Introducción a Sistemas Operativos: Concurrencia

Clips xxx
Francisco J Ballesteros

1. Semáforos con pipes

Vamos a implementar un semáforo que podremos usar en aplicaciones que compartan un proceso padre (o ancestro) común. La idea es utilizar un pipe como semáforo y guardar tantos bytes en el semáforo como tickets queramos tener: Para hacer un `down` leeremos y byte y para hacer un `up` escribiremos un byte.

Con esta idea, el único problema es que es preciso crear el semáforo antes de llamar a `fork` (para que todos lo procesos lo compartan), si es que usamos `fork`. Si usamos threads, dado que comparten los descriptores de fichero, no tenemos problemas.

Este es el interfaz para nuestros semáforos

`[sem.h]`:

```
typedef struct Sem Sem;
struct Sem {
    int fd[2];
};

int semdown(Sem *s);
int semup(Sem *s);
int semcreat(Sem *s, int val);
void semclose(Sem *s);
```

y esta es la implementación:

`[sem.c]`:

```
#include <stdio.h>
#include <unistd.h>
#include "sem.h"

int
semdown(Sem *s)
{
    char c;

    if (read(s->fd[0], &c, 1) != 1) {
        return -1;
    }
    return 0;
}
```

```
int
semup(Sem *s)
{
    if (write(s->fd[1], " ", 1) != 1) {
        return -1;
    }
    return 0;
}


void
semclose(Sem *s)
{
    close(s->fd[0]);
    close(s->fd[1]);
    s->fd[0] = s->fd[1] = -1;
}


int
semcreat(Sem *s, int n)
{
    int i;

    if (pipe(s->fd) < 0) {
        return -1;
    }
    for (i = 0; i < n; i++) {
        if (semup(s) < 0) {
            semclose(s);
            return -1;
        }
    }
    return 0;
}
```

Una vez los tenemos, podemos utilizarlos como en nuestro programa de ejemplo, que hemos adaptado para utilizar estos semáforos:

[semcnt.c]:

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

enum { Nloops = 10 };
static int nloops = Nloops;
static int cnt;
static Sem sem;
```

```
static void*
tmain(void *a)
{
    int i;

    for(i = 0; i < nloops; i++) {
        if (semdown(&sem) < 0) {
            err(1, "down");
        }
        cnt++;
        if (semup(&sem) < 0) {
            err(1, "up");
        }
    }
    return NULL;
}


int
main(int argc, char* argv[])
{
    int i;
    pthread_t thr[3];
    void *sts;

    if(argc > 1) {
        nloops = atoi(argv[1]);
    }
    if (semcreat(&sem, 1) < 0) {
        err(1, "sem creat");
    }
    for(i = 0; i < 3; i++) {
        if(pthread_create(thr+i, NULL, tmain, &cnt) != 0) {
            err(1, "thread");
        }
    }

    for(i = 0; i < 3; i++) {
        pthread_join(thr[i], &sts);
    }
    semclose(&sem);
    printf("cnt is %d\n", cnt);
    exit(0);
}
```

Ahora podemos compilarlo y ejecutarlo sin problemas:

```
unix$ cc -c sem.c
unix$ cc -c semcnt.c
unix$ cc -o semcnt semcnt.o sem.o
unix$ semcnt 10000
cnt is 30000
unix$
```

Un problema con estos semáforos es que no podemos hacer que el número de tickets supere el número de bytes que caben en el buffer del pipe. ¿Qué pasaría si lo hacemos? Una ventaja es que cuando nuestro programa termine su ejecución se cerrarán sus descriptores de fichero incluso si hemos olvidado llamar a `semclose` y el semáforo (el pipe) se destruirá sin dejar recursos perdidos en el sistema. Otra ventaja es que podemos usarlos tanto si creamos procesos que llamen a `fork` como si usamos `pthread_create`.