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: 2025 – 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`

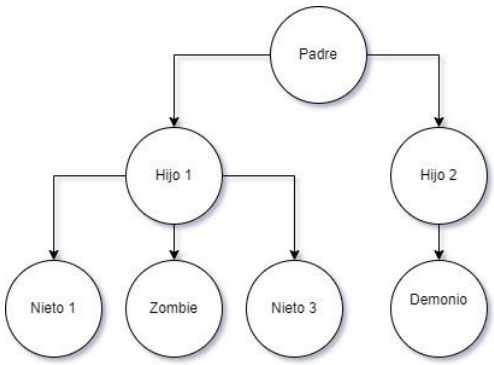
Criterios de corrección y evaluación generales para todos los ejercicios

- 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 simular la operatoria de una planta de logística. Para esto, se deberá crear un programa que reciba por parámetros 4 valores: el path a un directorio de procesamiento, cantidad de generadores de paquetes, cantidad de procesadores de paquetes y una cantidad de paquetes a procesar.

El proceso debe crear los threads solicitados, tanto para los generadores como los procesadores, y distribuir la cantidad de paquetes a generar entre ellos. Estos hilos se encargarán de generar archivos en el directorio de procesamiento con el siguiente nombre y formato en su contenido:

- Nombre: id.paq
- Contenido (una única línea): id_paquete;peso;destino
    - Id_paquete: número secuencial entre 1 y 1.000
    - Peso (Kilogramos): valor aleatorio decimal entre 0 y 300
    - Destino: id de la sucursal destino. Valor entero aleatorio entro 1 y 50

La operatoria será la siguiente:

- Al iniciar el proceso se limpiará el directorio para poder generar los nuevos paquetes.
- Los threads productores generarán los archivos de forma concurrente mientras haya lugar en el buffer virtual de 10 unidades.
- Los threads consumidores se encargarán de leer los archivos que generen los productores e ir acumulando el peso de los paquetes agrupados por sucursales de destino. Una vez procesado el archivo lo moveran a un subdirectorio {dir_procesamiento}/procesados para no volver a leerlo.
- Una vez finalizado el procesamiento se deberá mostrar por pantalla un resumen de la cantidad de paquetes procesados y el peso total calculado por cada sucursal.

Parámetros a recibir

Parámetro	Descripción
-d / --directorio <path>	Ruta del directorio a analizar. (Requerido)
-g / --generadores <nro>	Cantidad de threads a ejecutar concurrentemente para generarlos archivos del directorio (Requerido). El número ingresado debe ser un entero positivo.
-c / --consumidores <nro>	Cantidad de threads a ejecutar concurrentemente para procesar los archivos del directorio (Requerido). El número ingresado debe ser un entero positivo.
-p / --paquetes	Cantidad de paquetes a generar (Requerido). El número ingresado debe ser un entero positivo.
-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 una cola de impresión por FIFO. Para esto, varios procesos cliente pueden enviar un "trabajo de impresión" (archivo) a través de un FIFO de IMPRESION (/tmp/cola_impresion) y el proceso servidor de impresión se encargará de procesarlos.

Servidor:

- Limpia la cola de impresión y los resultados previos.
- Lee los trabajos desde el FIFO uno a la vez.
- Simula la impresión concatenando el contenido del archivo recibido por FIFO en un archivo log temporal /tmp/impresiones.log
- Previamente a concatenar el contenido, genera una línea indicando el nombre del archivo, el PID del proceso que generó la petición y la fecha y hora.
  Por ejemplo: PID {XXXXX} imprimió el archivo {XXXXXXXXXXX} el día {DD/MM/YYYY} a las {HH:MM:SS}
  A continuación escribirá el contenido del archivo enviado por FIFO.
- Luego de procesar el archivo, envía un mensaje al cliente por un FIFO privado (/tmp/FIFO_{PID}) para finalizar la operación de impresión del lado del cliente.
- Una vez procesados todos los archivos finaliza.

Cliente:

- Recibe por parámetro el archivo a imprimir.
- Escribe en el FIFO de impresión los siguientes datos para iniciar el proceso de impresión: {PID}:{PATH_ARCHIVO}
- Se queda esperando la confirmación de finalización por parte de Servidor en un FIFO privado (/tmp/FIFO_{PID})
- Una vez recibida la confirmación finaliza, limpiando los recursos generados.
  En caso de recibir un mensaje de error de parte del Servidor, mostrará el mensaje por pantalla y finalizará.

Aclaraciones:

- Los archivos serán rutas relativas o absolutas.
- Se debe validar la existencia de los archivos a imprimir.
- Un archivo vacío NO es un archivo válido y se deberá generar un mensaje de error correspondiente, tanto en el archivo de logs como en la confirmación al cliente.

Parámetros a recibir (Servidor):

Parámetro	Descripción
-i / --impresiones	Cantidad de archivos a imprimir (Requerido). Valor entero positivo.
-h / --help	Muestra la ayuda del ejercicio

Parámetros a recibir (Cliente):

Parámetro	Descripción
-a / --archivo	Archivo a imprimir. (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 “Ahorcado”, 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 saltos de línea, las frases disponibles para jugar y que enviará al proceso Cliente llevando el puntaje. El servidor debe hacer una partida de N intentos (por parámetro). El servidor debe quedar a la espera de nuevos clientes (cuando finalice la sesión en curso) y finalizará al recibir una señal. Al finalizar muestra el resultado con el cliente ganador (menor cantidad de tiempo en adivinar la frase).

Ejemplo del archivo de preguntas:

```
la computadora no se apaga
el archivo fue eliminado
proceso en estado bloqueado
memoria virtual agotada
el kernel gestiona los recursos
```

Secuencia del juego:

- Servidor: espera la conexión del cliente y le envía de forma aleatoria la frase a adivinar.
- Cliente: establece la conexión y recibir la frase a adivinar. Envía una letra y espera respuesta.
- Servidor: recibe la letra, verifica si coincide y envía al cliente el resultado actualizado (indicar la cantidad de intentos que le quedan en caso de error)
- Una vez identificada la frase o agotados los intentos se envía al cliente el resultado y guarda en el ranking la posición del jugador.
- Servidor: al recibir la señal de finalización, muestra los resultados y finaliza.

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. En ese caso, espera la finalización de la partida y luego finaliza.
- ✓ El servidor deberá finalizar al recibir una señal SIGUSR2. En caso de que exista una partida en progreso, la finaliza y luego muestra los resultados.

Parámetros del servidor:

Parámetro	Descripción
-a / --archivo	Archivo con las frases (Requerido).
-c / --cantidad	Cantidad de intentos 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 “Ahorcado” 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 máximos concurrentes. (Requerido)
-a / --archivo	Archivo con las frases (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