University of Calgary

CPSC 457 - Principles of Operating Systems

Professor: Dr. Pavol Federl

Assignment #5

Memory and Paging

Evan Loughlin

Student ID: 00503393

Tutorial Section: T04

TA: Sina Keshvadi (AKA: John Cena)

April 19, 2018

Q1

Assume an OS has five memory partitions of 100KB, 500KB, 200KB, 300KB, and 600KB.

free 100 KB	P10 30 KB	free 500KB	P11 30KB	free 200KB	P12 30KB	free 300KB	p13 30KB	free 600KB

The OS needs to place 4 new processes in memory in the following order:

- P1 of 212KB
- P2 of 417KB
- P3 of 112KB
- P4 of 426KB

Draw the diagrams of the partitions after the OS has placed the processes using 4 different algorithms:

1. First-fit
2. Best-Fit
3. Worst-Fit
4. Next-Fit

The resulting diagrams must show the size of each partition, and the status of each partition, similar to the figure above. If a process cannot be placed, indicate that below the diagram. Start the placement algorithms from the first partition.

First-Fit

Definition: *The memory manager scans along the list of segments until it finds a hole that is big enough. The hole is then broken up into two pieces, one for the process and one for the unused memory, except in the statistically unlikely case of an exact fit. First fit is a fast algorithm because it searches as little as possible.*

free 100 KB	P10 30 KB	P1 212 KB	P3 112KB	free 176KB	P11 30KB	free 200KB	P12 30KB	free 300KB	p13 30KB	P2 412KB	free 188KB

P4 cannot be placed - no available slot large enough.

Best-Fit

Definition: *Best fit searches the entire list, from beginning to end, and takes the smallest hole that is adequate. Rather than breaking up a big hole that might be needed later, best fit tries to find a hole that is close to the actual size needed, to best match the request and the available holes. It is slower than first fit, and surprisingly worse as it tends to fill memory with tiny, useless holes.*

free	P10	P2	free	P11	P3	free	P12	P1	free	p13	P4	free
100	30	412	88	30	112	88	30	212	88	30	426	174

(All sizes in KB)

- P1 of 212KB - smallest satisfiable free slot is 300KB.
- P2 of 412KB - smallest satisfiable free slot is 500KB.
- P3 of 112KB - smallest satisfiable free slot is 200KB
- P4 of 426KB - smallest satisfiable free slot is 600KB

Worst-Fit

Definition: *Worst fit always takes the largest available hole, so that the remaining hole will be big enough to be useful. Also not very good.*

free	P10	P2	free	P11	free	P12	free	p13	P1	P3	free
100	30	412	88	30	200	30	300	30	212	112	268

(All sizes in KB).

- P1 of 212KB - largest satisfiable free slot is 600 KB.
- P2 of 412KB - largest satisfiable free slot is 500 KB.
- P3 of 112KB - largest satisfiable free slot is 380 KB
- P4 of 426KB - no available slot large enough

Next-Fit

Definition: *Works the same as First-Fit, except that it keeps track of where it is whenever it finds a suitable hole. The next time it is called to fill a hole, it starts searching the list from the place where it left off last time, instead of always at the beginning. Simulations by Bays (1977) have shown that it gives slightly worse performance than first fit.*

free	P10	P1	free	P11	free	P12	free	p13	P2	P3	free
100	30	212	288	30	200	30	300	30	412	112	76

(All sizes in KB).

- P1 of 212KB - next available slot is 500KB
- P2 of 412KB - next available slot is 600KB (remainder = 188KB)
- P3 of 112KB - next available slot is the remainder 188KB.
- P4 of 426KB - no available slot large enough.

Q2

Consider a system with 1KB (1024 bytes) page size. What are the page numbers and offsets for the following addresses?

Address	Page Number	Offset
2375	2	327
19366	18	934
30000	29	304
256	0	256
16385	16	1

Page 0: 0 to 1023. Page 1: 1024 to 2047. Page 2: 2048 to 3071... Etc.

- Address 2375 will fall into page 3, because 2375/1024 = 2.3. Offset = 2375 mod 1024 = 327
- Address 19366 / 1024 = 18.9, so falls onto page 18. Offset = 19366 mod 1024 = 934.
- Address 30000 / 1024 = 29.2. Offset = 30000 mod 1024 = 304
- etc...

