# HOMEWORK 3
# SOLUTIONS

CSE 120 OPERATING SYSTEMS

FALL 2006

NB: Questions are in boldface and the solutions follow them.

**1.a) Assume a user process P1 of 5 MB size. A standard hard disk is used as a backing store. The given transfer rate of the hard disk is 20 MB per second. Assuming that the file is stored contiguously, and assuming an average seek latency of 5 milliseconds, what is the swap-in time for the process P1?**
**1.b) Now, consider a round-robin CPU-scheduling algorithm for long-term scheduling. What constraint would you impose on the time quantum for this RR algorithm, having computed the total swap time in (1.a)?**
[3 points]

Ans:

1.a)
The actual transfer of the 5 MB process to or from main memory takes:
5000 KB / 20000 KB per second = 1/4 second = 250 milliseconds
Assuming that the file is stored contiguously, and assuming an average seek latency of 5 milliseconds, the swap-in time is 255 milliseconds.
1.b)
Since we must both swap out and swap in, the total swap time is about 510 milliseconds. For efficient CPU utilization, we want the execution time for each process to be long relative to the swap time. Thus, in a RR CPU-scheduling algorithm, the time quantum should be substantially larger than 0.510 seconds.

**2) Why is it a good strategy to ensure that a process is completely idle (does not have any pending I/O – in particular) before swapping it out?**
[2 points]

Ans:

If we want to swap a process, we must be sure that it is completely idle. Of particular concern is any pending I/O. A process may be waiting for an I/O operation when we want to swap that process to free up memory. However, if the I/O is asynchronously accessing the user memory for I/O buffers, then the process cannot be swapped. Assume that the I/O operation is queued because the device is busy. If we were to swap out process P1 and swap in process P2, the I/O operation might then attempt to use memory that now belongs to process P2.

**3.a) One possible solution to the problem of external fragmentation is compaction – where the goal is to shuffle the memory contents so as to place all free memory together in one large block. However, compaction is not always possible. Mention one situation where compaction would not be possible.**
**3.b) What could be another possible solution to the problem of external fragmentation?**
**[1+1 points]**

**Ans:**

3.a) Compaction cannot be done if relocation is static and is done at assembly or load time.
3.b) To permit the logical address space of the processes to be noncontiguous, thus allowing a process to be allocated physical memory whenever the latter is available. (Paging and Segmentation)

**4) Some TLBs store address-space identifiers (ASIDs) or TLB tags in each TLB entry. What could be one advantage of using TLB tags with TLB entries?**
**[2 points]**

**Ans:**

Provide address-space protection:
An ASID uniquely identifies each process and s used to provide address-space protection for that process. When the TLB attempts to resolve virtual page numbers, it ensures that the ASID for the currently running process matches the ASID associated with the virtual page. If the ASIDs do not match, the attempt is treated as a TLB miss.
Allows the TLB to contain entries for several processes simultaneously:
If the TLB does not support separate ASIDs, then every time a new page table is selected (e.g. with each context switch), the TLB must be flushed (or erased) to ensure that the next executing process does not use the wrong translation information.

**5) The percentage of times that any page number is found in the TLB is called the hit ratio. Assume a TLB hit ratio of 92% and a single-level page table. If it takes 15**

**nanoseconds to lookup the TLB, and 120 nanoseconds to access memory, calculate the effective memory-access time.**
[2 points]

**Ans:**

TLB Hit: Page number is in the TLB:

- TLB search + Memory access for the desired byte: 15+120=135 ns

TLB Miss: Page number not in TLB:

- TLB search + Memory access for the page table and frame number + Memory access for the desired byte: 15+120+120=255 ns

Effective memory access time:
= 0.92 x 135 + 0.08 x 255 = 144.6 ns

**6) Explain how the use of a global page replacement algorithm can lead to thrashing. What would be one solution to limit the effects of thrashing in this case?**
[2 points]

**Ans:**

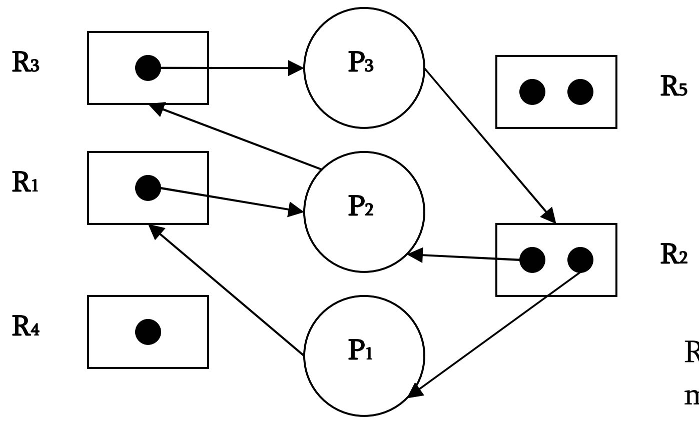
A global page replacement algorithm replaces pages without regard to the process to which they belong. Now suppose a process enters a new phase in its execution and needs more frames. It starts faulting and taking frames from other processes. These processes need those pages, however, and so they also fault, taking frames from other processes. These faulting processes must use the paging device to swap pages in and out. As they queue up for the paging device, the ready queue empties. As processes wait for the paging device, CPU utilization decreases. The CPU scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming as a result. The new process tries to get started by taking frames from running processes, causes more page faults and a longer queue for the paging device. Thrashing has occurred and, the system throughput plunges.
We can limit the effects of thrashing by using a local replacement algorithm. With local replacement, if one process starts thrashing, it cannot steal frames from another process and cause the latter to thrash as well.

