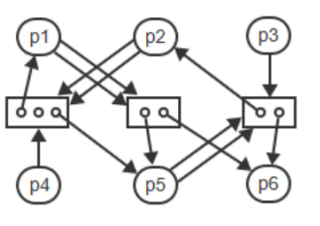
*Assignment 3*

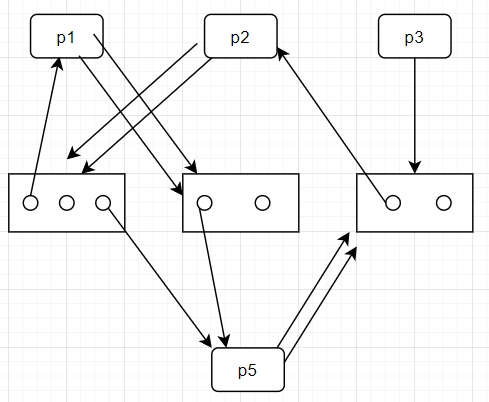
**Q1. Based on Exercise 5.2.1 – Graph Reduction**

**Using graph reduction, determine if the following graph contains a deadlock (2 points)**

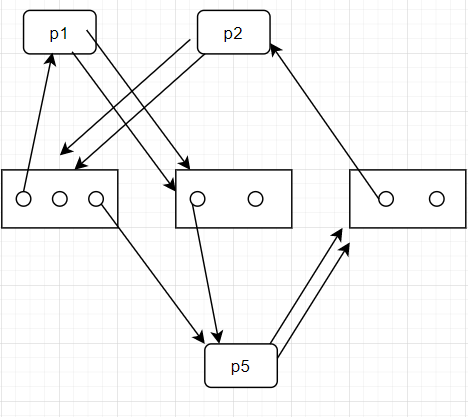


Using Graph reduction, we keep removing an unblocked process p, including all the requests and allocation edges connected to it.

p4 is not blocked because it needs one unit from Resource 1 which is available and thus request can be satisfied. P6 is not blocked because it has the required resource units from R2 and R3. Other processes remain blocked.



P3 is not blocked because it needs 1 unit from R3 which is now free. Other processes remain blocked.



P1 needs 2 units of R2 which is not possible because p5 holds one unit of two. P2 needs 2 units of R1 which is not possible because P5 holds 1 unit of R1 so not possible. P5 needs 2 units of R3 which is not possible because P2 holds one of two units of R3. Hence, these processes are in deadlock.

**Q2. Based on Exercise 6.1.5 – Comparison of Memory Allocation Algorithms**

**Assume a system’s memory contains 6 holes with the sizes: 190, 550, 220, 420, 650, and 110. A sequence of requests for 4 block is to be satisfied: A = 210, B = 430, C = 100, and D = 420.**

1. **Determine which holes are allocated to which request by each of the 4 allocation schemes. Further guidance: complete the following table to show which requests (A, B, C, D) are allocated to which hole(s) foreach of the four allocation schemes. Also give how the hole size has changed. For example, if A is allocated to the 650 hole for first-fit, put A (440) in the cell for First-Fit and 650. (4 points).**

(Hole size change to is written below the process)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Holes Before Allocation** | **190** | **550** | **220** | **420** | **650** | **110** |
| **After First Fit** | C  90 | A  210 |  | D  0 | B  220 |  |
| **After Next Fit** |  | A  210 |  | D  0 | B  220 | C  10 |
| **After Best Fit** |  | B  120 | A  10 | D  0 |  | C  10 |
| **After Worst Fit** |  | B  120 |  | C  320 | A  440 |  |

D does not get any allocation in the worst-case because no hole is left that has enough memory to accommodate D

**Q3. Based on Exercise 6.4.3 Space Trade-Offs Between Paging & Segmentation**

**Three functions, each of length 600 words, are linked together into one process and loaded into memory. Consider four possible combinations of paging and segmentation:**

