Introducción a Sistemas Operativos: Concurrencia

Clips xxx
Francisco J Ballesteros

1. Buffers compartidos: el productor/consumidor

Un problema que aparece a todas horas en programación concurrente es el del **productor-consumidor**, también conocido como el del **buffer acotado**. Se trata de tener procesos que producen cosas y procesos que las consumen, como puede verse en la figura 1. Cuando el buffer no tiene límite se denomina *productor-consumidor* al problema y en otro caso se le conoce como el problema del *buffer acotado*. ¡Los pipes son un caso de dicho problema!

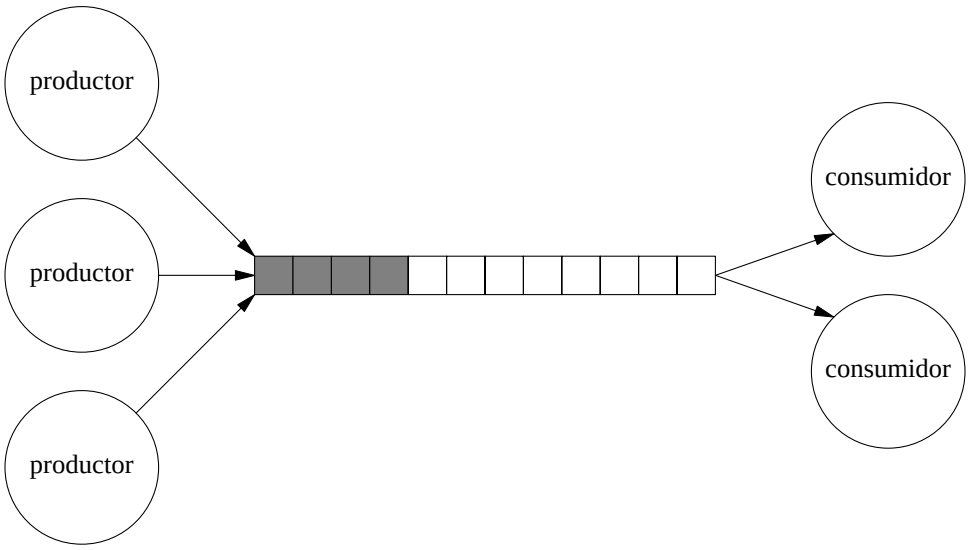

Figura 1: *El problema del productor consumidor con un buffer acotado.*

La solución es sencilla si pensamos en lo siguiente:

- Para producir un ítem es preciso tener un hueco en el buffer
- Para consumir in ítem es preciso tener un ítem en el buffer
- Para operar con el buffer es preciso utilizar un mutex.

La idea es tener un semáforo que represente los ítems en el buffer, otro que represente los huecos en el buffer y otro que represente el mutex:

- Cuando queramos un hueco basta pedirlo: `down(huecos)`.
- Cuando queramos un ítem basta pedirlo: `down(ítems)`.

El mutex ya sabes manejarlo. Naturalmente, si alguien produce un ítem deberá llamar a `up(ítems)` e, igualmente, si alguien produce un hueco deberá llamar a `up(huecos)`. Si piensas en los semáforos como en cajas con tickets, todo esto te resultará natural.

Primero vamos a implementar un buffer compartido que, en nuestro caso, será una cola de caracteres.

Vamos a mostrar y discutir el código en el mismo orden en que aparece en el fichero `pc.c`, secuencialmente de arriba a abajo.

`[pc.c]`:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <err.h>
#include "sem.h"

enum {QSIZE = 4};

typedef struct Queue Queue;
struct Queue {
    Sem mutex, nchars, nholes;
    char buf[QSIZE];
    int hd, tl;
};
```

En la cola `Queue` guardamos los tres semáforos que hemos mencionado, el buffer para guardar los caracteres y la posición de la cabeza (`hd`) y cola (`tl`) en el buffer. Insertaremos caracteres al final de la cola (en `tl`) y los tomaremos del principio, de `hd`.

Inicializar la cola requiere crear los semáforos y poco más:

