UOIT
CHALLENGE INNOVATE CONNECT
UNIVERSITY OF ONTARIO
INSTITUTE OF TECHNOLOGY

ENGR 3950U / CSCI 3020U: Operating Systems

Simulated Unix File System Project: Data Structure Design

Instructor

Dr. Kamran Sartipi

Faculty of Engineering and Applied Science
University of Ontario Institute of Technology
Canada

File system specifications

- File System structure: 512 blocks , 128 bytes per block
- Max number of the files (directory and regular): 64 files
- Max size of the files: 8 direct address blocks (1024 bytes)
- Max number of open files: 64 files
- Max number of directory entries: 63
- Max number of directory levels: 64
- Max number of the characters of the pathname component: 5 char and '\0'
- Block boundary crossing for read and write to the file is allowed.

Major Data Structure

1. Short int disk_bitmap[512]={1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}.

Keep track of the FREE and BUSY blocks of the "simulated file system" each element of this array represents "free" or "busy" of the corresponding block of the file system:

free=0, busy=1

2. short int super_blk_buf[128]

Used to hold the data of the SUPER BLOCKs (blocks 0 and 1). Compress the information of the "disk_bitmap[512]", each 4 contiguous elements of the "disk_bitmap[512]" is encoded to one integer number (between 0 and 15).

Major Data Structure (cntd)

3. short int file_blockno[8][64]

Two dimension array to hold the information of the "I_node table".

Disk blocks #2 to #9 (8 blocks) are allocated to "I_node table".

Each column of this array is allocated to one file (regular or directory) and the index number of that column is the "i_number" of the file.

4. short int file_pointer[64]

Maintain the information of the "file pointer table" (disk block #10).

Show the number of elements in the file.

Major Data Structure (cntd)

5. short int file_refcount[64]

This array is used to maintain the information of the "file reference count" in the memory. Like above each file has an entry in this array.

6. short int fd_table[64]

This array is the memory buffer for "file descriptor table" .

7. char buffer_cash[1024]

Used as the buffer for data transfer between file blocks and user memory addresses.

Major Data Structure (cntd)

8. int readdir_pointer[64]

Used in "sfs_readdir" as a counter to determine how many "directory entries" are so far read from the intended directory.

9. char io_buffer[512]

Defined in the main() program and in file system services .

10. char pathname_parse[64][7]

Two dimension array used to hold the parsed "pathname component".

Map of Simulated Disk

- SUPER BLOCK: Blocks #0 and #1 are allocated to the "super blocks“ which contain the information regarding the "free" or "busy" blocks of the file system.
- I_NODE TABLE: Blocks #2 to #9, contains information regarding the located blocks to the created files in the system.
- FILE POINTER TABLE: Block #10, contains the size of each created file.
- FILE DATA BLOCKS: Blocks #11 to #511, contain the data of the regular files and the directory entries of the directory files.

Structure of the Program

Two source files are responsible for providing the required lower level functions for the "file system interfaces" **:**

1. super_block.c: contains the functions related to super block handling

2. I_node.c: contains the rest of the low level functions.

Functions in "super_block.c" file

int put_super_blk():

1. Encodes each four locations (0 or 1) of disk_bitmap[512] into an integer number and puts in "super_blk_buf[128].
2. Puts the bytes of buffer (short int super_blk_buf[128]) in the super-block (blocks 0 and 1 of disk).

int get_super_blk() :

1. Gets super-block (blocks 0 and 1 of disk) as "short integer bytes" and puts them in the "super_blk_buf[128].
2. Decodes each integer(<15 & >0) in super_blk_buf[128] into 4 bits and puts them in disk_bitmap[512].

Functions in "super_block.c" file (cntd)

int get_empty_blk(int *free_blk_no)

Searches the super-block and if it has an empty block, marks it as a busy block and returns its block number to the calling function.

int release_block(int release_blk_no)

Releases the block as a free block to the system, this block was already allocated to a file; it also updates the disk_bitmap[512].

Functions in "I_node.c" file

int put_inode_table():

Puts the integer numbers from memory buffer "file_blockno[8][64]"to the I_NODE_TABLE blocks (blocks #2 to #9) of the disk.

int get_file_pointer(int i_number, int * file_ptr):

*G*ets the file pointer of the file whose i_number is sent as parameter,and returns it back in the variable "file_ptr".

Functions in "I_node.c" file (cntd)

int alloc_block_tofile(int i_number, int *allocated_blkno):

Allocates 1 block to a file whose i_number stored in "int i_number"and the number of the allocated block is returned in variable:"allocated_blk_no".

main functions:

1. Load the i_node table from disk to memory
2. Search to the list of direct blocks of the file to see how many blocks has already been allocated to the file.
3. Use "get_empty_blk()" search in the super block to find an empty block.
4. Allocate the empty block to file and save the i_node table in disk.
5. Return the number of the allocated block in variable "allocated_blkno".

Functions in "l_node.c" file (cntd)

int parse_pathname(char *path, int* no_components)

Parses the pathname in array "data_buffer_1[1024]".

The parsed pathname will be constructed in "pathname_parse[64][7]".

- Valid pathname component has 5 chars and delimiter '\0': "xxxxx \0".
- path[i]: 0 < i < 1023
- pathname_parse[j][k]: 0 < j < 63 0 < k < 5
- j counts individual pathname components.
- k counts the number of character in each pathname component.

Functions in "I_node.c" file (cntd)

int parse_dir_entry(int component_no, char *component, int *i_number):

Parses the directory entry and returns "pathname component" and "i_number"using "component_no“. It accesses the entry in "pathname_parse[64][]“ and parses the entry and RETURNS "component name" and "i_number".

return an integer with 2 digits: XX

GOOD LUCK!!