CIS 450 Homework #2

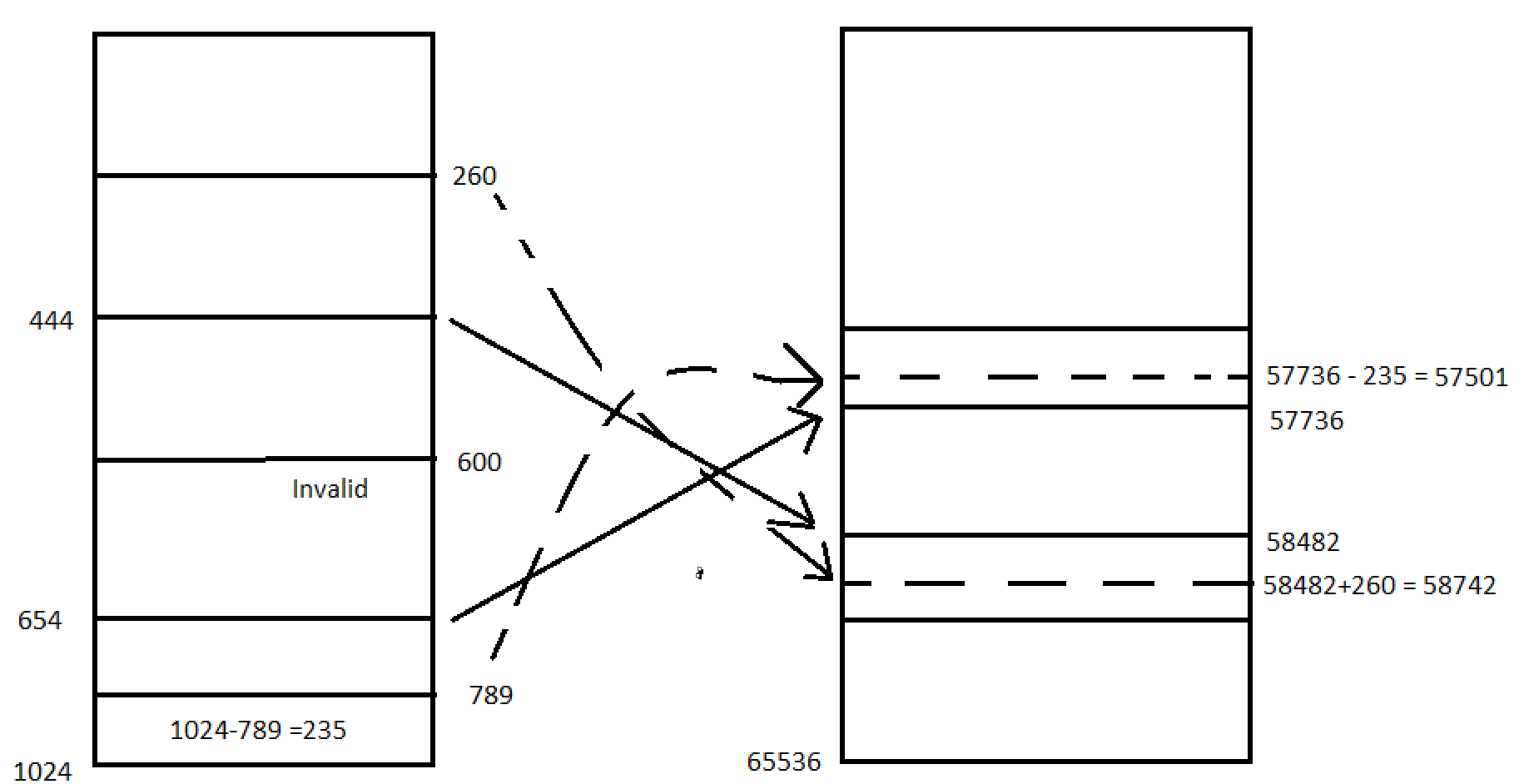
**Question 1:**

The elements involved in an address space of a process are the Program code (the code segment: Where the instructions are found), The heap (contains all the malloc’d or the allocated data that is used for dynamic structures), the stack which contains local variables, arguments to routines, return values, etc., and an allocated empty space for the stack to grow during the execution of the process.

