

# ***Introducción a los Sistemas Operativos***

## **Introducción a IPC**



# Concurrencia y Paralelismo

- ✓ Es común dividir un solución en diferentes “tareas o programas” que, independientemente o colaborativamente, solucionan el problema.
- ✓ Es común contar con un conjunto de procesadores para ejecutar nuestras soluciones de Forma Paralela



# Modelo de Procesos Concurrentes

## ☑ **Productor**

repeat

...

produce un *elemento*

...

Enviar *elemento* a  
consumidor

until false;

## ☑ **Consumidor**

repeat

Esperar que haya  
*elemento* a consumir

...

consume el *elemento*

...

until false;



# Concurrencia y Paralelismo

- ✓ Es común dividir un programa en diferentes “tareas” que, independientemente o colaborativamente, solucionan el problema
- ✓ Es común contar con un conjunto de procesadores para ejecutar nuestras soluciones de Forma Paralela
- ✓ Necesidades:
  - Comunicar Procesos
    - Compartir Información entre Procesos
  - Sincronizar Procesos
    - Acceso a información compartida



# Definición - Condición de carrera

- ✓ El resultado final depende del orden en que se ejecuten los procesos.
- ✓ Ejemplo:
  - ✓ Dos procesos P1 y P2 comparten la variable b y c.
  - ✓ Están inicializadas  $b=1$ ,  $c=2$
  - ✓ P1 ejecuta  $b=b+c$
  - ✓ P2 ejecuta  $c=b+c$
  - ✓ El valor final dependerá del orden de ejecución



# Definición - Sección Crítica

- ✓ Sección de código en un proceso que accede a recursos compartidos con otros procesos y que **no** puede ser ejecutada mientras otro proceso esté en **su** sección crítica
  - Se protegen datos, no código
  - El SO también presenta secciones críticas.



# Definición - Sección Crítica (cont.)

## ✓ Condiciones:

- **Exclusión Mutua:** Dos procesos no pueden estar simultáneamente dentro de sus SC.
- No se pueden hacer suposiciones en cuanto a velocidades o cantidad de CPUs
- Ningún proceso que se ejecute fuera de su SC puede bloquear otros procesos
- **Espera Limitada:** Ningún proceso tiene que esperar “por siempre” para entrar en su SC.



# *Definición - Sección Crítica (cont.)*

Estructura General de un Proceso:

Repeat

No Critical Section

Entry section

Critical Section

Exit section

No Critical Section

Until false;





- ✓ Inter-Process Communication
- ✓ Mecanismo para comunicar y sincronizar procesos.
- ✓ Consiste de:
  - ✓ Semáforos
  - ✓ Sistema de mensajes
  - ✓ Memoria Compartida



# Semáforos

- ✓ Es una herramienta de sincronización
- ✓ Sirve para solucionar el problema de la sección crítica.
- ✓ Sirve para solucionar problemas de sincronización.



# Semáforos (cont.)

☑ Es una variable entera

✓ Inicializada en un valor no negativo

☑ Dos operaciones:

✓ wait (también llamadas **down** o **p**)

♦ Decrementa el valor. El proceso no puede continuar ante un valor negativo, se bloquea.

✓ signal (también llamadas **up** o **v**)

♦ Incrementa el valor. Desbloqueo de un proceso que espere en el semaforo

☑ Operaciones atómicas

✓ Cuando un proceso modifica su valor, otros procesos no pueden modificarlo simultáneamente.



# Semáforos (cont.)

## *Esquema general de implementación*

☑ Wait(S)

While  $S \leq 0$  do

no op;

$S := S - 1$

☑ Signal(S)

$S := S + 1$



# Semáforos (cont.)

☑ Proceso P1

//código

wait(sem1)

// continua

☑ Proceso P2

//código

signal(sem1)

// continua



# Sol. Productor/consumidor usando semáforos

- ✓ Consideremos un pool de  $n$  buffers.
- ✓ Se usan los siguientes semáforos:  
vacíos, llenos y mutex.
- ✓ *vacíos* cuenta la cantidad de buffers vacíos (inicializado en  $n$ )
- ✓ *llenos* (inicializado en 0) y cuenta la cantidad de buffers llenos.
- ✓ *Mutex* es inicializado en 1.



