




SISTEMAS OPERATIVOS I

EL PROBLEMA DE LECTORES/ESCRITORES

DEL NEGRO, HUGO MARCELO

KALAYDJIAN, FRANCISCO

 UNIVERSIDAD CAECE	SISTEMAS OPERATIVOS I
<i>El problema de lectores/escritores</i>	

[El problema de lectores/escritores](#)

[Análisis del problema de lectores/escritores](#)

[Diagrama de clases](#)

[Manual Simulación](#)

[Requisitos](#)

[Iniciar la simulación](#)

[AGREGAR UN NUEVO PROCESO](#)


[MONITOREAR COLAS DE PLANIFICACIÓN](#)

[TERMINAR](#)

[PLANIFICADOR](#)

[LOG EJECUCIÓN](#)

[SIMULACIÓN](#)

 UNIVERSIDAD CAECE	SISTEMAS OPERATIVOS I
<i>El problema de lectores/escritores</i>	

El problema de lectores/escritores

El problema consiste en tener una **base de datos** como recurso crítico, procesos **lectores** y procesos **escritores** que deberán ser sincronizados para realizar operaciones sobre la misma. Los escenarios son los siguientes:

- Solamente puede haber un escritor al mismo tiempo, si un escritor o un lector quiere utilizar la base de datos y hay un escritor utilizándose, entonces debe esperar.
- Pueden haber varios lectores utilizando la base de datos al mismo tiempo, pero si un escritor quiere usar la base de datos mientras los lectores la usan, deberá esperar.
- El último lector en usar la base de datos debe habilitarla para que pueda ingresar el siguiente escritor que estaba esperando la base de datos o en todo caso al siguiente proceso que requiera usarla.

Análisis del problema de lectores/escritores

Antes de comenzar con la solución del simulador intentamos resolver el problema de los lectores/escritores utilizando una solución mediante semáforos como prueba de concepto para sincronizar distintas instancias de procesos lectores y escritores. La solución sincronizada que resultó efectiva es la siguiente.

contadorLectores	0
escribirMutex	1
sumarMutex	1

LECTOR	ESCRITOR
wait(sumarMutex)	wait(escribir)
contadorLectores++	
if (contadorLectores == 1) { wait(escribir) }	
signal(sumarMutex)	escribir()
leer()	
wait(sumarMutex)	signal(escribir)
contadorLectores--	
if (contadorLectores) == 0 {	
signal(escribir)	
}	
signal(sumarMutex)	

Teniendo el ejemplo sincronizado era posible deducir el funcionamiento de las diferentes colas de **ejecución**, **listos**, **nuevos** (correspondiente al planificador) y las colas correspondientes a los semáforos, según la estructura de datos citada en el libro de William Stallings.

```
typedef struct {  
    int count;  
    queueType queue;  
} SEMAPHORE;
```

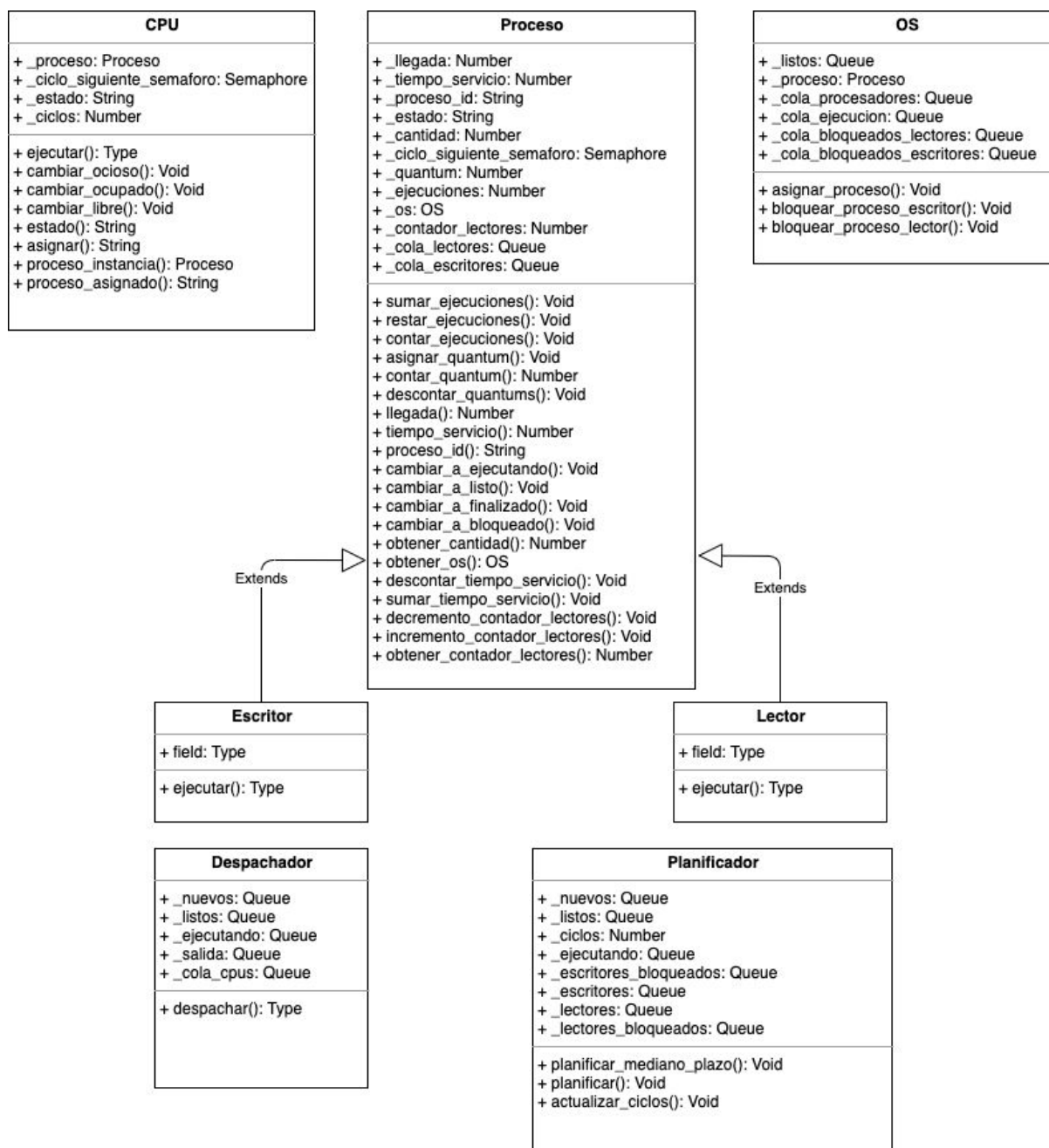
```
void semWait(semaphore s) {  
    s.count--;  
    if (s.count < 0) {  
        place P in s.queue;  
        block P;  
    }  
}
```

```
void semSignal(semaphore s) {  
    s.count++;  
    if (s.count ≤ 0) {  
        remove P from s.queue;  
        place P on ready list;  
    }  
}
```

