CSC 4420
Assignment 4
May Wandyez
Gq5426
7/19/2024

Question 1 (15 points): Consider the following page reference string:

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

How many page faults would occur for the listed replacement algorithms, assuming one, two, three, four, five, six, and seven frames? Remember all frames are initially empty, so your first unique pages will all cost one fault each.

- LRU replacement
- FIFO replacement
- Optimal replacement

A page fault occurs when the initial empty state of frames is tried to be loaded. Wow, this variation in frames is too complicated to do by hand, time for coding! Lists in C++ are annoying, so I used python instead:

```
#Hello brave astronaut, welcome to the CSC 4420 Assignment 4 Q 1code

#The plan is simple, define functions that return the total number of faults

#every time a page list with a given frame length is chosen


def lru_replacement(pages, frames):
    frame_list = []
    page_faults = 0
    for page in pages:
        if page not in frame_list:
            #this means we are looking for the page in pages,
            #but it isn't in the local frame list we are iterating through
            if len(frame_list) < frames:
                frame_list.append(page)
            #this means the list is full at the moment,
            else:
                #whoever is in the back is the least recently used
                #because otherwise it would have been moved to the front.
                #therefore we need to remove it, then put the new recently used
                #page in front.
                least_recent = frame_list.pop(0)
```

```
            frame_list.append(page)
        page_faults += 1
    else:
        #congrats! The page we are looking for already exists within frame list
        #therefore we should remove it from the frame, and put it in the front
        frame_list.remove(page)
        frame_list.append(page)
  return page_faults

def fifo_replacement(pages, frames):
  #fifo is easy, whichever was in there last goes out.
  frame_list = []
  page_faults = 0
  for page in pages:
    if page not in frame_list:
      if len(frame_list) < frames:
        #aww it's empty, fill it up
        frame_list.append(page)
      else:
        #remove the most back of the frame list, put the new one in the front
        frame_list.pop(0)
        frame_list.append(page)
      page_faults += 1
  return page_faults

def future_uses(pages, current_index, frame_list):
  # Initialize the frame to remove as the first frame in the list
  frame_to_remove = frame_list[0]
```

```
    max_future_index = -1

    # Iterate through each frame to consider its future use
    for frame in frame_list:
        # Check when this frame will be used next
        if frame in pages[current_index+1:]:
            future_index = pages[current_index+1:].index(frame)
        else:
            # If the frame is not used in the future, it should be removed
            future_index = float('inf')

        # Select the frame with the farthest future use
        if future_index > max_future_index:
            max_future_index = future_index
            frame_to_remove = frame

    return frame_to_remove

def optimal_replacement(pages, frames):
    frame_list = []
    page_faults = 0

    for i, page in enumerate(pages):
        if page not in frame_list:
            if len(frame_list) < frames:
                # Add page to frame if there is space
                frame_list.append(page)
            else:
```

```
            # Use the future_uses function to determine which page to replace
            frame_to_replace = future_uses(pages, i, frame_list)
            frame_list.remove(frame_to_replace)
            frame_list.append(page)
        page_faults += 1

    return page_faults

# Page reference string NOW A LIST TO ITERATE THROUGH yay
pages = [1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6]

#List containing number of frames so I only need one file
frames_list = [1, 2, 3, 4, 5, 6, 7]

#Put all the results in one place
print("LRU Replacement:")
for frames in frames_list:
    faults = lru_replacement(pages, frames)
    print(f"Frames: {frames}, Page Faults: {faults}")

#this is FIFO
print("\nFIFO Replacement:")
for frames in frames_list:
    faults = fifo_replacement(pages, frames)
    print(f"Frames: {frames}, Page Faults: {faults}")

#This represents the optimal thing
print("\nOptimal Replacement:")
```

CSC 4420
Assignment 4
May Wandyez
Gq5426
7/19/2024

```
for frames in frames_list:
    faults = optimal_replacement(pages, frames)
    print(f"Frames: {frames}, Page Faults: {faults}")
```

Faults Per Algorithm per Frame							
	Frames						
Algorithm	1	2	3	4	5	6	7
LRU	20	18	15	10	8	7	7
FIFO	20	18	16	14	10	10	7
Optimal	20	15	11	8	7	7	7

Question 2 (5 points): Consider a demand-paged computer system where the degree of multiprogramming is currently fixed at four. The system was recently measured to determine utilization of CPU and the paging disk. The results are one of the following alternatives. For each case, what is happening? Can the degree of multiprogramming be increased to increase the CPU utilization?

