CC4302 – Sistemas Operativos
Auxiliar 3

Profesor: Luis Mateu
Auxiliar: Diego Madariaga

8 de abril de 2020

1. P2 Control 1, 2010/1

Un grupo de programadores alternan su jornada diaria programando y comiendo pizza. Los hambrientos van a la pizzería, piden su pizza y esperan en la tienda hasta recibir su pizza. Los hackers son más eficientes porque ordenan por teléfono la pizza, luego programan y más tarde van a retirar su pizza. Estas actividades se ven reflejadas en estos procedimientos:

```
int hambriento() {
  for(;;) {
    Pizza* pizza = comprarPizza();
    comer(pizza);
    programar();
} }
```

```
int hacker() {
  for(;;) {
    Pizza* pizza;
    Orden* orden= ordenarPizza();
    programar1();
    pizza= esperarPizza(orden);
    comer(pizza);
    programar2();
} }
```

La implementación actual de la pizzería es ineficiente porque hornea una sola pizza a la vez (a pesar de que el horno permite hornear 4 pizzas simultáneamente) y las pizzas de los hackers solo comienzan a prepararse una vez que el cliente llega a la tienda:

```
typedef void Orden;
Horno* horno;
nSem sem;
Pizza *obtenerPizzaCruda();
void hornear(Horno* horno, Pizza* pizza_vec, int n_pizzas);
```

```
void iniciarPizzeria() {
sem= nMakeSem(1);}
Orden* ordenarPizza() {
  return NULL;}
Pizza* esperarPizza(Orden* orden) {
  return comprarPizza();}
```

```
Pizza* comprarPizza() {
  Pizza *pizza= obtenerPizzaCruda();
  Pizza *pizza_vec[1];
  pizza_vec[0]= pizza;
  nWaitSem(sem);
  hornear(horno, pizza_vec, 1);
  nSignalSem(sem);
  return pizza;
}
```

Usando los monitores de nSystem, implemente eficiente las siguientes funciones: `iniciarPizzeria`, `ordenarPizza`, `esperarPizza` y `comprarPizza`. Especifique también la estructura de datos `Orden`. Los demás procedimientos son dados. Ud. necesitará crear un thread adicional para operar el horno.

Considere que hornear es la única operación que toma mucho tiempo en ejecutarse. Restricciones (ordenadas de mayor a menor importancia):

1. La invocación de hornear debe ocurrir en exclusión mutua.
2. Invoque hornear para cocinar 4 pizzas suministradas en el arreglo `pizza_vec`. Si no han llegado suficientes clientes, puede hornear menos de 4 pizzas, pero hornee al menos una pizza. El horno no se abre hasta que hornear termina.
3. Su solución no debe producir hambruna.
4. Si el horno está disponible, comience a hornear las pizzas de los hackers antes de la invocación de `esperarPizza`, pero después del retorno de `ordenarPizza`.

2. P2 Control 1, 2017/2

Una empresa se dedica al transporte de contenedores. Sus clientes son tareas de nSystem que invocan la función transportar para solicitar el transporte del contenedor cont desde la ciudad orig hasta la ciudad dest. La implementación actual de esta función usa un semáforo para coordinar el uso del único camión que posee la empresa:

```
nSem mutex;                  //  = nMakeSem(1);
Camion *c;                   //  el único camión
Ciudad *ubic= &Santiago;     //  su ubicación actual
void transportar(Contenedor *cont,
                 Ciudad *orig, Ciudad *dest) {
  nWaitSem(mutex);
  conducir(c, ubic, orig);
  cargar(c, cont);
  conducir(c, orig, dest);
  descargar(c, cont);
  ubic= dest;                //  se actualiza la ubicación del camión
  nSignalSem(mutex);
}
```

Las funciones conducir, cargar y descargar son dadas y toman mucho tiempo. Observe que transportar espera cuando el único camión de la empresa está siendo ocupado por otra llamada concurrente de transportar. Antes de cargar el camión con el contenedor hay que conducir el camión desde su ubicación actual a la ciudad de origen del contenedor. La función solo retorna una vez que el contenedor se cargó en la ciudad de origen, se transportó y se descargó en su ciudad de destino. La empresa de transportes acaba de aumentar su flota a 8 camiones lo que le permite transportar hasta 8 contenedores en paralelo. Se le pide a Ud. reprogramar la función transportar para que se usen eficientemente estos 8 camiones. Ud. dispone de:

```
#define P 8
Camion *camiones[P];
double distancia(Ciudad *orig, Ciudad *dest);
```

Inicialmente todos los camiones están en Santiago. Agregue otras variables globales y programe una función de inicialización. Por simplicidad los transportes pendientes pueden ser atendidos en cualquier orden.

Restricciones:

- Un camión puede ser usado para un solo transporte a la vez.
- Un camión no puede permanecer ocioso si existe un transporte pendiente.
- Si en el momento de invocar transportar hay varios camiones ociosos, por razones de eficiencia Ud. debe elegir el camión cuya ubicación actual sea la más cercana a la ciudad de origen del contenedor. Use la función distancia para elegir el camión más cercano.

Resuelva el problema de sincronización usando los monitores de nSystem.