**Part 1**

* **7.9**Explain the difference between internal and external fragmentation.

Answer:

Internal fragmentation occurs in fixed size memory allocation while external fragmentation occurs in dynamic memory allocation. When an allocated partition is occupied by a program that is lesser than the partition , remaining space goes wasted causing internal fragmentation. When enough adjacent space cannot be found after loading and unloading of programs, due to the fact that free space is distributed here and there, this causes external fragmentation.

* **7.11**Given six memory partitions of 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, and 125 KB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of size 115 KB, 500 KB, 358 KB, 200 KB, and 375 KB (in order)? Rank the algorithms in terms of how efficiently they use memory.

Answer:

a. **First-fit**:

1. 115 KB is put in 300 KB partition, leaving (185 KB, 600 KB, 350 KB,

200 KB, 750 KB, 125 KB)

2. 500 KB is put in 600 KB partition, leaving (185 KB, 100 KB, 350 KB,

200 KB, 750 KB, 125 KB)

3. 358 KB is put in 750 KB partition, leaving (185 KB, 100 KB, 350 KB,

200 KB, 392 KB, 125 KB)

4. 200 KB is put in 350 KB partition, leaving (185 KB, 100 KB, 150 KB,

200 KB, 392 KB, 125 KB)

5. 375 KB is put in 392 KB partition, leaving (185 KB, 100 KB, 150 KB,

200 KB, 17 KB, 125 KB)

b. **Best-fit**:

1. 115 KB is put in 125 KB partition, leaving (300 KB, 600 KB, 350 KB,

200 KB, 750 KB, 10 KB)

2. 500 KB is put in 600 KB partition, leaving (300 KB, 100 KB, 350 KB,

200 KB, 750 KB, 10 KB)

3. 358 KB is put in 750 KB partition, leaving (300 KB, 100 KB, 350 KB,

200 KB, 392 KB, 10 KB)

4. 200 KB is put in 200 KB partition, leaving (300 KB, 100 KB, 350 KB, 0

KB, 392 KB, 10 KB)

5. 375 KB is put in 392 KB partition, leaving (300 KB, 100 KB, 350 KB, 0

KB, 17 KB, 10 KB)

c. **Worst-fit**:

1. 115 KB is put in 750 KB partition, leaving (300 KB, 600 KB, 350 KB,

200 KB, 635 KB, 125 KB)

2. 500 KB is put in 635 KB partition, leaving (300 KB, 600 KB, 350 KB,

200 KB, 135 KB, 125 KB)

3. 358 KB is put in 600 KB partition, leaving (300 KB, 242 KB, 350 KB,

200 KB, 135 KB, 125 KB)

4. 200 KB is put in 350 KB partition, leaving (300 KB, 242 KB, 150 KB,

200 KB, 135 KB, 125 KB)

5. 375 KB must wait

In this example, only worst-fit does not allow a request to be satisfied.

An argument could be made that best-fit is most efficient as it leaves the

largest holes after allocation. However, best-fit runs at time *O*(*n*) and

first-fit runs in constant time *O*(1).

* **7.18**Explain why address space identifiers (ASIDs) are used.

Answer:

ASIDs provide address space protection in the TLB as well as

supporting TLB entries for several different processes at the same time.

* **7.20**Assuming a 1-KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers):  a. 3085 b. 42095  c. 215201 d. 650000  e. 2000001

Answer:

a. page = 3; offset = 13

b. page = 41; offset = 111

c. page = 210; offset = 161

d. page = 634; offset = 784

e. page = 1953; offset = 129

* **7.23**Consider a logical address space of 256 pages with a 4-KB page size, mapped onto a physical memory of 64 frames.

1. How many bits are required in the logical address?
2. How many bits are required in the physical address?

Answer:

a. m =???

Size of logical address space = 2m = # of pages × page size

2m = 256 ×10244

2m = 28 × 212

2m =220  »» **m=20 bit**

b. Let **(x**) is number of bits in the physical address

x =???

