Programación de Sistemas CCPG1008

Federico Domínguez, PhD.

Unidad 6 – Sesión 6: Uso de semáforos para sincronizar acceso a recursos compartidos

Contenido

- Patrón de diseño: Productor – Consumidor
- Patrón de diseño: Escritores – Lectores

Aparte de la exclusión mutua, los semáforos permiten la sincronización de acceso a recursos compartidos.

Existen dos ejemplos clásicos que ilustran el uso de semáforos en el acceso de recursos compartidos:

- Productor – Consumidor
- Escritores – Lectores

El patrón de diseño **Productor – Consumidor** consiste en un proceso/hilo que produce un recurso y lo inserta en un buffer finito y otro proceso/hilo que extrae un recurso del buffer y lo consume.

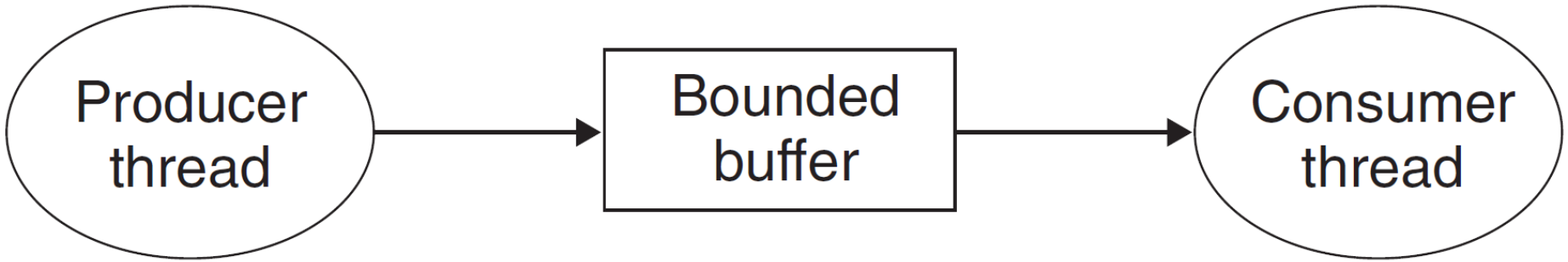
En este caso el recurso compartido es el buffer finito.

El buffer tiene *n* espacios y el proceso consumidor debe bloquearse si el buffer esta vacío.

El proceso productor debe bloquearse si el buffer esta lleno.

Ejemplos:

- Decodificación de *frames* de video
- Procesamiento de paquetes de red en ruteadores



Una solución al problema Productor – Consumidor es usando tres semáforos.

- Un semáforo funciona como binario y provee acceso exclusivo al buffer. (*mutex* en el ejemplo)
- Otro semáforo es inicializado con 0 y bloquea al proceso consumidor cuando el buffer esta vacío. (*items* en el ejemplo)
- Otro semáforo es inicializado en *n* (el número de espacios en el buffer) y bloquea al proceso productor cuando el buffer esta lleno. (*slots* en el ejemplo)

```c
/* Remove and return the first item from buffer sp */
int sbuf_remove(sbuf_t *sp)
{
    int item;
    P(&sp->items);                          /* Wait for available item */
    P(&sp->mutex);                          /* Lock the buffer */
    item = sp->buf[(++sp->front)%(sp->n)];  /* Remove the item */
    V(&sp->mutex);                          /* Unlock the buffer */
    V(&sp->slots);                          /* Announce available slot */
    return item;
}

/* Insert item onto the rear of shared buffer sp */
void sbuf_insert(sbuf_t *sp, int item)
{
    P(&sp->slots);                          /* Wait for available slot */
    P(&sp->mutex);                          /* Lock the buffer */
    sp->buf[(++sp->rear)%(sp->n)] = item;   /* Insert the item */
    V(&sp->mutex);                          /* Unlock the buffer */
    V(&sp->items);                          /* Announce available item */
}
```

El problema **Escritores – Lectores** surge cuando varios hilos necesitan leer un recurso mientras varios hilos necesitan escribir el recurso.

En este escenario, los hilos lectores pueden tener acceso concurrente al recurso mientras que un hilo escritor necesita acceso exclusivo al recurso.

Ejemplo:

- En un sistema de reserva de aerolíneas varios clientes pueden consultar simultáneamente la base de datos y revisar cuantos asientos hay disponibles en un vuelo.
- Al hacer una reserva, el cliente necesita acceso exclusivo a la base de datos.

Existen varias versiones del problema, dependiendo a quien se le da la prioridad: a los escritores o a los lectores.

El problema se puede resolver usando dos semáforos.

```c
/* Global variables */
int readcnt;    /* Initially = 0 */
sem_t mutex, w; /* Both initially = 1 */

void reader(void)
{
    while (1) {
        P(&mutex);
        readcnt++;
        if (readcnt == 1) /* First in */
            P(&w);
        V(&mutex);

        /* Critical section */
        /* Reading happens  */

        P(&mutex);
        readcnt--;
        if (readcnt == 0) /* Last out */
            V(&w);
        V(&mutex);
    }
}
```

```c
void writer(void)
{
    while (1) {
        P(&w);

        /* Critical section */
        /* Writing happens  */

        V(&w);
    }
}
```

Referencias

Libro guía *Computer Systems: A programmers perspective*. Secciones 12.5