Université Paris Dauphine Tunis

UNIVERSITÉ PARIS DAUPHINE

Bases de données avancées

Filière : **Master 2 Big Data**

Buffer Replacement Policies

Réalisé par

Hamza BEN MARZOUK

Année Universitaire : 2017/2018

Abstract

Un algorithme de remplacement de pages est un algorithme qui décide quel page du buffer remplacer et écrire sur le disque lorsque le buffer est saturé et a besoin de faire une reallocation de pages. Le remplacement de pages influe directement sur les performances du système vu que qu'il agit sur les opérations d'accès en lecture et écriture sur la mémoire disque[1].

Introduction

Dans le cadre du programme d'études en Master 2 Big Data à l'université Paris Dauphine Tunis, ce travail fait partie des projets réalisés au cours de l'étude des bases de données avancées. Ce projet nous amène à l'étude de différentes politiques de remplacement de cache utilisées par les Systèmes d'exploitation et les Systèmes de Gestion de Bases de Données, à savoir *FIFO : First In First Out*, *LRU :Least Recently used* et *Clock algorithm*. A travers ce document, nous expliquerons les démarches suivies pour la réalisation ainsi que les résultats de notre étude.

1 Étude théorique

Cette partie du rapport sera consacrée à la présentation théorique et l'explication des algorithmes utilisés.

1.1 FIFO : First In First Out

L'algorithme *FIFO* est un des algorithmes de remplacements les plus basiques et connus notamment en gestion de files d'attente. L'algorithme comme l'indique son nom se base sur le principe *Premier Arrivé, Premier Sorti* et utilise comme structure de données la structure *File*. Pour effectuer un remplacement de page, l'algorithme remplace toujours l'élément à la queue de la file correspondant au plus ancien élément introduit dans la file.

Le principal avantage de l'algorithme FIFO est sa facilité d'implementation et son utilisation réduite de mémoire puisqu'il ne requiert que la sauvegarde de l'index de la page à remplacer qui représente la page la plus anciennement utilisée.

Par contre, l'algorithme FIFO est en quelque sorte un algorithme sans mémoire et indépendant de l'utilisation réelle des données puisqu'il ne prend pas en compte l'utilisabilité des pages et leur fréquence d'appels d'où une dégradation des performances concrétisée par l'augmentation des nombres de pages non trouvées et un coût supplémentaire en lecture/écriture disque.

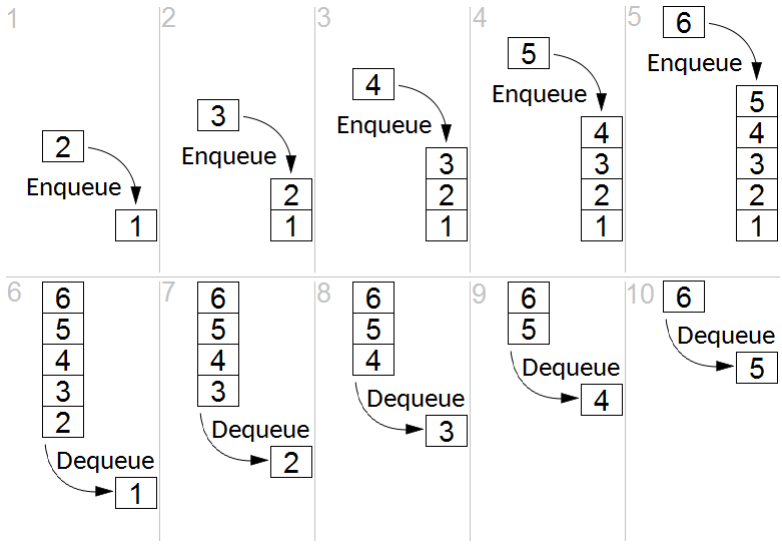

Figure 1 – Modelisation d'une file FIFO[2]

1.2 L.R.U : Least Recently Used

L'algorithme LRU : *Least Recently Used* ou Moins Récemment Utilisé, est un algorithme qui prend en considération l'utilisabilité des données et se base sur le principe de remplacement des pages dont l'utilisation est la moins récente. Moyennant un structure de file il sera facile d'implémenter un tel algorithme en triant les pages utilisées selon l'ordre d'utilisation et en mettant toujours en tete de file l'element le plus utilisé. Une fois la file saturée, le remplacement se fera en supprimant la page en queue de file et en décalant tous les éléments de la file pour mettre en tete la page récemment chargée. Une autre approche peut être adoptée pour implémenter cet algorithme et qui est d'enregistrer les occurrences d'appels de toutes les pages présentes dans le buffer avec un compteur incrémenté avec chaque appel et de remplacer celle ayant la valeur de compteur minimale.

Malgré sa simplicité d'implementation et ses bonnes performances, l'algorithme L.R.U reste assez coûteux en terme d'opérations notamment les opérations de tri de listes utilisées dans le buffer et en terme de mémoire pour avoir un suivi des appels de toutes les pages.

Pour notre projet, nous adopterons la premiere implementation de l'algorithme.