**Question 2:**

The base-and-bounds approach for memory allocation achieves memory protection be using registers to contain the process to valid places in memory. The process is loaded into the “base” register and its size is loaded into the “bound” register. Virtual addresses are added to the “base” register to generate the physical address. This address is then checked against the bounds register t check whether the process is accessing a valid or invalid memory location

**Question 3:**



Virtual address 260 points to 58742 in physical memory

Virtual address 789 points to 57501 in physical memory

Virtual address 600 points to an invalid piece of memory in physical memory

**Question 4:**

Processes: 96,44,22,78

First-Fit: The first process consumes 96 KB from Space 2 (107-96=11), the second process consumes 44 from Space 4 (135-44=91). The third is entered into Space 3 (33-22=11), and the final process is entered into Space 4 so (91-78=13)

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | 21 | Space 1 |
|  |  |
|  |  |  |
|  | p1 |  |
|  | 107 | Space 2 |
|  |  |
|  | 11 |  |
|  | p3 |  |
|  | 33 | Space 3 |
|  |  |
|  |  |  |
|  | p2 |  |
|  | 135 | Space 4 |
|  | p4 91 |
|  | 13 |  |
|  |  |  |
|  | 59 | Space 5 |
|  |  |
|  |  |  |

Next-fit: The first process consumes 96 KB from Space 2 (107-96=11), the second process consumes 44 from Space 4 (135-44=91). The third is entered into Space 5 (33-22=11), and the final process is entered into Space 4 so (91-78=13)

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | 21 | Space 1 |
|  |  |
|  |  |  |
|  | p1 |  |
|  | 107 | Space 2 |
|  |  |
|  | 11 |  |
|  |  |  |
|  | 33 | Space 3 |
|  |  |
|  |  |  |
|  | p2 |  |
|  | 135 | Space 4 |
|  | p4 91 |
|  | 13 |  |
|  | p3 |  |
|  | 59 | Space 5 |
|  | 37 |
|  |  |  |

Best-fit: The first process consumes 96 KB from Space 2 (107-96=11), the second process consumes 44 from Space 5 (59-44=91). The third is entered into Space 3 (59-22=11), and the final process is entered into Space 4 so (135-78=57)

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | 21 | Space 1 |
|  |  |
|  |  |  |
|  | p1 |  |
|  | 107 | Space 2 |
|  |  |
|  | 11 |  |
|  | p3 |  |
|  | 33 | Space 3 |
|  |  |
|  | 11 |  |
|  | p4 |  |
|  | 135 | Space 4 |
|  |  |
|  | 57 |  |
|  | p2 |  |
|  | 59 | Space 5 |
|  |  |
|  | 15 |  |

Worst-fit: The first process consumes 96 KB from Space 4 (135-96=39), the second process consumes 44 from Space 2 (107-44=63). The third is entered into Space 2 (63-22=41), and the final process can’t fit into memory

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 21 | Space 1 |  |  |
|  |  |  |
|  |  |  |  |
| p2 |  |  |  |
| 107 | Space 2 |  |  |
| p3 63 |  |  |
| 41 |  |  |  |
|  |  |  |  |
| 33 | Space 3 | p4 doesn’t fit | |
|  |  |  |
|  |  |  |  |
| p1 |  |  |  |
| 135 | Space 4 |  |  |
|  |  |  |
| 39 |  |  |  |
|  |  |  |  |
| 59 | Space 5 |  |  |
|  |  |  |
|  |  |  |  |

**Question 5:**

Paging does cause internal fragmentation and we can reduce this by using segmentation inside of the “Page Table” so that a whole page is not wasted when a very small amount of data is used for the entire page. Paging doesn’t cause external fragmentation because the memory is evenly cut throughout so there is no way for a part of memory to not be mapped to a part of a page.

