Sistemas Operativos: Procesos

Especialización en Sistemas Embebidos

Dr. Mario Marcelo Berón

Univerisidad Nacional de San Luis

October 13, 2017

El Problema de la Región Crítica

Concepto

- n procesos $P_i = 1, 2, .., n$ compiten por usar recursos compartidos.
- Cada proceso tiene un fragmento de código, llamado sección crítica (SC), en el que el proceso accede a los datos compartidos.
- Se debe asegurar que cuando un proceso se está ejecutando en su región crítica ningún otro proceso puede estar ejecutando en su región crítica.
- Se agrega código para entrar y acceder a la región crítica.

El Problema de la Región Crítica

Requisitos para la Solución de la Región Crítica

- Solo debe haber un proceso ejecutando en su región crítica (Exclusión Mutua).
- Un proceso que está fuera de su región crítica no debe bloquear a otro que quiere entrar (Progreso).
- Un proceso que quiere entrar a la región crítica no puede esperar indefinidamente (Espera Limitada).

Semáforos

Concepto

Tipo de dato abstracto que soporta tres operaciones atómicas: Inicialización, espera y señal.

Solución

Proporciona una solución a:

- El problema de la región crítica.
- La sincronización de procesos.

Semáforos

```
tipo semaforo=registro
 int valor;
 lista_de_procesos_bloqueados_en_el_semaforo L;
end

variable semaforo S;

wait (S):
 S.valor:=S.valor - 1;
 Si (S.valor < 0) entonces
  añadir a S.L el proceso que invoca la función;
  bloquear este proceso;
 fin_si;
```

Semáforos

```
signal (S):
 S.valor:=S.valor + 1;
 Si (S.valor <= 0) entonces
  extraer un proceso P de S.L;
  desbloquear P e insertarlo en lista de procesos preparados;
 fin_si;
```

Semáforos

Semáforos como Herramienta de Sincronización de Procesos

```
Datos compartidos:
variable semaforo S;
 sem_init (S,1);
 Proceso Pi
  wait (S);
  SC;
  signal (S);
```

Semáforos como Herramienta de Sincronización de Procesos

```
#include <pthread.h>
#include <stdio.h>
 ...
 int x=0;
 sem_t semaforo;
 void *fhilo1(void *arg){
  int i;
   for (i=0;i<3; i++) {
    sem_wait(&semaforo);
    x=x+1;
    sem_post(&semaforo);
    printf ("Suma 1\n");
    sleep (1);
   }
   pthread_exit (NULL);
  }
```

Semáforos como Herramienta de Sincronización de Procesos

```
void *fhilo2(void *arg){
 int i;
  for (i=0; i<3; i++) {
   sem_wait(&semaforo);
   x=x-1;
   sem_post(&semaforo);
   printf ("Resta 1\n");
   sleep (1);
  }
  pthread_exit (NULL);
}
```

Semáforos como Herramienta de Sincronización de Procesos

```
int main(){
 pthread_t hilo1, hilo2;
  printf ("Valor inicial de x: %d \n",x);
  sem_init (&semaforo,0,1);
  pthread_create(&hilo1, NULL,fhilo1, NULL);
  pthread_create(&hilo2, NULL,fhilo2, NULL);
  pthread_join(hilo1,NULL);
  pthread_join(hilo2,NULL);
  sem_destroy (&semaforo);
  printf("Valor final de x: %d \n",x);
  exit(0);
}
```

Semáforos

Semáforos como Herramienta de Sincronización de Procesos

```
Datos compartidos:
 variable semaforo sinc;
 sem_init (sinc,0);

Proceso Pi
 ...
 instrucción A;
 signal (sinc);
 ...

Proceso Pj
 ...
 wait (sinc);
 instrucción B;
 ...
```

Semáforos

Funciones

- int sem_init(sem_t *sem, int shared, int val);
- int sem_destroy(sem_t *sem);
- int sem_wait(sem_t *sem);
- int sem_post(sem_t *sem);

Valores de Retorno

Retornan 0 si terminan exitosamente -1 en otro caso.

Semáforos

Creación de un Semáforo

int sem_init (sem_t *sem, int shared, int val);

- Crea un semáforo indentificado por sem y le asigna el valor inicial val.
- shared indica si el semáforo se comparte entre los hilos de un proceso o entre procesos.

Destrucción de un Semáforo

```
int sem_destroy (sem_t *sem);
```

- Destruye el semáforo identificado por sem.

Semáforos

Operación wait sobre un Semáforo

```
int sem_wait (sem_t *sem);
```

Operación signal sobre un Semáforo

```
int sem_post (sem_t *sem);
```

Procesos

```
#include <unistd.h>
...
 int main() {
  int numero;
  pid_t p;
   printf("Ingrese un numero\n");
   scanf("%d",&numero);
   p= fork();
   if (p==-1) {
    printf("No se pudo crear el proceso");
    exit(1);
   }
   if (p==0)  printf ("Proceso hijo: %d\n",numero*2);
    else printf ("Proceso padre: %d\n",numero*5);
   }
   return 0;
 }
```

Procesos

```
#include <stdio.h>          /* printf()                    */
#include <stdlib.h>         /* exit(), malloc(), free()    */
#include <unistd.h>
#include <sys/types.h>      /* key_t, sem_t, pid_t         */
#include <sys/shm.h>        /* shmat(), IPC_RMID           */
#include <errno.h>          /* errno, ECHILD               */
#include <semaphore.h>      /* sem_open(), sem_destroy(),
                               sem_wait().. */
#include <fcntl.h>          /* O_CREAT, O_EXEC             */
#include <wait.h>
#define TAM 2
```

Procesos

```
int main() {
 int i;
 key_t clave;
 int   identificador;
 sem_t *semaforo1,*semaforo2;
 pid_t pid,pids[TAM];
 int   *vc;
```

Procesos

```
clave=ftok("/dev/null",5);
identificador=shmget(clave,sizeof(int),0644|IPC_CREAT);
if (identificador <0) {
 perror("Error no se pudo acceder al identificador");
 exit(1);
 }
 vc= (int*)shmat(identificador,NULL,0);
 *vc=0;
```

Procesos

```
semaforo1=sem_open("semaforo1",O_CREAT | O_EXCL, 0644, 1);
semaforo2=sem_open("semaforo2",O_CREAT | O_EXCL, 0644, 0);
for(i=0;i<TAM;i++){
 pids[i]=fork();
 if (pids[i]<0) {
  printf("Error en la creación de procesos");
  sem_unlink("semaforo1");
  sem_close(semaforo1);
  sem_unlink("semaforo2");
  sem_close(semaforo2);
 } else if (pids[i]==0) break;
}
```

Procesos

```
if (pids[0]==0) {
 while (1) {
  sem_wait(semaforo1);
   printf("El proceso 0 esta en la region critica\n");
   *vc=*vc+1;
   printf("Valor de la variable compartida: %d\n",*vc);
   sleep(1);
   sem_post(semaforo2);
 }
}
```

Procesos

```
} else if (pids[1]==0) {
  while(1) {
   sem_wait(semaforo2);
   *vc=*vc-1;
   sleep(1);
   sem_post(semaforo1);
  }
 }
```

Procesos

```
else {
 while (pid=waitpid(-1,NULL,0)){
  if (errno==ECHILD)
    break;
 }
 printf("Es el Padre\n");
 printf("Terminaron todos los procesos\n");
 sem_unlink("semaforo1");
 sem_close(semaforo1);
 sem_unlink("semaforo2");
 sem_close(semaforo2);
 }
}
```