COMP 2432 Operating Systems
Written Assignment 2
Submission to BlackBoard

Deadline: 07 April 2024

1. **Page Replacement.**

   Consider the following reference string of length 24 generated by a process for which **3 memory frames** are allocated for a process with 9 pages, 0, 1, 2, 3, 4, 5, 6, 7, 8. *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.

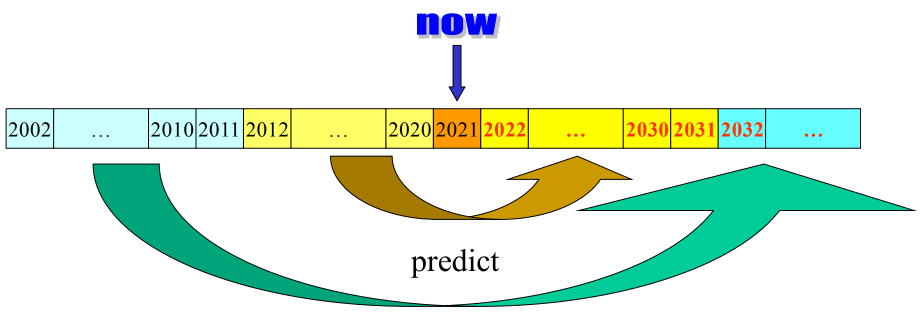
   0 1 2 3 4 5 2 4 3 2 0 4 8 1 0 2 4 5 1 2 4 3 2 0

   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 4** in **Tutorial 10**), the performance could remain or get worsen. Is it possible to *insert* one item in the reference string that will instead reduce the number of page faults for FIFO? If possible, indicate your reference string. If not possible, say so. Is it possible to *insert* one item in the reference string that will instead reduce the number of page faults for LRU? If possible, indicate your reference string. Is it possible to *insert* one item in the reference string that will instead reduce the number of page faults for **both** FIFO for LRU? If possible, indicate your reference string. If not possible, say so.

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 2021 should look like year 2011; then year 2022 could be predicted from year 2012; thus 2030 can be predicted from 2020; finally, 2031 can be predicted from 2021. This prediction is based on a 10-year cycle. However, he does not know how to predict 2032, since 2022 is in future. Stuart suggests that since 2012 has been used to predict 2022, he suggests to use *older information* of 2002 to wrap around to predict 2032, and use 2003 to predict 2033, and so on. This is depicted in the diagram below.



   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 2041 and that year 2042 *cannot* be predicted. But 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 $P = 8$ 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_0 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_0 is satisfied, another process P_x apologizes that it makes a mistake in ***under-reporting*** its maximum need in one of the resource types ***by one***. Due to noise in the network communication, only the last two bits of the identity of x is correctly received by the operating system, and the type of the resource under-reported is corrupted. However, the operating system can still mange to conclude that the corrected resource allocation state upon adjusting for this under-reporting ***remains safe***. *Determine* the possible identities of 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 distinct process P_y ($y \neq x$) apologizing that it under-reports its maximum need in one of the resource types Y by one. Knowing Y, the operating system is happy to conclude that the corrected resource allocation state upon adjusting for this under-reporting ***remains safe***. *Determine* the possible identities of Y. *Explain* how you arrive at your answer.

Q3	Allocation				Max			
	A	*B*	*C*	*D*	*A*	*B*	*C*	*D*
P_0	2	0	1	1	4	3	3	4
P_1	1	1	2	1	3	4	2	3
P_2	1	0	0	2	2	0	1	2
P_3	2	4	3	2	4	4	3	4
P_4	0	0	1	0	1	0	1	1
P_5	1	0	1	1	2	3	2	2
Avail	1	1	1	1				
Req_0	0	0	1	0				

Q4	Allocation				Request			
	A	*B*	*C*	*D*	*A*	*B*	*C*	*D*
P_0	2	0	1	1	1	0	1	1
P_1	1	2	1	0	1	0	0	1
P_2	1	0	0	2	0	1	2	3
P_3	0	1	0	0	2	3	2	2
P_4	1	0	1	1	1	2	3	4
P_5	2	0	1	1	2	4	3	2
Avail	1	1	0	1				

4. **Deadlock Detection.**

(*a*) *Show* that the resource allocation state on the right side is involved in a deadlock. *Indicate* the set of processes involved in the deadlock.

(*b*) Suppose that a process P_x reports to the oprating system that it is now willing to reduce its request for resource type X (A or B or C or D) by one instance. With this good news to the operating system, the system is now not deadlocked. *Determine* what x is and what X is.

(*c*) After the request by P_x for x is reduced by 1 as in (*b*), the system is not deadlocked. Now another process P_y makes a new request for an instance of A. Knowing y, the operating system has to declare that the system now enters a deadlocked state. *Determine* the identity of y and *explain* your answer.