UNIVERSIDAD ORT
Uruguay

# Universidad ORT Uruguay

## Facultad de Ingeniería

## Obligatorio 1

## -

## Sistemas Operativos


**Autores:**

Joaquín Costa - 330734

Ezequiel Rivero - 302985

Tomás Dotta - 326829

**Tutor:**

**Anuar Maubrigades**
**Enrique Latorres**


**2025**

UNIVERSIDAD ORT
Uruguay

UNIVERSIDAD ORT
Uruguay

# Ejercicio 1

El ejercicio consiste en desarrollar un sistema de gestión para la empresa Citadel, que fabrica pinturas para miniaturas. El sistema debe permitir la gestión de usuarios, ingreso y venta de productos y filtrado por tipo. En Bash.

## Decisiones de diseño

- Almacenamiento:
    - Usuarios en users.txt (formato: usuario contraseña).
    - Productos en productos.csv (formato: código,tipo,modelo,descripción,cantidad,precio).
- Archivos Temporales: se usa un archivo temporal session.tmp para saber si hay usuario logueado.
- Restricciones: se controlan duplicados, campos vacíos/no válidos y stock en ventas (tuvimos el problema de que se pueden ejecutar comandos de bash cuando se espera un input en consola).

## Decisiones por sección

- **“ $? ”** Se usa en la sección crearUser y otras funciones como login.
  ¿Qué hace **$?** devuelve el código de salida del último comando ejecutado:
  0 = éxito . Distinto de 0 = error o no coincidencia.
- **“-s”** session.tmp Se usa en la función validarLogin. **“-s”** verifica si el archivo existe y no está vacío.
- **“while IFS= read -r line; do ... awk ... xargs”** Se usa en ingProd, vendProd y filterProd.
    - While IFS= read -r line lee el archivo línea por línea.
    - awk -F',' '{print $N}' extrae la columna N del CSV.
    - xargs limpia espacios sobrantes (trim).

      Lo usamos para obtener datos de los productos ya que awk funciona para columnas de un csv

UNIVERSIDAD ORT
Uruguay

# Uso de IA en el Ejercicio 1

**awk para actualizar stock:**

```
241     col=5
242     valor=$((cantidad - cant))
243     # linea hecha con IA
244     awk -v r="$num" -v c="$col" -v v="$valor" 'BEGIN{FS=OFS=","} NR==r{$c=v}1' \
245     productos.csv > productos.csv.tmp && mv productos.csv.tmp productos.csv
246
```

En productos.csv

- -v r="$num": pasa a awk el número de fila que queremos editar .
- -v c="$col": pasa el número de columna a editar (5) que es el de la cantidad.
- -v v="$valor": pasa el nuevo valor (stock actualizado).
- BEGIN{FS=OFS=","}: fija el separador de campos de entrada y salida a la coma (CSV).
- NR==r{$c=v}1:
    - NR==r: cuando el número de registro (línea actual) es la fila r
    - {$c=v}:  asigna a la columna c el valor v .
    - 1:  significa “imprimir la línea”. Como no se filtra nada, imprime todas; con la edición aplicada en la fila r.

UNIVERSIDAD ORT
Uruguay

# Ejercicio 2

El ejercicio 2 plantea una variante del clásico problema de lectores y escritores, aplicada a un panel de información de un aeropuerto. En este escenario interactúan 100 pasajeros (lectores) y 5 oficinistas (escritores).
Los pasajeros consultan el cartel, mientras que los oficinistas realizan actualizaciones críticas que deben ejecutarse sin interferencias. El objetivo es diseñar un mecanismo de sincronización que garantice que:

- Los pasajeros pueden leer el cartel siempre que no exista una modificación en curso.
- Los oficinistas sólo puede n modificar el cartel cuando no haya lectores activos.
- Cada oficinista realiza al menos 3 actualizaciones, con tiempos aleatorios de espera (máx. 5 s para oficinistas y 3 s para pasajeros).