**Question 6:**

Va0x3b96 = 0011 1|100 10|01 011|0

7->7916 = 0111 1001 Valid Memory Access

The Answer is: 1100 1101 1100 1010

c b 9 6

Va0x046e=0000 0|100 01|10 111|0

0->7216 = 0111 0010 Valid Memory Access

The Answer is: 1001 0100 0110 1110

9 4 6 e

VA0x2580=0010 0|110 10|00 000|0

4 is an invalid memory access

**Question 7:**

EAT = (25+200) \* .8 + (25+2\*200) \* .2 = 180 + 25 = 265ns

Where a = .8

**Question 8:**

Page table is located at memory space 30.

VA1 = 0x7a02 = 0|111 10|10 000|0 0010

30 16 2

30->a3= 1010 0011

35->ae = 1010 1110

46-> 13 = 0001 0011

The Answer is: Address: 010111000010 value: 0011

VA2 = 0x57f0= 0|101 01|11 111|1 0000

21 31 16

30->e06 = 1110 0000

96->7f= 0111 1111 Invalid memory address

**Question 9:**

We can utilize the strengths of both methods by first using paging to (almost) completely get rid of external fragmentation. Then, from that point we can minimize internal fragmentation by cutting the page table into segments so that rather than using a whole page to have one reference to physical memory we can use a fraction of a page to do the same job.

**Question 10:**

Memory Access Time= 500ns

Swapping time = 50,000ns

TLB= 20ns

.8(500+20) + .05(500\*2 + 50,000) + .15(500\*2 +20) =

416 + 2550 + 150 = 3116ns

(Fix)

**Question 11:**

245015121405142

FIFO: 8 page faults

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| FIFO: | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 |
|  |  | 4 | 4 | 4 | 4 | 2 | 2 | 2 |
|  |  |  | 5 | 5 | 5 | 5 | 4 | 4 |
|  |  |  |  | 0 | 0 | 0 | 0 | 5 |

OPT: 7 page faults

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| OPT: | 2 | 2 | 2 | 2 | 2 | 0 | 0 |
|  |  | 4 | 4 | 4 | 4 | 4 | 2 |
|  |  |  | 5 | 5 | 5 | 5 | 5 |
|  |  |  |  | 0 | 1 | 1 | 1 |

LRU: 10 page faults

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LRU: | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 |
|  |  | 4 | 4 | 4 | 4 | 2 | 2 | 2 | 5 | 5 |
|  |  |  | 5 | 5 | 5 | 5 | 5 | 0 | 0 | 2 |
|  |  |  |  | 0 | 0 | 0 | 4 | 4 | 4 | 4 |

Second-Chance: 11 page faults

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Second-Chance: | 21 | 21 | 21 | 21 | 11 | 11 | 11 | 11 | 10 | 51 | 51 | 51 | 50 |
|  |  | 41 | 41 | 41 | 40 | 40 | 21 | 21 | 20 | 20 | 11 | 11 | 10 |
|  |  |  | 51 | 51 | 50 | 51 | 51 | 50 | 01 | 01 | 01 | 01 | 21 |
|  |  |  |  | 01 | 00 | 00 | 00 | 41 | 41 | 40 | 40 | 41 | 40 |

**Question 12:**

Thrashing occurs when there are too many processes for the computer to be able to access the memory (page table) and run them fast enough. The result of this is the scheduler spends more time searching for the pages for a job or process rather than actually running the instruction, which becomes a battle where the processes are constantly fighting for their pages to be executed so their jobs can be finished.

To combat this we use locality or the “sliding door” approach where there is a set of processes that can run uninterrupted based on the last (exact amount determined by developer) processes that have to have been ran. Then after that the “door” is slid over and new processes get the privilege of running uninterrupted.