Entendiendo como se encolan los procesos en la estructura de datos de los semáforos da una idea de cómo se podrían simular los movimientos y bloqueos de todo el problema entero.


Diagrama de clases

A continuación se expone el diagrama de clases con las abstracciones que fueron necesarias para construir el simulador y el detalle de cada operación de cada una de las abstracciones.





CPU	ejecutar	Ejecuta una unidad de tiempo de servicio del proceso
	cambiar_ocioso	Cambia el estado del CPU a LIBRE
	cambiar_libre	Cambia el estado del CPU a LIBRE
	estado	Retorna el estado del proceso actual
	asignar	Asignar el proceso mediante el despachador
	proceso_instancia	Obtener la instancia del proceso actual asignado al CPU
	proceso_asignado	Obtener el ID de proceso del proceso actual asignado al CPU
PROCESO	sumar_ejecuciones	Suma la cantidad de ejecuciones realizadas por el CPU para una instancia de un proceso dado
	restar_ejecuciones	Resta la cantidad de ejecuciones realizadas por

 UNIVERSIDAD CAECE	SISTEMAS OPERATIVOS I
<i>El problema de lectores/escritores</i>	

		el CPU para una instancia de un proceso dado
	contar_ejecuciones	Retorna la cantidad de ejecuciones actuales
	asignar_quantum	Utilizada por el planificador para asignar tiempo de procesador a una instancia de un proceso dado
	contar_quantum	Retorna la cantidad de quantum restante para el proceso antes de volver a ser planificado
	descontar_quantum	Descuenta una unidad de quantum cuando la instancia del proceso ha sido ejecutada
	llegada	Retorna el tiempo de llegada del proceso
	tiempo_servicio	Retorna el tiempo de servicio requerido por la instancia del proceso para completar la ejecución del mismo
	proceso_id	Retorna el ID de proceso de la instancia de proceso actual

	cambiar_a_ejecutando	Cambia el estado del proceso a EJECUTANDO
	cambiar_a_listo	Cambia el estado del proceso LISTO
	cambiar_a_finalizado	Cambia el estado del proceso FINALIZADO
	cambiar_a_bloqueado	Cambia el estado del proceso BLOQUEADO
	obtener_os	Obtener instancia del sistema operativo donde el proceso se encuentra ejecutando
	descontar_tiempo_servicio	Descuenta el tiempo de servicio del proceso tras la ejecución
	sumar_tiempo_servicio	Suma el tiempo de servicio a un proceso
OS	asignar_proceso	Cuando se quiere bloquear un proceso se le pasa el control del proceso al CPU para que simule el bloque de un semáforo.
	bloquear_proceso_escritor	Bloquea el proceso escritor y se encola en la cola de procesos escritores bloqueados



	bloquear_proceso_lector	Bloquea el proceso lector y se encola en la cola de procesos escritores bloqueados
PLANIFICADOR	planificar_mediano_plazo	Realiza una planificación de las colas de procesos escritores/lectores bloqueados
	planificar	Realiza una planificación a corto plazo de las colas listos y nuevos
	actualizar_ciclos	Notifica al planificador el ciclo de CPU actual
DESPACHADOR	despachar	Toma el primer proceso planificado por el planificador de la cola de listos y lo encola en la cola de procesos en ejecución y asigna al CPU el proceso para su ejecución
LECTOR	ejecutar	Ejecuta un tiempo de servicio de una instancia de proceso lector
ESCRITOR	ejecutar	Ejecuta un tiempo de servicio de una instancia de proceso escritor

Manual Simulación

A continuación se detalla el uso del simulador para resolver el problema de lectores/escritores.

