Facultad de Ingeniería en Electricidad y Computación

Programación de Sistemas CCPG1051

Federico Domínguez, PhD.

Unidad 6 – Sesión 7: Patrones de diseño con concurrencia

Agenda

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

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.

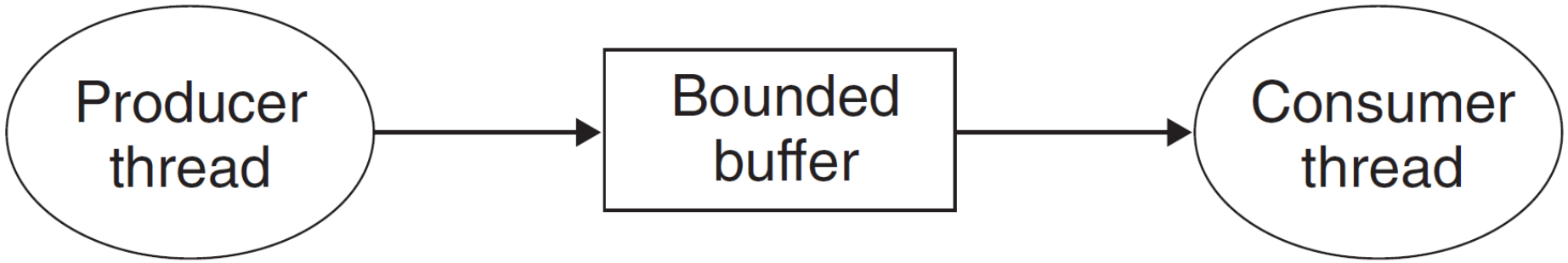
En este caso el recurso compartido es el buffer finito.

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)

```
typedef struct {
    int *buf;          /* Buffer array */
    int n;             /* Maximum number of slots */
    int front;         /* buf[(front+1)%n] is first item */
    int rear;          /* buf[rear%n] is last item */
    sem_t mutex;       /* Protects accesses to buf */
    sem_t slots;       /* Counts available slots */
    sem_t items;       /* Counts available items */
} sbuf_t;

/* Create an empty, bounded, shared FIFO buffer with n slots */
void sbuf_init(sbuf_t *sp, int n)
{
    sp->buf = Calloc(n, sizeof(int));
    sp->n = n;                       /* Buffer holds max of n items */
    sp->front = sp->rear = 0;        /* Empty buffer iff front == rear */
    Sem_init(&sp->mutex, 0, 1);      /* Binary semaphore for locking */
    Sem_init(&sp->slots, 0, n);      /* Initially, buf has n empty slots */
    Sem_init(&sp->items, 0, 0);      /* Initially, buf has zero data items */
}
```

```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.

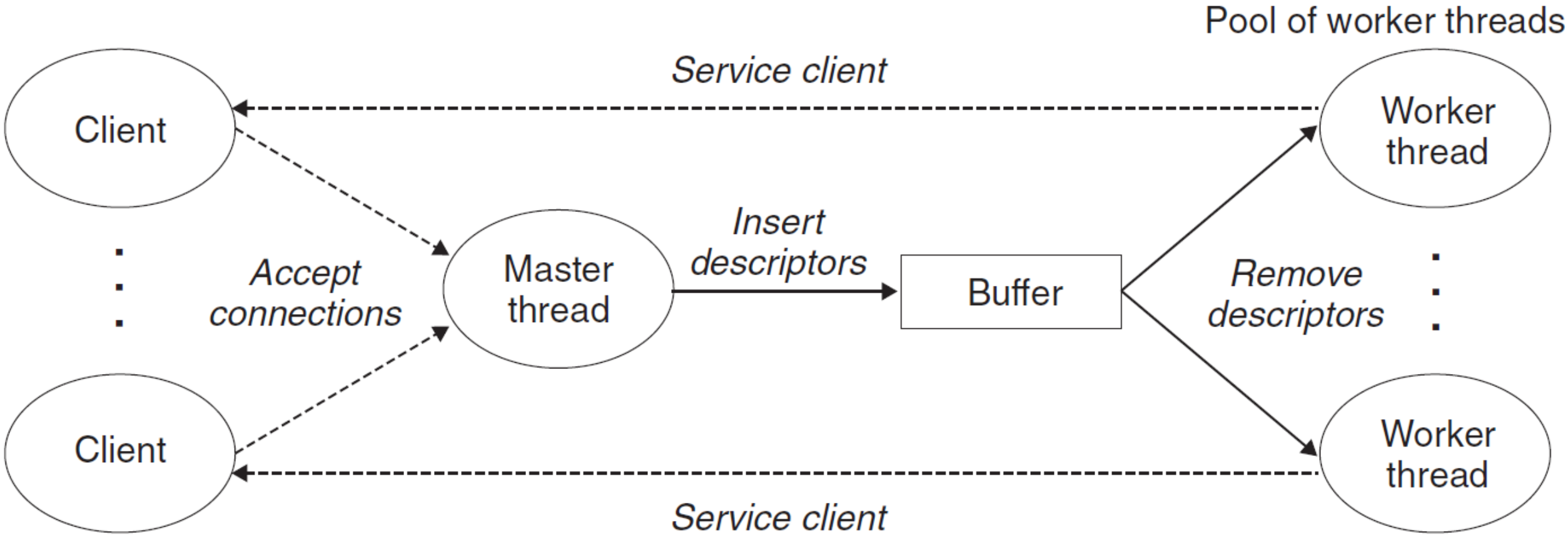
```c
/* Global variables */
int readcnt;    /* Initially = 0 */
sem_t mutex, w; /* Both initially = 1 */

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);
    }
}
```

Prethreading: Uso de **worker threads** para conexión simultánea de clientes.

Crear un hilo para cada cliente puede incurrir en costos computacionales altos en aplicaciones cliente - servidor con alto volumen de conexiones.

La técnica *prethreading* permite la creación de un número fijo de hilos al iniciar el servidor los cuales son reclutados para atender a diferentes clientes usando el patrón de diseño productor – consumidor.


Pool of worker threads
Service client
Client
Worker thread
Insert descriptors
Accept connections
Master thread
Buffer
Remove descriptors
Client
Worker thread
Service client

```c
int main(int argc, char **argv)
{
    int i, listenfd, connfd, port;
    socklen_t clientlen=sizeof(struct sockaddr_in);
    struct sockaddr_in clientaddr;
    pthread_t tid;

    if (argc != 2) {
        fprintf(stderr, "usage: %s <port>\n", argv[0]);
        exit(0);
    }
    port = atoi(argv[1]);
    sbuf_init(&sbuf, SBUFSIZE);
    listenfd = Open_listenfd(port);

    for (i = 0; i < NTHREADS; i++)  /* Create worker threads */
        Pthread_create(&tid, NULL, thread, NULL);

    while (1) {
        connfd = Accept(listenfd, (SA *) &clientaddr, &clientlen);
        sbuf_insert(&sbuf, connfd); /* Insert connfd in buffer */
    }
}

void *thread(void *vargp)
{
    Pthread_detach(pthread_self());
    while (1) {
        int connfd = sbuf_remove(&sbuf); /* Remove connfd from buffer */
        echo_cnt(connfd);                /* Service client */
        Close(connfd);
    }
}
```

Referencias

Libro guía *Computer Systems: A programmers perspective*. Secciones 12.5.4-5