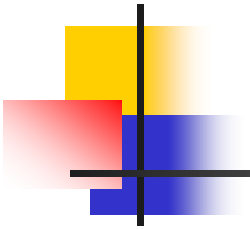
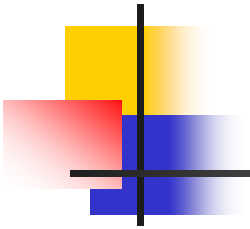


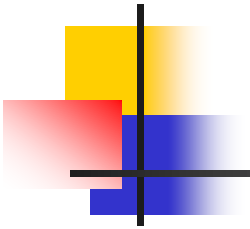
Sistemas Operativos 2021-2022

Productor-Consumidor Problema de concurrencia



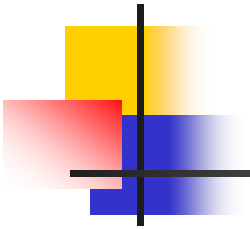
Productor-Consumidor
Descripción de los hilos

- Tenemos dos hilos
  1. ***Productor***
     - Produce datos de uno en uno y los va insertando en una cola FIFO
  2. ***Consumidor***
     - Consume , de uno en uno, los datos que va almacenando el productor → extrae de la cola FIFO
- Es un problema MODELO que se da en casi todas las comunicaciones (incluido la red Internet)



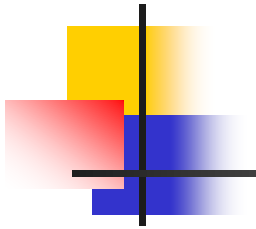
Productor-Consumidor
Detalles de implementación

- Vamos a crear dos funciones para sendos hilos:
  - **La función *Productor*() .-** que ejecutará el hilo del mismo nombre
  - **La función *Consumidor*() .-** que ejecutará el hilo del mismo nombre
- Por supuesto, también tendremos **la función *main*()** → Comienzo del programa con un único hilo principal



Productor-Consumidor
Detalles de implementación

- Declararemos las siguientes variables globales:
  - *cola_circular* → Una cola FIFO, de enteros, implementada mediante un vector circular
  - *cabeza* → variable entera con la cabeza de la cola FIFO
  - *Cola* → variable entera con la cola de la cola FIFO
  - *n_elementos* → variable entera con el nº de elementos de la cola FIFO



Productor-Consumidor Implementación

- **Primer paso**: Implementar el programa sin tener en cuenta los problemas de la concurrencia
- Este paso se muestra, a continuación completamente realizado

Productor-Consumidor Implementación

```
// Variables globales
int cola_circular[MAX_TAMANIO];
int cabeza=0, cola=0, n_elementos=0;

// Función para el hilo Productor
void Productor()
{
   // Variable local con el dato generado
   int dato;
   // El Productor está constantemente produciendo datos
   // (de uno en uno)
   while (true)
   {
```

Productor-Consumidor Implementación

```
    // Comprueba que la cola circular tenga aún sitio
    // (no esté llena)
    if (n_elementos < MAX_TAMANIO)
    {
       // La cola circular no está llena.
       // Produce un dato cualquiera
       dato = random();
      // Lo inserta en la cola circular (por la cola)
       cola_circular[cola] = dato;
       // Avanza la cola (de forma circular)
       cola = (cola + 1) % MAX_TAMANIO;
       // Incrementa el número de elementos
       n_elementos++;
    }
}
```

Productor-Consumidor Implementación

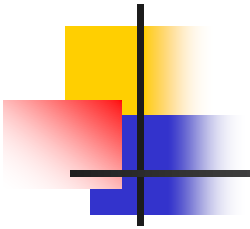
```
// Función para el hilo Consumidor
void Consumidor()
{
    // Variable local con el dato extraído (consumido)
    int dato;

    // El Consumidor está constantemente recogiendo datos
    // (de uno en uno)
    while (true)
    {
```

Productor-Consumidor Implementación

```
// Comprueba que la cola circular tenga elementos
// (no esté vacía)
if (n_elementos > 0)
{  // La cola circular no está vacía.
   // Obtiene un dato de la cola circular
   // (por la cabeza)
   dato = cola_circular[cabeza];
   // Avanza la cabeza
   cabeza = (cabeza + 1) % MAX_TAMANIO;
   // Decrementa el número de elementos
   n_elementos--;
   // Hace algo con el dato actual.
   // Por ejemplo imprimirlo
   System.out.println(dato);
}
```

