Carrera INGENIERIA EN INFORMATICA		
Asignatura SISTEMAS OPERATIVOS		
Trayecto Infraestructura		
Año académico 2025		
Responsable / Jefe de catedra Lic. Fabio E. Rivalta		
Carga horaria semanal 4	**Carga horaria total** 64	**Créditos**
Modalidad: presencial / virtual on-line		
Correlativas anteriores Arquitectura de Computadoras	**Correlativas posteriores** Virtualización de Hardware Seguridad Aplicada y Forensia	
Conocimientos necesarios		

Equipo docente		
Nombre	**Cargo**	**Titulo**
Fabio E. Rivalta	Profesor Adjunto	Lic. En Sistemas de Computación
Leonardo Catalano	Profesor Adjunto	Ing. En Informática
Alexis Villamayor	Jefe de trabajos prácticos	Ing. En Informática
Ramiro de Lizarralde	Jefe de trabajos prácticos	Ing. En Informática
Fernando Piubel	Ayudante de 1°	Ing. En Informática
Alejandro Rodriguez	Ayudante de 1°	Ing. En Informática

Trabajo Práctico (GRUPAL):

Entregables de cada ejercicio:

Para el diseño funcional:

- Abstract de la solución a implementar (máximo 50 palabras)
- Lenguaje a utilizar para la implementación.
- Enunciado con el diseño funcional de la problemática de ambos ejercicios.
- Diagramas y cómo será la resolución de la problemática.
- Formato de entrega: documento **word** subido a la plataforma **MIeL**.

Para la entrega final:

- Ejercicio resuelto, código fuente y binario (en caso de que aplique).
- Lote de prueba utilizado, junto con los archivos necesarios para poder probarlo, en caso de que sea necesario.
- Herramientas utilizadas para validar el monitoreo de los procesos.
- Todo lo relacionado a la entrega debe estar en un archivo comprimido **ZIP** y ser entregado por la plataforma **MIeL**.

Ejercicios:

Ejercicio 1: Coordinación de Tareas con Procesos y Memoria Compartida

Enunciado:

Desarrollen un programa que simule un sistema de procesamiento de datos en paralelo.
El sistema debe constar de un proceso padre que genera **N** procesos hijos (N >= 4).
Cada proceso hijo realizará una tarea específica sobre un conjunto de datos compartido en memoria.

Requisitos:

- El programa debe ser desarrollado en un lenguaje de programación de su elección, siempre y cuando sea compatible con *Linux* y permita la utilización de las llamadas al sistema *fork(),* memoria compartida y semáforos.
- Utilizar la llamada al sistema *fork()* para crear los procesos hijos.
- Implementar un mecanismo de memoria compartida para que los procesos hijos puedan acceder y modificar los datos.
- Emplear semáforos para sincronizar el acceso a la memoria compartida y evitar condiciones de carrera.
- Cada proceso hijo debe realizar una operación distinta, hasta que el padre le indique que debe finalizar, sobre los datos compartidos (por ejemplo, sumar, restar, multiplicar, ordenar, etc.), que debe ser informada como parte del diseño funcional de la problemática.
- El proceso padre debe quedar esperando hasta que reciba la orden del operador de finalizar y en ese momento debe indicarles a todos los procesos hijos que terminen y una vez que estos terminaron mostrar el resultado final de los datos compartidos.

Monitoreo en Linux:

- Utilizar comandos estándar de Bash (como *ipcs*, *ps*, *top*, *htop* o *vmstat*) para demostrar y documentar el uso de memoria compartida, semáforos y la creación/gestión de procesos durante la ejecución del programa.
- Debe poder visualizarse la concurrencia de procesos, la generación de los recursos de memoria compartida y semáforos.
- Luego de la ejecución no deben quedar recursos “basura” en el sistema. Se deben limpiar y eliminar todas las estructuras y archivos temporales generados.

Ejercicio 2: Comunicación Cliente-Servidor con Threads y Sockets

Enunciado:

Diseñen un sistema de comunicación cliente-servidor utilizando sockets y threads. El servidor debe ser capaz de atender múltiples clientes simultáneamente.

Requisitos:

- El programa debe ser desarrollado en un lenguaje de programación de su elección, siempre y cuando sea compatible con Linux y permita la utilización de sockets y threads.
- El servidor debe utilizar sockets para escuchar las conexiones entrantes de los clientes, e implementar un máximo de conexiones activas al mismo tiempo (a definir como parte del diseño). El resto de los clientes (excedentes del máximo permitido) deben quedar esperando hasta que el servidor tenga conexiones disponibles.
- Tanto el cliente como el servidor deben ser programas distintos y ser ejecutados manualmente
- Implementar un mecanismo de threads para que el servidor pueda manejar múltiples clientes de forma concurrente.
- Los clientes deben poder enviar mensajes al servidor. La acción a realizar con el mensaje y respuesta/s es parte de la solución funcional que se debe implementar. El cliente debe finalizar a pedido del usuario, NO automáticamente al recibir una respuesta del servidor.
- El servidor debe procesar los mensajes recibidos. El servidor debe terminar una vez que se retiraron todos los clientes.
- Como parte del diseño se debe definir el protocolo de comunicación que se utilizará para el intercambio de mensajes entre clientes y servidor.
- Manejo de cierre inesperado, tanto del servidor como de los clientes. El sistema debe mantenerse estable ante las caídas de los clientes o ante la caída del servidor.
- La utilización de sockets debe estar controlada y poder ejecutar una y otra vez sobre el mismo puerto.

Monitoreo en Linux:

- Utilizar comandos estándar de Bash (como *netstat*, *lsof*, *ps*, *top* o *htop*) para demostrar y documentar la creación/gestión de threads y el establecimiento/gestión de conexiones de sockets durante la ejecución del programa.
- Debe poder visualizarse la concurrencia de threads y la generación de los sockets.
- Luego de la ejecución no deben quedar recursos “basura” en el sistema. Se deben limpiar y eliminar todas las estructuras y archivos temporales generados.