1.3 Algorithme Horloge : Clock

L'algorithme de remplacement Horloge ou *Clock* tout comme l'algorithme L.R.U est un algorithme qui prend en considération l'utilisation des données. En effet, cet algorithme se base sur une modélisation cyclique du buffer et l'utilisation d'un itérateur (ou pointeur) faisant référence au dernier element remplacé avec une pondération positive (bit R) des noeuds existants

et appelés plus qu'une fois. Une fois le buffer saturé, le pointeur va parcourir le cycle cherchant un emplacement non pondéré et en supprimant les pondération des noeuds parcourus.Une fois un noeud remplacé, le pointeur sera incrémenté de façon à offrir une seconde chance à ce noeud et la prochaine recherche commencera à partir du noeud suivant.

L'algorithme est une variante de l'algorithme "Second Chance" qui lui même est une variante de FIFO qui prend en considération l'utilisation des pages en triant les pages par ordre d'utilisation et en affectant un bit R correspondant à un flag indicant que la page concernée a été récemment utilisée(lecture/écriture)[3].

La figure 2 illustre le principe de fonctionnement de l'algorithme Clock pour le remplacement de pages.

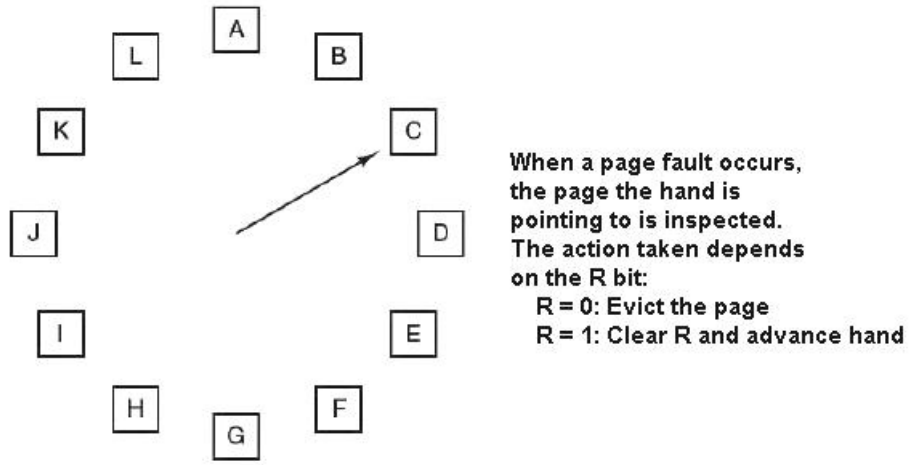

Figure 2. The clock page replacement algorithm.

Figure 2 – Fonctionnement de l'algorithme Clock [3]

2 Modélisation et Conception

Dans cette section nous présentons la phase de modélisation et de conception des éléments utilisés.

2.1 Modélisation

2.1.1 Page ou Frame

Une page ou Frame représente l'element unitaire et fondamentale pour les algorithmes de remplacement du buffer. Pour simplifier, nous considérons qu'une page est caractérisée par

son identifiant qui peut être son numéro, son R flag, et son pin count. Nous mettrons aussi l'hypothèse que les pages sont rapidement chargées et déchargées et leur pin count remis à 0.

Nous pourrons donc assimiler une page à un objet ayant 2 attributs : numéro et flag.

2.1.2 Buffer

Le *buffer* ou mémoire tampon, représente une mémoire intermédiaire pour mémoriser temporairement des données en cours de traitement[4]. Étant la capacité limitée du buffer, il est nécessaire d'adopter une politique de remplacement de pages pour assurer la continuité du traitement. C'est le buffer manager qui s'occupe de cette tache.

Pour modéliser le buffer, nous pouvons le considerer comme une série de cases mémoire de taille limitée. Puisqu'il s'agit d'un ensemble de pages (ou frames), pour faire abstraction nous considérerons le buffer comme une liste de pages que nous pourrons manipuler facilement.

2.2 Conception

Pour la conception de notre système, nous adopterons une modélisation orientée objet. Nous représentons par la figure 3 les classes utilisées et leurs liaisons par un diagramme de classes.

