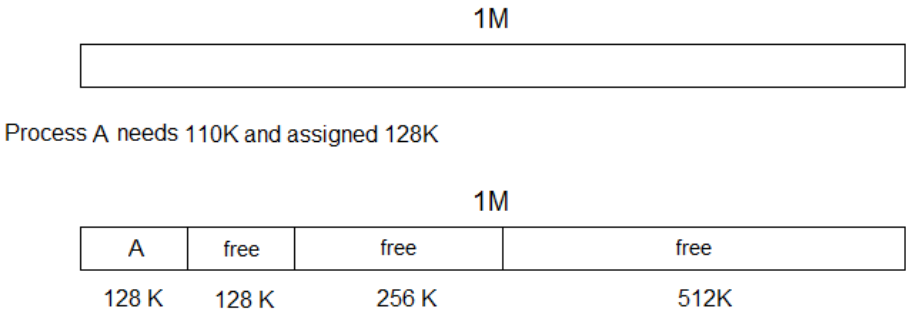
Problem Set 3 Answers:

[Q1] In this method, called buddy allocation, partition sizes are restricted to powers of 2. This makes the number of possible partition sizes limited and allows having a separate list for each size. This speeds up the search compared to variable partitions with a single common list. The disadvantage is that when process needs a memory size, it is assigned the nearest larger power of 2. This causes a waste as a result of unused space.

As an example of how the method operates, assume a memory size of 1M, which is initially free. Process A starts requiring 110K. It will be assigned the nearest larger power of 2, which is 128K. To obtain it, 1M area is split into two partitions of 512K each. One of these partitions is split again into two partitions of 256K each, and one of these is split again to obtain a 128K partition.



At this point, system has three free partitions of sizes 128K, 256K, and 512K. These are not merged together to restrict partition sizes to powers of 2. Only adjacent partitions of the same size can be merged.

[Q2] Effective instruction time will be $1 + (n/k)\ \mu s$. The page fault time n will depend on page size, and how processor accesses the page table. Page fault rate and number k will depend also on page size, and how much page frames are assigned to the process.

[Q3] (See lecture 8, slide 26) To fit page table in a single page means we need a table to have 4k/4=1024 entries. This means that size of page table entry will be 10 bits. Since total address size is 64 bits, of which 12 are offset inside page, leaving 52 bits, 6 levels will be needed to obtain the 10 bits page table entry.

[Q4] Both statements puts all elements of a two dimensional array to zero, but one does this row by row, and the other does this column by column. Two dimensional arrays are stored in the one-dimensional memory in the form of row by row (in most compilers, some store it column by column). Assume that one row fits in a page. The

version that works row by row will have a page fault that brings the first row in RAM. It then accesses the remainder or row without page faults. However, the version that works column by column will use one element only of the row, then moves to another row causing another fault. Thus this version will have much more faults and works slower. Of course this effect will depend on how many rows a page can hold.

[Q5] This method will remove from RAM a page that was not used since last page fault. Thus, it takes into account the use of pages as LRU does. It is much simpler as it needs to store one bit for each page and not the time of last access as needed in theoretical LRU. However, it selects one of the pages that was not accessed since last page fault, and not necessarily the one used earlier. Thus, it does not work exactly as LRU and is expected to result in more page faults than LRU.
You can find better and still simple methods in references.

[P1] When processes a,c, and e terminate, system will have free memory areas of sizes 200K, 100K, 120K, and 200K.

<u>in case of best-fit</u>
process g will be assigned 80K in the 2nd area (place of c), leaving free areas of sizes 200,20,120,200.
process h will be assigned 100K in the 3rd area (place of e), leaving free areas of sizes 200,20,20,200.
process i will be assigned 150K in the 1st area (place of a), leaving free areas of sizes 50,20,20,200.