Q3

Consider a system with a 32-bit logical address space and 4KB page size. The system supports up to 512MB of physical memory. How many entries are there in each of the following?

Conventional Single-Level Page Table

For a 32 bit logical address space, and 4KB page size, the offset needs to be capable of specifying 4KB (4096 bytes), which requires 12 bits. This leaves 20 bits for specifying page number. This allows for 2^{20} pages, or 1048576 pages, (1048576×4096 = 4GB of Virtual Memory) assuming that all 20 bits are there for page number. However, if we assume 1 bit for Caching/Disables, 1 bit for Referenced, 1 bit for Modified, 1 bit for Protection, and 1 bit for Present/Absent, this leaves 15 bits remaining for page number. 2^{15} = 32768 page numbers (128 MB virtual memory).

Inverted Page Table

In an inverted page table, one entry per page frame is stored directly on physical memory. The entry stores the corresponding page number, along with the process ID. For 512MB of RAM, and 4KB page sizes, then there are $524288KB/4096KB = 128$ page entries.

Pros of inverted page tables: they save a lot of space, at least when the virtual address space is much larger than the physical memory.
Cons of inverted page tables: Virtual-to-physical translation becomes much harder. Each memory reference requires a full search of page tables.

Q4

Consider a system where a direct memory reference takes 200ns.

If we add a single-level page table stored in memory to this system, how much time would it take to locate and reference a page in memory?

With this system, a virtual-to-physical mapping must be done on every memory reference. In order to complete a memory reference, therefore, this translation must first be completed. No information is given in this problem regarding the speed of this translation, so the following assumptions will be made:

- The design is of a single page table consisting of an array of fast hardware registers, with one entry for each virtual page, indexed by virtual page number. When a process is started up, the operating system loads the registers with the process' page table, taken from a copy kept in main memory. During process execution, no more memory references are needed for the page table. The advantages of this system are that no memory references are required during mapping.

With this system, assuming that the memory address translation speed is negligible, the time required to locate and reference a page in memory is roughly same as a direct memory reference (200ns).

One the other extreme, the following system might exist:

- A page table can be entirely in main memory. All the hardware needs then is a single register that points to the start of the page table. This design allows the virtual-to-physical map to be changed at a page fault by reloading one register. This has the disadvantage of requiring one or more memory references to read page table entries during the execution of each instruction, making it very slow.

  In this case, assuming that 2 memory references are required to locate and reference a page in memory, the total time would take 400ns.

If we also add a TLB, and 75% of all page-table references are found in the TLB, what is the effective access time? Assume that searching TLB takes 10ns.

The Translation Lookaside Buffer (TLB) is a widely implemented scheme for speeding up paging and handling large virtual address spaces. For this example, we will assume the second system (whereby page faultes are required, and each page table reference requires 2 direct memory lookups, for a total of 400ns).

On average, for every 4 page-table references, 3 will be found in the TLB, and 1 will not, needing the additional memory reference to find the page table. For the references found in the TLB, the time is 200ns + 10ns, and for the references not found in the TLB, the time is 200ns + 10ns + 200ns.

$$t_{lookup} = \frac{(200ns + 10ns) \times 3 + (200ns + 10ns + 200ns)}{4} = 260ns$$

This is a clear performance boost over the same system without a TLB. (260ns versus 400ns).

Q5

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.

Assume there are 3 frames in the physical memory and all frames are initially empty. Illustrate how pages are placed into the frames according to the LRU and the optimal replacement algorithms. How many page faults would occur for each algorithm?

Optimal Replacement Algorithm

The optimal replacement algorithm, which cannot actually be implemented (since the CPU would need to know exactly when each frame will be required next), works as follows:

- The frame to be removed is the one that is farthest away (in terms of cycles) from needing to be used again.

The algorithm can be visualized as follows:

State 1

Firstly, physical memory will be filled with the first three frames. This would be caused by 3 page faults, generated since the requested frames were not yet mapped into physical memory.

Physical Memory:

3
2
1

Next, the state of each of the frames is hypothetically known, so that the frame to be removed from memory is the one which won't be needed for the longest period of time.

