Universidad Nacional de La Matanza
Florencio Varela 1903 - San Justo - Buenos Aires - Argentina

Departamento de Ingeniería e Investigaciones Tecnológicas

Cátedra de Virtualización de Hardware (3654)

Jefe de Cátedra:	**Alexis Villamayor**
Docentes:	**Alexis Villamayor, Fernando Boettner**
Jefe de trabajos prácticos:	**Ramiro de Lizarralde**
Ayudantes:	**Alejandro Rodriguez, Fernando Piubel**

Año: **2024 – Primer Cuatrimestre**

Actividad Práctica de Laboratorio Nº 2

Procesos, comunicación y sincronización

Condiciones de entrega

- Se debe entregar por plataforma MIEL un archivo con formato ZIP o TAR (no se aceptan RAR u otros formatos de compresión/empaquetamiento de archivos), conteniendo la carátula que se publica en MIEL junto con los archivos de la resolución del trabajo.
- Se debe entregar el código fuente de cada uno de los ejercicios resueltos, junto con el Makefile necesario para su compilación y link-edición. Se rechazarán los ejercicios que no tengan su correspondiente Makefile. Además, se debe entregar un Dockerfile (para toda la APL) para la creación de la imagen que se utilizará al momento de la corrección.
- No se aceptarán binarios precompilados, en caso de entregar binarios serán ignorados. El ejercicio se probará solamente con los binarios generados al compilar los archivos de código fuentes entregados.
- No se aceptarán imágenes Docker ya creadas. Solo se utilizará la imagen creada a partir del Dockerfile entregado al momento de la corrección.
- En caso de que los archivos de código fuente no compilen debido a errores, no será posible la corrección del ejercicio debiendo reentregar una versión que compile correctamente para permitir su corrección.
- Se deben entregar lotes de prueba válidos para los ejercicios que reciban archivos o directorios como parámetro.
- Los archivos de código deben tener un encabezado en el que se listen los integrantes del grupo.

Consideraciones para la imagen de Docker

- Se recomienda usar la imagen `ubuntu:24.10` como imagen base, pero se puede optar por otra si lo cree conveniente.
- Al momento de la creación de la imagen, se deben copiar los archivos de la resolución del trabajo al directorio `/apl`. Es importante mencionar que los archivos se deben copiar y que no se deben utilizar volúmenes. Ejemplo de directorio:
  - `/apl`
    - `/apl/ejercicio1`
    - `/apl/ejercicio2`
    - `/apl/ejercicio3`
    - `/apl/ejercicio4`
    - `/apl/ejercicio5`

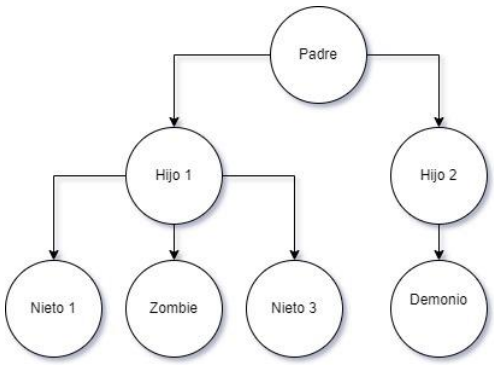
Criterios de corrección y evaluación generales para todos los ejercicios

