COMP 2432 Operating Systems

Written Assignment 2

Submission to BlackBoard

Deadline: 14 April 2023

1. Page Replacement.

Consider the following reference string of length 25 generated by a process for which **3 memory frames** are allocated for a process with 7 pages, 0, 1, 2, 3, 4, 5, 6. *Indicate* the content of the frames after each page indicated by the reference string is accessed, including the *page fault*. *How many* page faults are generated from the reference string? Answer the question based on 3 different page replacement algorithms: (*a*) FIFO, (*b*) Optimal, and (*c*) LRU based on stack implementation. Show the content of the stack for (*c*). For Optimal, there could be multiple possible answers.

0 1 2 3 4 5 2 4 3 2 5 4 2 5 6 1 3 2 1 0 2 4 3 2 1

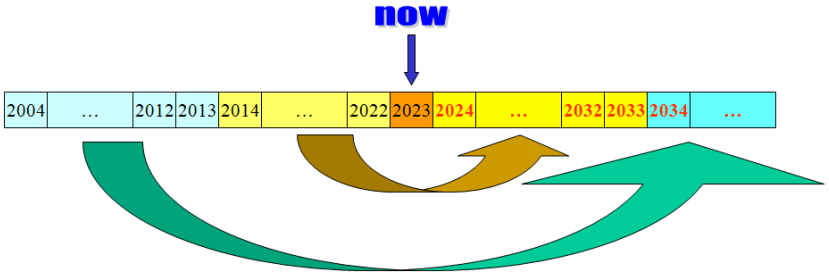
Repeat the question with **4 memory frames**.

(*d*) Consider the case with **3 memory frames**. One would expect that when a reference string is made longer through inserting an item in between (like **Question 5** in **Tutorial 10**), the performance could remain or get worsen. It is possible to *insert* one item in the reference string that will instead *reduce* the number of page faults for FIFO or LRU or **both**. Please indicate your revised reference string. Hint: consider inserting an item after one of the first 5 items.

(*e*) Consider the case with **3 memory frames**. One would expect that when a reference string is made shorter through deleting an item in between (like **Question 5** in **Tutorial 10**), the performance could remain or get better. Give a revised reference string by *deleting* one item from the reference string, which will *not decrease* the total number of page faults for FIFO and LRU.

2. Alternative Page Replacement.

Optimal looks into the future. LRU attempts to approximate Optimal by putting a mirror at the current moment and uses the history to predict the unknown future. Kevin notices that *history repeats itself* that they are bringing misery to all their villain bosses. He suggests that year 2023 should look like 2013; then year 2024 could be predicted from 2014; thus 2032 can be predicted from 2022; finally, 2033 can be predicted from 2023. This prediction is based on a **10-year cycle**. However, he does not know how to predict 2034, since 2024 is in future. Stuart suggests that since 2014 has been used to predict 2024, he suggests to use *older information* of 2004 to wrap around to predict 2034, and use 2005 to predict 2035, and so on. This is depicted in the diagram below, making use of a past 10-year period to predict a future 10-year period.



Bob comes and asks a question that he cannot use information of 1999, since all historical information before 2000 had been destroyed by the Millennium Bug that he himself released accidentally in one of their actions with the villains. Kevin replies that this prevents prediction beyond year 2043 and that year 2044 *cannot* be predicted. However, that does not matter, since Unix systems will face a problem in 2038 anyway. In general, prediction can be based on a period of P (P-year cycle) and $P = 10$ (10-year cycle) in this example. This observation is applied to approximate Optimal in this KBS ***algorithm***, named in honor of the three. Please watch out that the history used to predict the future will *change over time*.

Apply KBS algorithm on the reference string in **Question 1** for 3 memory frames, by selecting $P = 10$ and $P = 5$ respectively. *Indicate* the page faults. *Repeat* the question with 4 memory frames. Note that both LRU and KBS are similar in philosophy, in that LRU attempts to approximate Optimal by considering "*history is like a mirror*" and KBS attempts to approximate Optimal by considering "*history repeats itself*". How does KBS *compare* with LRU in terms of page fault performance in this case? Note that it is helpful as a *working step* to write down the "predicted future" for each page reference in determining the frame to be replaced.

