- Numere las hojas entregadas. Complete en la primera hoja la cantidad total de hojas entregadas.
- Entregue esta hoja junto al examen, la misma no se incluye en la cantidad total de hojas entregadas.
- Cada ejercicio debe realizarse en hojas separadas y numeradas. Debe identificarse cada hoja con nombre, apellido, LU y número de orden.
- **Cada código o pseudocódigo debe estar bien explicado y justificado en castellano. ¡Obligatorio!**
- Toda suposición o decisión que tome deberá justificarla adecuadamente. Si la misma no es correcta o no se condice con el enunciado no será tomada como válida y será corregida acorde.
- La devolución de los exámenes corregidos es personal. Los pedidos de revisión se realizarán por escrito, antes de retirar el examen corregido del aula.
- Los recuperatorios tienen dos notas: I (Insuficiente): 0 a 64 pts y A (Aprobado): 65 a 100 pts.

Ejercicio 1.- Señales. **(25 puntos)**

Se cuenta con un binario cuya ejecución a través de `strace` produce el siguiente output:

```
[88] pipe2([3, 4], 0) = 0
[88] clone(child_stack=NULL, flags=...) = 89
[88] clone(child_stack=NULL, flags=...) = 90
[89] rt_sigaction(SIGTERM, {sa_handler=0x5590,...}, 8) = 0
[88] read(3,  <unfinished ...>
[89] clone(child_stack=NULL, flags=...) = 91
[90] rt_sigaction(SIGTERM, {sa_handler=0x5590,...}, 8) = 0
[90] clone(child_stack=NULL, flags=...) = 92
[89] clock_nanosleep(tv_sec=1, ... <unfinished ...>
[91] clock_nanosleep(tv_sec=2, ... <unfinished ...>
[90] getpid() = 90
[92] clock_nanosleep(tv_sec=9999, ... <unfinished ...>
[90] clone(child_stack=NULL, flags=...) = 93
[90] clock_nanosleep(tv_sec=2, ... <unfinished ...>
[93] clock_nanosleep(tv_sec=3, ... <unfinished ...>
[89] <... clock_nanosleep resumed>0x7ffebb043460) = 0
[89] write(1, "Soy el hijo Pepito\n", 19) = 19
[89] wait4(91,  <unfinished ...>
[91] <... clock_nanosleep resumed>0x7ffebb043420) = 0
[91] write(1, "Soy el nieto Juanito\n", 21) = 21
[91] write(4, "Abuelaaaa, la, la, la, la, la\0", 30 <
     unfinished ...>
[88] <... read resumed>"Abuelaaaa, la, la, la, la, la\0",
     30) = 30
[91] <... write resumed>) = 30
[91] clock_nanosleep(tv_sec=9999, ... <unfinished ...>
[88] write(1, "Mensaje de mi nieto: Abuelaaa, la, la, la",
     51) = 51
[88] wait4(89,  <unfinished ...>
[90] <... clock_nanosleep resumed>0x7ffebb043460) = 0
[90] write(1, "Soy la hija Juanita\n", 20) = 20
[90] wait4(92,  <unfinished ...>
[93] <... clock_nanosleep resumed>0x7ffebb043450) = 0
[93] write(1, "No se quien soy... debo matar!!\n", 32) = 32
[93] kill(90, SIGTERM) = 0
[90] <... wait4 resumed>NULL, 0, NULL) = ? ERESTARTSYS
[93] exit_group(1 <unfinished ...>
[90] --- SIGTERM {si_pid=93, ...} ---
[93] <... exit_group resumed>) = ?
[90] write(1, "Oh no... mi hija nos quiere matar!"..., 35)
     = 35
[90] kill(89, SIGTERM)  = 0
[89] <... wait4 resumed>NULL, 0, NULL) = ? ERESTARTSYS
[90] kill(92, SIGKILL <unfinished ...>
[89] --- SIGTERM {si_pid=90, ...} ---
[92] <... clock_nanosleep resumed> <unfinished ...>) = ?
[90] <... kill resumed>) = 0
[89] write(1, "Oh no... mi hermana me quiere matar!", 37) =
     37
[92] +++ killed by SIGKILL +++
[90] rt_sigreturn({mask=[]} <unfinished ...>
[90] <... rt_sigreturn resumed>) = 61
[89] kill(91, SIGKILL) = 0
[91] <... clock_nanosleep resumed> <unfinished ...>) = ?
[90] wait4(92,  <unfinished ...>
[89] rt_sigreturn({mask=[]} <unfinished ...>
[91] +++ killed by SIGKILL +++
[90] <... wait4 resumed>NULL, 0, NULL) = 92
[89] <... rt_sigreturn resumed>) = 61
[90] wait4(93,  <unfinished ...>
[90] <... wait4 resumed>NULL, 0, NULL) = 93
[89] wait4(91,  <unfinished ...>
[90] exit_group(0 <unfinished ...>
[89] <... wait4 resumed>NULL, 0, NULL) = 91
[90] <... exit_group resumed>)   = ?
[89] exit_group(0 <unfinished ...>
[89] <... exit_group resumed>)   = ?
[88] <... wait4 resumed>NULL, 0, NULL) = ? ERESTARTSYS
[88] wait4(89, NULL, 0, NULL) = 89
[88] wait4(90, NULL, 0, NULL) = 90
[88] exit_group(0) = ?
```

