The value of the exam will be 100%. You can choose to answer questions to add up to 100, or answer all of them and have more chance to earn more points.

1. (20 points) Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 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?

Answer:

a. First-fit:
   - 212K is put in 500K partition
   - 417K is put in 600K partition
   - 112K is put in 288K partition (new partition 288K = 500K - 212K)
   - 426K must wait

b. Best-fit:
   - 212K is put in 300K partition
   - 417K is put in 500K partition
   - 112K is put in 200K partition
   - 426K is put in 600K partition

c. Worst-fit:
   - 212K is put in 600K partition
   - 417K is put in 500K partition
   - 112K is put in 388K partition
   - 426K must wait

In this example, Best-fit turns out to be the best.

2. (10 points) Compare the main memory organization schemes of contiguous-memory allocation, pure segmentation, and pure paging with respect to the following issues:

   a. external fragmentation
   b. internal fragmentation
   c. ability to share code across processes

Answer:

Contiguous memory allocation scheme suffers from external fragmentation as address spaces are allocated contiguously and holes develop as old processes die and new processes are initiated. It also does not allow processes to share code, since a process’s virtual memory segment is not broken into non- contiguous finegrained segments.

Pure segmentation also suffers from external fragmentation as a segment of a process is laid out contiguously in physical memory and fragmentation would occur as segments of dead processes are replaced by segments of new processes. Segmentation, however, enables processes to share code; for instance, two different processes could share a code segment but have distinct data segments.
Pure paging does not suffer from external fragmentation, but instead suffers from internal fragmentation. Processes are allocated in page granularity and if a page is not completely utilized, it results in internal fragmentation and a corresponding waste of space. Paging also enables processes to share code at the granularity of pages.

3. (20 points) Consider the following segment table:

<table>
<thead>
<tr>
<th>Segment</th>
<th>Base</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>219</td>
<td>600</td>
</tr>
<tr>
<td>1</td>
<td>2300</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>90</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>1327</td>
<td>580</td>
</tr>
<tr>
<td>4</td>
<td>1952</td>
<td>96</td>
</tr>
</tbody>
</table>

What are the physical addresses for the following logical addresses?
   a. 0,430
   b. 1,10
   c. 2,500
   d. 3,400
   e. 4,112

Answer:
   a. $219 + 430 = 649$
   b. $2300 + 10 = 2310$
   c. $500 > 100$. Illegal reference, trap to operating system.
   d. $1327 + 400 = 1727$
   e. $112 > 96$. Illegal reference, trap to operating system.

4. (20 points) Discuss situations under which the least frequently used page-replacement algorithm generates fewer page faults than the least recently used page replacement algorithm. Also discuss under what circumstance does the opposite holds.

Answer: Consider the following sequence of memory accesses in a system that can hold four pages in memory. Sequence: 1 1 2 3 4 5 1. When page 5 is accessed, the least frequently used page-replacement algorithm would replace a page other than 1, and therefore would not incur a page fault when page 1 is accessed again. On the other hand, for the sequence “1 2 3 4 5 2,” the least recently used algorithm performs better.

5. (20 points) Consider a demand-paging system with the following time-measured utilizations:

<table>
<thead>
<tr>
<th>Utilization Type</th>
<th>Utilization %</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU utilization</td>
<td>20%</td>
</tr>
<tr>
<td>Paging disk</td>
<td>97.7%</td>
</tr>
<tr>
<td>Other I/O devices</td>
<td>5%</td>
</tr>
</tbody>
</table>

Which (if any) of the following will (probably) improve CPU utilization? Explain your answer.
   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:
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. Get a faster CPU—No.
b. Get a bigger paging drum—No.
c. Increase the degree of multiprogramming—No.
d. Decrease the degree of multiprogramming—Yes.
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—Also an improvement, for as the disk bottleneck is removed by faster response and more throughput to the disks, the CPU will get more data more quickly.
g. Add prepaging to the page fetch algorithms—Again, the CPU will get more data faster, so it will be more in use. This is only the case if the paging action is amenable to prefetching (i.e., some of the access is sequential).
h. Increase the page size—Increasing the page size will result in fewer page faults if data is being accessed sequentially. If data access is more or less random, more paging action could ensue because fewer pages can be kept in memory and more data is transferred per page fault. So this change is as likely to decrease utilization as it is to increase it.

6. (20 points) A page-replacement algorithm should minimize the number of page faults. We can do this minimization by distributing heavily used pages evenly over all of memory, rather than having them compete for a small number of page frames. We can associate with each page frame a counter of the number of pages that are associated with that frame. Then, to replace a page, we search for the page frame with the smallest counter.

a. Define a page-replacement algorithm using this basic idea. Specifically address the problems of

   (1) what the initial value of the counters is,
   (2) when counters are increased,
   (3) when counters are decreased,
   (4) how the page to be replaced is selected.

b. How many page faults occur for your algorithm for the following reference string, for four page frames?

   1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2.

c. What is the minimum number of page faults for an optimal page replacement strategy for the reference string in part b with four page frames?
Answer:

a. Define a page-replacement algorithm addressing the problems of:
   1. Initial value of the counters—0.
   2. Counters are increased—whenever a new page is associated with that frame.
   3. Counters are decreased—whenever one of the pages associated with that frame is no longer required.
   4. How the page to be replaced is selected—find a frame with the smallest counter. Use FIFO for breaking ties.

b. 14 page faults

c. 11 page faults