Final Assignment

Andela Acic
82272881

1. Short Answer

1. (10 pts) List 3 algorithms for managing memory with contiguous segmented allocation. Do these algorithms require compaction to work? Why or why not?

1. First fit - we will allocate the first one in the list in which the process fits.

2. Best fit - we will allocate the smallest hole that is large enough to hold the process

3. Worst fit - we will allocate the biggest hole to the process

These algorithms do not require compaction to work because if they can't find a place

to the process, they can just wait until one of the processes finishes to start requesting the process.

2. (10 pts) Explain the difference between logical and physical addresses.

Physical addresses have to exactly correspond to a hardware location, but logical addresses are tied to a process and must be just mapped to a physical location. In paging, virtual adresses have page and offset.

3. (10 pts) What is swapping and when is it used?

Swapping is rolling out a process to the disk, releasing all the memory that it holds and when the process becomes active again, OS has to

reload it back into the memory. Swapping is used by the OS to manage how many processes are in the memory at once. When too many processes are in memory at once, we use swapping. Swapping takes time so performance is sacrificed a bit, but it enables other processes to run and get needed memory.

2. (20 pts) **Performance of Paged Systems.** This question asks you to compute the effective memory access time (*ema*) in symbolic terms for different hardware implementations of paging with and without virtual memory. Let *ma* be the time to read or write a value in memory. Define other terms as you need them. First assume the process and its page table fit and stay in memory while the program executes, and there is *no* TLB.

1. (5 pts) What is the *ema* for this system?
2. (5 pts) Now assume there is a TLB. What is the *ema* for this system? Include *l*, the time for the TLB look up in your equation.
3. (5 pts) Now consider a virtual memory system with a TLB where the process can grow larger than the memory, and the page may not be in memory. What is the *ema* now? Assume that the page table is always in memory.
4. (5 pts) Now what is the *ema* for the same system except that the page table entry itself may not be in memory?

1. $ema = 2 \cdot ma$
(memory reference + page table lookup)

2. $ema = h \cdot (ma + l) + 2 \cdot (l - h) \cdot ma$

h -> probability of a TLB hit

h · (ma + l) -> time with TLB hit

2 · (l - h) · ma -> time with tlb miss

↳ here, l is not included since memory page table lookup happens in parallel with the TLB lookup

3. $emat = h \cdot (ma + l) + (l - h)(2 \cdot y \cdot ma + (1 - y)(pf + 2 \cdot ma))$

h -> probability of a TLB hit

y -> probability that referenced page is in the memory

pf -> page fault time

h · (ma + l) -> time with TLB hit

Here, TLB miss time is sum of probability of a page hit ($2 \cdot y \cdot ma$) and the page miss ($pf + 2 \cdot ma$) which includes memory access and page fault time.

4. Lets define b for simplicity:

$$b = 2 \cdot y \cdot ma + (1-y)(pf + 2 \cdot ma)$$

variables are the same as previous question, now we have:

$$emat = h \cdot (ma + t) + (1-h)(z \cdot b + (1-z)(pf + b))$$

$z \Rightarrow$ probability of no page fault at the page table lookup

$2 * b \rightarrow$ hit

3. (25 pts) **Segmentation with Paging.** Design a memory management scheme that uses segmentation with paging.

1. Specify and justify the number of segments.
2. Specify the translation of logical to physical addresses.
3. Specify hardware support to make address translation fast. (Draw a picture.)