3. Deadlock Avoidance.

(*a*) *Show* that the request Req_2 on the left side can be *granted* with respect to the resource allocation state with deadlock avoidance. *Give* a possible *safe sequence* after the pretended allocation and indicate the *number* of possible safe sequences.

(*b*) After the request by P_2 is satisfied, another process P_x apologizes that it makes a mistake in ***under-reporting*** its maximum need in one of the resource types *X* ***by two***. Due to communication noise in the network, only the type of the resource *X* under-reported is received by the operating system, and the identity of P_x is corrupted. Furthermore, P_1 declares to the operating system that it has not under-reported any of its need. With the partial information received, the operating system can still mange to conclude that the revised resource allocation state upon adjusting for this under-reporting ***remains safe***. *Determine* the possible resource type *X*. *Explain* how you arrive at your answer.

(*c*) After a while, P_x comes back and reports to the operating system that its original report on maximum need was correct and that the mistake of under-reporting is a ***false alarm***. Back comes another report by process P_y apologizing that it under-reports its maximum need on two different resources Y_1 and Y_2 ***by one***. However, although the information about *y* and Y_1 are received correctly, the information about Y_2 is lost due to noise. Based on the information from (*b*), we know that P_y is not P_1. Knowing the identity of P_y and one of the under-reported need on resource Y_1, the operating system has to reluctantly conclude that the revised resource allocation state upon adjusting for the under-reporting ***becomes unsafe*** regardless what Y_2 is. *Determine* the possible identities of P_y and Y_1. *Explain* how you arrive at your answer.

Q3	Allocation				Max			
	A	*B*	*C*	*D*	*A*	*B*	*C*	*D*
P_0	1	0	1	2	2	2	2	2
P_1	2	1	0	1	3	1	1	1
P_2	1	2	0	1	2	4	3	2
P_3	2	1	1	0	3	2	1	2
P_4	1	1	1	2	2	3	2	3
P_5	2	0	2	2	3	1	2	2
Avail	1	1	1	0				
Req_2	0	1	0	0				

Q4	Allocation				Request			
	A	*B*	*C*	*D*	*A*	*B*	*C*	*D*
P_0	1	0	1	1	0	1	1	2
P_1	2	0	0	1	1	1	1	1
P_2	1	1	0	0	1	0	1	0
P_3	2	0	1	1	1	2	1	2
P_4	1	1	1	1	0	1	1	0
P_5	2	2	2	1	1	0	1	2
Avail	1	0	1	0				

4. Deadlock Detection.

(*a*) *Show* that the resource allocation state on the right side is not involved in a deadlock. *Indicate* the number of possible completion sequences.

(*b*) Process P_x reports a careless mistake that there should be two more *X* in its updated request than its original request. Knowing *x*, the operating system regrets to announce that the system is definitely in a ***deadlocked state***, no matter what *X* is. *Determine* the identity of P_x and *explain* your answer.

(*c*) P_x apologizes for its false alarm and that its original request is correct. Now another process P_y reports to the operating system that it has also made mistakes in its request: there should be one more Y_1 and one more Y_2 ($Y_1 \neq Y_2$) in its updated request than its original request. However, the information of Y_1 and Y_2 are lost in communication due to noise. For certain *y*, the operating system can be confident that there is no deadlock in the updated system state, and for some other *y*, there is unfortunately definitely a deadlock, no matter what Y_1 and Y_2 are. However, for this particular value of *y*, it remains ***uncertain*** whether there is a deadlock or not. Upon clarification, P_y confirms that Y_2 is *D*. With this new piece of information, the operating system declares that there is a deadlock, regardless of what Y_1 is. *Determine* the identity of P_y and *explain* your answer.