores negativos, ¿hay alguna diferencia en el efecto

de las dos definiciones cuando se utilizan para sincronizar procesos? Es decir, ¿es posible sustituir una definición por la otra sin alterar el significado del programa?

La decisión de uso de un tipo de semáforo u otro depende del criterio del desarrollador, de forma que solo éste conocería las limitantes del sistema sobre el cual se estaría trabajando. En cuestión del funcionamiento del programa, su ejecución no debería verse afectada más allá del tiempo de ejecución, ya que para el semáforo de enteros sin signo se requiere una mayor cantidad de líneas de código para llevar a cabo las primitivas del semáforo. Ambos métodos resuelven correctamente el mismo problema, por lo que se podría decir que sus definiciones son prácticamente intercambiables. ■

**3.-** La solución correcta al problema del productor consumidor que se muestra a continuación le hace falta considerar que el buffer es finito, es decir, que el tamaño del buffer es limitado, **usando semáforos binarios** haz las modificaciones y agrega los semáforos necesarios para que cuando haya N elementos en el buffer el productor se bloquee si quiere agregar más elementos hasta que el consumidor consuma un elemento del buffer y haya un espacio disponible.

```
int m,n;
SemaphoreBin s; // 1
SemaphoreBin retraso; // 0

SemaphoreBin buffS; // 0

int buffsize = 0;

Productor()
{
    while(forever)
    {
        producir;
        waitB(s);
        añadir;
        n++;

        buffsize = n;
        if (n==1) signalB(retraso);
        signalB(s);

        if (buffsize == TamañoBuffer) waitB(buffS);
    }
}

Consumidor()
{
    waitB(retraso);
    while(forever)
    {
        waitB(s);
        tomar;
```

```
        n--;
        m=n;

        if (n == TamañoBuffer - 1) signalB(buffS);
        signalB(s);
        consumir;
        if (m==0) waitB(retraso);
    }
}

main()
{
   n=0;
   initbsem(s,1);
   initbsem(retraso,0);
   cobegin {
      Productor();
      Consumidor();
   }
}
```

**4.-**Considere un programa concurrente con dos procesos P y Q, definidos a continuación. la impresión en pantalla de A, B, C, D y E son sentencias arbitrarias atómicas (indivisibles). Supóngase que el programa principal ejecuta concurrentemente los procesos P y Q. Comprueba tu solución empleando semáforos en Linux.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <time.h>

void imprime(char *s)
{
    usleep(rand()%100000);
    printf("%s\n",s);
    usleep(rand()%100000);
}

void P()
{
    imprime("A");
    imprime("B");
    imprime("C");
    exit(0);
}

void Q()
{
```

```
    imprime("D");
    imprime("E");
    exit(0);
}

int main()
{
    int p;

    srand(getpid());

    p=fork();
    if(p==0)
        P();

    p=fork();
    if(p==0)
        Q();

    wait(NULL);
    wait(NULL);
}
```

A. Cuántas combinaciones posibles hay en la intercalación de las sentencias A, B, C, D y E.
**R**: 10 posibles combinaciones.

B. Usando semáforos sincronice P y Q de manera que se asegure que las sentencias A y B se ejecutan antes que D.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <time.h>
#include "semaphoresarr.h"


int semarr;
#define SYNC 0


void imprime(char *s)
{
    usleep(rand()%100000);
    printf("%s\n",s);
    usleep(rand()%100000);
}


void P()
{
```

```
    imprime("A");
    imprime("B");
    semsignal(semarr, SYNC);
    imprime("C");
    exit(0);
}


void Q()
{
    semwait(semarr, SYNC);
    imprime("D");
    imprime("E");
    exit(0);
}


int main()
{
    int p;


    // Crear semáforos (Solicitar al OS un arreglo de semáforos)
    semarr = createsemarray(0x4321, 1);
    initsem(semarr, 0, 0); // Inicializar semáforo 0 con valor de 0


    srand(getpid());


    p=fork();
    if(p==0)
        P();


    p=fork();
    if(p==0)
        Q();


    wait(NULL);
    wait(NULL);


    erasesem(semarr);
}
```

c. Usando semáforos haga que el proceso P ejecute siempre las sentencia B inmediatamente después que A sin que cualquier sentencia de Q se ejecute intercalada entre ellas.

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

```c
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <time.h>
#include "semaphoresarr.h"


int semarr;
#define SYNC 0


void imprime(char *s)
{
    usleep(rand()%100000);
    printf("%s\n",s);
    usleep(rand()%100000);
}


void P()
{
    semwait(semarr, SYNC);
    imprime("A");
    imprime("B");
    semsignal(semarr, SYNC);
    imprime("C");
    exit(0);
}


void Q()
{
    semwait(semarr, SYNC);
    imprime("D");
    imprime("E");
    semsignal(semarr, SYNC);
    exit(0);
}


int main()
{
    int p;


    // Crear semáforos (Solicitar al OS un arreglo de semáforos)
    semarr = createsemarray(0x4321, 1);
    initsem(semarr, 0, 1); // Inicializar semáforo 0 con valor de 1


    srand(getpid());
```

```
    p=fork();
    if(p==0)
        P();


    p=fork();
    if(p==0)
        Q();


    wait(NULL);
    wait(NULL);


    erasesem(semarr);
}
```

**5.-** ¿Qué aprendiste?

Uso y abuso de los semáforos para la sincronización de procesos de carácter indistinto, así como múltiples técnicas de resolución para el problema de la concurrencia y la toma de secciones críticas. También, se pudo aprender de la implementación de estos mecanismos a través de código real en C, así como ejemplos reales y casos de uso pertinentes para la apropiación del conocimiento relacionado al tema de la sesión.