1. **Paging (no segmentation): page size: 1024 words, page table occupies 1 page.**
2. **Segmentation (no paging): segment table size: 1024 words.**
3. **Segmentation with paging (each function becomes a separate segment): page and segment size: 1024, page and segment tables occupy 1 page each.**
4. **Two-level paging (page table is paged): page size: 1024, all page tables occupy 1 page each.**
5. **For each system, determine the total amount of occupied memory space, including all page or segment tables. (2 points)**

* *Paging (no segmentation)   
  page size : 1024 words  
  page table occupies 1 page*

1 page table occupies 1\*1024 = 1024 words

Each function has 600 words, each function is allocated 1 page, so total 4 pages occupies 4\*1024 = 4096 words

So total occupied = 4096 + 1024 = 5120 words = 5120 bytes

* *Segmentation (no paging)  
  segment table size : 1024 words*

There is 1 page table occupying 1\*1024 = 1024 bytes

Now all 4 functions might combine into 1 segment or each function gets its own segment. Total space = 4\*600 = 2400 bytes

So total space occupied = 1024 + 2400 = 3424 bytes

* *Segmentation with paging   
  Page size : 1024  
  Segment size : 1024**page table : 1 page*

*segment table : 1 page*

There is 1 segment table occupying 1\*1024 = 1024 bytes

There is 1 page table for each function -> 4\*1024 = 4096 bytes

Each function occupies 1 page = 4\*1024 = 4096 bytes

Total = 1024 + 4096 + 4096 = 9216 bytes

* *Two-level paging   
  page size : 1024  
  page table : 1 page*

For first level page table -> 1\*1024 = 1024

Now 4 second level table = 4\*1024 = 4096

Each function occupies 1 page -> 4\*1024 = 4096

Total space = 1024 + 4096 + 4096 = 9216 bytes

1. **For each system, determine the amount of space wasted due to internal fragmentation. (2 points)**

* *Paging (no segmentation)   
  page size : 1024 words  
  page table occupies 1 page*

Each function is 600 words. So they need 4\*600 = 2400

Space occupied (each function taking one page) -> 4\*1024 = 4096

Wasted space = 4096 – 2400 = 1696 bytes

* *Segmentation (no paging)  
  segment table size : 1024 words*

Wasted space = page size\*page needed – size = 1024\*4 – 2400 = 1696 bytes

* *Segmentation with paging   
  Page size : 1024  
  Segment size : 1024**page table : 1 page  
  segment table : 1 page*

There is 1 page table for each function -> 4\*1024 = 4096 bytes  
 Required = 4\*600 = 2400

Each function occupies 1 page = 4\*1024 = 4096 bytes  
Required = 4\*600 = 2400

Wasted space = 4096 – 2400 + 4096 – 2400 = 3392 bytes

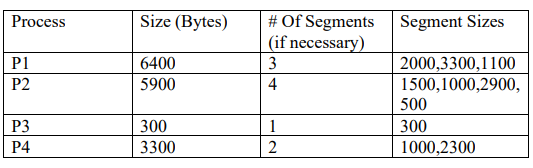
* *Two-level paging   
  page size : 1024  
  page table : 1 page*

Second level space occupied = 4096;  
Required = 2400

Wasted Space = 4096 – 2400 = 1696 bytes

**Q4. Paging & Segmentation – Chapter 6**

Consider the following list of processes and their associated sizes.



1. **Assuming that Paging is used, show the page table for each process under the following scenario: assume a page size of 1000 bytes and a main memory of size 32 frames with Frames 0, 1, 3, 5,8, 9, 15, 16, 17, 21, 24 and 31 currently utilized. Also assume the free frames are ordered by number in the free-space list. Each Page Table entry should contain Page# and Frame#. (2 points)**

Page size = 1000 bytes

Main memory size = 32 frames

Frames being utilized :

0, 1, 3, 5, 8, 9, 15, 16, 17, 21, 24, 31

Since each page has memory of 1000 bytes. P1 will need 7 pages (6\*1000 + 400). P2 will need 6 pages. P3 will need 1 page. P4 will need 4 pages. (The physical memory frames that are occupied are highlighted). Since, it is paging and not segmentation, we don’t really need the segments.

