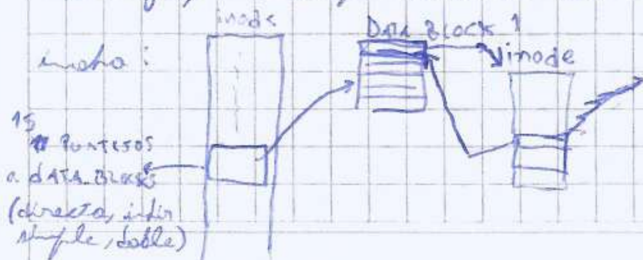
b) Solo contando escrituras a disco veamos lo que hay que hacer:
Voy recorriendo los inodos (partiendo desde root /) hasta llegar a la dir_entry correspondiente a /home/JUAN. Esta dir_entry supongo que tiene la estructura: dir_entry{

```
int inode //Su inodo
int record_lenght //Su tamaño
int name_lenght //largo del nombre
int FILE_TYPE //0x1 archivo, 0x2 directorio
char* name[]
}
```

Con esta estructura, podría modificar el nombre del directorio JUAN para renombrarlo a MARÍA (supongo que la diferencia de un caracter entre los nombres no trae problemas). Por lo tanto, tengo una escritura a disco para reescribir el bloque que contiene la dir_entry de /home/JUAN.
Una vez que modifique el nombre, voy al inodo correspondiente al directorio y busco el inodo del archivo "listado.txt". Una vez que lo encuentro el proceso es similar al de antes, renombro el nombre del inodo a "archivo.txt"
De esta forma tengo que hacer otra escritura a disco para modificar el bloque con el inodo de "listado.txt".
TOTAL = 2 escrituras          ok

c) Sí, se podría. Para hacer esto tendría que hacer que el inodo del archivo "se apunte a sí mismo". Esto podría realizarse si dentro de la ~~...~~ inodo del archivo existe una dir_entry asociada a ese mismo archivo. De esa forma, al intentar abrir el archivo, como busco a qué inodo ir (y leer sus bloques de datos) si el inodo es él mismo entraré en un loop infinito.
Por ejemplo el puntero del DataBlock del inodo podría apuntar a el mismo inodo:




(NO Tengo tiempo para profundizar más)

2) a) Para ver la vulnerabilidad de este código analicemos cada una de sus partes.

Analicemos la función "cambiarPassword()" que se invoca desde main(). En principio, la función "obtenerUsuarioActual()" es desconocida así que puedo suponer que es "segura". Ahora numeroDeUsuario tiene el valor correcto del usuario que quiere cambiar su password (PEPITO). Creo dos variables de 250 caracteres exactamente para guardar su password y la confirmación. Esto está bueno porque en principio limito la cantidad de caracteres que puedo almacenar en esas variables.

Luego el fgets lee de la stdin exactamente 250 caracteres para escribirlos en password. Esto está perfecto porque no permito un overflow del stack, no importa cuánto ingrese sólo voy a almacenar 250. Suponiendo que PEPITO es un usuario legítimo y tiene usuario y contraseña guardados (y que las funciones del if son "seguras") se le permite ingresar a la sección de cambio de contraseña.

De nuevo aquí, el fgets limita, con mucho criterio, la cantidad de caracteres que se leen de la stdin para escribir en password. Y ahora viene la vulnerabilidad en la función gets(). Esta función no limita la cantidad de caracteres que puedo escribir en passwordConfirmación entonces si PEPITO tiene malas intenciones podría ingresar +250 caracteres con una entrada determinada para realizar un stackoverflow. Veamos un diagrama del stack:

| |
|---|
| local vars |
| ret addr |
| registros |
| stack frame de main |

crecimiento del stack
registros → (varía dependiendo de la arquitectura)

Con este diagrama en mente podemos observar que es posible que al ejecutar la función gets y con una entrada determinada de PEPITO, se pise, por ejemplo, la ret addr (dirección de retorno) de la función, y termine en una dirección que ejecuta código malicioso. También es importante observar que se podrían pisar las variables correspondientes al usuario y la contraseña y quizás actualizar la contraseña de otro usuario ("admin"/"root" en el peor caso).