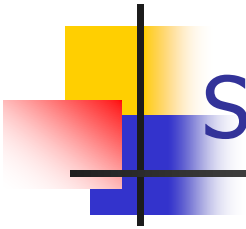
Productor-Consumidor Implementación

```
int main()
{
    // Se crean y lanzan los dos hilos (Productor y
    // Consumidor)
    . . .
}
```



Productor-Consumidor
Cuestiones de implementación

- ¿Qué recursos globales compartidos hay?
  - *cola_circular*
  - *n_elementos*
- Cuando el Productor tiene la cola llena y el Consumidor tiene la cola vacía, ¿qué hacen?
  - No hacen nada, están constantemente dando vueltas al bucle
  - ¿Qué tipo de espera están realizando, activa o pasiva?
    - Activa ☹ ☹ ☹



Productor-Consumidor
Solución al problema de la exclusion mutua

- **Segundo paso:** El programador identifica las **secciones críticas del Código**
- **¡Hágase!**

Productor-Consumidor
Secciones críticas

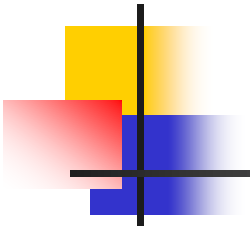
- Función Productor()

```
// Comprueba que la cola circular tenga aún sitio
// (no esté llena)
if (n_elementos < MAX_TAMANIO) // SC1
{
   // La cola circular no está llena.
   // Produce un dato cualquiera
   dato = random();
  // Lo inserta en la cola circular (por la cola)
   cola_circular[cola] = dato; // SC2
   // Avanza la cola (de forma circular)
   cola = (cola + 1) % MAX_TAMANIO;
   // Incrementa el número de elementos
   n_elementos++;  // SC3
```

Productor-Consumidor
Secciones críticas

- Función Consumidor()

```
// Comprueba que la cola circular tenga elementos
// (no esté vacía)
if (n_elementos > 0)  // SC4
{  // La cola circular no está vacía.
   // Obtiene un dato de la cola circular
   // (por la cabeza)
   dato = cola_circular[cabeza];  // SC5
   // Avanza la cabeza
   cabeza = (cabeza + 1) % MAX_TAMANIO;
   // Decrementa el número de elementos
   n_elementos--;  // SC6
   // Hace algo con el dato actual.
   // Por ejemplo imprimirlo
   System.out.println(dato);
```



Productor-Consumidor
Exclusión mutua con semáforos binarios

- **Tercer paso:** Implementar las **secciones de *entrada*** y ***salida*** de las *secciones crítica*s con el mecanismo/herramienta elegido → En este caso con **semáforos binarios**
- ¿Cuántos semáforos binarios necesitamos?
  - DOS → ***mutex_cola*** y ***mutex_n_elementos***
- ¿Qué valor inicial hay que poner en su contador?
  - El valor UNO
- ¡Impleméntense todas las *secciones de entrada* y *salida*!

Productor-Consumidor
Implementación con exclusion mutua

```
// Variables globales
int cola_circular[MAX_TAMANIO];
int cabeza=0, cola=0, n_elementos=0;
Semaphore mutex_cola, mutex_n_elementos;

// Función para el hilo Productor
void Productor()
{
   // Variable local con el dato generado
   int dato;
   // El Productor está constantemente produciendo datos
   // (de uno en uno)
   while (true)
   {
```

Productor-Consumidor
Implementación con exclusión mutua

```
// Comprueba que la cola circular tenga aún sitio
// (no esté llena)
P(mutex_n_elementos); // Sec. entrada SC1
if (n_elementos < MAX_TAMANIO)  // SC1
{
   // La cola circular no está llena
   V(mutex_n_elementos); // Sec. salida SC1
   // Produce un dato cualquiera
   dato = random();
  // Lo inserta en la cola circular (por la cola)
   P(mutex_cola); // Sec. entrada SC2
   cola_circular[cola] = dato; // SC2
   V(mutex_cola); // Sec. Salida SC2
```

