Práctica 2

Procesos – Señales – Sincronización

Datos útiles

Para realizar esta práctica puede utilizar la máquina virtual provista por la cátedra (se recomienda hacer un snapshot de la VM antes de empezar a usarla, para poder volver atrás en caso de que algo deje de funcionar).
El link de descarga se encuentra en el moodle junto a las credenciales de acceso. Además, la VM se encuentra instalada en las PCs de la Sala A del Lab. Bacarla donde se realizan las prácticas de la materia.

1) `fork()`, `getpid()` y `getppid()`
    a) Explique el funcionamiento de la función `fork()`.
    b) Si encuentra esta sentencia en un programa: `if ((pid=fork())!= 0)` ¿cuál será el hijo, la rama que va por **`then`** o la que va por **`else`**?
    c) Que obtengo con `getpid()` y con `getppid()`?
    d) Explique la diferencia entre `waitpid()` y `wait()`.
    e) Explique la utilidad de las macros `WIFEXITED()` y `WEXITSTATUS()`
    f) Compile y ejecute cada uno de los programas de ejemplo `padrehijo.c` e `hijosynietos.c`.
    g) Analice el comportamiento de cada uno de ellos y determine qué es lo que hace el proceso padre y el o los procesos hijos o nietos.
    s
2) Funciones `execve()`
    a) Busque en las páginas de manual las diferentes versiones de las funciones `exec()`.
    b) Explique las diferencias entre cada una de ellas.
    c) Compile y ejecute el ejemplo `ejemploexecv.c`. Experimente modificando los parámetros que se pasan a la función `ls` para obtener distintos formatos de salida.

d) Realice un programa que lea de la entrada estándar el nombre de un programa y cree un proceso hijo para ejecutar dicho programa. El padre debe esperar a que el proceso hijo termine.

3) Señales
   a) Describa que es una señal, y como es el manejo de esta por parte de un proceso.
   b) Describa cómo es la utilización de los comandos del sistema operativo para enviar y recibir señales (`kill`).
   c) Explique cómo se instala un manejador de señales.
   d) De una explicación del funcionamiento del programa de ejemplo `senial.c` visto en la clase.
   e) Explique el funcionamiento de las funciones `alarm()` y `pause()`.

4) Herencia
   a) Explique cómo los hijos heredan las variables existentes en el padre.
   b) Si uno de los hijos modifica el valor de una variable, ¿se modifica también en el padre?
   c) Si el padre modifica el valor de una variable mientras el hijo está activo, ¿se modifica también en el hijo?
   d) Analice los ejemplos `herencia1.c` y `herencia2.c`. Explique qué es lo que hereda el hijo del padre en cada caso.
   e) Un ingeniero afirma que los procesos hijos heredan los manejadores de señal y los archivos abiertos por su proceso padre. ¿Esto es cierto? Explique.

5) Sistemas IPC
   a) Explique el procedimiento para la obtención de claves únicas
   b) Explique el funcionamiento del comando `ipcs` y sus opciones
   c) Explique el funcionamiento del comando `ipcrm` y sus opciones

6) Memoria compartida
   a) Describa el proceso a seguir para utilizar una variable en memoria compartida

b) Describa los FLAGS que pueden utilizarse como parámetro en las funciones `shmget()`, `shmat()` y `shmctl()`. Explique cuál es su significado y de un ejemplo de su utilización

7)

a) Analice, comente, compile y ejecute los programas de ejemplo: `memcomp_1.c` y `memcomp2.c`.
b) Utilizando el comando `ipcs` analice para los casos anteriores, que sucede con los recursos en uso si:
   i) Al estar corriendo ambos programas uno u otro se termina abruptamente mediante una señal `kill` enviada desde otra consola,
   ii) Si se ejecutan simultáneamente dos instancias de un mismo programa,
   iii) Si se ejecuta primero un programa que utiliza (escribe o lee en un recurso) y luego se ejecuta el otro programa.

8)

a) Analice, comente, compile y ejecute los programas de ejemplo: `memcomp_1.c` y `memcomp3.c`.
b) Describa la utilidad de uso de `shmctl()` cuando se le pasa el argumento `IPC_STAT`, ¿Qué información nos permite obtener?

