Document de la tercera entrega del projecte 1 de PROSO

1. Disseny:

Per tal d'implementar l'entrada i sortida de dispositius hem hagut d'implementar diverses estructures que descrivim a continuació:

- Taula de Canals (TC): Una TC per procés. Cadascuna de les posicions d'aquesta taula conté una variable per identificar si el canal en qüestió es lliure o no i un punter a una posició de la Taula de Fitxers Oberts.
- Taula de Fitxers Oberts (TFO): Una TFO per tot el sistema. Cadascuna de les estrades d'aquesta taula conté el nombre de referències a aquesta entrada, la posició seqüencial dins el dispositiu (per aquells que l'utilitzin), el mode en que s'ha obert el dispositiu i un punter al Descriptor de Dispositiu (Dispositiu Lògic).
- Descriptor de Dispositiu: Cada Descriptor de Dispositiu conté una variable per indicar que l'entrada del directori del dispositiu en qüestió es lliure o no, el nom (PATH) del dispositiu, el nombre de referències (per aquells que ho necessitin), el mode d'accés, el primer bloc de disc corresponent al dispositiu en qüestió (només utilitzat per fitxers), el tamany (només utilitzat per fitxers) i un punter a la llista d'operacions dependents del dispositiu.

El motiu de realitzar la implementació d'aquestes estructures de la manera descrita es per arribar a una solució més simple i senzilla.

Per poder implementar la crida a sistema read per al teclat, hem hagut d'implementar un buffer circular per tal de recollir l'entrada del teclat dins del buffer i posteriorment servir als processos que sol·licitin llegir del teclat mitjançant aquest buffer circular.

El buffer circular s'ha implementat utilitzant un array de caràcters de KEYBUFF_SIZE (definit com 128 per convenció entre el grup) i tres variables, una per indicar la posició inicial de les dades vàlides dins l'array, una altre per indicar la posició final de les dades vàlides dins l'array i una altre per indicar el tamany de les dades vàlides dins l'array.

Per tal de realitzar una administració del buffer circular de manera senzilla i independent (podríem canviar la implementació de manera fàcil i la resta de codi no patiria canvis) hem implementat dues operacions:

- buff_keyboard_insert(char c): Aquesta operació insereix el caràcter c dins del buffer circular, tot actualitzant les tres variables de manera correcta i consistent.
- Buff_keyboard_get_next(): Aquesta operació retorna el següent caràcter vàlid dins del buffer circular, tot actualitzant les tres variables de manera correcta i consistent.

Per simular la part de disc al nostre sistema operatiu hem inclòs, principalment, dues estructures de dades molt relacionades entre si: ZeOSFAT i HardDisk.

ZeOSFAT es un array d'enters de tamany MAX_BLOCKS (definit com 50 per convenció entre el grup) que ens indica els enllaços entre els blocs que tenim a HardDisk.

HardDisk es una matriu de Bytes. En concret la matriu es de tamany MAX_BLOCK*BLOCK_SIZE (definit com 256 per convenció entre el grup) Bytes.

Parlant ara de la gestió dels blocs, tenim també una variable free_block que ens marca l'índex dins de la FAT del primer bloc dels blocs lliures (no ha de ser exactament el bloc lliure amb l'índex més baix dins de ZeOSFAT). Aquest bloc apuntat per free_block, apunta a la vegada a un altre block lliure i així fins a

arribar a l'últim lliure que prendrà per valor EOF (indicant el final de la llista de blocs lliures). En el cas que free_block prengui per valor EOF significarà que no tenim espai lliure. Així doncs, els blocs lliures formen una llista on el cap de la llista està indicat per la variable free_block. De la mateixa forma estan enllaçats els blocs de disc utilitzats per un mateix fitxer, acabant amb un bloc amb valor EOF (indicant que es tracta de l'últim bloc del fitxer).

A més de la funció initZeOSFat, hem implementat dues funcions per la gestió dels blocs del HardDisk. Aquestes funcions són Alloc_Block i Free_Blocks:

- initZeOSFat(): Inicialitza la FAT amb tots els blocs enllaçats entre ells ja que inicialment es troben tots lliures, inicialitza el directori '/' amb les DIR_ENTRIES entrades lliures i inicialitza el HardDisk amb zeros.
- Alloc_Block(): Ens retorna un bloc lliure i actualitza la llista de blocs lliures del disc.
- Free_Blocks(): Rep un bloc com a paràmetre (l'índex d'aquest dins de la FAT) i allibera aquest bloc a més de, en el cas que tingui, els següents blocs que estiguin enllaçats amb aquest. Per actualitzar l'espai lliure, el free_block passa a apuntar al bloc que ens passen per paràmetre i l'últim de la cadena que volem alliberar passa a apuntar al anterior free_block. Així fem fàcilment i de manera consistent la gestió de l'espai lliure.

A més d'aquestes operacions, algunes crides a sistema han estat modificades per a incloure la possibilitat d'utilitzar el disc i s'han afegit noves operacions auxiliars com, per exemple, createFile, que et crea un nou fitxer (només a l'open amb la opció O_CREAT) i li reserva un espai al disc.

Així doncs en aquesta tercera entrega hem hagut de fer canvis respecte la segona entrega. Els canvis més rellevants han sigut:

- Afegir la TC a en la task_struct
- La crida a sistema exit() ha de tancar tots els canals oberts de la TC del procés en qüestió.
- La crida a sistema fork() ha de copiar la TC del procés pare al procés fill i incrementar el nombre de referències al fitxer obert de la TFO.
- La funció init_task0() ha d'obrir els tres dispositius estàndard stdin, stdout i stderr als canals 0, 1 i 2 respectivament utilitzant els dispositius KEYBOARD y DISPLAY.
- La crida a sistema write() s'ha modificat de manera que utilitzi la funció write() dependent de cada dispositiu.
- La rutina d'interrupció al teclat s'ha modificat de manera que insereixi els caràcters al buffer circular i serveixi els caràcters als processos que ho han sol·licitat, desbloquejant-los quan aquest ja hi hagin estat servits.

2. Implementació:

Com aspectes d'implementació només destacar que hem fet us de la directiva "inline" en moltes de les operacions auxiliars, codi molt simple però que s'utilitza en diversos punts del codi i de manera que per fer el codi una mica reutilitzable l'hem encapsulat, utilitzant la directiva "inline" per no deixar de perdre eficiència pel fet d'haver de guardar el context de cada crida a la pila.

Els únics problemes amb els que ens hem trobat han sigut errates de codi, cap problema important ni cap error conceptual de manera que la realització d'aquesta tercera entrega ha sigut relaxada i senzilla per aquest mateix fet de no haver-nos trobat amb cap error important.

3. Descripció de les proves realitzades:

En aquesta tercera entrega les proves realitzades bàsicament han estat amb les llibreries de test proporcionades. Pel fet que les úniques parts que hem considerat més complexes de la entrega han sigut el read de teclat i la gestió de disc mitjançant el read i write de fitxers, a part de les llibreries de test proporcionades hem realitzat les següents proves:

- Pel read de teclat: Fer reads de 0 bytes, fer reads d'un sol byte, fer reads d'alguns bytes sense omplir el buffer circular i fer reads de molts bytes omplint el buffer circular.
- Per al read i write de fitxers: Fer reads i write de 0 bytes, fer reads i writes d'un byte, fer reads i writes d'alguns bytes de fitxers amb un sol bloc, fer reads i writes de fitxers amb diversos blocs els quals son consecutius a disc i fer reads i writes de fitxers amb diversos blocs els quals no son consecutius a disc.