Remaining frames: 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

Page #	Instructions until Next Call
1	3
2	2
3	9
4	infinity
5	4
6	5
7	10

The above table would be updated each state.
Page faults so far: 3

State 2

Next, frame 3 would be removed and replaced with frame 4.
Page faults so far: 4

Physical Memory: $\begin{array}{|c|}\hline 4 \\ \hline 2 \\ \hline 1 \\ \hline \end{array}$

Remaining frames: 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

State 3

Frame 2 already in physical memory, so no page fault required.
Page faults so far: 4

Physical Memory: $\begin{array}{|c|}\hline 4 \\ \hline 2 \\ \hline 1 \\ \hline \end{array}$

Remaining frames: 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

State 4

Frame 1 already in physical memory, so no page fault required.
Page faults so far: 4

Physical Memory: $\begin{array}{|c|}\hline 4 \\ \hline 2 \\ \hline 1 \\ \hline \end{array}$

Remaining frames: 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

State 5

page fault: Frame 5 replaces frame 4.
Page faults so far: 5

Physical Memory: $\begin{array}{|c|}\hline 5 \\ \hline 2 \\ \hline 1 \\ \hline \end{array}$

Remaining frames: 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

State 6

page fault: Frame 6 replaces frame 5.
Page faults so far: 6

$$\text{Physical Memory: } \begin{array}{|c|}\hline 6 \\ \hline 2 \\ \hline 1 \\ \hline \end{array}$$

Remaining frames: 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

States 7 thru 9

No page fault (for 2 1 2). Frames already in physical memory.
Page faults so far: 6

$$\text{Physical Memory: } \begin{array}{|c|}\hline 6 \\ \hline 2 \\ \hline 1 \\ \hline \end{array}$$

Remaining frames: 3, 7, 6, 3, 2, 1, 2, 3, 6.

State 10

Page Fault: Frame 1 replaced with frame 3.
Page faults so far: 7

$$\text{Physical Memory: } \begin{array}{|c|}\hline 6 \\ \hline 2 \\ \hline 3 \\ \hline \end{array}$$

Remaining frames: 7, 6, 3, 2, 1, 2, 3, 6.

State 11

Page Fault: Frame 2 replaced with frame 7.
Page faults so far: 8

$$\text{Physical Memory: } \begin{array}{|c|}\hline 6 \\ \hline 7 \\ \hline 3 \\ \hline \end{array}$$

Remaining frames: 6, 3, 2, 1, 2, 3, 6.

State 12

No page fault required. Frame 6 already in physical memory.
Page faults so far: 8

Physical Memory: $\begin{array}{|c|}\hline 6 \\ \hline 7 \\ \hline 3 \\ \hline \end{array}$

Remaining frames: 3, 2, 1, 2, 3, 6.

State 13

No page fault required. Frame 3 already in physical memory.
Page faults so far: 8

Physical Memory: $\begin{array}{|c|}\hline 6 \\ \hline 7 \\ \hline 3 \\ \hline \end{array}$

Remaining frames: 2, 1, 2, 3, 6.

State 14

Page Fault: Frame 7 replaced with Frame 2
Page faults so far: 9

Physical Memory: $\begin{array}{|c|}\hline 6 \\ \hline 2 \\ \hline 3 \\ \hline \end{array}$

Remaining frames: 1, 2, 3, 6.

State 15

Page Fault: Frame 6 replaced with Frame 1
Page faults so far: 10

Physical Memory: $\begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline 3 \\ \hline \end{array}$

Remaining frames: 2, 3, 6.

State 16

No page fault required.
Page faults so far: 10

$$\text{Physical Memory: } \begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline 3 \\ \hline \end{array}$$

Remaining frames: 3, 6.

State 17

No page fault required.
Page faults so far: 10

$$\text{Physical Memory: } \begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline 3 \\ \hline \end{array}$$

Remaining frames: 6.

State 18

page fault: tie: first frame found replaced with Frame 6.
Page faults so far: 11

$$\text{Physical Memory: } \begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline 6 \\ \hline \end{array}$$

Remaining frames: NULL.

Number of Page Faults

11. This includes the 3 page faults cased at the beginning when physical memory was empty.

LRU Algorithm

This algorithm throws out frames which have been least recently used. Thus, it's necessary to keep track of the number of instructions that have occurred since a frame's last access. This can be completed using a linked list.

State 1

Firstly, physical memory will be filled with the first three frames. This would be caused by 3 page faults, generated since the requested frames were not yet mapped into physical memory.

Physical Memory:

3
2
1

Remaining frames: 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.
Last Used: 3, 2, 1
Page Faults: 3

State 2

Page Fault: Replace 1 with 4.