Esto asegura que ningún pasajero lea el panel mientras está siendo actualizado.

## Decisiones de diseño

Para lograr una correcta sincronización entre lectores y escritores se decidió el uso de un sistema de tickets (funcionamiento semejante a una cola).
Dentro del código podemos encontrar dentro del sistema de tickets los siguientes elementos:

### Variables globales

```
int next_ticket = 0;
int turno_actual = 0;
pthread_mutex_t ticket_mutex = PTHREAD_MUTEX_INITIALIZER;
```

representando **next_ticket** el número del próximo turno a otorgar y **turno_actual** el número del turno que debe ejecutarse en ese momento. Acompañado de **ticket_mutex**
El cual sirve para asegurar la exclusión mutua de los hilos a la hora de modificar el siguiente turno (**next_ticket**).

UNIVERSIDAD ORT
Uruguay

## Funciones

```
void esperar_turno(int* mi_turno) {
    pthread_mutex_lock(&ticket_mutex);
    *mi_turno = next_ticket++;
    pthread_mutex_unlock(&ticket_mutex);

    while (turno_actual != *mi_turno) {
        sched_yield();
    }
}

void liberar_turno() {
    __sync_fetch_and_add(&turno_actual, 1);
}
```

Se tiene 2 funciones para lograr el intercalamiento al estilo de una cola:
**esperar_turno**:
La cual recibe como parámetro el turno actual, el cual será el turno del hilo en ejecución, y aumenta al ejecutarse en uno al siguiente turno para avanzar así al siguiente pasajero u oficinista siempre y cuando este la variable libre para su modificación con **ticket_mutex**. Además siempre que el **turno_actual** global no sea el mio voy a esperar a que llegue mi turno con **sched_yield()**(es una llamada al sistema que indica al planificador del sistema operativo que el hilo actual está dispuesto a ceder voluntariamente el procesador).
**liberar_turno**:
Que lo único que hace es **__sync_fetch_and_add(&turno_actual, 1)** una función la cual hace los siguientes pasos:

1. Lee el valor actual de **turno_actual**.
2. Suma 1.
3. Escribe el valor actualizado.
4. Devuelve el valor anterior.

Esto lo que logra es avanzar el turno actual en 1 para poder seguir con los demás.

## Hilos:

Se tienen los hilos escritor y lector los cuales su única función es esperar su turno, ejecutar su tarea la cual puede ser modificar o mirar el cartel y liberar su turno.

UNIVERSIDAD ORT
Uruguay

## Main:

El main se encarga de preparar y coordinar la creación de todos los hilos del sistema, logrando dividir su funcionamiento en 4 etapas principales:

1. **Inicialización de los datos**
   Se inicializan los arreglos con los identificadores de los lectores y escritores, además de un arreglo con todos los hilos del sistema y otro de roles. El cual cuenta con 100 posiciones de ‘L’ y 15 de ‘E’. Simulando las instancias de Lectores y Escritores respectivamente, siendo 15 escritores porque cada uno de los 5 hace 3 modificaciones quedando así en 15 instancias.

2. **Mezcla aleatoria de roles**
   Se realiza una mezcla de los roles para simular una situación real en la que cada lector y escritor llega en un momento aleatorio sin un orden fijo predecible, Dejando de esta forma el orden final de ejecución al sistema de tickets.

3. **Creación de hilos**
   se recorre el arreglo de roles y se crea un hilo lector o escritor dependiendo la letra que salga de roles. a cada hilo se le inicializa con un id numérico único.

4. **Espera de los hilos**
   Para finalizar se espera la liquidación de cada uno de los hilos ejecutados en el programa.

UNIVERSIDAD ORT
Uruguay

# Ejercicio 3

El ejercicio 3 se basa en la creación de una simulación de tiempos en las que 100 “vacas” van pasando por una zona de vacunación o ordeñe, y luego suben a uno de dos camiones de capacidad de 50 vacas.