- La corrección se realizará dentro del contenedor que se levantará a partir de la imagen creada desde el Dockerfile entregado.
- Los programas muestran una ayuda con el parámetro "-h". Deben permitir el ingreso de parámetros en cualquier orden, y no por un orden fijo.
- Cuando haya parámetros que reciban rutas de directorios o archivos se deben aceptar tanto rutas relativas como absolutas o que contengan espacios.
- No se debe permitir la ejecución del programa si al menos un parámetro obligatorio no está presente.
- Si alguna función utilizada en el programa da error, este se debe manejar correctamente y mostrar un error informando el problema de una manera amigable con el usuario, pensando que el usuario **no** tiene conocimientos informáticos.
- Si se generan archivos temporales de trabajo se deben crear en el directorio temporal /tmp; y se deben eliminar al finalizar el proceso, tanto en forma exitosa como por error, para no dejar archivos basura.
- Ante la finalización del proceso, tanto en forma normal como por las señales SIGTERM o SIGINT, se deben eliminar y/o cerrar correctamente los recursos que este hubiese creado (memorias compartidas, semáforos, FIFOs, sockets, etc.).
- En los ejercicios donde hay más de un proceso involucrado, la finalización de uno de los procesos debe manejarse correctamente desde el resto de los procesos. Ejemplo: ante el cierre del servidor, que los clientes informen correctamente de la situación y finalicen prolijamente.
- Salvo que se indique lo contrario en el enunciado, al finalizar el proceso principal no deben quedar procesos hijos como zombies o huérfanos.
- No se aceptan soluciones que utilicen la función system(), la misma es insegura y está completamente desaconsejado su uso. Tampoco el uso de popen(), se debe resolver la problemática utilizando bibliotecas de C/C++ y no comandos de Shell.
- Deseable:
    - Utilización de funciones en el código para resolver los ejercicios.
    - Modularización y uso de bibliotecas privadas.
    - Compilación sin warnings (advertencias).

Ejercicio 1

Objetivos de aprendizaje: Generación de procesos- manejo de jerarquías.

Se desea generar mediante el uso de la función fork la siguiente jerarquía de procesos.



Cada proceso deberá mostrar por pantalla la siguiente información:

Soy el proceso NOMBRE con PID nnnn, mi padre es PPID

Dado que los procesos no realizan ningún procesamiento en sí, realizar una espera hasta que se presione una tecla antes de finalizar para permitir verificar con los comandos ps o pstree la jerarquía de procesos generada.

Parámetros a recibir

Parámetro	Descripción
-h / --help	Muestra la ayuda del ejercicio

Ejercicio 2

Objetivos de aprendizaje: uso de threads, sincronización de threads

Se pretende realizar un proceso de búsquedas de cadenas en archivo similar a grep pero que utilice paralelismo para poder aprovechar el multiprocesamiento.

Se deberá generar un programa que reciba por parámetros 3 valores: un path a un directorio de entrada, el nivel de paralelismo (cantidad de threads a generar) y una cadena a buscar. No es necesario buscar con expresiones regulares, la cadena a buscar es un literal en cualquier parte del contenido del archivo, tampoco es necesaria la búsqueda recursiva de los archivos.

El proceso debe crear los threads solicitados, e ir pasándole de a un archivo por vez a procesar. Cuando el thread termina su proceso, se le asigna un nuevo archivo. El thread debe ser el mismo a lo largo de la vida del proceso (no se crea un thread por archivo), sino que son los mismos N threads desde el comienzo hasta que se agotan los archivos.

Se deberá mostrar por pantalla:

Nro de Thread:El nombre del archivo:El número de línea donde se encontró el string buscado

Parámetros a recibir

Parámetro	Descripción
-t / --threads <nro>	Cantidad de threads a ejecutar concurrentemente para procesar los archivos del directorio (Requerido). El número ingresado debe ser un entero positivo.
-d / --directorio <path>	Ruta del directorio a analizar. (Requerido)
-h / --help	Muestra la ayuda del ejercicio

Ejercicio 3

Objetivos de aprendizaje: uso de FIFO para IPC, opcionalmente select() o poll() para eventos de E/S

Se desea simular un sistema de sensores de huellas dactilares para aperturas de puertas y molinetes, que se comunican con un proceso centralizado que valida la huella si está registrada en un archivo y entonces le devuelve al sensor si la huella es válida o no.

Este proceso abre un FIFO para leer las entradas que le envían los distintos sensores y registra en un archivo de log (que toma como parámetro) la fecha y hora de la lectura, el número de sensor, la identificación de la huella o si no pudo identificarla. Debe quedar ejecutando como proceso demonio, y manejar correctamente su finalización a través de una señal.

A modo de representación numérica de la huella leída, se debe usar un número entero de 10 dígitos que es el que está registrado en el archivo.

Los procesos lectores toman por parámetro el número que les corresponde, y para hacer la simulación envían cada una cierta cantidad de segundos el mensaje vía el FIFO con su número de sensor y el ID a validar, que lo debe ir leyendo de un archivo de lecturas simuladas. El proceso de un lector finaliza luego de una cantidad de mensajes enviados, indicado también por parámetro. Puede haber varios lectores ejecutando al mismo tiempo.