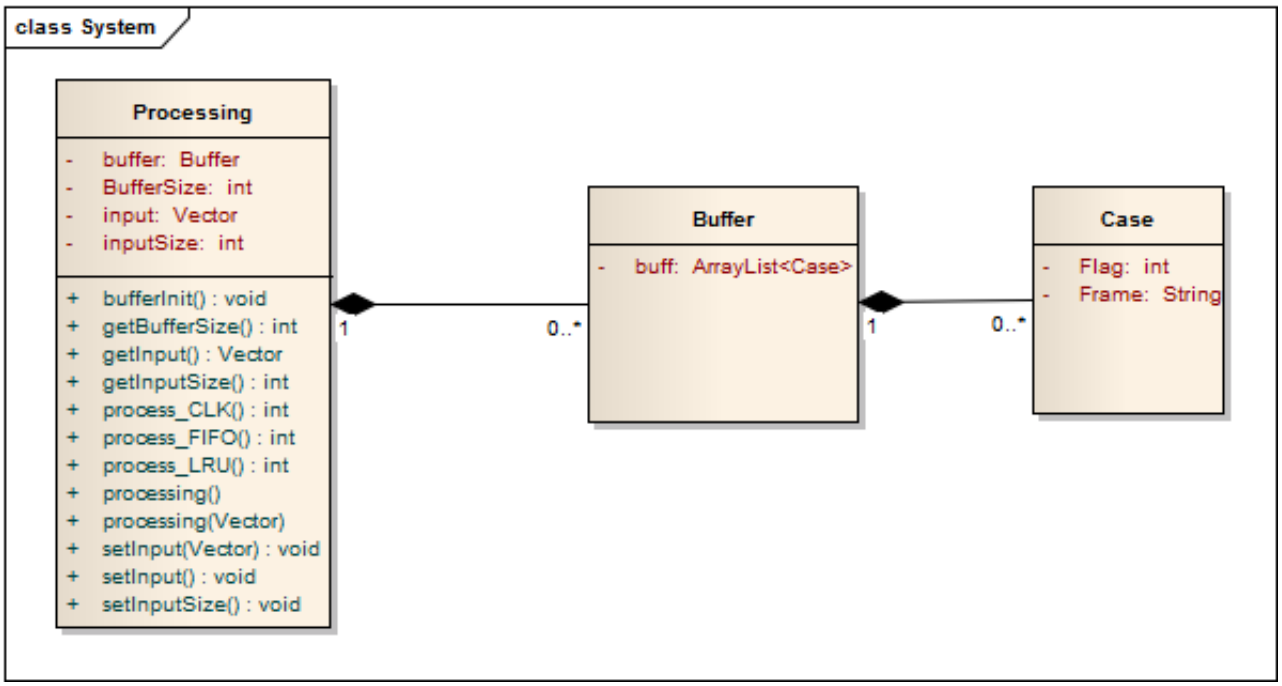

Figure 3 – Diagramme de classes du système

3 Implementation et résultats

3.1 Algorithme FIFO

Implementation

Pour l'implementation de l'algorithme FIFO, nous avons considéré le buffer comme une file d'attente. Lors de la saturation du buffer, la page à remplacer est simplement la page se trouvant à la queue de file référencée par une variable pointeur.

L'implementation de l'algorithme en langage Java et utilisant les classes définies précédemment est présentée par le code présent dans l'annexe.

Résultats d'execution

Pour l'execution et le test de l'algorithme nous utilisons deux suites d'entrées :

— la suite : 2,3,2,1,5,2,4,5,3,2,4,5,3,2,4,5,3,2,5,2

— la suite : 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

Nous avons rajouté des instructions de traçage dans le code pour pouvoir suivre l'execution des algorithmes.

Les résultats pour les 2 suites sont illustrés par les figures 4 et 5.

```
--------------------------------
( Frame number -1) ( Frame number -1) ( Frame number -1)  pointer on case 0
input is 2/ frame non existant
( Frame number 2) ( Frame number -1) ( Frame number -1)  pointer on case 1
input is 3/ frame non existant
( Frame number 2) ( Frame number 3) ( Frame number -1)  pointer on case 2
input is 2/ frame exists
( Frame number 2) ( Frame number 3) ( Frame number -1)  pointer on case 2
input is 1/ frame non existant
( Frame number 2) ( Frame number 3) ( Frame number 1)  pointer on case 0
input is 5/ frame non existant
( Frame number 5) ( Frame number 3) ( Frame number 1)  pointer on case 1
input is 2/ frame non existant
( Frame number 5) ( Frame number 2) ( Frame number 1)  pointer on case 2
input is 4/ frame non existant
( Frame number 5) ( Frame number 2) ( Frame number 4)  pointer on case 0
input is 5/ frame exists
( Frame number 5) ( Frame number 2) ( Frame number 4)  pointer on case 0
input is 3/ frame non existant
( Frame number 3) ( Frame number 2) ( Frame number 4)  pointer on case 1
input is 2/ frame exists
( Frame number 3) ( Frame number 2) ( Frame number 4)  pointer on case 1
input is 4/ frame exists
( Frame number 3) ( Frame number 2) ( Frame number 4)  pointer on case 1
input is 5/ frame non existant
( Frame number 3) ( Frame number 5) ( Frame number 4)  pointer on case 2
input is 3/ frame exists
( Frame number 3) ( Frame number 5) ( Frame number 4)  pointer on case 2
input is 2/ frame non existant
( Frame number 3) ( Frame number 5) ( Frame number 2)  pointer on case 0
input is 4/ frame non existant
( Frame number 4) ( Frame number 5) ( Frame number 2)  pointer on case 1
input is 5/ frame exists
( Frame number 4) ( Frame number 5) ( Frame number 2)  pointer on case 1
input is 3/ frame non existant
( Frame number 4) ( Frame number 3) ( Frame number 2)  pointer on case 2
input is 2/ frame exists
( Frame number 4) ( Frame number 3) ( Frame number 2)  pointer on case 2
input is 5/ frame non existant
( Frame number 4) ( Frame number 3) ( Frame number 5)  pointer on case 0
input is 2/ frame non existant
( Frame number 2) ( Frame number 3) ( Frame number 5)  pointer on case 1
end of processing fifo
( Frame number 2) ( Frame number 3) ( Frame number 5)  pointer on case 1
Fault Pages FIFO 13
```

