Ch9 Exercises

9.11 Explain the difference between internal and external fragmentation.

Answer:

a. Internal Fragmentation:

Internal fragmentation happens when the memory is split into mounted-sized blocks. Whenever a method is requested for the memory, the mounted-sized block is allotted to the method. In the case where the memory allotted to the method is somewhat larger than the memory requested, then the difference between allotted and requested memory is called internal fragmentation. We fixed the sizes of the memory blocks, which has caused this issue. If we use dynamic partitioning to allot space to the process, this issue can be solved.

b. External Fragmentation:

External fragmentation happens when there's a sufficient quantity of area within the memory to satisfy the memory request of a method. However, the process's memory request cannot be fulfilled because the memory offered is in a non-contiguous manner. Whether you apply a first-fit or best-fit memory allocation strategy it' ll cause external fragmentation.

c. Difference:

	Internal Fragmentation	External Fragmentation
1)	Fixed-sized memory, blocks square measure appointed to process.	Variable-sized memory blocks square measure appointed to the method.
2)	Happens when the method or process is smaller than the memory.	Happens when the method or process is removed.
3)	Solution is the best-fit block.	Solution is compaction and paging.
4)	Occurs when memory is divided into fixed-sized partitions.	Occurs when memory is divided into variable size partitions based on the size of processes.
5)	The difference between memory allocated and required memory is called Internal fragmentation.	The unused spaces formed between non-contiguous memory fragments are too small to serve a new process is called External fragmentation.
6)	Occurs with paging and fixed partitioning.	Occurs with segmentation and dynamic partitioning.
7)	Occurs on the allocation of a process to a partition greater than the process's requirement. The leftover space causes degradation system performance.	Occurs on the allocation of a process to a partition greater which is exactly the same memory space as it is required.
8)	Occurs in worst fit memory allocation method.	Occurs in best fit and first fit memory allocation method.

9.13 Given six memory partitions of 100 MB, 170 MB, 40 MB, 205 MB, 300 MB, and 185 MB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of size 200 MB, 15 MB, 185 MB, 75 MB, 175 MB, and 80 MB (in order)? Indicate which—if any—requests cannot be satisfied. Comment on how efficiently each of the algorithms manages memory.

Answer:

a. First-Fit:

1) 200 MB is put in 205 MB partition, leaving:
(100 MB, 170 MB, 40 MB, 5 MB, 300 MB, 185 MB).
2) 15 MB is put in 100 MB partition, leaving:
(85 MB, 170 MB, 40 MB, 5 MB, 300 MB, 185 MB).
3) 185 MB is put in 300 MB partition, leaving:
(85 MB, 170 MB, 40 MB, 5 MB, 115 MB, 185 MB).
4) 75 MB is put in 85 MB partition, leaving:
(10 MB, 170 MB, 40 MB, 5 MB, 115 MB, 185 MB).
5) 175 MB is put in 185 MB partition, leaving:
(10 MB, 170 MB, 40 MB, 5 MB, 115 MB, 10 MB).
6) 80 MB is put in 170 MB partition, leaving:
(10 MB, 90 MB, 40 MB, 5 MB, 115 MB, 10 MB).

b. Best-Fit:

1) 200 MB is put in 205 MB partition, leaving:
(100 MB, 170 MB, 40 MB, 5 MB, 300 MB, 185 MB).
2) 15 MB is put in 40 MB partition, leaving:
(100 MB, 170 MB, 25 MB, 5 MB, 300 MB, 185 MB).
3) 185 MB is put in 185 MB partition, leaving:
(100 MB, 170 MB, 25 MB, 5 MB, 300 MB, 0 MB).
4) 75 MB is put in 100 MB partition, leaving:
(25 MB, 170 MB, 25 MB, 5 MB, 300 MB, 0 MB).
5) 175 MB is put in 300 MB partition, leaving:
(25 MB, 170 MB, 25 MB, 5 MB, 125 MB, 0 MB).
6) 80 MB is put in 125 MB partition, leaving:
(25 MB, 170 MB, 25 MB, 5 MB, 45 MB, 0 MB).

c. Worst-Fit:

1) 200 MB is put in 300 MB partition, leaving:
(100 MB, 170 MB, 40 MB, 205 MB, 100 MB, 185 MB).
2) 15 MB is put in 205 MB partition, leaving:
(100 MB, 170 MB, 40 MB, 190 MB, 100 MB, 185 MB).
3) 185 MB is put in 190 MB partition, leaving:
(100 MB, 170 MB, 40 MB, 5 MB, 100 MB, 185 MB).
4) 75 MB is put in 185 MB partition, leaving:
(100 MB, 170 MB, 40 MB, 5 MB, 100 MB, 110 MB).

5) The request of 175 MB process can't be satisfied, leaving:
(100 MB, 170 MB, 40 MB, 5 MB, 100 MB, 110 MB).
6) 80 MB is put in 170 MB partition, leaving:
(100 MB, 90 MB, 40 MB, 5 MB, 100 MB, 110 MB).

d. Comparison on the efficiency of these 3 algorithms:

1) **First-Fit:** This algorithm searches along the list looking for the first segment that is large enough to accommodate the process. The segment is then split into a hole and a process. This method is fast as the first available hole that is large enough to accommodate the process is used.

