Sistemas Operativos 5º semestre. Grado II

Primer Parcial. Sistema de Ficheros. 6 de Noviembre de 2017.

SOLUCIÓN

Ejercicio 1 (2 puntos)

Realice un dibujo que muestre claramente las estructuras de datos donde el Sistema Operativo almacena los siguientes campos relativos a un fichero abierto. El dibujo debe ser consecuente con el valor indicado para cada campo. Añada las explicaciones necesarias.

- **a)** Descriptor de fichero. Valor: salida estándar.
- **b)** Número de inodo de dicho fichero. Valor: 12345.
- **c)** Número de duplicados o referencias (`ndups`). Valor: 3.
- **d)** Posición sobre el fichero. Valor: 9876.
- **e)** Modo de apertura. Valor: `O_WRITE|O_APPEND`.
- **f)** Número de veces abierto (`nopens`). Valor: 2.
- **g)** Número de enlaces o de nombres (`nlinks`). Valor: 2.
- **h)** Tipo de objeto: Valor: fichero regular.
- **i)** Propietario y Grupo. Valores: 0 y 0.
- **j)** Permisos de acceso. Valor: 0751 con SETUID activo.

Solución

Estructuras de datos:

Tabla de Procesos: (única en el sistema, con un BCP por cada proceso)
BCP del proceso: (con todos los atributos de ese proceso, y entre ellos la T. fds)
Tabla de descriptores de fichero:
[1] (salida estándar): apuntando a la entrada XX de la Tabla Intermedia.

Tabla Intermedia: (única, con una entrada por cada apertura de fichero realizada)
Campos: `[[ #inodo | ndups | posición | modo_open ]]`
[XX]: `[[ 12345 | 3 | 9876 | O_WRITE OAPPEND ]]`

Nota: ndups=3 indica que hay otros dos descriptores de fichero (duplicados) más (de ese proceso o de otros) apuntando a la entrada XX de la T. Intermedia.

Tabla de inodos: (única, conteniendo o referenciando a los inodos en uso)
Campos: `[[ #inodo | nopens | <<copia_del_inodo>> ]]`
[12345]: `[[ 12345 | 2 | <<mostrado aparte>> ]]`

Nota: nopens=2 indica que hay otra entrada más de la T.Intermedia apuntado a este inodo, correspondiente a una apertura independiente del mismo archivo.

Inodo 12345: (copia única de este inodo en memoria, llevado a memoria mientras se use)
Campos: `[[ <<permisos>> | <<propietario>> | nlinks | resto... ]]`
`[[ -,rwsr-x—x | (UID=0, GID=0) | 2 | ... ]]`

Nota: Los permisos contienen el tipo de objeto y los bists SETUID y SETGID, y UID y GID especifican la identidad del propietario del objeto (root en este caso).
Nota: nlinks=2 indica que hay dos entradas de directorio (o nombres de archivo o enlaces físicos) asociados a este número de inodo.

Ejercicio 2 (3 puntos)

Considere un sistema de ficheros tipo UNIX (basado en inodos) para un disco de 2 TiB, con sectores de tamaño estándar, bloques de 2 KiB y agrupaciones de 2 bloques. Explique con detalle:

- **a)** ¿A qué unidad "apuntan" los punteros directos e indirectos de un inodo? ¿Qué ancho deben tener estos punteros/direcciones en este sistema? ¿Por qué?
- **b)** ¿El mapa de bits para gestión de espacio libre utiliza un bit por cada...? (nombre la unidad) ¿Cuánto espacio de disco (en agrupaciones) ocupará este mapa de bits? Haga los cálculos.
- **c)** ¿Cuánto espacio de disco ocupan los datos de ficheros de: 0, 1, 3 y 5 KiB respectivamente?
- **d)** El otro mapa de bits ¿cuál es? ¿para qué sirve? ¿de qué decisión/es depende su tamaño?
- **e)** Exprese con una fórmula la capacidad máxima de direccionamiento (en bytes) del inodo.
- **f)** ¿Qué accesos a disco se necesitan para leer el byte 409600000 de un archivo recién abierto?

Solución

a) Las direcciones del Sistema de Ficheros apuntan a espacio asignado, esto es, a agrupaciones. Los punteros directos apuntan a agrupaciones de datos y los indirectos a agrupaciones de indirección. Hay que calcular cuantas agrupaciones tiene este SF:

2^41(B/disco) / 2^12(B/agrup) = 2^29(agrup/disco)

Luego necesitaríamos 29 bits, que redondeamos a potencia de 2 (32 bits) para definir el ancho necesario para los campos de dirección (o punteros).

b) El mapa de bits para gestión de espacio libre utiliza un bit por cada agrupación tenga este SF. Este mapa almacena 2^29(bits), que son 2^26(B), que a 2^12(B/agrup) son 2^14(agrup).