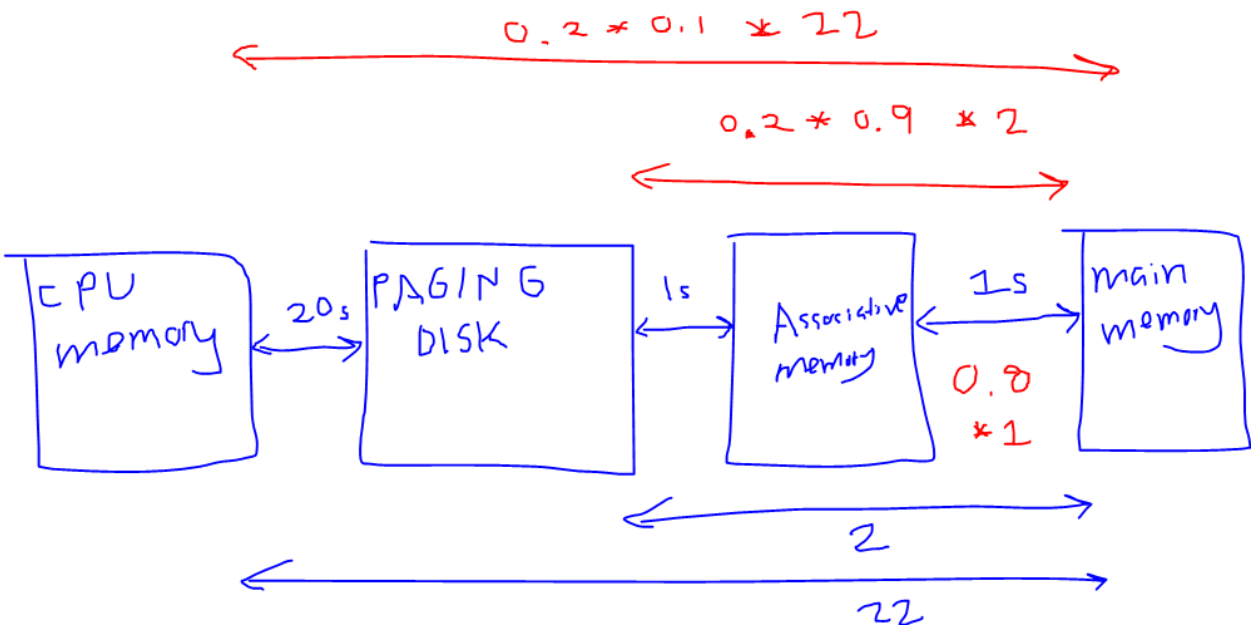
1. CPU utilization 13 percent; disk utilization 97 percent
2. CPU utilization 87 percent; disk utilization 3 percent
3. CPU utilization 13 percent; disk utilization 3 percent

1.) There's a lot of page faulting going on, more time is being spent swapping pages in and out of memory than actual calculating. This indicates that the degree of multiprogramming is too high, and increasing the amount of multiprogramming will just increase the number of faults without increasing CPU utilization. My current laptop is experiencing this I bet I could fix it with a solid state drive.
2.) The CPU is busy which is good! However, the disk is not, meaning we could increase the amount of multiprogramming to access information on the disk to increase CPU utilization further even though there is not much to be gained.
3.) The CPU isn't being used, and the disk isn't being used. This means your program is not taxing enough on the system to use all of it at once. Increasing multiprogramming would increase CPU utilization and would help it run faster since the CPU is barely being used at all. There is a lot to gain!

CSC 4420
Assignment 4
May Wandyez
Gq5426
7/19/2024

Question 3 (5 points): Consider a demand-paging system with a paging disk that has an average access and transfer time of 20 milliseconds. Addresses are translated through a page table in main memory, with an access time of 1 microsecond per memory access. Thus, each memory reference through the page table takes two accesses. To improve this time, we have added an associative memory that reduces access time to one memory reference, if the page-table entry is in the associative memory.

Assume that 80 percent of the accesses are in the associative memory and that, of the remaining, 10 percent (or 2 percent of the total) cause page faults. What is the effective memory access time?



0.8*1 + 0.18*2 + 0.2*0.1*22 = 1.6 ms