1. I would choose 3 segments:

(1.) Segment for code
↳ so we can share it between the processes

(2.) Heap
↳ so we can share it between threads

(3.) Stack
↳ so the threaded programs can share it

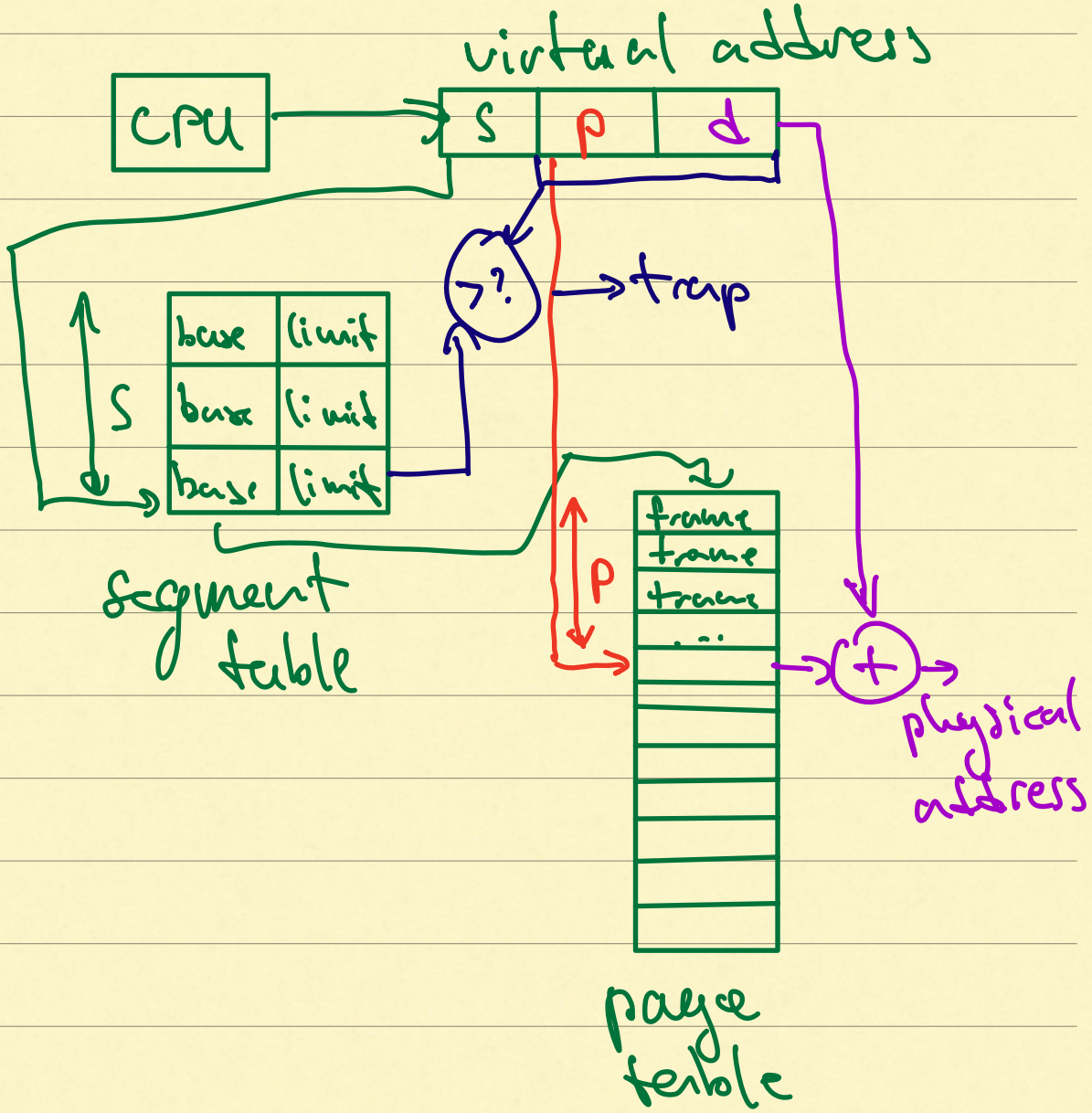
2. Segment number will have 2^n bits. Compiler will put these 2 bits on front of the logical address. This method is compatible with virtual memory and the rest of the address can grow larger than memory. The translation of the rest of the address splits logical address into 2 more parts:

page → p
offset into the page → d

A page will contain 2^d bytes. Number of pages will depend on the logical space. With a physical memory of size 2^k and no virtual memory, our system will need at most k bits.

With virtual memory it can be larger.

3. We will use a TLB for page table entries and registers (associative memory) for the 3 segment entries.



4. (25 pts) **Memory Management versus Disk Management.** Memory management and disk management have some key similarities and differences that result from their physical characteristics and from the access patterns each expects and supports well. Using at least 3 points of comparison, describe some of their similarities and differences.

1. Memory and disk are optimized for different sizes of objects. Most processes are big and bigger than a page, and they are logically contiguously allocated. Most files are small and smaller than a block, so most data fits in one block on the disk. Both memory and disk usually choose some internal fragmentation over external fragmentation to make freeing & allocation simpler

2. There is a big physical difference between memory and disk. Random access to memory takes the same time as sequential access, but for the disk, all accesses require a seek, so random access is much slower than access with sequential locality.

3. One thing they have in common is that they both expect to see sequential access and that is why they store more than one addressable word. Memory uses pages and disks use sectors/blocks.

4. Physical differences between memory and disk have big influence on where we can put data. In memory, it doesn't really matter where OS places the page, but on disk, we want OS to place most frequently used files or folders in the middle of the disk to keep seek times as low as possible.

5. (20 pts) **Short Answer.**

1. (5 pts) What type of naming scheme does the web use and what advantage does it have?
2. (5 pts) What does RPC stand for and how does it work?

3. (5 pts) Name the four components of disk access time.
4. (5 pts) What are the differences between user threads and kernel threads?

1. The web uses exact location names. These names have no location transparency or independance exept if they are implemented at a certain cite. The biggest advantage of this approach is that looking up the name is easy and fast.

2. RPC -> remote procedure call

It works by providing stubs at both server and client. First, RPC services register with the OS. Then, client invokes the procedure/system call. Then OS uses client stub to send a message to the server stub.

Then, server does the procedure work and sends result back to the client.

(3.) - rotational delay
- seek time
- latency
- CPU

(4.) User threads are implemented in user libraries. OS is not aware of user threads so they can have special scheduling algorithms in the code that suit the specific application. On the other hand, since OS doesn't know about them the OS may preempt the whole process when only one of its threads does IO even

where there are threads that can run. Kernel threads are threads that OS is aware of and because of that OS can schedule them even if other threads from the same process are doing IO or are blocked.