COMPE571 – Embedded Operating Systems – Fall 2025 – Homework 3

Skye Russ

**File systems**

1 – (10 pts) What is the maximum file size supported by a file system with 16 direct blocks, a single, a double, and a triple indirection blocks? The block size is 1KB. Disk block numbers can be stored in 4 bytes.

16 Blocks * 1KB = 16KB Direct Blocks
Single Indirect = 1KB / 4 byte addresses = $2^8$ Entries = 256KB Additional
Double indirect = $2^8$ * $2^8$ * 1KB Blocks = $2^{16}$KB additional
Triple indirect = $2^8$ * $2^8$ * $2^8$ * 1KB Blocks = $2^{24}$KB additional
TOTAL: 16 + $2^8$ + $2^{16}$ + $2^{24}$ = 16,843,024KB

2 – (10 pts) Answer the following questions in 2-3 sentences.

a) Give an example of a scenario that might benefit from a file system supporting an append-only access write.
   If a file system is used for logging of information and wants no information to be overwritten, append-only access write would be appropriate.
b) Give a scenario where choosing a large file system block size might be a benefit; give an example where it might be a disadvantage.
   If the file system uses very few large files it would be good to have a large block size. If a file system has many small files, the block size would result in a lot of wasted space.
c) What file allocation strategy is most appropriate for random access files? Give an example implementation that uses the allocation strategy you answered.
   Indexed allocation is best for random access. UNIX uses this strategy.
d) Why are the direct blocks stored in the i-node itself?
   Direct blocks that need to be accessed frequently or are convenient for smaller files to not need multiple I-node reads to access the file.
e) Given that the maximum file size of combination of direct, single indirection, double indirection, and triple indirection in an i-node-based file system is approximately the same as a file system solely using triple indirection, why not simply use only triple indirection to locate all file blocks?
   Using only triple indirection would make the number of i-nodes required to be read consistent but would remove the speedup that is achieved from direct blocks,

single and double indirects. Essentially, it would be slower to access most of the file.

3 – (10 pts) Given a block size of 16KB and a disk that holds 100GB of data (using 32 bit addresses), how big can a file be if you have single index block in an indexed file management scheme?

32 bit addresses = 4 byte addresses
1 index block = 16KB / 4byte = 4096 Blocks * 16KB block size = 65536KB file size

4 – (10 pts) A disk request queue has requests for blocks on the following cylinders (ordered by time of arrival):

18, 50, 45, 69, 9, 49

The disk has 100 cylinders numbered trough 0 to 99. The disk head is currently at cylinder 46 and is moving towards cylinder 99. Calculate the total seek distance for each of the following disk seek algorithms: FCFS, SSTF and SCAN.
(Note: For SCAN, you should assume that the disk head should go until the last cylinder in one direction and then reverse from the last cylinder.)

FCFS: START 46 ->18->50->45->69->9->49
Distance = (46-18) + (50-18) + (50-45) + (69-45) + (69-9) + (49-9) = 189 Seeks

SSTF: START 46 ->45->49->50->69->18->9
Distance = (46-45) + (49-45) + (50-49) + (69-50) + (69-18) + (18-9) = 85 Seeks

SCAN: START 46 ->49->50->69->99->45->18->9
Distance: (49-46) + (50-49) + (69-50) + (99-69) + (99-45) + (45-18) + (18-9) = 143 Seeks

5 – (5 pts) Write down the number of disk accesses required to fetch the 5th block of a file with:

**Contiguous allocation:**
Number: 1 Why: Any index can be accessed as an offset of the start of the file in a contiguous allocation file system.

**Linked-list allocation:**

Number: 5 Why: The disk must be accessed for each block of the file (because each block contains the location of the next block). (This assumes the first block of the file is 1 not zero, for a zero-based system the number would be 6).

**Indexed allocation:**
Number: 2 Why: The disk must first access the index block which will have a value for the 5th block of a file.