Figure 4 – Trace d'execution Fifo suite 1

```
( Frame number -1) ( Frame number -1) ( Frame number -1)  pointer on case 0
input is 7/ frame non existant
( Frame number 7) ( Frame number -1) ( Frame number -1)  pointer on case 1
input is 0/ frame non existant
( Frame number 7) ( Frame number 0) ( Frame number -1)  pointer on case 2
input is 1/ frame non existant
( Frame number 7) ( Frame number 0) ( Frame number 1)  pointer on case 0
input is 2/ frame non existant
( Frame number 2) ( Frame number 0) ( Frame number 1)  pointer on case 1
input is 0/ frame exists
( Frame number 2) ( Frame number 0) ( Frame number 1)  pointer on case 1
input is 3/ frame non existant
( Frame number 2) ( Frame number 3) ( Frame number 1)  pointer on case 2
input is 0/ frame non existant
( Frame number 2) ( Frame number 3) ( Frame number 0)  pointer on case 0
input is 4/ frame non existant
( Frame number 4) ( Frame number 3) ( Frame number 0)  pointer on case 1
input is 2/ frame non existant
( Frame number 4) ( Frame number 2) ( Frame number 0)  pointer on case 2
input is 3/ frame non existant
( Frame number 4) ( Frame number 2) ( Frame number 3)  pointer on case 0
input is 0/ frame non existant
( Frame number 0) ( Frame number 2) ( Frame number 3)  pointer on case 1
input is 3/ frame exists
( Frame number 0) ( Frame number 2) ( Frame number 3)  pointer on case 1
input is 2/ frame exists
( Frame number 0) ( Frame number 2) ( Frame number 3)  pointer on case 1
input is 1/ frame non existant
( Frame number 0) ( Frame number 1) ( Frame number 3)  pointer on case 2
input is 2/ frame non existant
( Frame number 0) ( Frame number 1) ( Frame number 2)  pointer on case 0
input is 0/ frame exists
( Frame number 0) ( Frame number 1) ( Frame number 2)  pointer on case 0
input is 1/ frame exists
( Frame number 0) ( Frame number 1) ( Frame number 2)  pointer on case 0
input is 7/ frame non existant
( Frame number 7) ( Frame number 1) ( Frame number 2)  pointer on case 1
input is 0/ frame non existant
( Frame number 7) ( Frame number 0) ( Frame number 2)  pointer on case 2
input is 1/ frame non existant
( Frame number 7) ( Frame number 0) ( Frame number 1)  pointer on case 0
end of processing fifo
( Frame number 7) ( Frame number 0) ( Frame number 1)  pointer on case 0
Fault Pages FIFO 15
```

Figure 5 – Trace d'execution Fifo suite 2

3.2 L.R.U

Implementation

L'algorithme LRU est en quelque sorte une variante de l'algorithme FIFO prenant en considération l'utilisation des pages. En utilisant un modèle en file, nous pouvons considerer que

LRU se base sur le placement de la page la plus récente toujours en tete de file et en supprimant, en cas de saturation, toujours la page se trouvant à la fin de la file ce qui correspond bien à la page la moins récemment utilisée ou la plus anciennement utilisée.

Pour son implementation, nous nous sommes basés sur l'algorithme FIFO en rajoutant les instructions de tri de file par ancienneté. Le code de l'algorithme implémenté en Java est présent dans l'annexe 19.

Résultats d'execution

Pour les résultats, nous utilisons les mêmes suites d'entrées pour FIFO. Les traces d'execution sont présentés par les figures 6 et 7.

```
*********************************
( Frame number -1) ( Frame number -1) ( Frame number -1)  pointer on case 0
input 2( Frame number -1) ( Frame number -1) ( Frame number 2)  pointer on case 0
input 3( Frame number -1) ( Frame number 2) ( Frame number 3)  pointer on case 0
input 2( Frame number -1) ( Frame number 3) ( Frame number 2)  pointer on case 0
input 1( Frame number 3) ( Frame number 2) ( Frame number 1)  pointer on case 0
input 5( Frame number 2) ( Frame number 1) ( Frame number 5)  pointer on case 0
input 2( Frame number 1) ( Frame number 5) ( Frame number 2)  pointer on case 0
input 4( Frame number 5) ( Frame number 2) ( Frame number 4)  pointer on case 0
input 5( Frame number 2) ( Frame number 4) ( Frame number 5)  pointer on case 0
input 3( Frame number 4) ( Frame number 5) ( Frame number 3)  pointer on case 0
input 2( Frame number 5) ( Frame number 3) ( Frame number 2)  pointer on case 0
input 4( Frame number 3) ( Frame number 2) ( Frame number 4)  pointer on case 0
input 5( Frame number 2) ( Frame number 4) ( Frame number 5)  pointer on case 0
input 3( Frame number 4) ( Frame number 5) ( Frame number 3)  pointer on case 0
input 2( Frame number 5) ( Frame number 3) ( Frame number 2)  pointer on case 0
input 4( Frame number 3) ( Frame number 2) ( Frame number 4)  pointer on case 0
input 5( Frame number 2) ( Frame number 4) ( Frame number 5)  pointer on case 0
input 3( Frame number 4) ( Frame number 5) ( Frame number 3)  pointer on case 0
input 2( Frame number 5) ( Frame number 3) ( Frame number 2)  pointer on case 0
input 5( Frame number 3) ( Frame number 2) ( Frame number 5)  pointer on case 0
input 2( Frame number 3) ( Frame number 5) ( Frame number 2)  pointer on case 0
end of processing LRU
( Frame number 3) ( Frame number 5) ( Frame number 2) Fault Pages LRU 15
```