## Decisiones de diseño

Para lograr el objetivo principal del ejercicio, el manejo de hilos y procesos, se decidió solo hacer una procedure main, pero que cada tarea sea una task, las salas y el pasillo son tasks que reciben parámetros según la vaca y la acción, estas solo devuelven que la vaca está realizando su debida acción, luego la task vaca genera un numero random para decidir qué acción va a hacer, y se ejecuta con su id (si es posible y no redundante) .

UNIVERSIDAD ORT
Uruguay

# Decisiones de Código

## Subtype y range

Define un subtipo restringido de Integer con un rango válido.

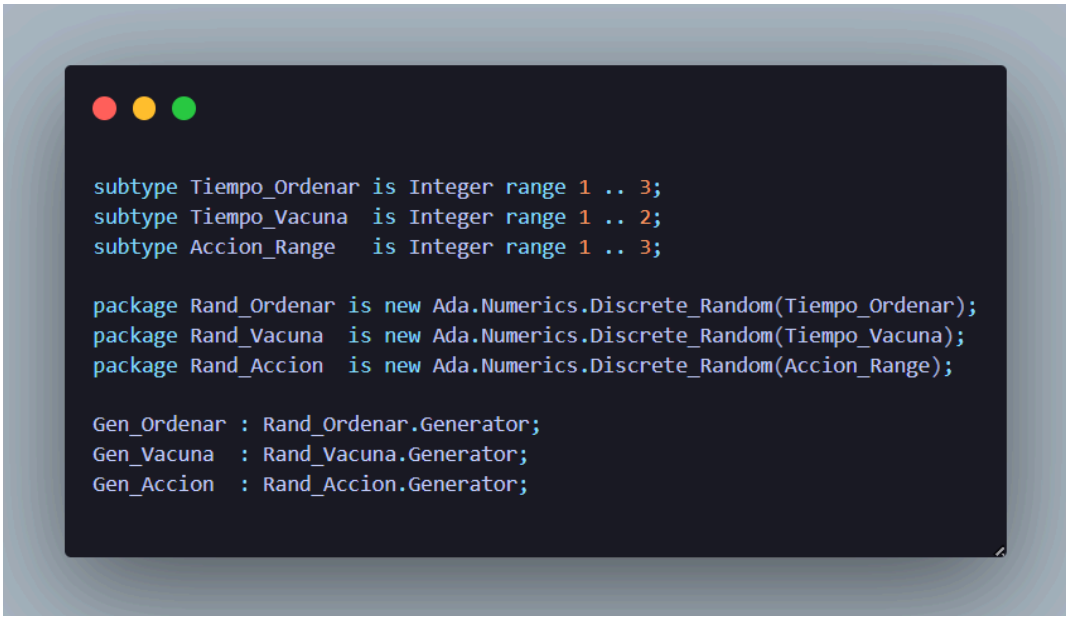
Esto fuerza que cualquier valor de ese subtipo esté entre X e Y,lo usamos para instanciar generadores aleatorios con límites, por ejemplo el tiempo de una vaca en un tambo está entre 1 y 5.

## Instancias de paquetes genéricos

“Discrete_Random” es un paquete genérico que necesita un tipo discreto.
is new crea una instancia especializada para generar números aleatorios en el rango del subtipo.

## Generadores

Cada paquete instanciado define un tipo Generator.
Se usa Random(Gen) para obtener un valor aleatorio dentro del rango  y Reset(Gen) para inicializar la semilla.



```
subtype Tiempo_Ordenar is Integer range 1 .. 3;
subtype Tiempo_Vacuna  is Integer range 1 .. 2;
subtype Accion_Range   is Integer range 1 .. 3;

package Rand_Ordenar is new Ada.Numerics.Discrete_Random(Tiempo_Ordenar);
package Rand_Vacuna  is new Ada.Numerics.Discrete_Random(Tiempo_Vacuna);
package Rand_Accion  is new Ada.Numerics.Discrete_Random(Accion_Range);

Gen_Ordenar : Rand_Ordenar.Generator;
Gen_Vacuna  : Rand_Vacuna.Generator;
Gen_Accion  : Rand_Accion.Generator;
```

