FISOP - Parcialito TP1

Puntos totales 69/100

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

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

0 de 0 puntos

Antes de arrancar, dejanos tus datos.

Y tu nombre completo (apellido y nombre) *

Luis Condori

Ingresá tu padrón: *

98237

Preguntas 69 de 100 puntos

Son 20 preguntas en total.

✗ ¿Cómo se produce la expansión de variables? * 0/5

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

Respuesta correcta

- ◉ La shell reemplaza toda ocurrencia del patrón “$VARIABLE” por el valor de la misma, antes de llamar a “fork”.

✓ En el modo de lectura “no canónico”: * 5/5

- ☑ El usuario tiene acceso a todos los caracteres que se reciben por teclado. ✓
- ☐ El kernel no realiza ninguna acción especial frente a un determinado caracter.
- ☐ El efecto de “echo” por la pantalla sigue estando activo
- ☐ Todas las anteriores

Respuesta correcta

- ☑ El usuario tiene acceso a todos los caracteres que se reciben por teclado.
- ☑ El kernel no realiza ninguna acción especial frente a un determinado caracter.

✓ Cuando se 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

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

- (●) Luego de crear el proceso para ejecutar el comando, la shell hace un setenv de cada variable. ✓
- ( ) Antes de crear el proceso para ejecutar el comando, la shell hace un setenv de cada variable.
- ( ) En el proceso hijo se pasan esos únicos valores como tercer argumento a la syscall “exec”.
- ( ) Ninguna de las anteriores

✓ 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 secuencialmente: 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

✓ ¿Cómo se logra la redirección de un flujo estándar en 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

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

- ○ Falso
- ◉ Verdadero ✓

✗ Los valores de las variables “mágicas”: * 0/5

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

Respuesta correcta

- ◉ Se obtienen en runtime de acuerdo al estado de la shell

✗ En el modo de lectura “canónico”,: * 0/5

- ○ El kernel procesa ciertos caracteres que nunca llegan al usuario.
- ○ La entrada estándar contiene todos los caracteres posibles que el usuario ingresó en el teclado
- ◉ El usuario recibe cada caracter tan pronto como se escribe en el teclado ✗
- ○ Solamente se reciben los caracteres ASCII pertenecientes a las letras

Respuesta correcta

- ◉ El kernel procesa ciertos caracteres que nunca llegan al usuario.

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

- ○ 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

✓ El modo “canónico” solamente envía los caracteres leídos del teclado cuando se ingresa el “salto de línea”: 5/5 *

- ◉ Verdadero ✓
- ○ Falso

✓ Las “flechas” del teclado se codifican como secuencias de “escape” conformadas por varios bytes: 5/5 *

- ○ Falso
- ◉ Verdadero ✓

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

- ○ La shell no espera por ninguno y devuelve el prompt inmediatamente
- ◉ 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 izquierda
- ○ La shell solamente espera a que termine el comando de más a la derecha

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

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

✗ Todo proceso siempre comienza con tres “file descriptors” abiertos: Entrada estándar | Salida estándar | Un pipe para comunicarse con el padre 0/5 *

- [ ] Falso
- [x] Verdadero ✗

Respuesta correcta

- [x] Falso

✗ Cuando creo un nuevo proceso con “fork”: * 4/5

- [x] El código 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.
- [x] Las variables de entorno del proceso nuevo se resetean (no comparte ninguna con el padre) ✗
- [ ] Todas las anteriores

Respuesta correcta

- [x] El código 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).

✓ Sobre el comando “pwd”: * 5/5

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

✗ Sobre el comando “cd”: * 0/5

- ○ 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 perfectamente como binario ejecutable.

Respuesta correcta

- ◉ Debe ser un built-in de la shell para que cumpla su cometido.

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

- ○ 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.
- ○ No existe ninguna diferencia y son aliases una de la otra por motivos de compatibilidad con versiones anteriores de la libc.
- ◉ Realiza algunas tareas de mantenimiento relacionadas con estructuras creadas por la libc (biblioteca estándar de C) antes de llamar a la syscall exit. ✓

✗ La expansión de una variable que no existe, resulta en: * 0/5

- ○ Imprimir un “ ” (espacio)
- ○ Imprimir un caracter no visible
- ◉ Imprimir un caracter vacío ✗
- ○ No imprimir absolutamente nada en su lugar

Respuesta correcta

- ◉ No imprimir absolutamente nada en su lugar