CSE 430 Homework 6

Ryan Dougherty

Question 8.9

Explain the difference between internal and external fragmentation. When areas of memory are allocated, space can be wasted in two general ways. Wasted space inside of an allocated area of memory is referred to as internal fragmentation. External fragmentation refers to the wasted space outside of the allocated areas of memory.

Question 8.11

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. Let M_i correspond to the ith memory partition (in the order they are), and let P_i be the ith process (in the order they are).

- First-fit
  - $P_1 = 115KB$, so choose $M_1 = 300KB$. Change: $M_1 = 185KB$.
  - $P_2 = 500KB$, so choose $M_2 = 600KB$. Change: $M_2 = 100KB$.
  - $P_3 = 358KB$, so choose $M_5 = 750KB$. Change: $M_5 = 392KB$.
  - $P_4 = 200KB$, so choose $M_3 = 350KB$. Change: $M_3 = 150KB$.
  - $P_5 = 375KB$, so choose $M_5 = 392KB$. Change: $M_5 = 17KB$.
- Best-fit
  - $P_1 = 115KB$, so choose $M_6 = 125KB$. Change: $M_6 = 10KB$.
  - $P_2 = 500KB$, so choose $M_2 = 600KB$. Change: $M_2 = 100KB$.

  - $P_3 = 358KB$, so choose $M_5 = 750KB$. Change: $M_5 = 392KB$.
  - $P_4 = 200KB$, so choose $M_4 = 200KB$. Change: $M_4 = 0KB$.
  - $P_5 = 375KB$, so choose $M_5 = 392KB$. Change: $M_5 = 17KB$.

- Worst-fit
  - $P_1 = 115KB$, so choose $M_5 = 750KB$. Change: $M_5 = 635KB$.
  - $P_2 = 500KB$, so choose $M_5 = 635KB$. Change: $M_5 = 135KB$.
  - $P_3 = 358KB$, so choose $M_2 = 600KB$. Change: $M_2 = 242KB$.
  - $P_4 = 200KB$, so choose $M_3 = 350KB$. Change: $M_3 = 150KB$.
  - $P_5 = 375KB$, and since no hole is large enough, P_5 will have to wait until there is space available.

Question 8.13

Compare the memory organization schemes of contiguous memory allocation, pure segmentation, and pure paging with respect to the following issues:

(a) External fragmentation

- Contiguous memory allocation - There is external fragmentation (as address spaces are allocated contiguously and holes develop as finished processes release its space and new processes are allocated and the size of the new process is almost smaller than the old one).
- Pure segmentation - There is external fragmentation (fragmentation would occur as segments of finished processes are replaced by segments of new processes, and the size of the new process is almost smaller than the old one).
- Pure paging - There is no external fragmentation.

(b) Internal fragmentation

- Contiguous memory allocation - There is no internal fragmentation.
- Pure segmentation - There is no internal fragmentation.
- Pure paging - There is internal fragmentation (it appears in the last frame because the process size almost not a multiplex of page size).

(c) Ability to share code across processes

- Contiguous memory allocation - It does not allow processes to share code.
- Pure segmentation - Able to share code between processes.
- Pure paging - Able to share code between processes.

Question 8.25

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

(a) If a memory reference takes 50 nanoseconds, how long does a paged memory reference take? Here, we have 2 memory accesses: 1 for the page lookup, and another for the actual access. Since each takes 50ns, the whole paged memory reference takes 100ns.

(b) If we add TLBs, and 75 percent of all page-table reference are found in the TLBs, what is the effective memory reference time? (Assume that finding a page-table entry in the TLBs takes 2 nanoseconds, if the entry is present.) 75%*(TLB hit time + finding page table entry) + 25%*(TLB miss time + finding page table entry) $= 75\% * (50 + 2) + 25\% * (100 + 2) = 64.5$ns.

Question 9.17

What is the copy-on-write feature, and under what circumstances is its use beneficial? What hardware support is required to implement this feature? Copy-on-write allows processes to share pages rather than each having a separate copy of the pages. However, when one process tried to write to a shared page, then a trap is generated and the OS makes a separate copy of the page for each process. This is commonly used in a fork() operation where the child is supposed to have a complete copy of the parent address space. Rather than create a separate copy, the OS allows the parent and child to share the parent's pages. However, since each is supposed to have its own private copy of the pages, the pages are copied when one of them attemps a write. The hardware support required to implement is simply the following: on each memory access, the page table needs to be consulted to checkwhether the page iswrite-protected. If it is indeedwrite-protected, a trap would occur and the operating system could resolve the issue.

Question 9.19

Assume that we have a demand-paged memory. The page table is held in registers. It takes 8 milliseconds to service a page fault if an empty frame is available or if the replaced page is not modified and 20 milliseconds if the replaced page is modified. Memory-access time is 100 nanoseconds. Assume that the page to be replaced is modified 70 percent of the time. What is the maximum acceptable page-fault rate for an effective access time of no more than 200 nanoseconds? Let p be the page fault rate (i.e. the probability that a memory access results in a page fault). Then 1-p is the probability that a memory access costs 100ns. The probability that a page fault costs 20ms is 0.7*p and the probability that a page fault costs 8ms is 0.3*p. Since 1ns = 10^6ms, (1-p)*100 + 0.7*p*2*10^6 + 0.3*p*8*10^6 = 200. Therefore, (14*10^6 + 2.4*10^6 - 100)*p = 100. Then, p = $\frac{100}{16400100}$ = 0.00061%.

Question 9.27

Consider a demand-paging system with the following time-measured utilizations:...For each of the following, indicate whether it will (or is likely to) improve CPU utilization. Explain your answers.

(a) Install a faster CPU. Unlikely to have an effect, since available memory per program is limited.

(b) Install a bigger paging disk. Also unlikely to have an effect.

(c) Increase the degree of multiprogramming. This decreases CPU utilization because less memory is available to each program. Therefore, the probability for a page fault increases.

(d) Decrease the degree of multiprogramming. This increases CPU utilization because it keeps more of the working set of each program in memory. Therefore, the probability for a page fault decreases.

(e) Install more main memory. This increases CPU utilization because there will be less paging taking CPU time to service.

(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 will increase the system's responsiveness. However, since the CPU context switches to other programs, this may not increase CPU utilization. It is possible that the faster page retrieval limits the number of context switches, but thrashing could still occur.

(g) Add prepaging to the page-fetch algorithms. This increases CPU utilization because it avoids page faults by having the pages pulled into memory before they are needed.

(h) Increase the page size. This increases CPU utilization because spatial locality will reduce the number of page faults. However, there is a cost of more internal fragmentation. If the page size is increased too much, then the number of programs that can have a working set in memory will be reduced.

Question 9.33

Is it possible for a process to have two working sets, one representing data and another representing code? Explain. Yes, in fact many processors provide two TLBs for this very reason. As an example, the code being accessed by a process may retain the same working set for a long period of time. However, the data the code accesses may change, thus reflecting a change in the working set for data accesses.