UNIVERSIDAD ORT
Uruguay

task , entry e accept

```
task body Sala is
      Cant : Integer := 0;
   begin
      loop
         accept Operar(ID : Integer; Accion : String; Exito : out Boolean) do
            if Accion = "Entrar" then
               if Cant < 15 then
                  Cant := Cant + 1;
                  Exito := True;
                  Put("Vaca "); Put(ID); Put_Line(" entra a la sala");
               else
                  Exito := False;
               end if;
            else
               Cant := Cant - 1;
               Exito := True;
               Put("Vaca "); Put(ID); Put_Line(" sale de la sala");
            end if;
         end Operar;
      end loop;
   end Sala;

   task Mangas is
      entry Operar(ID : Integer; Accion : String; Exito : out Boolean);
   end Mangas;
```

**task mangas is entry Operar()**

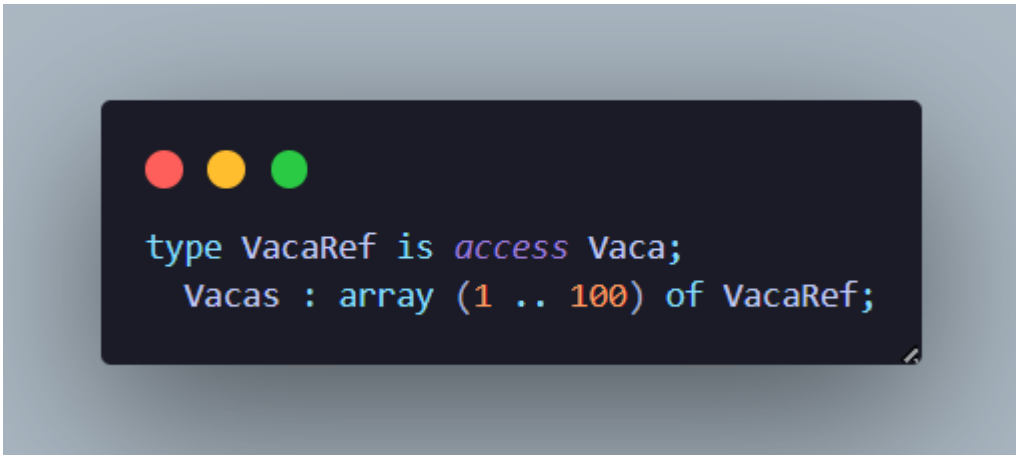
Declara una tarea concurrente con una entrada (entry) que otros pueden invocar. Las entradas son puntos de sincronización para comunicación entre tareas, permitiendo pasar parámetros y recibir resultados.

**accept**

La tarea se bloquea hasta que otra invoque la entrada.
Se ejecuta el bloque do ... end, esto es lo que nos asegura la sincronización

UNIVERSIDAD ORT
Uruguay

## Tipos acceso y creación dinámica




**Access** define un puntero seguro a una tarea y **new Vaca(I)** crea dinámicamenteuna tarea concurrente con el parámetro ID desde el 1 al 100.

Esto fue extraído de la pagina link, para resolver el problema de que se ejecutaban los tasks secuencialmente. ya que trabajamos con el objeto vaca, para poder usar estrategias con las que estamos más familiarizados.

# Control de Concurrencias

Las salas, mangas y camiones se manejan con variables internas (Cant, C1, C2, etc) y accept.
Si no hay espacio, la entrada devuelve Exito=False y la tarea reintenta tras un delay.

UNIVERSIDAD ORT
Uruguay

# Ejercicio 4

## Documento Técnico

### Arquitectura implementada

El sistema está diseñado bajo una arquitectura de microservicios contenerizados con Docker. La solución incluye tres servicios principales que trabajan de forma coordinada para garantizar modularidad y escalabilidad.

