McGill University

COMP-310 – Operating Systems – ECSE-427

Assignment #4

Due: December 4, 2008 at 23:55 on Web CT

(Secret late date Dec. 11, 2008 without penalty)

About File Systems

QUESTION 1: Big-Oh Run-Time Calculations

The OS manages files and the directory structure. Two common disk access methods exist: the disk block method and the classical semi-contiguous file using pointers. In either case the File Allocation Tables were basically similar. The file is written contiguously until there is no more space, then a pointer is used to refer to the next free space where data continues to be written (or read for that matter). The difference between these two methods is that the block method writes and is formatted by blocks and the classical method is organized by byte.

For this question discuss the:

- Best-case, and
- Worst-case

run-times for both the blocked disk access method and semi-contiguous access method, both using pointers.

In other words assume the worst-case situation for how a file might exist on disk in the semi-contiguous and block files methods. Then compute using Big-Oh notation the performance of loading such a file entirely into memory. Do this again for the best-case situation.

Finally discuss and compare your findings. Which is better, in what case and why?

QUESTION 2: Programming Problem

This question expands the program you created from assignment number 3. In this installment of the OS building exercise you will be asked to do one thing:

- Create a One-level contiguous storage directory based file system

This assignment must continue from either your assignment #3 code or the one provided by the teacher. Please maintain the functionality stated for the program from assignment #3 unless specifically changed in this assignment.

You are permitted to implement these features as you see fit. The only flexibility you do not have is when I specifically ask you to do something in a particular way or I outline some features that must exist.

Disk File Management

When your OS simulator first starts up it will simulate a disk format operation before it shows the user the command-line prompt. This format operation will occur as follows:

1. Create two new/empty files on disk. One file is called OSFAT.txt and the other is called OSFILES.txt. Both files will be text files. Create these two files when your simulated OS begins, before the prompt is displayed.

2. The OSFAT.txt file will be your simulated file allocation table. Each record in this FAT will have the following fields: a unique file name, and a pointer to the start address of the file on disk. The following structure details this:

```
struct OSFAT
{
   char filename[20]; // file names are at most 19 characters with /0
   int address; // an offset value from the start of the simulated disk
}
```

3. The OSFILES.txt file will be your simulated disk. You will "format" it by inserting 1000 zeros into this file. This will be the maximum physical space of your disk, 1000 bytes. In other words this file must always be 1000 bytes long, never longer, never shorter. So you will insure this by writing 1000 ASCII zeros into it. You will then consider this to be an "empty" disk drive of 1000 bytes. As you create files you will write them into this simulated disk drive space. Note that your simulated disk drive space can become full – when you've changed all 1000 zeros into ASCII data by your WRITE command. You can delete a file using your DELETE command (more about this later). You can overwrite an existing file by closing and then opening and then writing to it.

New script commands:

- OPEN FILENAME AT INDEX
  The OPEN command will call an internal OS library function (that you create) called GETRESOURCE INDEX to open at file at the requested INDEX number. Your OS will implement a global file pointer table, accessible and managed only by the OS. This table will allow a total of 5 valid file pointers for the entire system. This new table will be called "int FileHandles[5][2]" and will store the offset pointer to the file in your simulated disk and the process ID number of the process assigned to the INDEX column. The offset will represent the location in the file you are currently reading from or writing to. If the index number is in use the process is put to sleep by removing it's PCB from the Ready queue and adding it to a Waiting queue. Once the INDEX is acquired the offset cell is assigned the offset number of the beginning of the simulated file from

the OSFAT.txt file. If the filename does not exist, the file is created and a new entry is added into the OSFAT.txt file and an offset pointer is assigned to the next free space. If there is no free space left, then this instruction is ignored and the INDEX is not assigned to this PCB. Note that this simulation is with only contiguous files, so space can run out. In this case the process is terminated by the OS and a run-time error message is displayed: "Process NAME terminate. No more disk space.". PCBs on the Waiting queue are scanned during the "overhead" portion of the kernel execution loop to see if the resource they want is available. If so, the PCB is added to the tail of the Ready queue to await its turn.

- WRITE INDEX, STRING
  Takes STRING and writes it to the OSFILES.txt file at the offset indicated by the cell in INDEX. Each character is written and INDEX offset cell is updated. Before this is done though, WRITE checks if the requesting PCB owns the INDEX and if the INDEX was opened. If not, then nothing is written and the instruction is ignored. Note: the write command does not write ASCII digits. If any exist in your STRING they are skipped and ignored. Note: we will use ASCII digits in this simulation to represent file control information: zero means empty and one means EOF. If while writing you run out of space then WRITE automatically calls CLOSE INDEX to terminate the file and insert the EOF marker. Note: if write was used when opening an existing file make sure to overwrite the existing file.

- CLOSE INDEX
  This does two operations. First, an EOF character – number 1, terminates the file. Then internal OS library function that you created called FREERESOURCE INDEX is called to release the resource for someone else.

- DELETE FILENAME
  If the file name exists, then the entry is removed from the OSFAT.txt file and the simulated file in the OSFILES.txt file is overwritten by zeros.

Notes:

- The file's name and start address will be located in the OSFAT file. If the file name already exists in the OSFAT file the new file will overwrite it.
- The OSFAT file address will be an integer number representing the offset in bytes from the start of the OSFILES file to the location of the first byte of your file written in your simulated disk drive.
- The file will be written byte-by-byte contiguously. The last byte of the file will be the EOF marker designated as the ASCII digit 1.
- If you run out of contiguous space for the file the OS will automatically put an EOF marker at the end of the file and the process will not be permitted to write any more data to the file. Do not use pointers to overcome this problem.

WHAT TO HAND IN

On Web CT in electronic form:

- A single WORD, or PDF document answering the non-programming questions.
- One source files.
- Three ASCII programs to run on your OS (small size, medium size and large size). Note that the TA will probably also run their own programs.
- Also include a text file describing what operating system your program was developed under. It would be good if you tested your program on our machines on the third floor of Trottier.
- Please provide comments in your code so that it will be easier to grade.

HOW IT WILL BE GRADED

This assignments is worth 20 points, distributed as follows:

- Question 1: 5 points (one point for each sub question)
- Question 2: Total of 15 points

Resource data structure and functions	5 points
File access commands and scripting	5 points
Disk free space management	5 points