2025

GRodriguezAR

Proyecto de programación concurrente y paralela en C

Diseño funcional

Sistema de facturación concurrente

**Índice**

[Sistema de Facturación 2](#_Toc197525976)

[Abstract 2](#_Toc197525977)

[Escenario planteado 2](#_Toc197525978)

[Estructura 2](#_Toc197525979)

[Lenguaje utilizado 3](#_Toc197525980)

[Relación entre procesos 4](#_Toc197525981)

[Memoria compartida y su estructura 5](#_Toc197525982)

[Servicio meteorológico 6](#_Toc197525983)

[Abstract 6](#_Toc197525984)

[Escenario planteado 6](#_Toc197525985)

[Estructura 6](#_Toc197525986)

[Lenguaje utilizado 7](#_Toc197525987)

[Relación entre procesos 8](#_Toc197525988)

[Memoria compartida, estructuras de datos y semáforos 9](#_Toc197525989)

[Procesador de Texto 10](#_Toc197525990)

[Abstract 10](#_Toc197525991)

[Escenario Planteado 10](#_Toc197525992)

[Estructura 10](#_Toc197525993)

[Protocolo de Comunicación 11](#_Toc197525994)

[Flujo del sistema 12](#_Toc197525995)

[Sistema de Cálculo Distribuido 13](#_Toc197525996)

[Abstract 13](#_Toc197525997)

[Escenario Planteado 13](#_Toc197525998)

[Estructura 13](#_Toc197525999)

[Protocolo de comunicación 15](#_Toc197526000)

[Flujo del sistema 16](#_Toc197526001)

**Sistema de Facturación Concurrente**

## Abstract

Sistema de multiproceso para facturación. Simula pedidos entrantes con un generador, se validan y actualizan los stocks correspondientes, y se generan las facturas detalladas. Se generan logs para alertar sobre stock bajo, y periódicamente se simula una entrada de mercadería.

## Escenario planteado

Se presenta un escenario donde una tienda de computadoras desea administrar la venta de sus productos: requiere un software con el que pueda administrar todos los pedidos que sus clientes realizan en tiempo real.

Se plantea entonces, un sistema que recibe pedidos (generados de forma simulada en este caso), valida si existe stock suficiente para aprobarlos, actualiza su estado y genera la factura. Adicionalmente, alerta sobre productos con un stock bajo generando un archivo .log y, periódicamente, repone una cantidad determinada de los mismos.

## Estructura

El sistema está estructurado por un proceso padre encargado de inicializar la memoria compartida, semáforos, inventario inicial y de crear los procesos hijos:

