

CS423UG Homework 4 Solution

November 7, 2005

1. (4.5)

- first fit
(a) 20KB (b) 10KB (c) 18KB
- best fit
(a) 12KB (b) 10KB (c) 9KB
- worst fit
(a) 20KB (b) 18KB (c) 15KB
- next fit
(a) 20KB (b) 18KB (c) 9KB

2. (4.15)

- one-level
Each 4KB page has 2^{12} bytes. Therefore, the computer has $\frac{2^{32}}{2^{12}} = 2^{20}$ virtual pages. In a one-level page table, all virtual pages need an entry. Therefore, the number of page table entries needed is 2^{20} .
- two-level
The first-level table has 2^{10} entries, each of which points to a second-level table. Each second-level table has 2^{10} entries, each pointing to a 4KB page. From the assumption of this question, the program text and data pages are pointed by the same second-level table, while the stack pages need another table.
Therefore, we need one first-level page, and two second-level pages, which gives us the total number of page table entries $3 \times 2^{10} = 3072$.

3. (4.16)

- Page reference string
Load word at location 6144 into register 0
→ Inst. Addr: 1020, Data Addr: 6144
Push register 0 onto the stack
→ Inst. Addr: 1024, Data Addr: 8188
Call a procedure at 5120, stacking the return address
→ Inst. Addr: 1028, Data Addr: 8184 (to store return addr.)
Subtract the immediate constant 16 from the stack pointer
→ Inst. Addr: 5120, Data Addr: nil.
Compare the result of previous operation to the immediate constant 4
→ Inst. Addr: 5124, Data Addr: nil.
Jump if equal to 5152.
→ Inst. Addr: 5128, Data Addr: nil.
So the sequence of addresses generated are: 1020, 6144, 1024, 8188, 1028, 8184, 5120, 5124, 5128.
Given that the page size is 512 bytes, the corresponding page reference string is 1, 12, 2, 15, 2, 15, 10, 10, 10.

- Number of page faults under LRU with 3 frames
Total 5 page faults (1st, 2nd, 3rd, 4th, 7th)

4. (4.31)

- 4096-byte page
65536-byte address space is composed of 16 pages. On the other hand, program text is 8 pages, data is 5 pages, and stack requires 4 pages. Since it requires total 17 pages, it does not fit to physical memory.
- 512-byte page
65536-byte address space is composed of 128 pages. On the other hand, program text is 64 pages, data is 33 pages, and stack requires 31 pages. Since it requires total 128 pages, it fits to physical memory.

5. (5.11)

- floppy disk
Average time to transfer 1 sector is 77 (average seek time) + 100 (half rotation time) + 22 (transfer time) = 199ms. Since 1 sector has 512 bytes, this gives us 20.58kbps as follows, which is slower than 56kbps modem.

$$\frac{512 \cdot 8}{199 \cdot 10^{-3}} = 20.58$$

- hard disk
Average time to transfer 1 sector is 6.9 (average seek time) + 4.165 (half rotation time) + 0.017 (transfer time) = 11.08ms. Since 1 sector has 512 bytes, this gives us 369.61kbps as follows, which is faster than 56kbps modem, yet slower than 100Mbps fast ethernet.

$$\frac{512 \cdot 8}{11.08 \cdot 10^{-3}} = 369.61$$

6. (5.12)

A packet is copied for 4 times during this process. There are also two interrupt. Therefore, along with the transmission delay, it takes 6.9152ms to pump one 1024-byte packet, as follows.

$$\frac{4 \cdot 1024 \cdot 0.001 + 2 \cdot 1 + 1024 \cdot 8}{10 \cdot 10^6 \cdot 10^{-3}} = 6.9152$$

Therefore, the maximum data rate is $\frac{1024 \cdot 8}{6.9152 \cdot 10^{-3}} \approx 1.185\text{Mbps}$.

7. (5.24)

- FCFS
The arm accesses in the order of 10, 22, 20, 2, 40, 6, 38, which gives $(10 + 12 + 2 + 18 + 38 + 34 + 32) \cdot 6 = 876\text{ms}$
- Closest cylinder first
The arm accesses in the order of 20, 22, 10, 6, 2, 38, 40, which gives $(0 + 2 + 12 + 4 + 4 + 36 + 2) \cdot 6 = 360\text{ms}$
- Elevator algorithm (initially moving forward)
The arm accesses in the order of 20, 22, 38, 40, 10, 6, 2, which gives $(0 + 2 + 16 + 2 + 30 + 4 + 4) \cdot 6 = 348\text{ms}$

8. We can use soft timer for polling I/O in the same idea for the clock. Whenever the system enters kernel mode, the soft-timer facility checks for any pending I/O events before it exits, and invokes the associated handlers when appropriate.

The advantage of this scheme is same as the original one. It reduces number of interrupts because it avoids interrupts when kernel is running. Thus, this scheme can avoid the overhead of interrupts. However, this scheme can be disadvantageous especially for high-frequency devices, as I/O requests now cannot be processed when the system does not enter the kernel mode. This leads some delay for such requests. Also, the polling time increases linearly as the number of devices increases.