

Reporte Práctica #2 **Sincronización de semáforos**

Introducción:

La sincronización es necesaria cuando dos o más procesos se van a disputar un recurso en común, ya sea escribiendo o leyendo de una zona crítica. Para el caso de los hilos, existe un mecanismo de sincronización que se llama “semáforo”. Existen diferentes tipos de semáforos, y en esta práctica se usaron los “system v”.

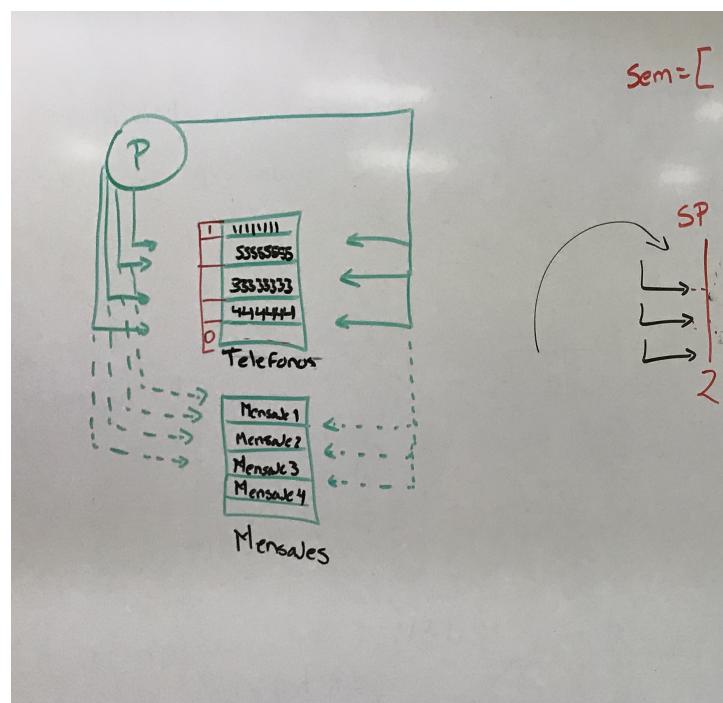
Para crear un semáforo system V, se debe crear una llave con la función ftok(...) donde se debe especificar un “pathname” válido.

Luego se crea al semáforo con la función semget(...), si la función falla, puede que el semáforo ya exista así que hay que ligarse a este utilizando smget(...) pero sin el “IPC_CREAT”.

Después hay que inicializar al semáforo con semctl(...), donde se especifica el valor que tendrá este, =0 cerrado, o >0 abierto.

Objetivo:

La práctica consistía en generar ‘n’ hilos productores y ‘m’ hilos consumidores. Los hilos productores tenían que escribir en una zona crítica cada uno un teléfono diferente ‘x’ veces y luego escribir un mensaje ‘x’ veces. Los hilos consumidores tenían que sincronizarse con los hilos productores para leer estos teléfonos y mensajes.



Desarrollo:

Se creó una sección crítica global para guardar 5 cadenas de longitud 8, y otra sección exactamente igual a la que se va a acceder a través de la primera sección crítica.

```
}
```

```
else{
```

```
    semctl(idSem,0,SETVAL,5); //el grandote de la izquierda
```

```
    semctl(idSem,1,SETVAL,1);
```

```
    semctl(idSem,2,SETVAL,1);
```

```
    semctl(idSem,3,SETVAL,1);
```

```
    semctl(idSem,4,SETVAL,1);
```

```
    semctl(idSem,5,SETVAL,1);
```

```
    semctl(idSem,6,SETVAL,0);
```

```
    semctl(idSem,7,SETVAL,0);
```

```
    semctl(idSem,8,SETVAL,0);
```

```
    semctl(idSem,9,SETVAL,0);
```