La primera capa es la de Presentación y Balanceo de Carga. Aquí se utiliza Nginx como proxy inverso y como balanceador de tráfico. Este servicio es el punto de entrada público. Recibe las peticiones en el puerto 8080 y las distribuye entre las instancias de la aplicación.

El archivo nginx.conf se incluye en el directorio ‘./nginx’ del proyecto. El servicio Nginx lo carga mediante el volumen declarado en el archivo docker-compose.yaml. Este archivo define el bloque upstream necesario para distribuir el tráfico entre las instancias de la aplicación y la directiva proxy_pass que redirige las solicitudes hacia el backend en el puerto interno 5000.

La estructura debe respetar el formato estándar de Nginx para evitar errores de arranque o reinicios del servicio.

La estructura debe respetar el formato estándar de Nginx para evitar errores de arranque o reinicios del servicio.

La segunda capa corresponde a la Aplicación. Está formada por una aplicación desarrollada en Node.js. Esta capa es apátrida y permite escalar de forma horizontal. Cada instancia corre en el puerto interno 5000 y ejecuta toda la lógica de negocio.

La tercera capa es la de Datos. Redis se utiliza como base de datos en memoria. Para asegurar la persistencia, se utilizan volúmenes de Docker que mantienen la información incluso si el contenedor se reinicia.

### Decisiones Técnicas

Se eligieron imágenes base Alpine para los contenedores de Node, Nginx y Redis. Estas imágenes reducen el tamaño de cada servicio y mejoran los tiempos de despliegue. También disminuyen la superficie de ataque, lo cual aporta un nivel mayor de seguridad.

```
FROM node:alpine
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 5000
CMD ["node", "server.js"]
```

La arquitectura utiliza dos redes internas diferenciadas.

- La red front-tier conecta únicamente a Nginx con la aplicación.
- La red back-tier conecta la aplicación con la base de datos Redis.

Gracias a esta separación, Redis no es accesible desde el exterior ni desde Nginx, lo que incrementa la protección del sistema.

El archivo nginx.conf fue modificado para agregar un bloque upstream. Esto permite que Nginx detecte las instancias disponibles de la aplicación mediante su hostname y reparta el tráfico entre ellas. Así, el sistema puede escalar sin necesidad de reconfigurar el servidor manualmente.

Para la persistencia, se definió un volumen nombrado llamado redis_data. Esto desacopla el ciclo de vida de los datos, del ciclo de vida del contenedor. De esta forma, el contenido almacenado persiste aunque Redis sea reiniciado o actualizado.

## Comunicación entre servicios

El flujo general de interacción entre los servicios se puede describir de la siguiente manera:

- El usuario realiza una petición al puerto 8080.
- Nginx recibe la solicitud y la envía a una de las instancias de Node.js a través de la red front-tier, siempre utilizando el puerto interno 5000.
- La aplicación procesa la petición. Si necesita información, consulta a Redis por la red back-tier utilizando el puerto 6379.
- Redis devuelve los datos solicitados.
- La respuesta vuelve hacia Nginx y desde allí se entrega al usuario.

UNIVERSIDAD ORT
Uruguay

## Análisis de implementación

### Ventajas

La arquitectura tiene la capacidad de escalar de forma inmediata. Gracias a la configuración de upstream en Nginx y a las funcionalidades de Docker Compose, es posible aumentar el número de instancias de Node.js con un único comando.

El uso de redes separadas reduce el riesgo de exposición. La base de datos no está accesible desde el exterior, lo que aporta un nivel adicional de seguridad.

Cada servicio está configurado con restart always. Esto asegura que, en caso de error o caída, el sistema restablezca los contenedores de manera automática.

### Desventajas

A medida que se agregan más servicios, la manipulación de tantos contenedores se vuelve más compleja. La gestión de logs, el monitoreo y la trazabilidad requieren herramientas adicionales dado a la gran magnitud de datos que se manejarían en dicha situación.

