Ingeniería en Sistemas Computacionales

Fundamentos de Sistemas Operativos

Actividad 15

IS727272 - Marco Ricardo Cordero Hernández

Jal., 19 de junio de 2023

1.- Del problema de los 4 jugadores de dominó sentados en una mesa jugando dominó, se van turnando para tirar una ficha o pasar si no pueden jugar en el sentido contrario a las manecillas del reloj. Ahora represente a los cuatro jugadores con procesos desde P(0) hasta P(3) y como sincronizaría los turnos usando monitores con variables de condición.

```
monitor juego {

      int turno = 0;



      void esperarTurno(int i) {

            while (turno != i) cwait(condicion);

      }



      void cederTurno(int n) {

            turno = n;

            notifyAll();

      }

}



void jugador (int i) {

      int next = (i + 1) % 4;

      while (jugar) {

            juego.esperarTurno(i);

            tirar || pasar

            juego.cederTurno(next);

      }

}
```

2.- En el caso del problema del productor consumidor que se muestra a continuación, la sincronización se hace con un monitor con notificación definido a partir de semáforos. Para esto fue necesario implantar las funciones:

- `entrar_monitor()`: Permite que solo un proceso/hilo entre a esa sección ya considerada como parte del monitor.
- `cwait()`: Bloquea un proceso/hilo dentro del monitor.
- `cnotify()`: Desbloquea un proceso/hilo que esté bloqueado en la zona de espera del monitor.
- `leave_monitor()`: Permite que otro proceso/hilo entre al monitor.

El problema es que la librería donde se definen esta funciones se extravió, por lo que es necesario re-definirlas.

Utilizando pseudocódigo defina usando semáforos las funciones: `entrar_monitor()`, `cwait()`, `cnotify()` y `leave_monitor()` para que, `agrega_al_buffer()` y `tomar_del_buffer()` se comporten como procedimientos de un monitor.

```
int agrega_al_buffer(e)
{
    enter_monitor();

    while(elementos==MAX_CAPACIDAD)
        cwait();

    ent++;
    end=ent%MAX_CAPACIDAD;
    buffer[ent]=e;
    elementos++;

    cnotify();
    leave_monitor();
}

int tomar_del_buffer()
{
    enter_monitor();

    while(elementos==0)
        cwait();

    sal++;
    sal=sal%MAX_CAPACIDAD;
    e=buffer[sal];
    elementos-- ;

    cnotify();
    leave_monitor();
    return(e);
}

int Productor()
{
    int e;
    while(forever)
    {
        e=Produce_elemento();
        agrega_al_buffer(e);
```

```
    }
}

int Consumidor()
{
    int e;
    while(forever)
    {
        e=tomar_del_buffer();
        consumir(e);
    }
}
```

R: Pseudocódigo

```
Semaphore exmut = 1;

Semaphore cq = 0;


void enter_monitor() {

      semwait(exmut);

}


void leave_monitor() {

      semsignal(exmut);

}


void cwait() {

      semsignal(exmut);

      semwait(cq);

      semwait(exmut);

}


void cnotify() {

      if (cq.cont < 0) // ¿Hay procesos bloqueados en cq?

            semsignal(cq);

}
```

3.- ¿Qué aprendiste?

Se vieron formas prácticas de implementar monitores a través de ejemplos revisados previamente, así como una porción adicional de refuerzo para el tema pasado de semáforos. Adicional a esto, se revivió un poco del conocimiento adquirido en programación orientada a objetos a través de Java y algunos modificadores que existen para sus clases; también, se introdujo el tema de métodos síncronos a través del mismo lenguaje de programación mencionado.