<u>in case of first-fit</u>
process g will be assigned 80K in the 1st area (place of a), leaving free areas of sizes 120,100,120,200.
process h will be assigned 100K in the 1st area (place of a), leaving free areas of sizes 20,100,120,200.
process i will be assigned 150K in the 4th area (free place), leaving free areas of sizes 20,100,120,50.

<u>in case of next-fit</u>
process g will be assigned 80K in the 4th area (free place), leaving free areas of sizes 200,100,120,120.
process h will be assigned 100K in the 4th area (free place), leaving free areas of sizes 200,100,120,20.
process i will be assigned 150K in the 1st area (place of a), leaving free areas of sizes 50,100,120,20.

Note that best-fit leaves smaller fragments.

[P2]

a) Using SPN, A runs then D then B then A and the average waiting time will be 4.5.
b) Using first-fit, when A starts, it will be assigned 4M from the first 8M, leaving 4M, then B will be assigned these 4M then C and D will be assigned the 6M area. Thus, all processes can start immediately and using RR with q=1, the average waiting time will be 9.25.
c) Using best-fit, A will be assigned 4M from the 6M area leaving 2M. B and C will be assigned 7M from the 8M area, leaving 1M. Thus D will not be able to run when it arrives and must wait for the first process to terminate to start running.

Using RR with q=1, the average waiting time will be 8.25.

[P3] This problem illustrates how sequences of addresses are generated when programs run. Note that addresses are given in decimal. Thus to find the number of page in which any address is found, we divide the address by 512 and take the integer part (we always start from page 0). The sequence of accessed addresses are:

- First instruction is fetched from address 1020, which is in page 1.
- Data is obtained from location 6144, which is in page 12.
- Second instruction is fetched from address 1024, which is in page 2.
- Data is written in stack at address 8191, which is in page 15.
- Third instruction is fetched from address 1028 in page 2.
- Return address is written in stack in page 15.
- Next instruction is fetched from address 5120 in page 10 (program jumped to this address). No further memory accesses in this instruction.
- Next instruction is fetched from address 5124 in page 10.
- Next instruction is fetched from address 5128 in page 10.

From the above, the numbers of pages accessed (page reference string) is:

1- 12- 2- 15- 2- 15- 10- 10- 10

[P4] With 4 frames and FIFO

	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
0	1	1	1	1	1	1	5	5	5	5	5	3	3	3	3	3	1	1	1	1
1		2	2	2	2	2	2	6	6	6	6	6	7	7	7	7	7	7	3	3
2			3	3	3	3	3	3	2	2	2	2	2	6	6	6	6	6	6	6
3				4	4	4	4	4	4	1	1	1	1	1	1	2	2	2	2	2
fault	x	x	x	x			x	x	x	x		x	x	x		x	x		x	

Which results in 14 faults.

With 4 frames and LRU

	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
0	1	1	1	1	1	1	1	1	1	1	1	1	1	6	6	6	6	6	6	6
1		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2			3	3	3	3	5	5	5	5	5	3	3	3	3	3	3	3	3	3
3				4	4	4	4	6	6	6	6	6	7	7	7	7	1	1	1	1
fault	x	x	x	x			x	x				x	x	x			x			

Which results in 10 faults.

The virtual address in page 6 will be translated by replacing the page number 6 by the number of frame in which page 6 is loaded (0, 1, 2, or 3 as above).

Similarly, for 5 frames FIFO will have 10 faults and LRU will have 8 faults. With 6 frames FIFO will have 10 faults and LRU will have 7 faults (the minimum number since 7 pages are demanded).

[P5] Solved in lecture**.**

[P6] Remember that virtual page 2 of process A is different from page 2 of process B. Each process independently replaces its pages in the 3 frames assigned to it (this is called local replacement in lecture). Thus, effectively we have two problems similar to the previous problems, one for pages of process A and one for pages of process B.

Process A will have 6 faults and process B will have 4 faults. Assigning 4 frames to one process and 2 frames to the other will not reduce the total number of page faults.