Productor-Consumidor
Implementación con exclusión mutua

```
            // Avanza la cola (de forma circular)
            cola = (cola + 1) % MAX_TAMANIO;
            // Incrementa el número de elementos
            P(mutex_n_elementos); // Sec. entrada SC3
            n_elementos++; // SC3
            V(mutex_n_elementos); // Sec. salida SC3
         } // Del if (n_elementos < … )
         else
         {
            V(mutex_n_elementos); // Sec. salida SC1
         }
      } // Del while(true)
   } // De la función Productor
```

Productor-Consumidor
Implementación con exclusión mutua

```
// Función para el hilo Consumidor
void Consumidor()
{
    // Variable local con el dato extraído (consumido)
    int dato;

    // El Consumidor está constantemente recogiendo datos
    // (de uno en uno)
    while (true)
    {
```

Productor-Consumidor
Implementación con exclusión mutua

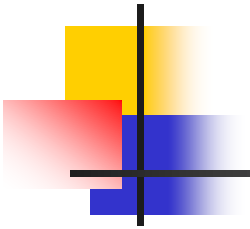
```
// Comprueba que la cola circular tenga elementos
// (no esté vacía)
P(mutex_n_elementos); // Sec. entrada SC4
if (n_elementos > 0) // SC4
{  // La cola circular no está vacía.
   V(mutex_n_elementos); // Sec. salida SC4
   // Obtiene un dato de la cola circular
   // (por la cabeza)
   P(mutex_cola); // Sec. entrada SC5
   dato = cola_circular[cabeza]; // SC5
   V(mutex_cola); // Sec. salida SC5
   // Avanza la cabeza
   cabeza = (cabeza + 1) % MAX_TAMANIO;
```

Productor-Consumidor
Implementación con exclusión mutua

```
            // Decrementa el número de elementos
            P(mutex_n_elementos); // Sec. entrada SC6
            n_elementos--;  // SC6
            V(mutex_n_elementos); // Sec. salida SC6
            // Hace algo con el dato actual.
            // Por ejemplo imprimirlo
            System.out.println(dato);
         } // Del if (n_elementos > 0)
         else
         {
            V(mutex_n_elementos); // Sec. salida SC4
         }
      } // Del while(true)
   } // De la función Consumidor
```

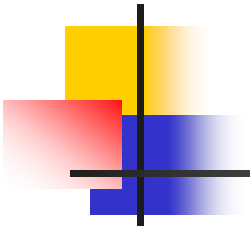
Productor-Consumidor
Implementación con exclusión mutua

```
int main()
{
    // Se inicializan los semáforos
    Init(mutex_cola, 1);
    Init(mutex_n_elementos, 1);

    // Se crean y lanzan los dos hilos (Productor y
    // Consumidor)
    . . .
}
```



Productor-Consumidor
Sincronización con semáforos binarios

- **Cuarto paso:** Identificar si es necesario realizar alguna ***sincronización***
- Recordemos que tanto el *Productor* como el *Consumidor* tienen que esperar por "algo"
  - El *Productor* debe esperar si la cola está llena
  - El *Consumidor* debe esperar si la cola está vacía
- Las esperas anteriores son activas ☹ ☹ ☹



Productor-Consumidor
Sincronización con semáforos binarios

- SOLUCIÓN: Realizar DOS sincronizaciones:
  1. El *Consumidor* espera si la cola está vacía y el *Productor* le "avisa" cuando haya UN dato (estando la cola vacía)
  2. El *Productor* espera si la cola está llena y el *Consumidor* le "avisa" cuando consuma UN dato (estando la cola llena)
- ¿Cuántos semáforos binarios necesitamos?
  - DOS → ***sinc_cola_vacia*** y ***sinc_cola_llena***
- ¿Qué valor inicial hay que poner en su contador?
  - El valor CERO
- ¡Impleméntense las dos sincronizaciones!