Aunque Redis utiliza volúmenes, toda la persistencia depende de un único host. En un entorno distribuido real, esto debería reemplazarse por almacenamiento compartido o un clúster de Redis.

## Recomendaciones

La implementación de healthchecks podría ser una buena implementación ya que gestionaría de forma más eficiente el tráfico de información. Evitando situaciones en las que se envían datos a un componente que quizás no está funcionando debidamente, esto antes que actúe el reinicio de docker del componente en cuestión.

Otra buena implementación es la idea de una base de datos más especializada, redis es útil como memoria de alta velocidad, más no como medio para respaldar el servidor de forma completa. Por lo que se recomiendo utilizar alguna base de datos u otro método para almacenar la información del servidor, facilitando así el almacenamiento y la busqueda de información.

UNIVERSIDAD ORT
Uruguay

# Manual de Usuario

## Requisitos Previos

Antes de ejecutar el sistema, asegúrese de que su equipo tenga instalados Docker y Docker Compose. Ambos deben estar configurados correctamente para permitir la creación y ejecución de contenedores. Para la debida ejecución de los comandos se debe tener en todo momento la aplicación de Docker abierta.

El sistema utiliza el puerto 8080 para exponer la aplicación hacia el navegador. Por este motivo, es importante verificar que dicho puerto no esté siendo utilizado por otro programa o servicio.

## Inicio Servicio

Para iniciar la infraestructura debe de seguir los siguientes estos pasos:

- Abra una terminal o ventana de comandos.
- Luego acceda al directorio principal del proyecto, que es aquel donde se encuentra el archivo docker-compose.yaml.

Una vez ubicado en la carpeta correcta, ejecute en la consola ‘ docker compose up ‘

Este comando descarga las imágenes necesarias, construye los contenedores y muestra en la terminal los registros de ejecución, permitiéndole ver el estado de cada servicio a medida que inicia.

## Ingreso a la Web

Cuando los servicios de la aplicación estén activos y sin errores, puede ingresar desde su navegador.

Abra su navegador web y diríjase a ‘ http://localhost:8080 ‘

En esa dirección podrá ver la interfaz principal del sistema. Todas las acciones que realice se procesarán automáticamente mediante el balanceador de carga y se almacenarán en la base de datos.

Si desea detener la ejecución del entorno, vuelva a la terminal donde se encuentra corriendo Docker Compose y presione  CTRL + C

Los contenedores se detendrán de manera ordenada.

Si además quiere liberar el entorno y eliminar los contenedores generados, utilice

‘ docker compose down ‘

## Solución de Problemas

### Puerto Ocupado

Si al iniciar el sistema aparece un mensaje indicando que el puerto 8080 está en uso, significa que otro programa lo está ocupando. Para solucionar este inconveniente se recomienda dirigirse al archivo docker-compose.yaml y modificar en el apartado de app, donde dice 8080:80, en Nginx, modificarlo por 8081:80. Si esto no soluciona el problema intentar aumentar uno hasta encontrar un puerto libre. Luego ingrese a la web pero modifique el término 8080 por el último número que usted ingresó para solucionar el problema.

### Problemas con la configuración de Nginx

Si el contenedor de Nginx entra en un ciclo de reinicios, revise que el archivo nginx.conf se encuentre en la ubicación esperada por el servicio. También verifique que su estructura sea válida y coincida con lo indicado en el documento técnico. Se adjunta como debe de lucir dicho archivo:

```
events {
    worker_connections 1024;
}

http{
    upstream loadbalancer {
        server node-app1:5000;
        server node-app2:5000;
    }
    server {
        listen 80;
        server_name localhost;
        location / {
            proxy_pass http://loadbalancer;

            proxy_set_header Host $host;

            proxy_set_header X-Real-IP
$remote_addr;
        }
    }
}
```