]Alberto Josué Ortega Girón 200718898 [USAC]Sistemas Operativos 1

# 1 Problema Productor Consumidor

El problema Productor Consumidor es un ejemplo clásico de un algoritmo que hace uso de concurrencia, recibe este nombre debido a que funciona de la siguiente manera:
Se encuentra el productor y pone en una pila el producto, este producto es almacenado en la pila hasta que el consumidor lo toma.
En nuestro caso el producto son datos generados por algun proceso, que son puestos en una cola (buffer) y son tomados por otro proceso. Se debe garantizar que:

1. El productor puede generar sus datos en cualquier momento.
2. El consumidor puedo obtener un dato solamente cuando hay uno
3. Para el intercambio de Datos se utiliza un solo buffer, donde el productor y el consumidor tienen acceso.
4. Ningun dato puede tener dos estados al mismo tiempo, es decir es consumido o es producido.

```
producer:                  consumer:
  forever                    forever
    spacesLeft.wait()          itemsReady.wait()
    produce(item)              mutex.wait()
    mutex.wait()               take(item)
    place(item)                mutex.signal()
    mutex.signal()             consume(item)
    itemsReady.signal()        spacesLeft.signal()
```

## 1.1 Aplicación en un Caso Concreto

Una aplicacón en un caso de "la vida real" es el que se da en un *Web Service*, El Web Service recibe la petición de información *http*, esta petición es puesta en una cola interna.
El hilo que se se esta encargando de manejar la petición toma estos datos y se pone a trabajar con ellos, esto se da en cada petición *http* que se realiza, la petición es puesta en cola y el hilo del servidor se encarga de tomar la información que será procesada de regreso al cliente.

# 2 Algoritmo de Dekker

El algoritmo de Dekker es un algoritmo concurrente para exclusión mutua(*mutex*), diseñado por Esdger Dijkstra.Este permite a dos procesos o hilos compartir recursos sin conflictos, fue uno de los primeros de este tipo. Basicamente lo que el algoritmo hace es que si dos procesos intentan entrar a la sección critica siultáneamente el algoritmo elige un proceso según una variable de turno.
Existen cinco versiones del algoritmo:

1. **Versión 1:** *Alternancia Estricta* EL problema con esta version es que hace que los procesos se acomplen a la fuerza, siendo el caso donde un proceso es mas lento que el otro causando que se atrase.

```
P0                                P1
a:  loop                              loop
b:    wait until v equals P0            wait until v equals P1
c:    critical section                  critical section
d:    set v to P1                       set v to P0
e:    non-critical section              non-critical section
f:  endloop                           endloop
```

2. **Versión 2:** *Problema de Interbloqueo* Los procesos pueden quedar en el mismo estado y no salir de ahí *deadlock*.

```
P0                                P1
a:  loop                              loop
b:    wait until v1 equals false        wait until v0 equals false
c:    set v0 to true                    set v1 to true
d:    critical section                  critical section
e:    set v0 to false                   set v1 to false
f:    non-critical section              non-critical section
g:  endloop                           endloop
```

3. **Versión 3:** *Colisión en la región critica* No se garantiza la exclusión mutua.

```
P0                                P1
a:  loop                              loop
b:    set v0 to true                    set v1 to true
c:    wait until v1 equals false        wait until v0 equals false
```

```
d:    critical section                  critical section
e:    set v0 to false                   set v1 to false
f:    non-critical section              non-critical section
g:  endloop                           endloop
```

4. **Versión 4:** *Postergación indefinida* Los procesos pueden quedar esperando mutuamente un evento que probablemente nunca sucederá (*livelock*).

```
P0                                    P1
a:  loop                                  loop
b:    set v0 to true                      set v1 to true
c:    repeat                              repeat
d:      set v0 to false                     set v1 to false
e:      set v0 to true                      set v1 to true
f:    until v1 equals false               until v0 equals false
g:    critical section                    critical section
h:    set v0 to false                     set v1 to false
i:    non-critical section                non-critical section
j:  endloop                           endloop
```

5. **Versión 5:** Esta versión es una mezcla de la primera con la cuarta, creando así el algoritmo que mas funciona.

```
Initially:  v0 is equal false
v1 is equal false
v  is equal P0 o P1

P0                                    P1
a:  loop                                  loop
b:    set v0 to true                      set v1 to true
c:    loop                                loop
d:      if v1 equals false exit             if v0 equals false exit
e:      if v equals P1                      if v equals P0
f:        set v0 to false                     set v1 to false
g:        wait until v equals P0              wait until v equals P1
h:        set v0 to true                      set v1 to true
i:      fi                                  fi
j:    endloop                             endloop
k:    critical section                    critical section
l:    set v0 to false                     set v1 to false
```

```
m:    set v to P1                    set v to P0
n:    non-critical section           non-critical section
```