~~Aunque esto último quizás no sea tan directo. La mayor vulnerabilidad del código es que brinda la posibilidad de realizar un stack overflow y pisar la dir de retorno de la función.~~

b) Por ese riesgo, si un atacante logra explotar la vulnerabilidad quiere decir que puede llegar a pisar las variables locales de la función (password y nombre de usuario) para que password y passwordConfirmacion sean iguales y el usuario sea alguno determinado. Lo más grave sería un escalamiento de privilegios que permitiría cambiar la contraseña de root, por ejemplo. Pero en principio todas las contraseñas corren el riesgo de ser cambiadas por un usuario cualquiera del sistema (de a una por vez, claramente).

Bolling, Outs, DMA

071177 | B | MUY BIEN | N: 3

3) a)

i) En este caso no tendría mucho sentido utilizar un buffer para almacenar una mayor cantidad de entrada del usuario (o salida). Pues al ser la pantalla táctil de un celular (con Android :)) la interacción (y experiencia de) con el usuario ~~no es tan inmediata (si hubiera)~~ se vería considerablemente degradada. Uno como usuario quisiera tener una interacción lo más fluida posible con la pantalla y como programador de un driver para la pantalla debería tener en cuenta su contexto de uso.

ii) En este caso tendría sentido tener un buffer en memoria. Porque un disco virtual debería ser capaz de ir acumulando entrada de usuario (por ejemplo si se suben gran cantidad de archivos (y si son pesados)) e ir "subiéndola" o "escribiéndola" a medida que pueda. Pero la acumulación de entrada no debería afectar a la interacción con el usuario, o hacerlo lo menos posible. Un ejemplo que se me ocurre si no tengo un buffer es que pueda escribir de a un archivo por vez y no poder ir "subiendo" más hasta que se complete dicha "escritura". Eso no sería un buen escenario en este contexto.

iii) ~~Este caso considero que es LIGERAMENTE más ambiguo. En primer lugar, si el contexto de la multiplicación de matrices es frecuente~~

Dado el contexto de que la GPU se utiliza exclusivamente para multiplicar matrices en cálculos científicos considero que tendría sentido utilizar un buffer para ir almacenando una mayor cantidad de entrada del usuario (o salida) antes de enviarla (o recibirla) a el dispositivo. La razón es porque si son multiplicaciones de matrices es muy probable que tenga que hacer cálculos numéricos extensos (si son matrices grandes, que en este contexto asumo que sí) y al mismo tiempo ir guardando los resultados para seguir con el resto de las operaciones, en el caso de

que, por ejemplo, sean varios multiplicaciones juntas (seguidas): [illegible]

Entonces, en este contexto un buffer podría ser de utilidad siempre y cuando esté "bien" manejado por el driver. Es decir, no debería hacer que las cuentas y los resultados pierdan mucha "performance" (enlenteciendo mucho el proceso por ejemplo).

Por otro lado, si estamos en otro contexto, por ejemplo en un videojuego con procesamiento gráfico considerable, quizás convendría que la entrada se intercambie en cuanto esté disponible para que sea "fluido" y no perder performance. En ese contexto es común el uso de operaciones matriciales (por parte de la GPU) para la renderización gráfica.

*) b) i) Como mecanismo de entrada/salida en este contexto considero que las interrupciones son la mejor opción. Esto debido a que si usara Polling (por ejemplo, estaría "gastando" CPU de manera innecesaria pues la interacción no es tan frecuente. Además eso implicaría, entre otras cosas, que se gaste más rápido la batería del celular. Por eso, manejar la pantalla táctil con interrupciones es una buena opción, si utilizo handlers óptimos puedo obtener respuestas rápidas ante la interacción con un usuario. DMA queda descartado principalmente porque no tendría sentido dedicar tanto hardware y CPU a algo que no maneja intercambio de información de grandes cantidades.

ii) Considero que en este contexto definitivamente la peor opción sería utilizar Polling pues sería un gasto innecesario de CPU esperando acciones que no ocurren con frecuencia. Por otro lado, dependiendo de la magnitud del disco, los archivos que se manejen y la capacidad del hardware disponible podría utilizar DMA o interrupciones. Voy a considerar que este disco virtual está dentro del contexto de una computadora "normal" (quizás con buen hardware pero de uso cotidiano/profesional) entonces utilizar DMA para el manejo del disco virtual quizás sea

*) Considero los mecanismos de E/S principales: DMA, Polling, interrupciones.