|  |
| --- |
| 0 |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |
| 11 |
| 12 |
| 13 |
| 14 |
| 15 |
| 16 |
| 17 |
| 18 |
| 19 |
| 20 |
| 21 |
| 22 |
| 23 |
| 24 |
| 25 |
| 26 |
| 27 |
| 28 |
| 29 |
| 30 |
| 31 |

Page tables

P1 PT1

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

|  |
| --- |
| 0 |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |

P2 PT2

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |

|  |
| --- |
| 0 |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |

P3 PT3

|  |
| --- |
|  |

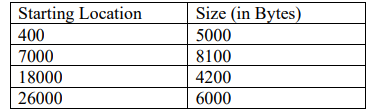
|  |
| --- |
| 0 |

P4 PT4

|  |
| --- |
|  |
|  |
|  |
|  |

|  |
| --- |
| 0 |
| 1 |
| 2 |
| 3 |

1. **Assuming that Segmentation is used, show the Segment Table for each process (if first-fit allocation is used) under the following scenario: given a main memory size of 32,000 bytes with the following areas currently available. Assume a segment is added at the beginning of the free hole (a segment of 2000 bytes would start at address 400 instead of finishing at 5000):**

****

**Each segment table entry should contain the segment #, size, and MM address. (2 points)**

According to the previous table, the segment sizes required for each process has been given. So using that and the second table that specifies, the range of segments empty :

|  |  |
| --- | --- |
| Starting location + size | Ending location |
| 400 + 5000 | 5400 |
| 7000 + 8100 | 15100 |
| 18000 + 4200 | 22200 |
| 26000 + 6000 | 32000 |

**First fit :**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Process | Explanation | Segment # | Size | MM address |
| P1 | 2000 – from 400 | Segment 1 | 2000 bytes | 400 – 2400 |
|  | 3300 – from 7000 | Segment 2 | 3300 bytes | 7000 – 10300 |
|  | 1100 – from 2400 | Segment 1 | 1100 bytes | 2400 – 3500 |
|  |  |  |  |  |
| P2 | 1500 – from 3500 | Segment 1 | 1500 bytes | 3500 – 5000 |
|  | 1000 – from 10300 | Segment 2 | 1000 bytes | 10300 – 11300 |
|  | 2900 – from 11300 | Segment 2 | 2900 bytes | 11300 – 14200 |
|  | 500 – from 14200 | Segment 2 | 500 bytes | 14200 – 14700 |
|  |  |  |  |  |
| P3 | 300 – from 5000 | Segment 1 | 300 bytes | 5000 – 5300 |
|  |  |  |  |  |
| P4 | 1000 – from 18000 | Segment 2 | 1000 bytes | 18000 – 19000 |
|  | 2300 – from 19000 | Segment 3 | 2300 bytes | 19000 - 21300 |
|  |  |  |  |  |

1. **Repeat Part (b) assuming best-fit allocation is used. (2 points)**

**Best fit :**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Process | Explanation | Segment # | Size | MM address |
| P1 | 2000 – from 18000 | Segment 3 | 2000 bytes | 18000 - 20000 |
|  | 3300 – from 400 | Segment 1 | 3300 bytes | 400 – 3700 |
|  | 1100 – from 20000 | Segment 3 | 1100 bytes | 20000 - 21100 |
|  |  |  |  |  |
| P2 | 1500 – from 26000 | Segment 4 | 1500 bytes | 26000 – 27500 |
|  | 1000 – from 21100 | Segment 3 | 1000 bytes | 21100 – 22100 |
|  | 2900 – from 27500 | Segment 4 | 2900 bytes | 27500 – 30400 |
|  | 500 – from 3700 | Segment 1 | 500 bytes | 3700 – 4200 |
|  |  |  |  |  |
| P3 | 300 – from 4200 | Segment 1 | 300 bytes | 4200 – 4500 |
|  |  |  |  |  |
| P4 | 1000 – from 30400 | Segment 4 | 1000 bytes | 30400 – 31400 |
|  | 2300 – from 7000 | Segment 2 | 2300 bytes | 7000 – 9300 |
|  |  |  |  |  |