**Memory**

6 – (10 pts) Consider a paging system with a single-level page table. And the page table stored in main memory.

a) If a single main memory reference takes 200 nanoseconds, how long does a paged memory reference take?
   A paged memory reference takes 400 nanoseconds (200 for the page table, 200 for the memory access).
b) If we add a set of special registers (with access time 10 nanoseconds), and 50 percent of all page-table references are found in these registers, what is the effective memory reference time?
   Hit: 210ns
   Miss: 410ns
   Chance: 50%
   0.5*210 + 0.5*410 = 310 Nanoseconds effective memory reference time

7 – (10 pts) Virtual memory address translation:

a) Consider a machine with a physical memory of 8 GB, a page size of 4 KB, and a page table entry size of 4 bytes. How many levels of page tables would be required to map a 52-bit virtual address space if every page table fits into a single page?
   Offset: 12 bits, with 40 bits left so you have 4 pages available. 4 levels of page tables are available.
b) Without a cache or TLB, how many memory operations are required to read or write a page in physical memory?
   Because you have to go through 4 levels of page tables you need 5 memory accesses.

c) How much physical memory is needed for a process with three pages of virtual memory?
4 virtual pages and 3 physical pages so 28MB of physical memory.

8 – (10 pts) Answer the following questions in 2-3 sentences.

a) Describe the difference between external and internal fragmentation.
External fragmentation is empty memory space between tasks (frequently seen in segmented memory). This is a result of space being left between tasks to allow for variable task sizes / task boundaries. Internal fragmentation is normally seen in paging systems and is empty / unused memory space within an existing task's memory bounds (ex. A 24k task in a 10k page system has a 6k empty space in one of the pages that is internal fragmentation).

b) Give some advantages of a system with page-based virtual memory compared to a simple system with base-limit registers that implements swapping.
A page-based virtual memory system has better security than base-limit registers because the access of memory locations is managed by the OS rather than the user. Also, a page-based virtual memory system can use more of the memory with less loss due to the bounds between processes in a base-limit register. Multiple levels of pages allows for tasks to expand much easier than a base-limit register system.

c) What is a page fault?
When a program attempts to access a location in a virtual page that has not been loaded into the main memory and might be in the disk.

d) Write down pros and cons (2 each) for increasing the page size.
Pros:
- Less accesses to get a larger section (multiple level page tables require more memory accesses whereas a larger page table can directly reference more memory locations)
- More accessible frequently used data sections (more memory locations that can be accessed directly).
Cons:
- More overhead space for each task (less actually usable memory space)
- Larger page initialization phase (smaller page tables can be initialized faster)

e) Describe two virtual memory page fetch policies that we saw in class.
One policy is to start the system off with no virtual pages loaded and then bring the pages on demand and use FIFO to replace the pages. Alternatively, the system can try to preload pages by organizing them and predicting which page is going to be required next.

9 – (5 pts) In class, we saw that in a C program memory map is divided between several parts. Given the following code piece, write down where each variable is stored in the memory map. You should consider slide set 16 - slide 19 to answer this question.

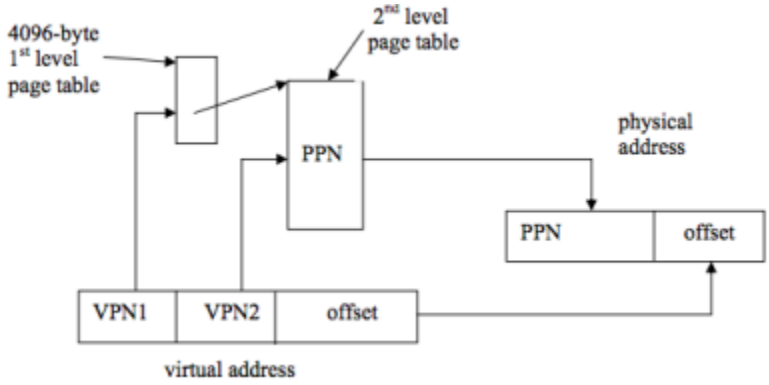
```
#include <stdio.h>
#include <stdlib.h>

int var1 = 100;
const int var2 = 0;
int* var4;

int foo(){
      int var3 = 500;
      return var3;
}
void main(){
      var1 += foo() + var2;
      var4 = (int*)malloc(50*sizeof(int));
      printf("%d\n", var1);
}
```