OTd:77 H: 4

menos conveniente que manejar la E/S con interrupciones. Creo que utilizar interrupciones es lo suficientemente eficiente en el contexto dado (sin más detalles, como dije) porque aunque con DMA las "escrituras" y "lecturas" al disco virtual probablemente se hagan más rápido, puede ser demasiado para lo que necesito.

iii) En este caso supongo que en el contexto científico puedo tener una computadora que quiero que haga unas cuentas de la manera más rápida posible. Con eso en mente, utilizar DMA ~~[illegible]~~ es la mejor opción pues es probable que necesite escribir y leer cantidades considerables (y tamaños) de datos de manera eficiente. Claramente depende de la magnitud de los datos, porque si se tratan de datos extremadamente grandes quizás tenga inconvenientes a nivel hardware para manejar el acceso a la memoria y los buses de datos.

Polling, de nuevo, sería la peor opción en este contexto sería la peor opción por la misma razón que los casos anteriores. Interrupciones no es una mal opción pues creo que podría cumplir en contextos donde las multiplicaciones no son muy grandes o complejos.

Papel de fibra de caña de azúcar.

OTD:77 B H.5

4) a) Con las características dadas el protocolo no puede cumplir con el requisito iv). Esto es así porque en sistemas distribuidos y concurrentes como el planteado en el enunciado, es muy difícil determinar el orden cronológico de los eventos. Si se contara con timestamps de cuándo se realizó cada oferta seguiría sin estar garantizado que el lote se va a asignar al proceso que oferte primero. Los clock de cada proceso podrían no estar sincronizados a la perfección y que el proceso que primero oferte figure al subastador como que no ofertó primero (y sí lo haya otro que haya ofertado después). Además de esto, también podría ocurrir que el 1er mensaje de oferta tarde en llegar al subastador por algún inconveniente en la red/conexión y otra oferta que se realizó después llegue primero al subastador.

b) Con lo que dice el enunciado podemos suponer que el requisito V) puede enfrentar algunos problemas. En primer lugar, vale la pena cuestionar el modo de acceso a los discos remotos: ¿Qué sucede si se pierde el mensaje del subastador? Si suponemos que es posible que se pierdan mensajes entre cliente-servidor quizás podría ocurrir que el subastador se quede esperando una respuesta del servidor (ya sea porque se perdió el mensaje del cliente o el del servidor). Si utiliza algún timeout (lo cual es probable) esto no debería ocurrir.

Por otro lado, para el correcto cumplimiento del requisito V), además de suponer que no se pierden mensajes (y que la lectura/escritura se realiza en un tiempo razonable que no caiga en timeout) debería suponer que una vez que el subastador elige al ganador, hasta no obtener respuesta (negativa o positiva) del servidor, no se asigna el mismo lote a otro proceso, para evitar condiciones de carrera. Es decir no podría ocurrir que

un proceso que oferte, compre un lote que ya se vendió. Igualmente, el enunciado no da muchos indicios de que esto pueda ocurrir. Pero me parece pertinente mencionar los aspectos y problemas que pueden surgir por la concurrencia. En definitiva, lo principal tiene que ver con la pérdida de mensajes y el manejo de este tipo de casos.

c) Un algoritmo que podría servir para realizar una implementación de este protocolo es el 2 Phase COMMIT (2PC). Si bien la versión analizada en clase consideraba que todos los procesos realizaban una operación mandada por un proceso coordinador, en este caso me interesaría que solo un proceso la haga. En particular, el que ofertó primero. Lo que busco es solucionar el problema planteado en b) (la pérdida de mensajes) mediante un protocolo que lo considere. El problema del orden cronológico[(a)] es bastante complejo ya que lo que se busca es un clocking distribuido que permita determinar si un proceso ofertó antes que otro con seguridad. En principio, este protocolo no trata el problema de la manera buscada, pero podría, en la fase de "Commit Request", modificarlo para implementar algo como lo planteado por Lamport. Que se basa en la idea de que no importa el "cuándo", sino en qué orden se realizaron las operaciones. En este caso las ofertas