FISOP - Parcialito TP1

Puntos totales 100/100

Parcialito sobre el TP1 de la materia Sistemas Operativos (FIUBA)

Se ha registrado el correo del encuestado (**jrehl@fi.uba.ar**) al enviar este formulario.

0 de 0 puntos

Antes de arrancar, dejanos tus datos.

Y tu nombre completo (apellido y nombre) *

Juana Rehl

Ingresá tu padrón: *

112185

Preguntas 100 de 100 puntos

Son 20 preguntas en total.

✓ La función exit() a diferencia de _exit(): * 5/5

- (●) Realiza algunas tareas de mantenimiento relacionadas con estructuras creadas por "libc" (biblioteca estándar de C) antes de llamar a la syscall exit. ✓
- ( ) No existe ninguna diferencia y son aliases una de la otra por motivos de compatibilidad con versiones anteriores de la libc.
- ( ) Libera la memoria y “file descriptors” alocados por el proceso para que, al terminar, el sistema operativo no pierda memoria de manera permanente.
- ( ) Es meramente un wrapper de la syscall exit.

✓ ¿Cómo se logra la redirección de un flujo estándar a un archivo? * 5/5

- (●) Se "apunta" el flujo estándar al archivo deseado ✓
- ( ) Se envía un argumento extra como parte de la syscall “exec”
- ( ) Se envía un argumento extra como parte de la syscall “open” al abrir el archivo
- ( ) Ninguna de las anteriores

✓ Para un comando de tipo “pipe”: * 5/5

- (●) La shell espera a que terminen ambos procesos para devolver el prompt ✓
- ( ) La shell solamente espera a que termine el comando de más a la derecha
- ( ) La shell no espera por ninguno y devuelve el prompt inmediatamente
- ( ) La shell solamente espera a que termine el comando de más a la izquierda

✓ Sobre el comando “cd”: * 5/5

- ( ) Se implementa con la syscall "cd" (mismo nombre)
- ( ) Debe ser un built-in de la shell por motivos de performance.
- (●) Debe ser un built-in de la shell para que cumpla su cometido. ✓
- ( ) Puede implementarse como binario ejecutable manteniendo su comportamiento.

✓ Las características de un “file descriptor” son: * 5/5

- (●) Es una referencia al archivo subyacente (independientemente de la naturaleza de ese archivo). ✓
- ( ) No se puede duplicar
- ( ) Es el archivo abierto “per se”.
- ( ) Cuando se cierra, se elimina directamente el archivo físico relacionado.

✓ Cuando se llama "waitpid(0, ...)" el comportamiento es: * 5/5

- ( ) Es un argumento inválido y la syscall falla
- (●) Espera por todos los procesos hijos cuyo PGID sea el mismo que el del proceso que ejecuta la syscall ✓
- ( ) Esperar por cualquier proceso hijo
- ( ) Esperar por el proceso hijo cuyo PID es el cero

✓ ¿Cuál es el mecanismo para setear las variables de entorno temporales? * 5/5

- (●) En el proceso ejecutor del comando, se hace un setenv de cada variable (antes de hacer "exec") ✓
- ( ) Antes de crear el proceso ejecutor del comando, se hace un setenv de cada variable.
- ( ) En el proceso ejecutor del comando, se pasan esos únicos valores como tercer argumento (eargv) a la syscall “exec”.
- ( ) Ninguna de las anteriores

✓ La configuración de los handlers de señales: * 5/5

- [ ] Se preserva a través de un "exec(2)"
- [ ] No se preserva a través de un "fork(2)"
- [x] Se preserva a través de un "fork(2)" ✓
- [x] No se preserva a través de un "exec(2)" ✓

✓ La ejecución de los comandos en “pipe”: * 5/5

- (●) Ocurren en simultáneo: es decir, el comando de la izquierda escribe mientras el comando de la derecha ya está leyendo. ✓
- ( ) Ocurren en secuencia: es decir, el comando de la derecha tiene que esperar a que termine el de la izquierda para poder ser ejecutado.
- ( ) Ocurre en orden inverso: es decir, el comando de la derecha se ejecuta antes que el izquierdo pueda iniciar.
- ( ) Ninguna de las anteriores