**7) To keep track of free disk space, the system maintains a free-space list. Give 3 common methods of implementing a free-space list in a system. What are the pros and cons of each approach?**
[1+1+1 points]

Ans:

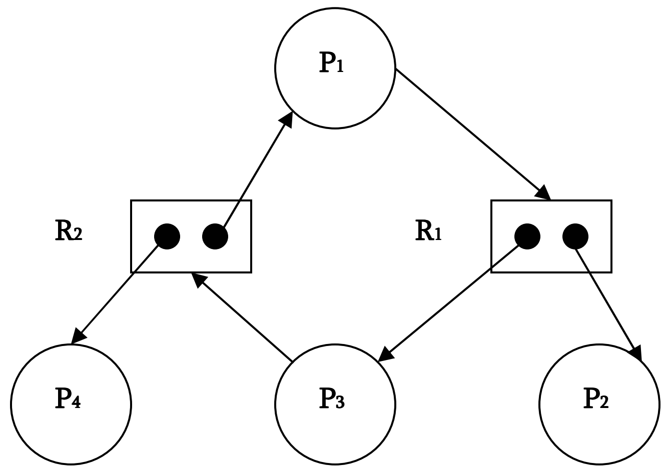
1) Bit vector
2) Linked list
3) FAT

**8) Look at the resource allocation graphs below. Point out the cycles that exist in each of the graphs and demonstrate if they have a possible deadlock or not.**



Graph 1

Resources $R_2$ and $R_5$ have multiple instances



Graph 2

Resources $R_1$ and $R_2$ have multiple instances

[3 points]

Ans:

Graph 1: Cycle: P1-R1-P2-R3-P3-R2-P1 P2-R3-P3-R2-P2
Processes P1, P2, P3 are deadlocked. P2 is waiting for the resource R3, which is held by process P3. Process P3 is waiting for either process P1 or process P2 to release resource R2. In addition, process P1 is waiting for process P2 to release resource R1.
Graph 2: Cycle: P1-R1-P3-R2-P1
There is no deadlock. P4 may release its instance of resource type R2. That resource can be allocated to P3, breaking the cycle.

**9) [Silberschatz Galvin Gagne – 7th Edition – Chapter 8 – Problem 8.3] Given 5 memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, 600 KB (in order), how would each of the first-fit, best-fit and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)? Which algorithm makes the most efficient use of memory?**
**(Assume searching always starts at the beginning of the set of partitions)**
[3 points]

Ans:

| PARTITIONS | FIRST FIT | | BEST FIT | | WORST FIT | |
|---|---|---|---|---|---|---|
| | (Used) KB | (Free) KB | (Used) KB | (Free) KB | (Used) KB | (Free) KB |
| **100 KB** | | 100 | | 100 | | 100 |
| **500 KB** | 417 | 83 | 417 | 83 | 417 | 83 |
| **200 KB** | 112 | 88 | 112 | 88 | | 200 |
| **300 KB** | 212 | 88 | 212 | 88 | 300 | 0 |
| **600 KB** | 426 | 174 | 426 | 174 | 212+112 +126 | 150 |
| | | | | | | |
| TOTAL FREE | | 533 | | 533 | | 533 |

In all the three cases, 533 KB of memory is unused (free). The first-fit and best-fit algorithms generate the exact same results.

However, in this case, the worst fit algorithm does not suffer from as much internal fragmentation as the best-fit and first-fit algorithms. Hence, to minimize the effects of internal fragmentation within the given partitions, it would be a better idea to use worst-fit as compared to the other two.

**10) Assume that in a system with a 32-bit address space, the page size is 4KB. How many bits are required for representing the page-offset and the virtual page number? Consider the following page table entries:**

| VPN | Frame Number |
|---|---|
| | |
| 0x7 | 0x0005 |
| 0x8 | 0x0002 |
| 0x9 | 0x0001 |
| 0xA | 0x0007 |
| 0xB | 0x0003 |
| 0xC | 0x0006 |
| 0xD | 0x0004 |

**What would be the physical address for the virtual address 0x0000A579?**
**[3 points]**

**Ans:**

Page size is 4KB. Thus, 12 lower-order bits are used for the page offset.
32-12=20 higher-order bits are used for representing the virtual page number.
Virtual address is 0x0000A579.
Thus, the VPN is 0x0000A and the offset is 0x00000579

The page table entry for 0xA says 0x0007
Multiplying by the page frame base 0x1000, we get 0x7000 or 0x00007000

Physical address= 0x00007000+0x00000579=**0x00007579**