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.
- No se aceptarán binarios precompilados, en caso de entregar binarios serán ignorados. El ejercicio se probará solamente con los binarios generados al compilar los fuentes entregados.
- En caso que los fuentes 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.
- La entrega se realizará en formato zip, con un directorio para cada uno de los ejercicios, sin espacios en su nombre.
    - Ejercicio1
    - Ejercicio2
    - Ejercicio3
    - Ejercicio4
    - Ejercicio5

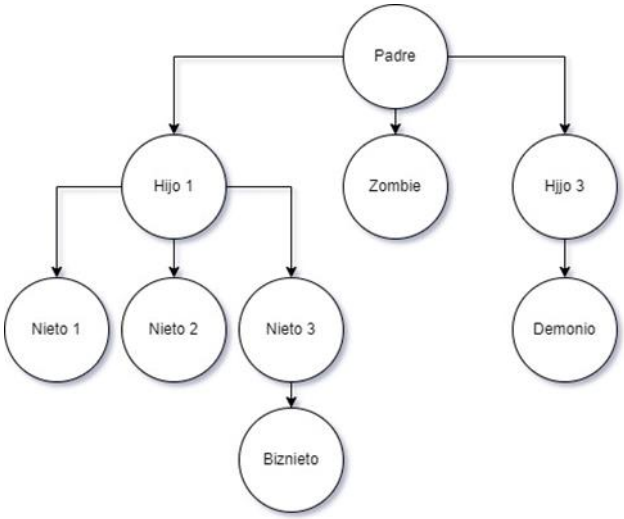
Criterios de corrección y evaluación generales para todos los ejercicios

- La corrección se realizará sobre Ubuntu 22.04 LTS de Windows (WSL).
- 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.
- 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: creación de distintos tipos de procesos, uso de fork y wait

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 desea desarrollar un programa capaz de contar la cantidad de números del 0 al 9 que aparece en todos los archivos de texto (con extensión .txt) que se encuentra en un determinado directorio. Para ello deberá recibir un número N que indique el nivel de paralelismo a nivel de threads y el path al directorio. Cada hilo deberá leer un archivo y contabilizar la cantidad de número que leyó. Adicionalmente al final se deberá mencionar la cantidad de números leídos totales.

El criterio es que se individualice a cada carácter entre letra y número, en caso de pertenecer al conjunto numérico se contabilizará. Por ejemplo: la palabra: “Hola C-3PO, soy R2-D2”, sumaría una aparición al número “3” y dos apariciones al número “2”.

Opcionalmente, si se recibe el parámetro -o / --output se generará un archivo con los resultados de los archivos procesados.

La salida por pantalla debe ser:

- Thread 1: Archivo leído test.txt. Apariciones 0=${cantCeros}, 1=${cantUnos}, etc
- Thread 2: Archivo leído prueba.txt. Apariciones 0=${cantCeros}, 1=${cantUnos}, etc
- Thread 1: Archivo leído pepe.txt. Apariciones 0=${cantCeros}, 1=${cantUnos}, etc
- Finalizado lectura: Apariciones total: 0=${cantTotalCeros}, 1=${cantTotalUnos}, etc

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.
-i / --input <directorio>	Ruta del directorio a analizar. (Requerido)
-o / --output <archivo>	Ruta del archivo con los resultados del procesamiento. (Opcional)

Ejercicio 3

Objetivos de aprendizaje: uso de FIFO para IPC

Se desea simular un sistema de toma de mediciones de sensores en una fábrica, que se comunican con un proceso centralizado que registra los mensajes enviados por los sensores para mostrar un tablero de control del proceso de fabricación.

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 y la medición (que es un número entero). Debe quedar ejecutando como proceso demonio, y manejar correctamente su finalización a través de una señal.

Los procesos sensores toman por parámetro el número que les corresponde, y para hacer la simulación generan cada una cierta cantidad de segundos una medición que generan en forma aleatoria, informando el mensaje vía el FIFO con su número de sensor y la medición a registrar. El proceso de un sensor finaliza luego de una cantidad de mensajes enviados, indicado también por parámetro. Puede haber varios sensores 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)

Ejercicio 4

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

Implementar el clásico juego de la memoria “Memotest”, pero alfabético.

Para ello deberá crear dos procesos no emparentados que utilicen memoria compartida y se sincronicen con semáforos.

Deberá existir un proceso “Cliente”, cuya tarea será mostrar por pantalla el estado actual del tablero y leer desde teclado el par de casillas que el usuario quiere destapar.

También existirá un proceso “Servidor”, que será el encargado de actualizar el estado del tablero en base al par de casillas ingresado, así como controlar la finalización partida.

Características del diseño:

- ✓ El tablero tendrá 16 casillas (4 filas x 4 columnas)
- ✓ Se debe garantizar que no se pueda ejecutar más de un cliente a la vez conectado al mismo servidor
- ✓ Se deberá garantizar que solo pueda haber un servidor por computadora
- ✓ Cada vez que se genere una nueva partida, el servidor deberá rellenar de manera aleatoria el tablero con 8 pares de letras mayúsculas (A-Z). Cada letra seleccionada solo deberá aparecer dos veces en posiciones también aleatorias
- ✓ 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
- ✓ El cliente deberá mostrar por pantalla el tiempo para saber cuánto se tarda en resolver el juego

Ejemplo la interfaz del cliente:

Esta interfaz es solo a nivel de ejemplo y quedará a criterio del grupo el diseño de la misma, por ejemplo, sería válido si quisieran hacer el ingreso de la fila y la columna en dos entradas separadas.

```
  0 1 2 3
0 - - - -
1 - - - -
2 - - - -
3 - - - -

Ingrese Fila-Columna: 0-0
```

```
  0 1 2 3
0 A - - -
1 - - - -
2 - - - -
3 - - - -

Ingrese Fila-Columna:
```

Ejercicio 5

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

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

El servidor debe tomar por parámetro el puerto y la cantidad de clientes para iniciar la partida, 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. Los clientes deben ver el estado actualizado del tablero cuando ocurran aciertos y solo se permitirá una jugada por turno de cada cliente.
4. 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)
-j / --jugadores	Cantidad de jugadores a esperar para iniciar la sala. (Requerido)

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)