c) El espacio asignado para los datos de los ficheros será un número entero de agrupaciones. Para ficheros de 0, 1, 3 y 5 KiB, serán 0, 1, 1 y 2 agrupaciones respectivamente.

d) El otro mapa de bits es el de inodos, y se utiliza para saber el estado de libre u ocupado de cada inodo. Su tamaño depende del número máximo de inodos que vayamos a querer tener en este SF, y esto a su vez suele calcularse como el número de archivos de tamaño medio que entrarían en este SF, o lo que es igual, tamaño del disco entre tamaño medio del archivo.

e) El número máximo de agrupaciones que puede llegar a direccionar un inodo, es función del número de punteros de cada nivel de indirección que tenga y del número de punteros por agrupación de indirección, según la siguiente fórmula: $CMD(agrup) = \#pd + \#psi*dpa + \#pdi*dpa^2 + \#pti*dpa^3$

donde los punteros por agrupación son: dpa = 2^12(B/agrup) / 2^2(B/direc) = 2^10(direc/agrup)

normalmente hablamos de 10 punteros directos y tres niveles, #pd = 10 y #psi = #pdi = #pti = 1

y finalmente habría que pasarlo a bytes, CMD(B) = CMD(agrup)*Tam.Agrp.(B)

f) Se trata de acceder directamente a un byte, un acceso directo que de ningún modo precisa el acceso secuencial a toda la información anterior. Considerando que las numeraciones empiezan desde cero, el byte 409600000 será el primero de la agrupación 100000 del fichero. Con los 10 punteros directos no llega (10 < 100000), ni con el simple indirecto (1024 < (100000-10)), pero con el doble indirecto podremos acceder (2^20 > (100000-10-1024)).

Luego, considerando que el inodo ya se encuentra en memoria, necesitaremos 3 accesos a disco. Usamos el puntero doble indirecto del inodo y accedemos a una agrupación de indirección con punteros de tipo simple indirecto. Se calcula la entrada adecuada (la 96) y accedemos a una agrupación de indirección con punteros directos. Se calcula la entrada adecuada (la 662) y accedemos a la agrupación de datos que contendrá el byte 409600000 del fichero.

Ejercicio 3 (5 puntos)

a) Implemente en C y para UNIX el mandato `cat01` (equivalente a un `cat` sin argumentos) que emite por la salida estándar todo lo que lee de su entrada estándar, usando un buffer de 1024 bytes.

b) Ponga y explique un ejemplo de uso del mandato `cat01` desde el intérprete de mandatos, que fuerce la situación en la que el proceso correspondiente a este mandato recibiría la señal SIGPIPE.

Considere ahora el bucle interno de copia de `cat01` y describa cómo será su avance según sea la naturaleza del objeto asociado a su entrada estándar, indicando:

- ¿qué valores devolverá cada llamada `read`?
- ¿cuál será el total de bytes retransmitidos?
- ¿cuánto durará (aproximadamente) la ejecución de `cat01`?

Todo ello para cada uno de los tres casos siguientes donde el objeto asociado a la entrada estándar es:

c) Un archivo de 3500 bytes.

d) Un terminal por el que un usuario introduce los 7 días de la semana (lunes, martes, etc.), uno por línea y uno cada minuto.

e) Una tubería por la que llega una línea de texto cada segundo: la primera vacía, la segunda con una letra, la tercera con dos, y así sucesivamente, hasta 1000 líneas.

Solución

a)

```
int main(void) {
     int cnt;
     char buf[1024];
     while((cnt=read(0,buf,1024))>0)
          write(1,buf,cnt);
     return 0;
}
```

b) `$  echo “Algo para leer” | cat01 | true`

El mandato `true` termina bien inmediatamente, de manera que el pipe a la salida de `cat01` se quedará sin lectores y cuando `cat01` escriba en dicho pipe recibirá SIGPIPE. Pero para que `cat01` escriba primero debe leer (véase el bucle de trabajo), es por eso que hemos conectado su entrada con otro mandato que le proporcione algo para leer.

c) Un archivo se recorrerá secuencialmente, devolviendo: 1024, 1024, 1024, 428 y finalmente 0, este último indicando fin de fichero y permitiendo salir del bucle y terminar el programa. Se habrán transmitido los 3500 bytes. La duración no está acotada, pero será muy muy breve (milisegundos).

d) De un terminal se leen líneas completas, devolviendo: 6, 7, 10, 7, 8, 7, 8 y finalmente 0, o la longitud de cada linea (fin de línea incluido) más la indicación de EOF (Ctrl-D por terminal). Total 53 bytes en 7 minutos aproximados.

e) De una tubería se lee lo que haya en cada momento. Como llega una línea de texto cada segundo, cada una de ellas se leerá por separado devolviendose el número de caracteres leídos que incluirá los caracteres de fin de línea: 1, 2, 3, … 999, 1000 y finalmente 0, (la indicación de EOF). Total de 1001*1000/2 bytes en 1000 segundos.