CSE 430 Homework 6

Ryan Dougherty

Question 8.9

Explain the difference between internal and external fragmentation. When areas of memory are allocated, space could be wasted in 2 ways: internal and external fragmentation. Internal fragmentation is wasted space inside of an allocated area of memory, and external fragmentation is 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.

- I will define the ranking of algorithms based on the average percentage of memory for each memory partition is still available.
  - For first-fit, the ratios of memory left are: $M_1 = 185/300, M_2 = 100/600, M_3 = 150/350, M_4 = 200/200, M_5 = 17/750, M_6 = 125/125$. The average is: 53.9%.
  - For best-fit, the ratios of memory left are: $M_1 = 300/300, M_2 = 100/600, M_3 = 350/350, M_4 = 0/200, M_5 = 17/750, M_6 = 10/125$. The average is: 37.8%.
  - For worst-fit, the ratios of memory left are: $M_1 = 300/300, M_2 = 242/600, M_3 = 150/350, M_4 = 200/200, M_5 = 135/750, M_6 = 125/125$. The average is: 66.9%.

- Therefore, best-fit most efficiently uses memory, first-fit next, and last is worst-fit. We could also define our ranking to be whether the algorithm requires a wait for a process. In that case, we could say that best-fit and first-fit are the most efficient, and worst-fit is the less efficient (P_5 has to wait).

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. This happens because address spaces are allocated contiguously, and "holes" develop as old processes terminate and new processes start.
- Pure segmentation - There is external fragmentation. It occurs as new processes' segments replace those of finished processes, and as the new process's size almost is smaller than the old process.
- 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. If a page is not completely utilized, it results in internal fragmentation and a corresponding wastage of space because processes are allocated in page granularity.

(c) Ability to share code across processes

- Contiguous memory allocation - It is not possible to share code between processes.
- Pure segmentation - It is possible to share code between processes.
- Pure paging - It is possible 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? The copy-on-write feature allows processes to share pages rather than each having a separate copy. However, when one process tries to write to a shared page, a trap is generated and the OS makes a separate copy of the page for each process.

It use is beneficial when doing a fork() system call. There, the child process should have be a copy of the parent (including address space, etc.). However, what needs to be copied can be quite large. To counteract this, the OS allows both the parent and child to share the pages of the parent. However, since both the child and parent should have their own private copies of those pages, they are copied when either the parent or child attempts a write.

The hardware support required to implement this feature is that on each memory access, the OS needs to check the page table to see if the page has write protections. If so, a trap occurs and the OS can 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%, or approximately 1 fault for every 164k pages.

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. This will not likely have any effect (available memory/program is limited).

(b) Install a bigger paging disk. This is also unlikely to have any effect.

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

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

(e) Install more main memory. This increases CPU utilization because there will be less paging that takes up CPU time.

(f) Install a faster hard disk or multiple controllers with multiple hard disks. This decreases time spent waiting for pages to be brought in ⇒ system's responsiveness increases. However, since the CPU does context switches to other programs, this may not increase CPU utilization.

(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 less.

Question 9.33

Is it possible for a process to have two working sets, one representing data and another representing code? Explain. Yes. Some processors have 2 TLBs for this scenario, i.e. the code accessed by some process possibly can have the same working set for awhile. However, there may be a change with respect to the working set (regarding data accesses) because the accessed data may change.