| **Variable** | **Memory Location** |
|---|---|
| Var1 | Initialized data segment |
| Var2 | Read Only Data Section |
| Var3 | Stack frame of foo() (heap) |
| Var4 | The pointer is in the BSS and the memory is in the Heap |

10 – (10 pts) Multi-level virtual page tables: To avoid excessively large page tables, many virtual memory systems have multi-level page tables. For this problem, we would like to figure out the details of a 2-level paging system with the following characteristics:

- Both virtual and real (physical) memory addresses are 32 bits each
- Pages are 4KB each
- Page table entries are 4 bytes each, containing the physical page frame number, plus additional control bits including the valid bit.
- Each page table fits in a single page (both the first-level page table and each second-level page tables)

The page table structure and address translation can be diagramed like this (remember the lecture slides).




Fill in the blanks with the correct numbers to match the given constraints (with calculations):

- Number of entries in the first level page table: $2^{10}$ (1K)
  Why: The first level page table contains entries for all second level page tables (and each page table is addressable by a 4byte address).
- Number of bits in the VPN1 part of the virtual address: 10 bits
  Why: To address each second level page table we need 10 bits.
- Number of bits in the offset of field of each address: 12
  Why: Pages are 4KB which requires 12 bits to address
- Number of bits in the VPN2 part of the virtual address: 10 bits
  Why: $2^{10}$ entries in the Virtual page table 2
- Number of total page table entries in 2nd level page tables: $2^{10} * 2^{10}$ ($2^{20}$K)
  Why: Page tables have 4 bytes per entry and page tables fit within 4KB leaving 1K entries. Two levels squares the size.

11 – (10 pts) Real time operating systems

Analyze what task/thread scheduling algorithms are used in the systems listed below. Explain why particular schedulers were chosen and what applications they work best for.

a) Linux kernel (2.6.35)
    a. "Completely Fair Scheduler" – The goal is to provide equal CPU access to all processes. This is not ideal for real-time or high-performance applications. However, this is great for high multitasking systems because all tasks get generally the same CPU time.
    b. Some kernels also use "Nice Values" which allow for certain tasks to be preempted more frequently than other tasks.
b) RTLinux
    a. RTLinux uses a preemptive scheduler that is designed to have as much code in the kernel's control as possible. Interrupt handlers are threaded so they can be preempted as well. This is ideal for real time applications because nearly every process can be preempted in favor of a real time application.
c) VxWorks
    a. Priority based preemptive scheduling with round robin for each priority with the additional lock to force a single task to be top priority. This is best for a real time system that needs multiple real time tasks to be the same priority and resources given equally between them.
d) TinyOS
    a. TinyOS uses a First In First Out scheduler because it expects few tasks and wants the scheduling overhead to be as minimal as possible. This optimizes CPU sleep time if the task list is low. This would not be good for a system with a lot of periodic tasks, but rather for a system that has few tasks and small capabilities.
e) Symbian OS
    a. Symbian OS is optimized for single core devices (mostly cellular devices). Because of this, they use an Active Scheduler that has all active processes run on a non-pre-emptive single thread. The order of execution is based on object priority.
f) Android OS
    a. Android OS uses Nice values like the Linux kernel however it also enforces groups for foreground and background processes. The kernel automatically assigns background processes a lower priority and ensures the current foreground task has priority. This is essentially the Linux "Completely Fair Scheduler" with an additional layer of priority based scheduling. This is best

for prioritizing what the user is looking at while still allowing the background processes to complete (therefore being good for mobile devices).

Note: For some of these you will need to do some research.