2) **Best-Fit:** Best-fit searches the entire list and uses the smallest hole that is large enough to accommodate the process. The idea is that it is better not to split up a larger hole that might be needed later. Best-fit is slower than first-fit as it must search the entire list every time. It has also be shown that best-fit performs worse than first-fit as it tends to leave lots of small gaps.

3) **Worst-Fit:** As best-fit leaves many small, useless holes it might be a good idea to always use the largest hole available. The idea is that splitting a large hole into two will leave a large enough hole to be useful. It has been shown that this algorithm is not very good either.

4) **In Conclusion:** Best-fit and worst-fit are slower than first-fit.

Ch10 Exercises

10.26 Discuss situations in which the least frequently used (LFU) page-replacement algorithm generates fewer page faults than the least recently used (LRU) page-replacement algorithm. Also discuss under what circumstances the opposite holds.

Answer:

1) **Least Frequently Used:** In an LFU cache, the entry that has been used the fewest number of times is evicted to make room for new data. When a new entry is added, its frequency count is set to 1. Each time an entry is accessed, its frequency count is incremented. This algorithm assumes that data that is frequently accessed is more likely to be needed in the future than data that is rarely used.

2) **Least Recently Used:** In an LRU cache, the cache entry that was least recently used is evicted to make room for new data. When a new entry is added, it is considered the most recently used. Each time an entry is accessed, its status is updated to reflect that it is now the most recently used. This algorithm assumes that the data that was recently used will be the data that is needed in the near future.

3) The situation that **LFU** generates fewer page faults than **LRU**:
LFU is better than LRU in situations where there is a mix of short-term and long-term memory access patterns. LFU is better suited for situations where the frequency of access to a page is more important than the recency of the access. Additionally, LFU can be beneficial when there are sudden changes in access patterns, as it can quickly adapt to the new frequency of page accesses compared to LRU.

4) The situation that **LRU** generates fewer page faults than **LFU**:
Otherwise, LRU is better than LFU, which are the situations where the recency of access to a page is more important than the frequency of the access, like much more short-term memory access patterns than long-term.

10.29 Consider a demand-paging system with the following time-measured utilizations:

CPU utilization	20%
Paging disk	97.7%
Other I/O devices	5%

For each of the following, indicate whether it will (or is likely to) improve CPU utilization. Explain your answers.

a. Install a faster CPU.

b. Install a bigger paging disk.

c. Increase the degree of multiprogramming.

d. Decrease the degree of multiprogramming.

e. Install more main memory.

f. Install a faster hard disk or multiple controllers with multiple hard disks.

g. Add prepaging to the page-fetch algorithms.

h. Increase the page size.

Answer:

a. **Install a faster CPU: NO.**
Unlikely to improve CPU utilization. The current CPU is underutilized. A faster one won't help.

b. **Install a bigger paging disk: NO.**
Unlikely to improve CPU utilization. The size of the backing store can effect the maximum degree of multiprogramming that the system can support, since a bigger backing store can hold more swapped-out pages. However, that does not seem to be the problem here. The backing store is very busy, and a bigger disk won't help that.

c. **Increase the degree of multiprogramming: NO.**
Very unlikely to improve CPU utilization. It appears this system is thrashing. Increasing the degree of multiprogramming is the exact wrong thing to do.

d. **Decrease the degree of multiprogramming: YES.**
Likely to improve CPU utilization. It appears that jobs are queued up, waiting on the backing store to serve their page faults. Swapping out entire jobs would allow the remaining processes to be allocated enough frames to reduce their page fault rates so they can actually spend time executing on the 5CPU, and thus finish. Once those processes finish, memory will be freed up, and the kernel can begin gradually swapping processes back in.

e. **Install more main memory: YES.**
Likely to improve CPU utilization. More memory would allow us to allocate more frames to each process, hopefully capturing more of each process's locality in memory, thus reducing its page-fault rate.

f. **Install a faster hard disk or multiple controllers with multiple hard disks:**
May slightly improve CPU utilization. A faster disk may reduce page-fault service time, leading to a decrease in effective access time and higher CPU throughput.

g. **Add prepaging to the page-fetch algorithms: NO.**
Prepaging is unlikely to increase CPU utilization. The problem with this system is that too many pages are fighting over too few frames. Bringing more pages into memory won't solve the problem.

h. **Increase the page size:**
If data and code are accessed sequentially, a large page size might capture more of a process's locality in memory, reducing the page fault rate and thus increasing the CPU utilization.

However, if the process does not access memory sequentially, a larger page size is unlikely to improve CPU utilization. A larger page size would tend to capture more code and data than is contained in the process's current locality, meaning code and data which are not being used are stored in memory, wasting memory and increasing the page fault rate.

For example, imagine a process that is traversing a massive tree in a breadth-first manner from root to leaves. Parent and child nodes of this tree are unlikely to reside on the same page. To the kernel, the process will appear to be accessing memory in a non-linear fashion.