1. **Assuming that Paged-Segmentation is used, show the Segment and Pages Tables for each process using the same scenario for main memory as described in Part (a). The Segment Tables entries for this part should contain Segment#, Size, and Page Table id (start counting at 0). (2 points)**

*Segmentation with paging   
Page size : 1000  
Segment size : differs*

There is 1 page table for each function

Each function occupies 1 segment which has a page table – there can be multiple page tables depending on memory and address

The memory location for pages are used as per question 1

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Page table memory id | Number of pages | Size |
| P1 | 2, 7, 4, 6, 14, 13,25 | 7 | 6\*1000 + 1\*400 |
| P2 | 2, 7, 4, 6, 14, 13,25 | 6 | 5\*1000 + 1\*900 |
| P3 | 30 | 1 | 1\*300 |
| P4 | 27, 28, 29, 23 | 4 | 3\*1000 + 1\*300 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Process | Page Table ID | Page table memory id | Segment # | Size |
| P1 | 0 | 2, 7, 4, 6, 14, 13,25 | 1 | 6400 |
| P2 | 1 | 2, 7, 4, 6, 14, 13,25 | 2 | 5900 |
| P3 | 2 | 30 | 3 | 300 |
| P4 | 3 | 27, 28, 29, 23 | 4 | 3300 |

**Q5. Based on Exercise 7.2.1 – The Reference String**

1. **A process in a paged system accesses the following virtual addresses:10, 11, 104, 170, 73, 309, 185, 245, 246, 434, 458, 364. Derive the corresponding reference string if the page size is 100 words, 200 words, and 300 words. (2 points)**

Depending on the page, the reference string is decided. For instance, for page size 100, the o-99th addresses refer to the 0th page and so on. For page size 100, 0th page will be 0-199, 1st page 200, 399 and so on. For page size 300, 0-299 is 0th page. An easy way to determine the reference string is using a little math – virtual address/page size.

For the given addresses :

10, 11, 104, 170, 73, 309, 185, 245, 246, 434, 458, 364

*Page Size = 100*

0, 0, 1, 1, 0, 3, 1, 2, 2, 4, 4, 3

*Page Size = 200*

0, 0, 0, 0, 0, 1, 0, 1, 1, 2, 2, 1

*Page Size = 300*

0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1

**Q6.** - **Page Reference Strings – Chapter 7   
This question is not based on any exercise from the textbook.**

**Consider the following page reference string: 1 2 1 3 4 0 2 3 0 1 3 4 2 0 4 2 1 0.**

1. **How many page faults would occur for the Optimal, FIFO, and LRU algorithms in a three-frame system (all frames are initially empty). Make sure you show your work. (2 points)**

Page Reference Strings :

1 2 1 3 4 0 2 3 0 1 3 4 2 0 4 2 1 0

Three-frame system

Optimal : 9 Page Faults //wrong

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Time t | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| RS |  | 1 | 2 | 1 | 3 | 4 | 0 | 2 | 3 | 0 | 1 | 3 | 4 | 2 | 0 | 4 | 2 | 1 | 0 |
| Frame |  |  |  |  | 3 | 3 | 3 | 3 | 3 | 3 | 3 | ~~3~~ | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| Frame |  |  | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | ~~2~~ | 1 | 1 |
| Frame |  | 1 | 1 | 1 | ~~1~~ | ~~4~~ | 0 | 0 | 0 | ~~0~~ | 1 | 1 | 1 | ~~1~~ | 0 | 0 | 0 | 0 | 0 |
| Page Fault |  | \* | \* |  | \* | \* | \* |  |  |  | \* |  | \* |  | \* |  |  | \* |  |

At time 4, page 3 is referenced which causes page fault. 1 is the page referenced in the most possible distant future (0 and 2 are referenced before). At time 5, 4 is referenced which will replace. 3 is the most distant future. The small numbers in brackets represent at which page fault they were replaced.