Parámetros a recibir:

Parámetro	Descripción
-l / --log	Archivo de log donde se irán escribiendo los mensajes. En caso de no existir es necesario crearlo. (Requerido)
-n / --numero	Número del sensor (Requerido)
-s / --segundos	Intervalo en segundos para el envío del mensaje (Requerido)
-m / --mensajes	Cantidad de mensajes a enviar (Requerido)
-i / --ids	Archivo con los números de ID a informar desde el lector (Requerido)
-h / --help	Muestra la ayuda del ejercicio

Ejercicio 4

Objetivos de aprendizaje: uso de memoria compartida y semáforos para IPC, sincronización de procesos.

Implementar el juego “Preguntados”, para ello deberá crear dos procesos no emparentados que se comuniquen con memoria compartida y se sincronicen con semáforos.

Deberá existir un proceso Servidor que lea de un archivo separado por comas las preguntas, el número de respuesta correcta y 3 opciones a mostrar; y que va enviando al proceso Cliente las mismas y llevando el puntaje. El servidor debe hacer una partida de N preguntas (por parámetro) y llegado a la cantidad finaliza la partida y se debe quedar esperando nuevas partidas.

Ejemplo del archivo de preguntas:

```
¿De qué color era el caballo blanco de San Martín?,2,Negro,Blanco,Marrón
```

El proceso Cliente debe identificar al jugador al comenzar, va mostrando las preguntas y las opciones, y envía la respuesta al Servidor, mostrando luego el resultado final del puntaje.

Características del diseño:

- ✓ Se debe garantizar que no se pueda ejecutar más de un cliente a la vez
- ✓ Se deberá garantizar que solo pueda haber un servidor por computadora
- ✓ El servidor se ejecutará y quedará a la espera de que un cliente se ejecute
- ✓ Tanto el cliente como el servidor deberán ignorar la señal SIGINT (Ctrl-C)
- ✓ El servidor deberá finalizar al recibir una señal SIGUSR1, siempre y cuando no haya ninguna partida en progreso

Parámetros del servidor:

Parámetro	Descripción
-a / --archivo	Archivo con las preguntas (Requerido).
-c / --cantidad	Cantidad de preguntas por partida (Requerido).
-h / --help	Muestra la ayuda del ejercicio

Parámetros del cliente:

Parámetro	Descripción
-n / --nickname	Nickname del usuario (Requerido).
-h / --help	Muestra la ayuda del ejercicio

Ejercicio 5

Objetivos de aprendizaje: uso de sockets para IPC, comunicación de procesos remotos

Implementar juego “Preguntados” pero a través de conexiones de red, pudiendo admitir más de un cliente por servidor.

El servidor debe tomar por parámetro el puerto, mientras que el cliente debe solicitar la dirección IP (o el nombre) del servidor y el puerto del mismo.

Aclaraciones:

1. Si el servidor se cae (deja de funcionar) o es detenido, los clientes deben ser notificados o identificar el problema de conexión y cerrar de forma controlada.
2. Si alguno de los clientes se cae o es detenido, el servidor debe poder identificar el problema y cerrar la conexión de forma controlada y seguir funcionando hasta que solo quede un cliente.
3. Se deberá llevar un marcador indicando cuantos aciertos realizó cada jugador y al final mostrar el ganador.

Parámetros del Servidor:

Parámetro	Descripción
-p / --puerto	Número de puerto (Requerido)
-u / --usuarios	Cantidad de usuarios a esperar para iniciar la sala. (Requerido)
-a / --archivo	Archivo con las preguntas (Requerido).
-c / --cantidad	Cantidad de preguntas por partida (Requerido).
-h / --help	Muestra la ayuda del ejercicio

Parámetros del Cliente:

Parámetro	Descripción
-n / --nickname	Nickname del usuario (Requerido).
-p / --puerto	Nro de puerto (Requerido)
-s / --servidor	Dirección IP o nombre del servidor (Requerido)
-h / --help	Muestra la ayuda del ejercicio