9) Escriba un programa que cada vez que reciba la señal `SIGUSR1` cree un hijo, que imprimirá su `PID` y el del padre cada cinco segundos. El proceso padre deberá escribir cada 10 segundos el número de procesos creados y su pid, esto se debe hacer mediante una alarma (usando `alarm`). Transcurrido el tiempo de la alarma se imprimirá el mensaje por pantalla. Cuando reciba la señal `SIGUSR2`, el programa deberá terminar la ejecución de todos los hijos creados usando `kill(pid, sig)`

10) **(Ejercicio entregable)**

Escriba un programa que cada vez que reciba la señal `SIGUSR1` cree un hijo, que imprimirá su PID y el del padre cada cinco segundos. Por otra parte, cada vez que reciba la señal `SIGUSR2`, cree un hijo que deberá imprimir su PID y ejecutar, mediante una función de la familia `exec()`, un programa de los que reside en `/bin` (p. ej., `ls`, `date`, `time`, `cd`, `pwd`, ...), luego de esto dicho hijo terminará. Si el programa original, recibe la señal `SIGTERM`, terminará

todos los procesos hijos que fueron creados con `SIGUSR1`, a razón de uno por segundo, indicando el PID de cada uno de los procesos terminados y por último terminará él mismo.

11) **(Ejercicio entregable)**

Escriba un programa que utilice un arreglo de 20 `int`, compartido entre dos procesos A y B de manera tal que A escriba los primeros 10, y B los lea solamente después que se completó la escritura. De la misma manera B escribirá los 10 últimos enteros del arreglo y A los leerá solamente después que B complete la escritura. Este proceso se repetirá hasta que el Proceso A reciba la señal `SIGTERM`, y cuando la reciba debe a su vez hacer terminar el proceso B, y liberar todos los recursos compartidos utilizados.

Los datos serán números entre 0 y 26, (representando letras entre la **a** y la **z**), que serán cifrados previo a la escritura mediante una función de cifrado de la forma $f(x) = (ax + b)\ mod\ n$. En donde (a,b) serán claves públicas que estarán disponibles en memoria compartida. Para el descifrado de los datos se usará la función inversa $f^{-1}(x) = (kx + c) mod\ n$, en donde (k,c) son claves privadas conocidas solo por el proceso lector. Como función de cifrado se usará $f(x) = (4x + 5)\ mod\ 27$ y para el descifrado$f^{-1}(x) = (7x + 19)\ mod\ 27$

Ambos procesos realizarán primero la escritura de los datos en la zona correspondiente y luego la lectura. Ninguno de los dos procesos puede escribir en su zona hasta que el otro no haya leído los datos, excepto la primera vez. La escritura en ambos casos se realizará a razón de un valor por segundo, mientras que la lectura se realizará tan rápido como se pueda y se imprimirá en pantalla junto con la identificación del proceso que está leyendo.

El programa debe ser único, es decir el mismo programa puede actuar como proceso A o como proceso B, dependiendo de un parámetro que se pase en la línea de comando. (por ejemplo “programa a” o “programa b”). Realice su implementación basada en memoria compartida.

Pautas de entrega

La resolución debe hacerse en base a las pautas explicadas en clase, es decir:

- Seleccione una plataforma de control de versiones (ej: GitLab, GitHub, Bitbucket, …)
- Dentro de la misma, debe crear un grupo:
  - Nombre: **`SOyR 2024 Grupo X`** (**`X`** representa el número de grupo).
  - Descripción: apellido, nombre y número de alumno de cada integrante del grupo.
  - Debe agregar al docente asignado con el rol de `Owner`.
- Dentro del grupo, debe crear un repositorio:
  - Nombre: **`Entregable Práctica X`** (`X` representa el número de práctica)
  - El repositorio debe contener lo siguiente:
    - Archivo de programa **`eX.Y`** (**`X`** representa el número de ejercicio. **`Y`** representa la extensión del archivo, ejemplo: **`c`**, **`sh`**) correctamente estructurado y con los comentarios pertinentes.
    - Archivo **`README.md`**. Usando sintaxis **`markdown`** incorporar lo siguiente:
      - Copia del enunciado
      - Interpretación del problema. (qué debo hacer, qué datos tengo y cómo se organizan, cuál es la interfaz con el usuario)
      - Propuesta de solución (Cómo resuelvo lo pedido) describo los modelos utilizados: algoritmos, matemáticos o conceptuales. Aproximar la solución con un diagrama o pseudocódigo, modularizar.
      - Instrucciones de uso
    - Directorio **`imagenes`** ó **`figuras`** (opcional: en caso de que se requiera adjuntar imágenes en el **`README.md`**) con las figuras correspondientes.
    - **`.gitignore`** (opcional: en caso de que haya trabajado con más archivos en local)