# *Sol. Productor/consumidor usando semáforos*

## **PRODUCTOR**

```
repeat
...
produce un ítem en
nextp
...
wait(vacios);
wait(mutex);
...
agrega nextp al buffer
...
signal(mutex);
signal(llenos);
until false;
```

## **CONSUMIDOR**

```
repeat
    wait(llenos);
    wait(mutex);
...
saca el ítem del buffer a
    nextc
...
signal(mutex);
signal(vacias);
...
consume el ítem en
    nextc
...
until false;
```



# *Pasaje de Mensajes*

- ✓ Dos primitivas básicas
  - ✓ send y receive.
- ✓ Se establece un link de comunicación entre dos o mas procesos.
- ✓ Ese link puede ser
  - unidireccional o bidireccional
  - simétrico o asimétrico.
- ✓ La operación send puede ser:
  - por copia o por referencia;
- ✓ Los mensajes de medida fija o variable.





# *Pasaje de Mensajes (cont.)*

## ☑ Comunicación directa:

- Cada proceso que quiere comunicarse con otro deberá explícitamente indicar quien recibe o manda la comunicación
- Send (P, mensaje) Envía un mensaje al proceso P
- Receive (Q, mensaje) Recibe un mensaje desde el proceso Q



# *Pasaje de Mensajes (cont.)*

## ☑ **Productor**

repeat

...

produce un ítem en  
nextp

...

send(consumidor, nextp);

until false;

## ☑ **Consumidor**

repeat

receive(producer, nextc);

...

consume el ítem en  
nextc

...

until false;



# *Pasaje de Mensajes (cont.)*

## ☑ Comunicación Directa – Naming Asimétrico

- Send (P,message)      Envía un mensaje a P
- Receive (id, message)      Recibe un mensaje desde cualquier proceso. Id identifica el nombre del proceso con el que se ha establecido la comunicación.



# *Pasaje de Mensajes (cont.)*

## ☑ Comunicación Indirecta

- usa un mailbox o port
- Un mailbox puede verse como un objeto donde se ponen y sacan mensajes
- Cada mailbox tiene una identificación única

Send (A, mensaje) Envía un mensaje al mailbox A

Receive (A, mensaje) Recibe un mensaje desde el mailbox A



# *Pasaje de Mensajes (cont.)*

## ☑ **Productor**

repeat

...

produce un ítem en  
nextp

...

send(Mailbox, nextp);  
until false;

## ☑ **Consumidor**

repeat

receive(Mailbox,nextc)  
;

...

consume el ítem en  
nextc

...

until false;



# *Pasaje de Mensajes (cont.)*

- ☑ Comunicación Indirecta (cont.):
  - El sistema operativo debe proveer los mecanismo para que un proceso pueda:
    - Crear un nuevo mailbox
    - Compartir un mailbox
    - Enviar y recibir mensajes a través del mailbox
    - Destruir un mailbox



# *Pasaje de Mensajes (cont.)*

## ☑ Comunicación Indirecta (cont.):

- Capacidad del Link: ¿Cuántos mensajes puede mantener el link?
  - Cero: no puede haber mensajes esperando. Es lo que se llama Rendezvous: el emisor debe esperar que el receptor reciba el mensaje para poder mandar otro. Hay sincronismo.
  - Capacidad limitada: la cola tiene una longitud finita.
  - Capacidad ilimitada: tiene una longitud “infinita”. El emisor nunca espera.



# *Pasaje de Mensajes (cont.)*

- ✓ Emisor y receptor pueden ser bloqueantes o no bloqueantes.
- ✓ Caso receptor:
  - Si el mensaje ya se mandó, lo recibe.
  - Si no hay mensajes: o se bloquea o continua sin recepción
- ✓ Caso emisor:
  - Si hay un proceso esperando o hay capacidad en el link, envía
  - Si no hay un proceso esperando o el link se lleno: o se bloquea o continua su ejecución sin enviar





# Memoria Compartida

- ✓ Tradicionalmente cada proceso cuenta con su espacio de direcciones
  - ✓ Direcciones Virtuales
- ✓ Un proceso NO puede acceder al espacio de otro
  - ✓ Protección de la memoria
- ✓ Los procesos siguen “viendo” un espacio virtual
  - ✓ Cada región compartida puede estar en diferente lugar del Espacio de Direcciones de cada proceso.



# Memoria Compartida (cont.)

- ☑ La técnica permite a dos o mas procesos compartir un segmento de memoria.
- ☑ Permite la transferencia de datos entre procesos
  - ✓ Comunicación
- ☑ Se requieren mecanismos de Sincronización
  - ✓ Semáforos