Figure 6 – Trace d'execution LRU suite 1

```
( Frame number -1) ( Frame number -1) ( Frame number -1)  pointer on case 0
input 7( Frame number -1) ( Frame number -1) ( Frame number 7)  pointer on case 0
input 0( Frame number -1) ( Frame number 7) ( Frame number 0)  pointer on case 0
input 1( Frame number 7) ( Frame number 0) ( Frame number 1)  pointer on case 0
input 2( Frame number 0) ( Frame number 1) ( Frame number 2)  pointer on case 0
input 0( Frame number 1) ( Frame number 2) ( Frame number 0)  pointer on case 0
input 3( Frame number 2) ( Frame number 0) ( Frame number 3)  pointer on case 0
input 0( Frame number 2) ( Frame number 3) ( Frame number 0)  pointer on case 0
input 4( Frame number 3) ( Frame number 0) ( Frame number 4)  pointer on case 0
input 2( Frame number 0) ( Frame number 4) ( Frame number 2)  pointer on case 0
input 3( Frame number 4) ( Frame number 2) ( Frame number 3)  pointer on case 0
input 0( Frame number 2) ( Frame number 3) ( Frame number 0)  pointer on case 0
input 3( Frame number 2) ( Frame number 0) ( Frame number 3)  pointer on case 0
input 2( Frame number 0) ( Frame number 3) ( Frame number 2)  pointer on case 0
input 1( Frame number 3) ( Frame number 2) ( Frame number 1)  pointer on case 0
input 2( Frame number 3) ( Frame number 1) ( Frame number 2)  pointer on case 0
input 0( Frame number 1) ( Frame number 2) ( Frame number 0)  pointer on case 0
input 1( Frame number 2) ( Frame number 0) ( Frame number 1)  pointer on case 0
input 7( Frame number 0) ( Frame number 1) ( Frame number 7)  pointer on case 0
input 0( Frame number 1) ( Frame number 7) ( Frame number 0)  pointer on case 0
input 1( Frame number 7) ( Frame number 0) ( Frame number 1)  pointer on case 0
end of processing LRU
( Frame number 7) ( Frame number 0) ( Frame number 1) Fault Pages LRU 12
```

Figure 7 – Trace d'execution LRU suite 2

3.3 Clock Algorithm

Implementation

L'algorithme Horloge ou Clock considère le buffer en forme circulaire en prenant en considération le Bit R qui donne une information sur l'utilisation d'une page donnée. Tout comme l'algorithme *Second Chance*, l'algorithme Clock donne à chaque page récemment introduite dans le buffer une seconde chance avant d'etre inspectée pour remplacement en cas de saturation du buffer. Pour la recherche de page à remplacer Clock parcoure le buffer cherchant une page avec bit R à 0, tout en mettant les bits R des autres pages à 0 en parcourant le buffer. Son implementation est un peu complexe par rapport aux autres algorithmes puisqu'il utilise un modèle circulaire du buffer et un pointeur.

Pour notre implementation, nous avons utilisé un second pointeur *"last"* permettant de referencer la page dernièrement introduite. Le code de l'algorithme implémenté en Java est présent dans l'annexe 21.