```
static int
qinit(Queue *q)
{
    memset(q, 0, sizeof *q);
    if (semcreat(&q->mutex, 1) < 0) {
        return -1;
    }
    if (semcreat(&q->nchars, 0) < 0) {
        semclose(&q->mutex);
        return -1;
    }
    if (semcreat(&q->nholes, QSIZE) < 0) {
        semclose(&q->mutex);
        semclose(&q->nchars);
        return -1;
    }
    return 0;
}
```

Tomamos la precaución de dejar todos los campos a valores nulos llamando a `memset` para que todo esté bien inicializado.

Terminar de usar la cola requiere cerrar nuestros semáforos:

```
static void
qterm(Queue *q)
{
    semclose(&q->mutex);
    semclose(&q->nchars);
    semclose(&q->nholes);
    q->hd = q->tl = 0;
}
```

¡Ahora hay que hacer la parte interesante! Para poner un carácter en la cola necesitamos y hueco, luego ponerlo mientras tenemos el mutex (para evitar condiciones de carrera en el uso de la cola) y además hemos de echar un ticket al semáforo que indica cuántos ítems hay en la cola:

```
static int
qput(Queue *q, int c)
{
    if (semdown(&q->nholes) < 0) {
        return -1;
    }
    if (semdown(&q->mutex) < 0) {
        return -1;
    }
    q->buf[q->tl] = c;
    q->tl = (q->tl+1)%QSIZE;
    if (semup(&q->mutex) < 0) {
        return -1;
    }
    if (semup(&q->nchars) < 0) {
        return -1;
    }
    return 0;
}
```

Tomar un carácter de la cola es simétrico con respecto a ponerlo. Esta vez pedimos ítems y generamos huecos:

```
static int
qget(Queue *q)
{
    int c;

    if (semdown(&q->nchars) < 0) {
        return -1;
    }
    if (semdown(&q->mutex) < 0) {
        return -1;
    }
    c = q->buf[q->hd];
    q->hd = (q->hd+1)%QSIZE;
    if (semup(&q->mutex) < 0) {
        return -1;
    }
    if (semup(&q->nholes) < 0) {
        return -1;
    }
    return c;
}
```

Ya podemos declarar la cola:

```
static Queue q;
```

Un *productor* será un proceso que pone caracteres en la cola. En nuestro caso cada productor pondrá 10:

```
static void*
tput(void *a)
{
    char *s;
    int i;

    s = a;
    for (i = 0; i < 10; i++) {
        if (qput(&q, *s) < 0) {
            err(1, "qput");
        }
    }
    return NULL;
}
```

Hemos pasado como parámetro un puntero a un `char` para que cada productor ponga un carácter distinto en la cola.

El consumidor va a sacar de la cola todo lo que pueda hasta que obtenga un `0`, con lo que marcaremos el final de los datos:

```
static void*
tget(void *a)
{
    int c;
    char buf[1];

    for (;;) {
        c = qget(&q);
        if (c == 0) {
            break;
        }
        if (c < 0) {
            err(1, "qget");
        }
        buf[0] = c;
        if (write(1, buf, 1) != 1) {
            err(1, "write");
        }
    }
    return NULL;
}
```

Por último, nuestro programa principal inicializará la cola y creará dos productores y un consumidor.

```
int
main(int argc, char* argv[])
{
    pthread_t p1, p2, g;
    void *sts;

    if (qinit(&q) < 0) {
        err(1, "qinit");
    }
    if (pthread_create(&p1, NULL, tput, "a") != 0) {
        err(1, "thread");
    }
    if (pthread_create(&p2, NULL, tput, "b") != 0) {
        err(1, "thread");
    }
    if (pthread_create(&g, NULL, tget, NULL) != 0) {
        err(1, "thread");
    }
    pthread_join(p1, &sts);
    pthread_join(p2, &sts);
    if (qput(&q, 0) < 0) {
        err(1, "qput");
    }
    pthread_join(g, &sts);
    write(1, "\n", 1);
    qterm(&q);
    exit(0);
}
```

¡Listos para ejecutarlo!

```
unix$ pc
bbaabaaaaababababbabbb
unix$
```

Este problema es muy importante. En casi todos los programas utilizarás algún tipo de buffer compartido y necesitarás código que resuelva el problema del productor-consumidor. Además, es un buen problema para terminar de entender cómo se utilizan los semáforos y cómo funcionan. Observa que si el buffer se llena, el productor espera a tener hueco. Igualmente, si el buffer se vacía, el consumidor espera a tener algo que consumir. ¡Los pipes funcionan de este modo!