FIFO : 12 Page Faults

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Time t | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| RS |  | 1 | 2 | 1 | 3 | 4 | 0 | 2 | 3 | 0 | 1 | 3 | 4 | 2 | 0 | 4 | 2 | 1 | 0 |
| Frame |  |  |  |  | 3 | 3 | ~~3~~ | 2 | 2 | 2 | 2 | ~~2~~ | 4 | 4 | 4 | 4 | ~~4~~ | 1 | 1 |
| Frame |  |  | 2 | 2 | 2 | ~~2~~ | 0 | 0 | 0 | ~~0~~ | 1 | 1 | 1 | ~~1~~ | 0 | 0 | 0 | 0 | 0 |
| Frame |  | 1 | 1 | 1 | ~~1~~ | 4 | 4 | ~~4~~ | 3 | 3 | 3 | 3 | ~~3~~ | 2 | 2 | 2 | 2 | 2 | 2 |
| Page Fault |  | \* | \* |  | \* | \* | \* | \* | \* |  | \* |  | \* | \* | \* |  |  | \* |  |

LRU : 13 Page Faults

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Time t | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| RS |  | 1 | 2 | 1 | 3 | 4 | 0 | 2 | 3 | 0 | 1 | 3 | 4 | 2 | 0 | 4 | 2 | 1 | 0 |
| Frame |  |  |  |  | 3 | 3 | ~~3~~ | 2 | 2 | ~~2~~ | 1 | 1 | ~~1~~ | 2 | 2 | 2 | 2 | 2 | 2 |
| Frame |  |  | 2 | 2 | ~~2~~ | 4 | 4 | ~~4~~ | 3 | 3 | 3 | 3 | 3 | ~~3~~ | 0 | 0 | ~~0~~ | 1 | 1 |
| Frame |  | 1 | 1 | 1 | 1 | ~~1~~ | 0 | 0 | 0 | 0 | 0 | ~~0~~ | 4 | 4 | 4 | 4 | 4 | ~~4~~ | 0 |
| Page Fault |  | \* | \* |  | \* | \* | \* | \* | \* |  | \* |  | \* | \* | \* |  |  | \* | \* |

**Q7. Based on Exercise 7.4.1 - Comparing Page Replacement Algorithms   
Physical memory is initially empty. The following reference string is processed: 0 1 4 0 2 3 0 1 0 2 3 4 2 3**

1. **Show which pages are resident in an optimal working set with d = 3. Indicate when page faults occur. Determine the average working set size. (2 points)**

d = 3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 0 | 1 | 4 | 0 | 2 | 3 | 0 | 1 | 0 | 2 | 3 | 4 | 2 | 3 |
| 1. |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 4 |
| 2. |  |  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 |
| 3. |  |  |  | 4 | 4 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| Page Fault |  | \* | \* | \* |  | \* | \* |  |  |  | \* |  | \* |  |  |

Number of Page Faults = 7

Average Fault = 7/14 = ½

Average Working Set = 1 – 7/14 = 1/2

1. **Show which pages are resident under the working set page replacement algorithm with d = 3. Indicate when page faults occur. Determine the average working set size. (2 points)**

d = 3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 0 | 1 | 4 | 0 | 2 | 3 | 0 | 1 | 0 | 2 | 3 | 4 | 2 | 3 |
| 1. |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 3 |
| 2. |  |  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3. |  |  |  | 4 | 4 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 4 | 4 | 4 |
| Page Fault |  | \* | \* | \* |  | \* | \* |  |  |  | \* |  | \* |  | \* |

Number of Page Faults = 8

Average Fault = 8/14 = 4/7

Average Working Set = 1 – 8/14 = 6/14 = 3/7

**Q8.** - **Based on Exercise 8.2.1 – Operations on File Directories**

1. **Given the file directory in the participation activity titled "A tree-structured directory hierarchy", show all changes to the directory after the following sequence of operations has been executed (5 points):**
2. **rename n N** - No path name is specified //wrong
3. **delete ../t –** Deleted D5
4. **create /x/T** – In D1
5. **change m**  -change the main directory to m
6. **move c /y** - f1 moves from c to Root (under y)