✓ Un comando ejecutado en “background”: * 5/5

Nota: tener en cuenta una implementación correcta

- ( ) Es un proceso al cual nunca se le hace “wait”
- (●) Se lo “monitorea” para que cuando finalice no quede zombie ✓
- ( ) No puede tener redirección de su flujo estándar
- ( ) Todas las anteriores

✓ Cuando creo un nuevo proceso con “fork”: * 5/5

- [x] El código binario del proceso nuevo es el mismo que el del padre ✓
- [x] Los “file descriptors” son un duplicado de los que tenía el padre (referencian a los mismos archivos). ✓
- [ ] La ejecución arranca desde el comienzo del programa.
- [ ] Las variables de entorno del proceso nuevo se resetean (no comparte ninguna con el padre)
- [ ] Todas las anteriores

✓ La syscall “exec” reemplaza todo el *address space* del proceso actual (datos + código binario) pero preserva el estado de los “file descriptors”: *5/5

- (●) Verdadero ✓
- ( ) Falso

✓ Todo proceso siempre comienza con tres “file descriptors” abiertos: * 5/5

- Entrada estándar
- Salida estándar
- Un pipe para comunicarse con el padre

- (●) Falso ✓
- ( ) Verdadero

✓ ¿Qué ocurre cuando una señal interrumpe la ejecución de una syscall? * 5/5

- ○ No es un comportamiento que esté definido
- ○ La syscall se reanuda automáticamente cuando termina la ejecución del handler de la señal
- ◉ La syscall se reanuda únicamente cuando se configuró el handler apropiadamente ✓
- ○ La syscall nunca se reanuda y falla con el error EINTR

✓ Los valores de las variables “mágicas”: * 5/5

- ◉ Se obtienen en runtime de acuerdo al estado de la shell ✓
- ○ Se obtienen de variables de entorno especiales que dispone el kernel
- ○ La syscall “exec” es capaz de obtener esos valores y expandirlos.
- ○ Se cargan en la inicialización de la shell, para luego ser consumidas

✓ ¿Cómo se produce la expansión de variables? * 5/5

- ○ La syscall “exec” reemplaza toda ocurrencia del patrón “$VARIABLE” por el valor de la misma.
- ○ En el proceso ejecutor, antes de hacer “exec”, se reemplaza toda ocurrencia del patrón “$VARIABLE” por el valor de la misma
- ○ El binario que se termina ejecutando las reemplaza como parte de su código
- ◉ La shell reemplaza toda ocurrencia del patrón “$VARIABLE” por el valor de la misma, antes de llamar a “fork”. ✓

✓ Sobre el comando “pwd”: * 5/5

- ○ Existe solamente como built-in
- ○ Existe solamente como binario ejecutable
- ○ No es un comando válido de la shell
- ◉ Se puede implementar tanto como binario ejecutable como built-in ✓

✓ Cuando la shell realiza la redirección de la salida estándar (*stdout*) en un archivo, los datos se envían tanto a la pantalla como al archivo: *5/5

- ◉ Falso ✓
- ○ Verdadero

✓ La expansión de una variable que no existe, por ejemplo "echo hola $NO_EXISTE", resulta en que a "exec" le llegue: *5/5

- ○ exec("/bin/echo", ["/bin/echo", "hola", "\n", NULL])
- ○ exec("/bin/echo", ["/bin/echo", "hola", " ", NULL])
- ○ exec("/bin/echo", ["/bin/echo", "hola", "", NULL])
- ◉ exec("/bin/echo", ["/bin/echo", "hola", NULL]) ✓

✓ Si los procesos en segundo plano están en un "process group" igual que el de la shell: *5/5

- ○ Al hacer "exec" se genera un error con los flujos de redirección estándar
- ◉ El "wait" del handler de SIGCHILD puede hacerse usando "0" como pid ✓
- ○ Al hacer "fork" se genera un error
- ○ Ninguna de las anteriores