Productor-Consumidor
Implementación añadiendo sincronizacion

```
// Variables globales
int cola_circular[MAX_TAMANIO];
int cabeza=0, cola=0, n_elementos=0;
Semaphore mutex_cola, mutex_n_elementos;
Semaphore sinc_cola_vacia, sinc_cola_llena;

// Función para el hilo Productor
void Productor()
{
   // Variable local con el dato generado
   int dato;
   // El Productor está constantemente produciendo datos
   // (de uno en uno)
   while (true)
   {
```

Productor-Consumidor
Implementación añadiendo sincronizacion

```
// Comprueba que la cola circular tenga aún sitio
// (no esté llena)
P(mutex_n_elementos); // Sec. entrada SC1
if (n_elementos < MAX_TAMANIO)  // SC1
{
   // La cola circular no está llena
   V(mutex_n_elementos); // Sec. salida SC1
   // Produce un dato cualquiera
   dato = random();
  // Lo inserta en la cola circular (por la cola)
   P(mutex_cola); // Sec. entrada SC2
   cola_circular[cola] = dato; // SC2
   V(mutex_cola); // Sec. Salida SC2
```

Productor-Consumidor
Implementación añadiendo sincronizacion

```
        // Avanza la cola (de forma circular)
        cola = (cola + 1) % MAX_TAMANIO;
        // Incrementa el número de elementos
        P(mutex_n_elementos); // Sec. entrada SC3
        // Si no hay elementos en la cola avisamos
        // al Consumidor de que ya no está vacía
        if (n_elementos == 0)
           V(sinc_cola_vacia);
        n_elementos++; // SC3
        V(mutex_n_elementos); // Sec. salida SC3
    } // Del if (n_elementos < … )
```

Productor-Consumidor
Implementación añadiendo sincronizacion

```
        else
        {
            // La cola está llena
            V(mutex_n_elementos); // Sec. salida SC1
            // Debemos esperar a que se consuma un dato
            P(sinc_cola_llena);
        }
    } // Del while(true)
} // De la función Productor
```

Productor-Consumidor
Implementación añadiendo sincronizacion

```
// Función para el hilo Consumidor
void Consumidor()
{
    // Variable local con el dato extraído (consumido)
    int dato;

    // El Consumidor está constantemente recogiendo datos
    // (de uno en uno)
    while (true)
    {
```

Productor-Consumidor
Implementación añadiendo sincronizacion

```
// Comprueba que la cola circular tenga elementos
// (no esté vacía)
P(mutex_n_elementos); // Sec. entrada SC4
if (n_elementos > 0) // SC4
{  // La cola circular no está vacía.
   V(mutex_n_elementos); // Sec. salida SC4
   // Obtiene un dato de la cola circular
   // (por la cabeza)
   P(mutex_cola); // Sec. entrada SC5
   dato = cola_circular[cabeza]; // SC5
   V(mutex_cola); // Sec. salida SC5
   // Avanza la cabeza
   cabeza = (cabeza + 1) % MAX_TAMANIO;
```

Productor-Consumidor
Implementación añadiendo sincronizacion

```
            // Decrementa el número de elementos
            P(mutex_n_elementos); // Sec. entrada SC6
            // Si la cola estaba llena avisamos al
            // Productor de que ya no está llena
            if (n_elementos == MAX_TAMANIO)
               V(sinc_cola_llena);
            n_elementos--;  // SC6
            V(mutex_n_elementos); // Sec. salida SC6
            // Hace algo con el dato actual.
            // Por ejemplo imprimirlo
            System.out.println(dato);
         } // Del if (n_elementos > 0)
```

Productor-Consumidor
Implementación añadiendo sincronizacion

```
        else
        {
            // La cola circular está vacía
            V(mutex_n_elementos); // Sec. salida SC4
            // Debemos esperar a que se produzca un dato
            P(sinc_cola_vacia);
        }
    } // Del while(true)
} // De la función Consumidor
```

Productor-Consumidor
Implementación añadiendo sincronizacion

```
int main()
{
    // Se inicializan los semáforos
    Init(mutex_cola, 1);
    Init(mutex_n_elementos, 1);
    Init(sinc_cola_llena, 0);
    Init(sinc_cola_vacia, 0);

    // Se crean y lanzan los dos hilos (Productor y
    // Consumidor)
    . . .
}
```