Résultats d'execution

Pour les résultats, nous utilisons les mêmes suites d'entrées pour FIFO et LRU. Les traces d'execution sont présentés par les figures 8 et 9.

```
(-1,0)(-1,0)(-1,0)pointer on case 0 last on case -1
input is 2/ frame non existant
(2,0)(-1,0)(-1,0)pointer on case 1 last on case 0
input is 3/ frame non existant
(2,0)(3,0)(-1,0)pointer on case 2 last on case 1
input is 2/ frame exists
(2,1)(3,0)(-1,0)pointer on case 2 last on case 1
input is 1/ frame non existant
(2,1)(3,0)(1,0)pointer on case 0 last on case 2
input is 5/ frame non existant
(2,0)(5,0)(1,0)pointer on case 2 last on case 1
input is 2/ frame exists
(2,1)(5,0)(1,0)pointer on case 2 last on case 1
input is 4/ frame non existant
(2,1)(5,0)(4,0)pointer on case 0 last on case 2
input is 5/ frame exists
(2,1)(5,1)(4,0)pointer on case 0 last on case 2
input is 3/ frame non existant
(2,0)(5,0)(3,0)pointer on case 0 last on case 2
input is 2/ frame exists
(2,1)(5,0)(3,0)pointer on case 0 last on case 2
input is 4/ frame non existant
(2,0)(4,0)(3,0)pointer on case 2 last on case 1
input is 5/ frame non existant
(2,0)(4,0)(5,0)pointer on case 0 last on case 2
input is 3/ frame non existant
(3,0)(4,0)(5,0)pointer on case 1 last on case 0
input is 2/ frame non existant
(3,0)(2,0)(5,0)pointer on case 2 last on case 1
input is 4/ frame non existant
(3,0)(2,0)(4,0)pointer on case 0 last on case 2
input is 5/ frame non existant
(5,0)(2,0)(4,0)pointer on case 1 last on case 0
input is 3/ frame non existant
(5,0)(3,0)(4,0)pointer on case 2 last on case 1
input is 2/ frame non existant
(5,0)(3,0)(2,0)pointer on case 0 last on case 2
input is 5/ frame exists
(5,1)(3,0)(2,0)pointer on case 0 last on case 2
input is 2/ frame exists
(5,1)(3,0)(2,1)pointer on case 0 last on case 2
end of CLOCK processing
(5,1)(3,0)(2,1)pointer on case 0 last on case 2
Fault Pages Clock 14
```

Figure 8 – Trace d'execution CLK suite 1

```
7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1
(-1,0)(-1,0)(-1,0)pointer on case 0 last on case -1
input is 7/ frame non existant
(7,0)(-1,0)(-1,0)pointer on case 1 last on case 0
input is 0/ frame non existant
(7,0)(0,0)(-1,0)pointer on case 2 last on case 1
input is 1/ frame non existant
(7,0)(0,0)(1,0)pointer on case 0 last on case 2
input is 2/ frame non existant
(2,0)(0,0)(1,0)pointer on case 1 last on case 0
input is 0/ frame exists
(2,0)(0,1)(1,0)pointer on case 1 last on case 0
input is 3/ frame non existant
(2,0)(0,0)(3,0)pointer on case 0 last on case 2
input is 0/ frame exists
(2,0)(0,1)(3,0)pointer on case 0 last on case 2
input is 4/ frame non existant
(4,0)(0,1)(3,0)pointer on case 1 last on case 0
input is 2/ frame non existant
(4,0)(0,0)(2,0)pointer on case 0 last on case 2
input is 3/ frame non existant
(3,0)(0,0)(2,0)pointer on case 1 last on case 0
input is 0/ frame exists
(3,0)(0,1)(2,0)pointer on case 1 last on case 0
input is 3/ frame exists
(3,1)(0,1)(2,0)pointer on case 1 last on case 0
input is 2/ frame exists
(3,1)(0,1)(2,1)pointer on case 1 last on case 0
input is 1/ frame non existant
(3,0)(1,0)(2,0)pointer on case 2 last on case 1
input is 2/ frame exists
(3,0)(1,0)(2,1)pointer on case 2 last on case 1
input is 0/ frame non existant
(0,0)(1,0)(2,0)pointer on case 1 last on case 0
input is 1/ frame exists
(0,0)(1,1)(2,0)pointer on case 1 last on case 0
input is 7/ frame non existant
(0,0)(1,0)(7,0)pointer on case 0 last on case 2
input is 0/ frame exists
(0,1)(1,0)(7,0)pointer on case 0 last on case 2
input is 1/ frame exists
(0,1)(1,1)(7,0)pointer on case 0 last on case 2
end of CLOCK processing
(0,1)(1,1)(7,0)pointer on case 0 last on case 2
Fault Pages Clock 11
```

Figure 9 – Trace d'execution CLK suite 2

4 Etude expérimentale

Afin de comparer les 3 algorithmes utilisés nous utiliserons comme facteur de performance le nombre de page faults qui représente le nombre de fois où l'algorithme ne trouve pas une page donnée dans le buffer et fait une lecture écriture de la mémoire disque.

Avec les suites d'entrées précédemment utilisées nous avons trouvé les résultats suivants pour un buffer de taille 3 :

— série 1 : Clock 14 FIFO 13 LRU 15

— série 2 : Clock 11 FIFO 15 LRU 12

Pour un buffer de taille 4 :

— série 1 : Clock 6 FIFO 7 LRU 6

— série 2 :Clock 8 FIFO 10 LRU 8

Nous pouvons remarquer que les résultats ne sont pas concluants par rapport à la performance des algorithmes. Pour cela, nous allons modifier les entrées du système pour pouvoir étudier le comportement des algorithmes avec ces variations.