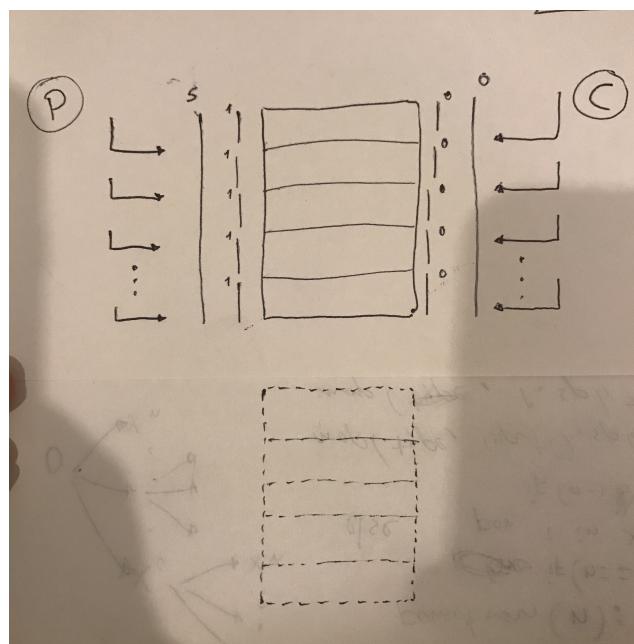
```
    semctl(idSem,10,SETVAL,0);
```

```
    semctl(idSem,11,SETVAL,0); //el grandote de la derecha
```

```
    printf("\n\ninicia %d %c
```

```
}
```

Se crearon 12 semáforos, 1 para regular a los hilos productores, otro para regular la entrada de los hilos consumidores, y 5 pares de semáforos cruzados para regular la lectura y escritura de la zona crítica.



Los hilos se crearon de forma dinámica introduciendo los valores desde consola.

```

int valors[10]={0,1,2,3,4,5,6,7,8,9};

int prod=0,cons=0;
for(int i=0 ; i<numProductores ; i++){
    pthread_create(&hilos[i], NULL, productor, (void *)&valors[i]);
}

for(int i=0 ; i<numConsumidores ; i++){
    pthread_create(&hilos[i+numProductores], NULL, consumidor, (void *)&valors[i]);
}

```

La función de los hilos productores obtiene como parámetro, un entero que será el número que identificará a ese hilo en la producción, así si obtiene ‘1’ escribirá el teléfono 11111111 y “mensaje1”

El hilo productor entra a la zona crítica, decrementa al semáforo de los productores, itera sobre la zona crítica y si encuentra un semáforo abierto entra y lo cierra para poder escribir en esa sección de la zona crítica, luego escribe en la zona de los mensajes, disminuye su conteo de cuántas veces debe escribir y al terminar deja cerrado el semáforo del lado de los productores, abre del lado de los consumidores e incrementa una vez el semáforo de todos los consumidores.

```

void * productor(void* argv){
    int *arg=(int *)argv;
    char letra=(*arg+1)+'0';

    int cont=producciones;
    while(cont!=0){//producciones en la sección crítica
        for(int i=0;i<5;i++){
            cierre(0);
            if(semctl(idSem,i+1,GETVAL,NULL)==1){
                cierre(i+1);//cierro 1 a 5 Sem de productor
                printf("\nP%c %d %d %d %d %d %d %d %d %d",letra,semctl(idSem,0,GETVAL,NULL),semctl(idSem,1,6
                for(int j=1; j<9 ;j++){
                    secCrit[i][j]=letra;
                }
                secCrit2[i][0]='m';secCrit2[i][1]='e';secCrit2[i][2]='n';secCrit2[i][3]='s';secCrit2[i][4]='a';secCrit2[i][5]='j';se
                secCrit2[i][7]=letra;
                cont=cont-1;
                apertura(i+6); //abro 6 a 10 Sem de consumidor
                apertura(11);
                break;
            }
            else{
                apertura(0);
            }
        //}
        //if(cont==0){ apertura(0);break; }
        //apertura(0);
    }
    printf("\nSoy el productor%c y voy a salir...",letra);
    pthread_exit(NULL);
}

```

La función de los consumidores es muy parecida. La función del hilo consumidor recibe un número para identificar mejor al hilo a la hora de imprimir lo que leyó.