Size of physical address space = 2x

Size of physical address space = # of frames × frame size (Frame size = page size)

Size of physical address space = 64 ×10244

2x =26 × 212

2x = 218

»» number of required bits in the physical address=**x =18 bit**

**Part 2**

* **8.14**Assume that a program has just referenced an address in virtual memory. Describe a scenario in which each of the following can occur. (If no such scenario can occur, explain why.)  • TLB miss with no page fault • TLB miss and page fault • TLB hit and no page fault • TLB hit and page fault

Answer:

* TLB miss with no page fault page has been brought into memory, but has been removed from the TLB.
* TLB miss and page fault page fault has occurred.
* TLB hit and no page fault page is in memory and in the TLB. Most likely a recent reference.
* TLB hit and page fault cannot occur. The TLB is a cache of the page table. If an entry is not in the page table, it will not be in the TLB.
* **8.15**A simplified view of thread states is ***Ready***, ***Running***, and ***Blocked***, where a thread is either ready and waiting to be scheduled, is running on the processor, or is blocked (for example, waiting for I/O). This is illustrated in Figure 8.31. Assuming a thread is in the Running state, answer the following questions, and explain your answer:
  1. Will the thread change state if it incurs a page fault? If so, to what state will it change?
  2. Will the thread change state if it generates a TLB miss that is resolved in the page table? If so, to what state will it change?
  3. Will the thread change state if an address reference is resolved in the page table? If so, to what state will it change?

Answer:

1. On a page fault the thread state is set to blocked as an I/O operation is required to bring the new page into memory.
2. On a TLB-miss, the thread continues running if the address is resolved in the page table.
3. The thread will continue running if the address is resolved in the page table.

* **8.18**A certain computer provides its users with a virtual memory space of 232 bytes. The computer has 222 bytes of physical memory. The virtual memory is implemented by paging, and the page size is 4,096 bytes. A user process generates the virtual address 11123456. Explain how the system establishes the corresponding physical location. Distinguish between software and hardware operations.

Answer:

The virtual address in binary form is 0001 0001 0001 0010 0011 0100 0101 0110.

Since the page size is 212, the page table size is 220. Therefore the low order 12 bits “0100 0101 0110” are used as the displacement into the page, while the remaining 20 bits “0001 0001 0001 0010 0011” are used as the displacement in the page table.

* **8.21**Consider the following page reference string:  
   7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0 , 1.    
  Assuming demand paging with three frames, how many page faults would occur for the following replacement algorithms?  • LRU replacement • FIFO replacement • Optimal replacement

Answer:

* LRU replacement : 18
* FIFO replacement : 17
* Optimal replacement: 13
* **8.27**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.

1. Install a faster CPU.
2. Install a bigger paging disk.
3. Increase the degree of multiprogramming.
4. Decrease the degree of multiprogramming.
5. Install more main memory.
6. Install a faster hard disk or multiple controllers with multiple hard disks.
7. Add prepaging to the page-fetch algorithms.
8. Increase the page size.

Answer:

The system obviously is spending most of its time paging, indicating over-allocation of memory. If the level of multiprogramming is reduced resident processes would page fault less frequently and the CPU utilization would improve. Another way to improve performance would be to get more physical memory or a faster paging drum.

a. Install a faster CPU—No. This will likely have no effect. The limiting factor is available memory per program.

b. Install a bigger paging disk—No. This should have no affect really.

c. Increase the degree of multiprogramming—No. This typically decreases CPU utilization because less memory is available to each program and the chances of page faults increase.

d. Decrease the degree of multiprogramming—Yes. This typically increases CPU utilization by keeping more of the working set of each program in memory, thereby reducing the number of page faults.

e. Install more main memory—Likely to improve CPU utilization as more pages can remain resident and not require paging to or from the disks.

f. Install a faster hard disk or multiple controllers with multiple hard disks—This will decrease the time spent waiting for pages to be brought in so it’ll increase responsiveness of the system, but since the CPU context switches to other programs anyway, this might not increase CPU utilization that much, if at all. It’s possible that the faster page retrieval limits the number of context switches but, thrashing will still be occuring.

g. Add prepaging to the page fetch algorithms—This is increase CPU utilization by avoiding page faults by having the pages pulled into memory before they’re needed.

h. Increase the page size—This will increase CPU utilization because spatial locality will reduce the number of page faults. This comes at the cost of more internal fragmentation. If taken too far, can reduce the number of programs that can have a working set in memory.

**Part 3**

Consider a paging system with the page table stored in memory.

a. If a memory reference takes 200 nanoseconds, how long does a paged memory reference take?

b. If we add TLBs, and 75 percent of all page-table references are found in the TLBs, what is the effective memory reference time? (Assume that finding a page-table entry in the TLBs takes zero time, if the entry is there.)

Answer:

1. 2 memory accesses: page lookup followed by actual access => 2\*200ns = **400ns**
2. 75% \* TLB hit-time + 25% \* TLB miss-time = 75% \* 200ns + 25% \* 400ns = **250ns**