Requisitos

Para poder utilizar el planificador es necesario tener instalado PERL 5, en la cual fue desarrollado.

Iniciar la simulación

Para iniciar la simulación utilizamos el comando siguiente dentro del directorio **src**:

```
$ perl simulador.pl

=====
== PLANIFICADOR CPU - SIMULADOR 🍏 ==
=====

AYUDA GESTOR DE PROCESOS

1) AGREGAR UN NUEVO PROCESO
2) MONITOREAR COLAS DE PLANIFICACION
3) TERMINAR

+ INGRESAR OPCION:
```

Tras la ejecución del simulador, se presenta un menú con tres operaciones al usuario:

1. AGREGAR UN NUEVO PROCESO

- Permite al usuario agregar un nuevo proceso a la cola de procesos nuevos para que el planificador lo considere en los tiempos configurados.

2. MONITOREAR COLAS DE PLANIFICACIÓN

- Es el modo de monitor activo que permite visualizar las colas en tiempo real.

3. TERMINAR

- Finalizar simulación

AGREGAR UN NUEVO PROCESO

Esta opción permite agregar en tiempo real un nuevo proceso a la cola de procesos nuevos y luego el planificador eventualmente colocara estos procesos nuevos en la cola de procesos listos para su ejecución.

```
== PLANIFICADOR CPU - SIMULADOR 🖥️ ==
```

```
AYUDA GESTOR DE PROCESOS
```

- 1) AGREGAR UN NUEVO PROCESO
- 2) MONITOREAR COLAS DE PLANIFICACION
- 3) TERMINAR

```
+ INGRESAR OPCION: 1
```


```
AGREGAR UN NUEVO PROCESO
```

```
INGRESAR TIPO PROCESO L (LECTOR) / E (ESCRITOR): L
```

```
INGRESAR PID: X30
```

```
INGRESAR TIEMPO LLEGADA: 10
```

```
INGRESAR TIEMPO DE SERVICIO: 4
```

 UNIVERSIDAD CAECE	SISTEMAS OPERATIVOS I
<i>El problema de lectores/escritores</i>	

MONITOREAR COLAS DE PLANIFICACIÓN

Esta opción permite ver el uso de las diferentes colas por las cuales se mueven los procesos e información adicional:

1. CPU CICLO

- Muestra el ciclo actual de CPU (siempre ciclando)

2. PROCESOS NUEVOS

- Muestra cuántos procesos están en la cola de nuevos procesos.

3. PROCESOS LISTOS

- Muestra cuántos procesos están en la cola de listos procesos.

4. ESTADO DEL PROCESADOR

- Muestra el estado del procesador en el ciclo actual.

5. ÚLTIMO PROCESO EN EJECUCIÓN

- Muestra el último proceso que tuvo asignado el CPU.

6. CANTIDAD DE ESCRITORES ESPERANDO

- Muestra la cantidad de procesos escritores bloqueados por algún otro proceso lector o escritor.

7. CANTIDAD DE LECTORES ESPERANDO

- Muestra la cantidad de procesos lectores bloqueados esperando por la finalización de procesos escritores.



MONITOREANDO COLAS DE PLANIFICACION (presione Enter para salir...)

+ CPU CICLO 🕒 : 4
+ PROCESOS NUEVOS: 0
+ PROCESOS LISTOS: 1
+ ESTADO DEL PROCESADOR: LIBRE
+ ULTIMO PROCESO EN EJECUCION: L3
+ CANTIDAD DE ESCRITORES ESPERANDO: 0
+ CANTIDAD DE LECTORES ESPERANDO: 2

COLA: LISTOS

Proceso	Llegada	Servicio
E1	0	1

COLA: NUEVOS


Proceso	Llegada	Servicio
---------	---------	----------

COLA: ESCRITORES

Proceso	Llegada	Servicio
---------	---------	----------

COLA: LECTORES

Proceso	Llegada	Servicio
L2	0	3
L3	0	3

 UNIVERSIDAD CAECE	SISTEMAS OPERATIVOS I
<i>El problema de lectores/escritores</i>	

TERMINAR

Finalizar la simulación

PLANIFICADOR

El planificador utiliza un algoritmo Round-Robin de Quantum = 2.

LOG EJECUCIÓN

El proceso guarda la traza de ejecución de la carpeta de logs en caso de querer visualizar el proceso completo.

SIMULACIÓN

Dejamos una grabación en video del simulador en ejecución:

[simulacion.mov](#)