Physical Memory:

3
2
4

Remaining frames: 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.
Last Used: 4, 3, 2, 1
Page Faults: 4

State 3

No page fault required. (Frame 2 in memory).

Physical Memory:

3
2
4

Remaining frames: 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.
Last Used: 2, 4, 3, 1
Page Faults: 4

State 4

page fault: Replace Frame 3 with 1

Physical Memory:

1
2
4

Remaining frames: 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.
Last Used: 1, 2, 4, 3
Page Faults: 5

State 5

page fault: Replace Frame 4 with 5

Physical Memory: $\begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline 5 \\ \hline \end{array}$

Remaining frames: 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.
Last Used: 5, 1, 2, 4, 3
Page Faults: 6

State 6

page fault: Replace Frame 2 with 6

Physical Memory: $\begin{array}{|c|}\hline 1 \\ \hline 6 \\ \hline 5 \\ \hline \end{array}$

Remaining frames: 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.
Last Used: 6, 5, 1, 2, 4, 3
Page Faults: 7

State 7

page fault: Replace Frame 1 with 2

Physical Memory: $\begin{array}{|c|}\hline 2 \\ \hline 6 \\ \hline 5 \\ \hline \end{array}$

Remaining frames: 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.
Last Used: 2, 6, 5, 1, 4, 3
Page Faults: 8

State 8

page fault: Replace Frame 5 with 1

Physical Memory: $\begin{array}{|c|}\hline 2 \\ \hline 6 \\ \hline 1 \\ \hline \end{array}$

Remaining frames: 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.
Last Used: 1, 2, 6, 5, 4, 3
Page Faults: 9

State 9

No page fault.

	2
Physical Memory:	6
	1

Remaining frames: 3, 7, 6, 3, 2, 1, 2, 3, 6.
Last Used: 2, 1, 6, 5, 4, 3
Page Faults: 9

State 10

Page Fault: Replace frame 6 with 3.

	2
Physical Memory:	3
	1

Remaining frames: 7, 6, 3, 2, 1, 2, 3, 6.
Last Used: 3, 2, 1, 6, 5, 4
Page Faults: 10

State 11

Page Fault: Replace frame 1 with 7.

	2
Physical Memory:	3
	7

Remaining frames: 6, 3, 2, 1, 2, 3, 6.
Last Used: 7, 3, 2, 1, 6, 5, 4
Page Faults: 11

State 12

Page Fault: Replace frame 2 with 6.

	6
Physical Memory:	3
	7

Remaining frames: 3, 2, 1, 2, 3, 6.
Last Used: 6, 7, 3, 2, 1, 5, 4
Page Faults: 12

State 13

No page fault.

Physical Memory: $\begin{array}{|c|}\hline 6\\ \hline 3\\ \hline 7\\ \hline\end{array}$

Remaining frames: 2, 1, 2, 3, 6.
Last Used: 3, 6, 7, 2, 1, 5, 4
Page Faults: 12

State 14

Page Fault: Replace frame 7 with 2.

Physical Memory: $\begin{array}{|c|}\hline 6\\ \hline 3\\ \hline 2\\ \hline\end{array}$

Remaining frames: 1, 2, 3, 6.
Last Used: 2, 3, 6, 7, 1, 5, 4
Page Faults: 13

State 15

Page Fault: Replace frame 6 with 1.

Physical Memory: $\begin{array}{|c|}\hline 1\\ \hline 3\\ \hline 2\\ \hline\end{array}$

Remaining frames: 2, 3, 6.
Last Used: 1, 2, 3, 6, 7, 5, 4
Page Faults: 14

State 16

No page fault.

Physical Memory: $\begin{array}{|c|}\hline 1\\ \hline 3\\ \hline 2\\ \hline\end{array}$

Remaining frames: 3, 6.
Last Used: 2, 1, 3, 6, 7, 5, 4
Page Faults: 14

State 17

No page fault.

	1
Physical Memory:	3
	2

Remaining frames: 6.
Last Used: 3, 2, 1, 6, 7, 5, 4
Page Faults: 14

State 18

page fault: Replace frame 1 with 6.

	6
Physical Memory:	3
	2

Remaining frames: NULL
Last Used: 6, 3, 2, 1, 7, 5, 4
Page Faults: 15

Number of Page Faults

15.

Q6

Refer to pagesim.cpp

Q7

Refer to impl.cpp