Pour modéliser les pages en entrée, nous utiliserons une distribution de pages selon la loi normale (centrée sur 100 pour avoir des numéros de page positifs) et en changeant la variance qui correspondra en quelque sorte à un facteur de diversification des pages : Moins la variance est grande plus les pages sont semblables et répétitives. Nous adopterons un facteur pas très élevé pour que les pages soient répétitives et moyennement variées.

Nous utiliserons comme variable la taille du buffer qui augmentera à chaque itération et nous aurons comme sortie le nombre de page faults pour chaque algorithme.

Pour implémenter cela, nous utiliserons le code Java d'évaluation présenté dans l'annexe qui utilisera une entrée aléatoire et une taille de buffer incrémentée. Le code enregistrera le résultat dans un fichier CSV qu'on exploitera pour visualiser le comportement des algorithmes.

Un exemple d'output donné par le programme avec un échantillon de 10k pages avec une loi de distribution N(100,20).

```
size,Clck,FIFO,LRU
3 9610 9602 9605
4 9440 9451 9442
5 9306 9296 9308
6 9141 9162 9158
7 9023 9037 9029
8 8906 8925 8914
```

Figure 10 – Exemple de trace d'évaluation

Afin de visualiser les résultats obtenus, nous utiliserons un script en langage R permettant d'avoir les courbes d'évolution des page faults pour les différents algorithmes. Le script R est

présenté dans l'annexe.

4.1 Une répétition élevée, variance = 5

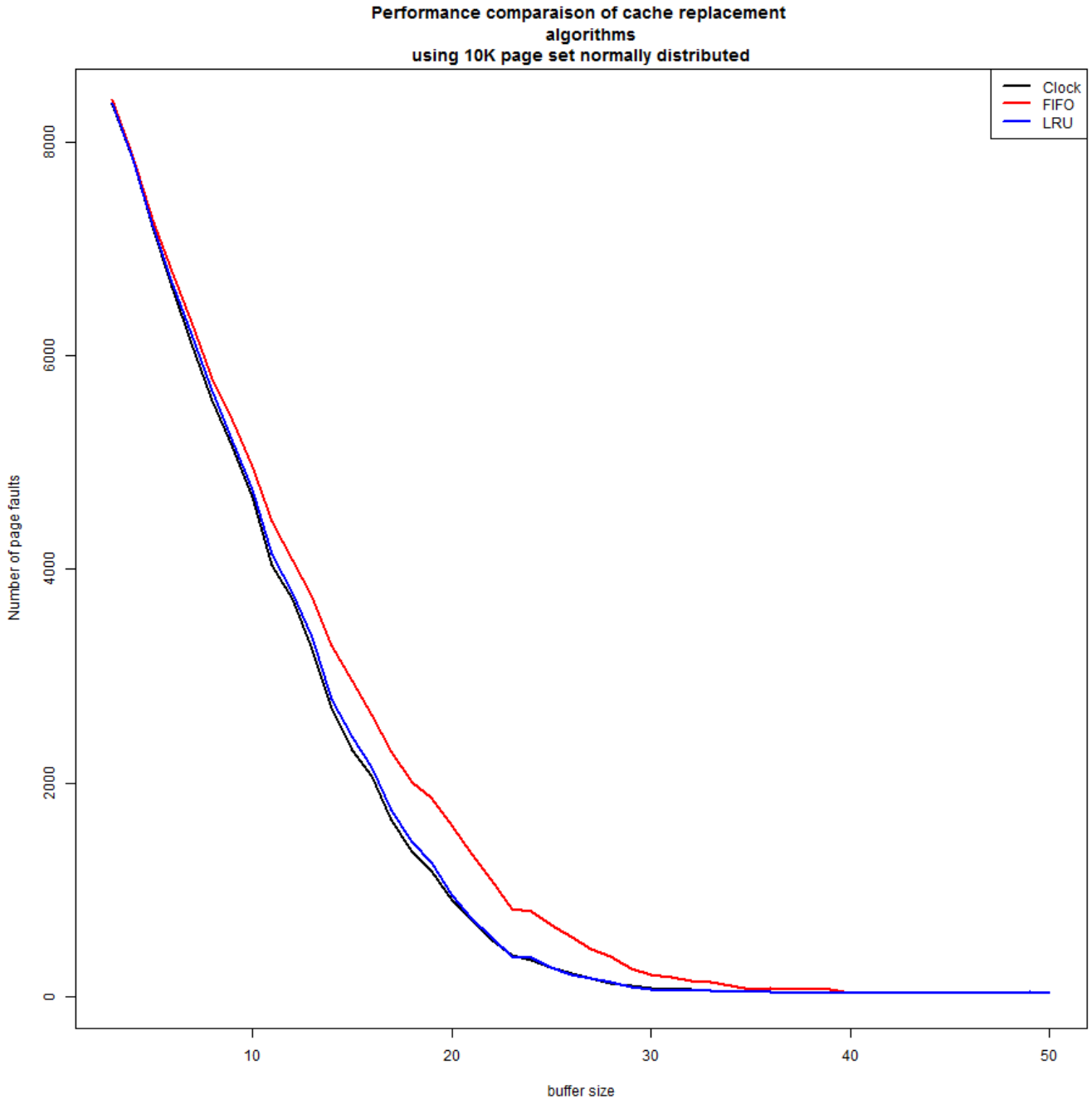

Figure 11 – Evolution fault pages, variance = 7

Avec une grande répétition des pages, nous pouvons remarquer que les deux algorithmes

L.R.U et Clock ont une courbe logarithmique décroissante avec une décroissance rapide et donc une rapidité à converger en terme de page faults plus importante que le FIFO. Ceci revient à la caractéristique de prise en compte de l'utilisation des pages par ces 2 algorithmes.

