Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur
End-semester Examination, Spring 2017
CS30002: Operating Systems

Full Marks: 100 **Time: 3 Hours**

Answer ALL questions. All parts of Q. 3 and Q. 4 must be answered consecutively.

1. **(a)** Define a wait-for graph in the context of deadlock. What property should exist in the wait-for graph for a deadlock to be present? (2 + 1)

   (b) In a demand paged OS, is it necessary that the code for the scheduler always need to be in main memory? If yes, justify (in 1-2 sentences) why and state how it can be ensured that it is always in main memory. If no, justify (in 1-2 sentences) why not and what action is taken if the scheduler code is not found. (3)

   (c) Consider an OS that has 128 different priority levels for processes from 1 to 128, with lower level indicating higher priority. A process with some priority can be scheduled only if there is no higher priority process present in the ready queue. Scheduling between processes at the same level is FCFS. The priorities are set statically during creation of the process and not changed after that. How will you organize the ready queue to efficiently choose the next process to schedule? Justify your design briefly. (3)

   (d) State one advantage and one disadvantage of a FAT-based filesystem over an indexed-allocation based filesystem. (2)

   (e) What is the difference between the SCAN and C-SCAN disk scheduling algorithm? Explain with an example. (3)

   (f) Is it possible to design an OS in which bytes of a file can be read or written by a user without opening the file first? Justify your answer briefly. (3)

2. **(a)** Consider the page reference string `0 1 3 6 2 4 5 2 5 0 3 1 2 5 4 1 0`. The number of page frames is 4. What would be the number of page faults generated for each of the following algorithms (i) FIFO, (ii), LRU, and (iii) second chance algorithm with one reference bit. Show the contents of the page frames at each page reference for each of the algorithms, identifying the page faults. (9)

   (b) Consider a demand paged system with a 64-bit address space, 4 KB page size, 512 GB RAM, and an additional 8 bits per page table entry (including valid, dirty and other bits). Show the total page table sizes for (i1) a 1-level page table, (ii) a 2-level page table, and (iii) a 3-level page table. Show all calculations, showing how do you go from 1-level to 2-level and 2-level to 3-level. Assume that the memory is aligned at 4-byte boundaries. (9)

3. Consider a file system that uses the indexed-allocation technique, with block size of 1 KB (file block size and disk block size are the same). The disk partition holding the filesystem has size 200 GB. The filesystem keeps one *i-node* for each file and one *d-node* for each directory. Each i-node has some file attributes, 6 direct blocks, 3 single-indirect blocks, 2 double-indirect blocks, and 1 triple-indirect block. Each d-node has some directory attributes and a linear list of

files/subdirectories under it. The maximum file/directory name size is 15 characters. The maximum number of files and subdirectories (combined) under a single directory is 50. The maximum number of directories in the system can be 500. However, the maximum number of files in the system can be 20000. The filesystem supports read/write/execute permissions for each file and directory, and hard links for both files and directories.

(i) What is the maximum size of a file that can be supported? (3)

(ii) Write a C-like typedef INODE for an i-node, with a 1-line description within /*..*/ before each field stating what is stored in the field and its use. (4)

(iii) Write a C-like typedef DNODE for a d-node, with description of fields as above. (3)

(iv)Show in a diagram how the filesystem components are organized in the disk partition (showing the exact sizes and order of the components). Assume that bitmap method is used to manage free space for i-nodes, d-nodes, and data blocks. (4)

(v) What will be cached in main memory when the filesystem is initialized? (3)

(vi)List the steps that that occur when the call *write(fd, buff, size)* is made, which writes *size* bytes starting from memory address *buff* to the end of an already open file with file descriptor *fd*. Assume that the system supports both a per-process and a system-wide open file table. (8)

The file and directory attributes you choose must support the above operations efficiently. Align all fields in the data structures defined to byte boundaries. You can assume that routines to read/write blocks from/to disks already exist that you can call.

4. Consider an OS using a demand-paged (1-level paging) memory management scheme with a 32-bit virtual address space, 8 KB page size, and 1 GB RAM. Dynamic allocation is supported by *malloc*() and *free*() calls with usual meaning, but for simplicity, assume that all malloc requests are always for exact multiples of pages. The system also supports shared pages (do not worry about how they are created or deleted). The page replacement policy used is a simple policy with 1 reference bit that is set to 1 on access and reset periodically to 0 by a clock interrupt. Any page frame with reference bit set to 0 can be chosen for replacement. If there is no such page (all reference bits 1), any page frame can be chosen for replacement.

(i) Write C-like typedefs for the page table, and any other memory management data structures that you will need to support the above operations. For each field, add a comment in /* ...*/ format just before it stating what is stored in the field and its use. Align all fields in the data structures defined to byte boundaries. (6)

(ii) Write the psuedocode (referring to the data structures) for implementing *malloc*(). (4)

(iii)Write the psuedocode (referring to the data structures) for implementing *free*(). (4)

(iv)List step by step (referring to the data structures) how a page is brought into memory on a page fault when no free page frame is found. You can assume that routines for disk read/writes of pages already exist that you can call. (6)

5. **(a)** Consider a set S of N processes ($N < 100$), with process ids 0 to N-1. A function *void Barrier(int pid)* is defined over the set S such that if any one process t in S calls *Barrier(t)*, it is blocked until all processes in S have called *Barrier*() with their process ids. Write the pseudocode for the *Barrier*() function using only semaphores, you cannot use any shared variables; however, you can use any number of semaphores. First list the semaphores that you will use and their initial values (do not worry about who creates and initializes them). Then show the code for *Barrier*() that uses the semaphores declared by making *P/wait* and *V/signal* calls on them. (10)

(b) Describe what happens when a system call is made, including how the kernel code for the specific system call is invoked with the correct parameters passed and any other relevant details. (10)