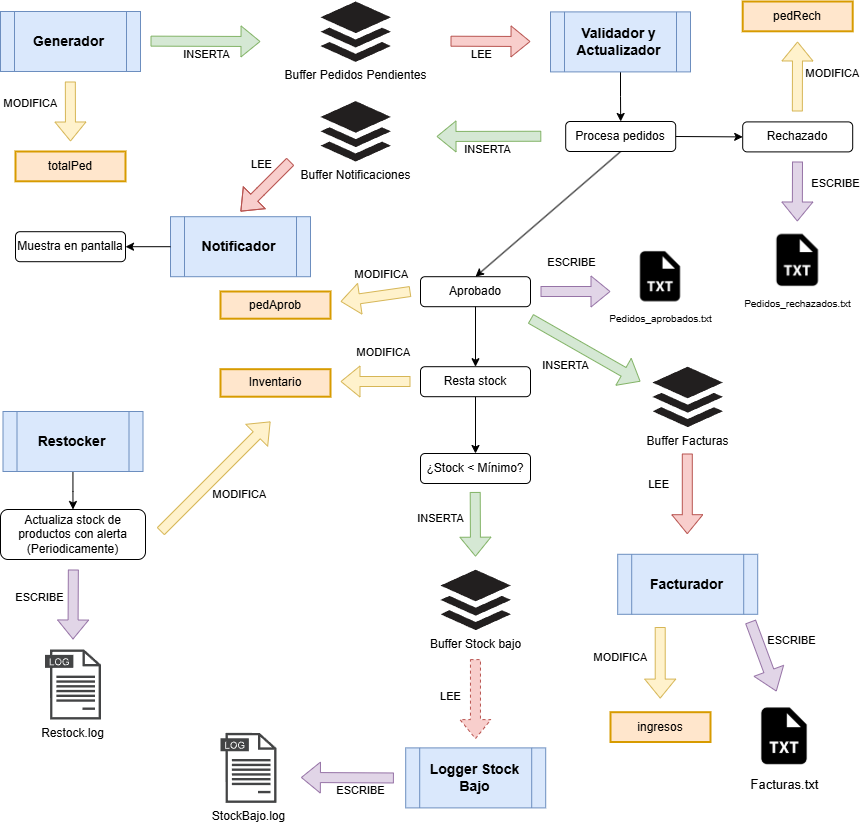
1. Proceso 1 – Generador: Crea pedidos cada un periodo de tiempo predeterminado: cada pedido posee una cantidad aleatoria de ítems (productos), cada uno con su respectiva cantidad. Los productos son cargados en un buffer a medida que se generan.
2. Proceso 2 – Validador y Actualizador: Lee el buffer de pedidos y analiza si, teniendo en cuenta todos los productos y cantidades pedidas, es posible aprobarlo o no. En caso positivo, actualiza el stock en el inventario y si este queda por debajo del umbral mínimo, lo carga en el buffer de stock bajo. En caso de no aprobarse el pedido, lo marca como rechazado. En ambos casos, el pedido es cargado en un buffer para notificarse posteriormente y es escrito en un log de pedidos aprobados o rechazados según corresponda.
3. Proceso 3 – Logger Stock Bajo: Lee el buffer de stock bajo y genera un log de alerta en tiempo real indicando la cantidad actual de stock del producto.
4. Proceso 4 – Restocker: Cada un intervalo de tiempo predeterminado, es decir periódicamente, aumenta el stock solo de aquellos productos que fueron marcados con stock por debajo del umbral. Genera un log con el stock antiguo y el nuevo, simulando una entrada de mercadería.
5. Proceso 5 – Facturador: Genera un archivo de texto donde por cada pedido aprobado se indican todos los datos de la compra así como el total y el cliente correspondiente.
6. Proceso 6 – Notificador: Muestra por pantalla en tiempo real cuales pedidos son aprobados y cuales rechazados.

Al finalizar el programa, el proceso padre se encarga de detener los demás procesos, limpiar la memoria y los semáforos. Genera además un informe final con el total recaudado y la cantidad total de pedidos aprobados, rechazados y recibidos.

## Lenguaje utilizado

El lenguaje a utilizar será ‘C debido al control total sobre la gestión de memoria y sincronización de semáforos de System V que están nativamente soportados, facilitando el monitoreo desde el bash de Ubuntu.

## Relación entre procesos



En el gráfico, las fechas verdes indican escrituras en el buffer y las rojas indican lecturas. Ambas operaciones requieren del uso de los tres semáforos. SEM\_buffer\_MUTEX, SEM\_buffer\_VACIO, SEM\_buffer\_LLENO.

Las flechas amarillas indican escritura o lectura de variables en memoria compartida (que no son buffers). En este caso, la única que variable que necesita un semáforo es la de inventario (SEM\_INVENTARIO\_MUTEX). Esto se debe a que las demás son accedidas únicamente por un solo proceso (por ejemplo, totalPedidos es accedida únicamente por el proceso Generador), y luego son accedidas por el padre una vez que los hijos finalizaron.

Por último, las flechas violetas indican escritura en archivos externos. El tipo del mismo se especifica en el gráfico, junto con su nombre.

## Memoria compartida y su estructura