Nota: Algunas líneas están modificadas por simplicidad.

a) (5p) Explicar en lenguaje natural cuáles son las syscalls involucradas, y qué hace cada una de ellas. Dibujar el árbol de procesos resultante. Para cada proceso explicar qué acciones realiza el mismo, y en qué orden. Considerar el manejo de señales en su respuesta.

b) (20p) Proponer una implementación en C tal que al ser ejecutada usando `strace` genere un output equivalente al anterior. El código entregado debe respetar el árbol genealógico de los procesos y el orden parcial de las syscalls.

Ejercicio 2.- Scheduling. **(25 puntos)**

Un sistema operativo cuenta con tres schedulers: (i) un Round Robin que utiliza un `quantum` de 3 unidades de tiempo, (ii) un FIFO, y (iii) un Shortest Remaining Time (SRT). Considere los siguientes procesos:

Proceso	Instante de llegada	Ráfaga de CPU
P_1	1	9
P_2	2	5
P_3	6	8
P_4	5	4

Se requiere:

a) Decidir cuál es el mejor scheduler si se considera el *turnaround promedio* como métrica de comparación. Asuma que los cambios de contexto tienen una duración nula.

b) Realizar los **diagramas de Gantt** correspondientes para justificar la respuesta del punto anterior.

Ejercicio 3.- Gestión de Memoria. **(25 puntos)**

Considere la siguiente secuencia de referencias a páginas: $7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0, 1$.

a) Realice el seguimiento de cada uno de los algoritmos de reemplazo listados abajo, considerando que el sistema cuenta con 3 *frames* (todos ellos inicialmente libres).

   i. LRU
   ii. FIFO
   iii. Segunda Oportunidad

b) Indique el *hit-rate* para cada algoritmo.

Ejercicio 4.- Concurrencia - sincronización. **(25 puntos)**

a) Se tiene una página web de un diario digital donde se permite a lxs usuarixs suscribirse para poder recibir las últimas noticias. Luego del registro a la página, debe recibir un mail confirmando su suscripción. En el servidor de la plataforma se tienen varios procesos que, entre otras tareas, se encargan de procesar las solicitudes de suscripción y de envío de mails. Hay un proceso (`Proceso 1`) que se encarga de procesar cada solicitud de suscripción. En una parte de su ejecución, se encarga de dejar en una cola de mensajes de tamaño N, un mensaje con información del mail del usuario. La cola de mensajes solo tiene las funciones `add` y `pop`, con su comportamiento esperado. Luego, otros procesos (`Procesos 2`) se encargan de desencolar los mensajes de la cola y mandar un mail al usuario que se registró (con la función `enviarMail` que recibe un mensaje). Una vez que el tamaño de la cola de mensajes esté *llena, no se deben colocar nuevos* mensajes. Sabemos que la cola de mensajes no es atómica. La interacción de cada *proceso con la cola se implementó* hace poco y su comportamiento se puede representar con el siguiente pseudo *código:*

```
// Variables compartidas
mutex = sem_init(1)
mails = sem_init(0)
buffer cola(N) // buffer de tamanio N

// Proceso 1
mensaje = esperarSubscripcionMailUsuario()
mutex.wait ()
        cola.add(mensaje)
mutex.signal()
mails.signal()

// Procesos 2
mutex.wait ()
mails.wait ()
        mensaje = cola.pop()
mutex.signal ()
enviarMail(mensaje)
```

Sin embargo, se reportó que esta parte del código no está funcionando correctamente y que no cumple exactamente con lo pedido. Nos encargaron la tarea de identificar qué está ocurriendo. Se pide:

- Encontrar qué problema/s tiene este código. Justificar detalladamente la/s causa/s y corregirlo, cumpliendo con la consigna del problema. También se pide justificar la corrección.

b) Dado el siguiente código ejecutado concurrentemente por N procesos, encontrar qué problemas de concurrencia puede haber y arreglarlo/s para que funcione correctamente. Considerar que este código está dentro de un *loop* y se tiene que cumplir su correctitud en cada iteración. Justificar cada decisión tomada.

```
// Compartidas
int N = cant_threads()
int count = 0
mutex = sem_init(1)
turnstile = sem_init(0)

// Proceso
while(1) {
        mutex.wait ()
                count += 1
        mutex.signal ()
        if count == N : turnstile.signal()
        turnstile.wait ()
        turnstile.signal ()

        // sección crítica

        mutex.wait ()
                count -= 1
        mutex.signal()
        if count == 0: turnstile.wait()
}
```