Hay un valor global llamado consumosT que viene siendo el total de veces que todos los hilos van a leer y es la multiplicación del número de hilos productores por el total de veces que van a producir.

Los hilos entran a un loop donde por cada vez que lean se va a ir decrementando el valor de consumosT hasta que este sea 0. Al entrar al for, decrementan el semáforo de los consumidores, si encuentran un semáforo de la zona crítica abierto del lado de los consumidores, el hilo va a entrar, lo va a decrementar y va a leer lo que encuentre, así como también el valor en la zona de los mensajes, al terminar, va a decrementar a consumosT, va a abrir el semáforo de lado contrario y también va a incrementar una vez el semáforo de los productores.

```
void * consumidor(void * argv){// en el for de aqui no podemos saber cuanto va a consumir cada consumidor... debe ser
    int * arg=(int*) argv;
    char letra=(*arg+1)+'0';
    while(consumosT!=0){
        printf("\n\tConsumosT=%d",consumosT);
        for(int i=0;i<5;i++){
            cierre(11);
            if(semctl(idSem,i+6,GETVAL,NULL)==1){
                cierre(i+6);//cierra de 6 a 10
                printf("\nC%c %d %d",letra,semctl(idSem,0,GETVAL,NULL),secCrit[i][1],secCrit[i][2],secCrit[i][3],secCrit[i][4],secCrit[i][5],secCrit[i][6],secCrit[i][7],secCrit[i][8],secCrit[i][9],secCrit[i][10],secCrit[i][11],secCrit[i][12]);
                printf("\nSoy el consumidor%c y leo:%c%c%c%c%c\n",letra,secCrit[i][1],secCrit[i][2],secCrit[i][3],secCrit[i][4],secCrit[i][5],secCrit[i][6]);
                printf("\nSoy el consumidor%c y leo: %c%c%c%c%c%c%c\n",letra,secCrit2[i][0],secCrit2[i][1],secCrit2[i][2],secCrit2[i][3],secCrit2[i][4],secCrit2[i][5],secCrit2[i][6],secCrit2[i][7]);
                consumosT=consumosT-1;
                apertura(i+1);//abre 1 a 5 sem de productor
                apertura(0);
                if(consumosT<numConsumidores){ pthread_exit(NULL); }
                break;
            }
            else{
                apertura(11);
            }
        }
        printf("\nSoy el consumidor%c y voy a acabar...%d",letra,consumosT);
        pthread_exit(NULL);
    }
}
```

Para resolver el problema de que consumosT también es una zona crítica, si el número de consumosT es menor que el número de consumidores, el hilo va a morir después de decrementar a consumosT.

Los semáforos se matan mandando a llamar una función cuyo parámetro es el número total de semáforos.

```
void matar_semaforos(int ns){
    for(int i=0; i<ns; i++){
        semctl(idSem,i,IPC_RMID,0);//mato a los semáforos
    }
}
```

Conclusiones:

Ana Paola Nava Vivas

Los semáforos son una herramienta muy importante en la sincronización de procesos hilos. Se puede cumplir con la exclusión mútua si los semáforos están bien implementados. La gran ventaja de los semáforos es que los procesos no pueden acceder a la zona crítica y se duermen, quedando a la espera de que se abra el semáforo que los detiene, esto mal programado podría generar un “deadlock”.

Gustavo Josue Robles Martínez

Es bastante interesante las formas de implementar los semáforos, hay maneras más sencillas y complicadas, aunque siento que hay veces que las implementaciones necesitan conocimientos previos de análisis a mayor profundidad, seguro hay todo un tema muy completo de semáforos para diferentes propósitos, hace unos días me encontré con un log de semáforos de php.

