CIS 520 - Operating Systems I – Homework #4

Due: Wednesday, Nov. 6th, by 11:59 pm, upload via K-State OnLine

1. Dynamic Memory Management: A dynamic memory allocator uses a linked list to track free blocks. Suppose that the free list contains just two blocks, of size 24 and 16 bytes, in that order. Ignore any space required for bookkeeping overhead.

   (a) List a sequence of malloc( ) calls that would succeed using first-fit allocation, but fail with best-fit, or explain why such a sequence cannot exist.

   12, 12, 16, first fit would place 12 in the first block with 12 left over, then 12 in the left over portion of the first block, and finally 16 in the second block; whereas, best fit would put 12 in the second block with 4 left over, 12 in the first block with 12 left over, and then not enough left over to satisfy the request for 16. Other examples are possible.

   (b) List a sequence of malloc( ) calls that would succeed using best-bit, but fail with first-fit, or explain why such a sequence cannot exist.

   8, 24, first fit would put 8 in the first block of size 24 with 16 left over, and there wouldn't be enough room to satisfy the request for 24; whereas, best fit would put 8 in the second block, and 24 in the first.

   (c) List a sequence of malloc( ) calls that would succeed on the original free list, with free blocks of size 24 and 16, using worst-fit, but fail with first-fit, or explain why such a sequence cannot exist.

   12, 6, 10, 8, both would put 12 in the first block, with free blocks of size 12 and 16 left over, then first fit would put 6 in the first block (with 6 left over), and 10 in the second block (with 6 left over), so the final request for 8 could not be satisfied; whereas, worst fit would put 6 in the second block (with 10 left over), 10 in the first block (with 2 left over), and 8 in the second block (with 2 left over).

   (d) In a pure paging system, processes are allocated a fixed number of pages. Each page has a fixed size. So, typically, half of the last page is wasted when memory is allocated. Is this an example of internal or external fragmentation? Explain briefly.

   Internal, because it is memory wasted inside of the memory allocated to the process

   (e) For dynamic memory management, if the process calls malloc( ) with a request for 14 bytes and the allocator is using the best-fit algorithm, the request for 14 bytes will be satisfied by breaking the block of size 16 into two parts, and returning the remaining 2 bytes to the free list. This small block of size 2 bytes is sometimes referred to as "sawdust" because it will probably never be used. Is this an example of internal or external fragmentation? Explain briefly.

   External, because the wasted "sawdust" memory is not allocated to a process.

2. Paging and Page Replacement Algorithms:

   (a) Discuss situations in which the most recently used (MRU) page replacement algorithm generates fewer page faults than the least recently used (LRU) page replacement algorithm. Also, discuss under what circumstances the opposite holds.

   If the pages in memory are accessed like a circular queue or in a round robin fashion so that the page that has been in memory the longest is accessed first.

   Likewise, LRU generates fewer faults if pages that were recently loaded are accessed in the near future. Then, the pages that are least recently used are evicted first.

   (b) Discuss the philosophy behind the second chance (clock) algorithm and how it approximates LRU. Drawings are fine here to help with your explanation.

   The second chance (clock) algorithm looks at pages in turn around the clock to find one that has not been accessed recently. If the Accessed flag is set A=1, as the hand goes by, it is cleared, A is set to 0. If we get all the way around the clock without finding a page that has it's A bit cleared (set to 0), then the first page we set A=0 will be evicted, but at least it got a "second chance". Instead of storing a complete history to determine which page was Least Recently Used, the clock algorithm just approximates LRU by using the last time interval since we ran the algorithm as the "history".

   (c) What is Belady's anomaly?

   Increasing the number of page frames doesn't mean we won't increase the number of page faults.

   (d) Can the FIFO algorithm exhibit Belady's anomaly?

   Yes.

   (e) Can the LRU algorithm exhibit Belady's anomaly?

   No.

3. System Calls and Swapping:

   (a) Your teammates suggests that, in order to check the validity of the buffer passed to the write( ) system call (which is declared as **write(int fd, const void *buffer, size_t length)**, you could simply check whether the beginning of the buffer ('**buffer**') and the end of the buffer (' **buffer+length–1**') are in the user virtual address range and have valid page table entries. Is this correct? Either briefly explain why this approach is correct or give an example user program for which this approach fails.

   No, this is not correct. Suppose that we dynamically allocate three segments of memory to be used as the buffer, and free the middle segment. Then, the addresses in the first segment and last segment are valid, but the memory in between could have been allocated to another process.

(b) In Unix terminology, a child process becomes a ‘zombie’ if it calls exit() before its parent calls wait(). Your system administrator claims that too many zombie processes are bogging down the machine. Is he correct? Briefly justify your answer.

No. Zombies don’t use system resources other than taking up space in a process control block. So, unless the process table is completely full, zombie processes will not have any significant effect on performance.

(c) In Pintos, if the parent process calls wait() after the child has already called exit(), how can we ensure that the parent knows that the child process is a zombie (has already exited)? Likewise, how can we block the parent if the parent calls wait() before the child calls exit()?

The standard approach is to use a semaphore initialized to 0 associated with the child thread. The parent will call sem_wait( ) = sema_down( ) on the semaphore to wait on the child, and when the child exits, it will signal the parent by calling sem_signal( ) = sema_up( ) on its semaphore. In this way, if the child exits first, the semaphore count will be incremented to 1 to ensure that the parent is not blocked when it calls wait – it can just record the child’s exit code and return.