4.2 Diversification des pages

En augmentant la variance des pages utilisées, nous constatons une augmentation des fault pages pour les 3 algorithmes mais toujours avec décalage entre LRU et Clock d'un coté et FIFO d'un autre coté ce qui prouve l'importance de la prise en considération de l'utilisation des pages. Une conséquence assez triviale de l'augmentation de la variance est l'affaiblissement des performances des algorithmes avec des courbes qui deviennent presque linéaires (voir figures 12 et 13).

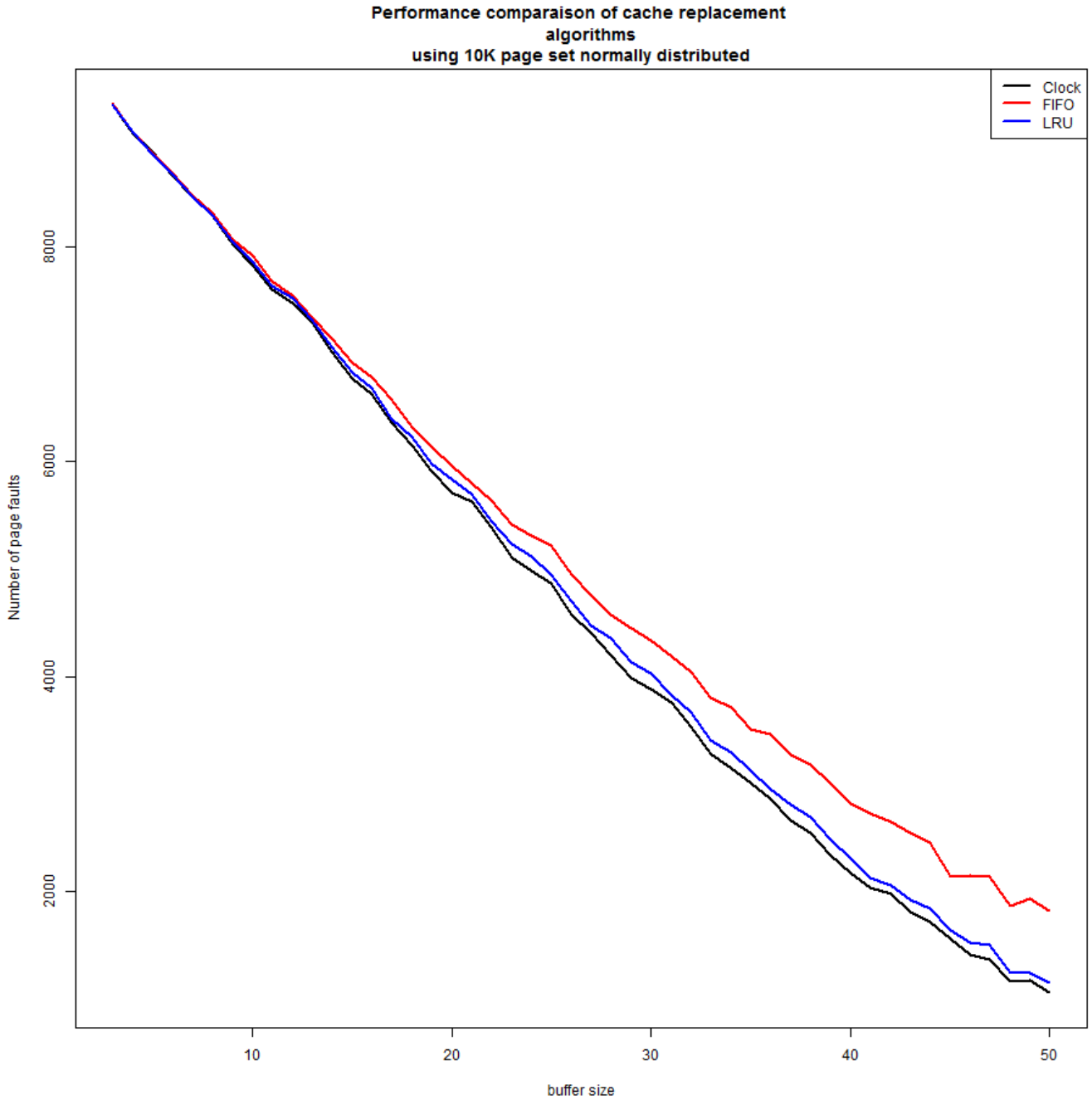

Figure 12 – Evolution fault pages, variance = 13

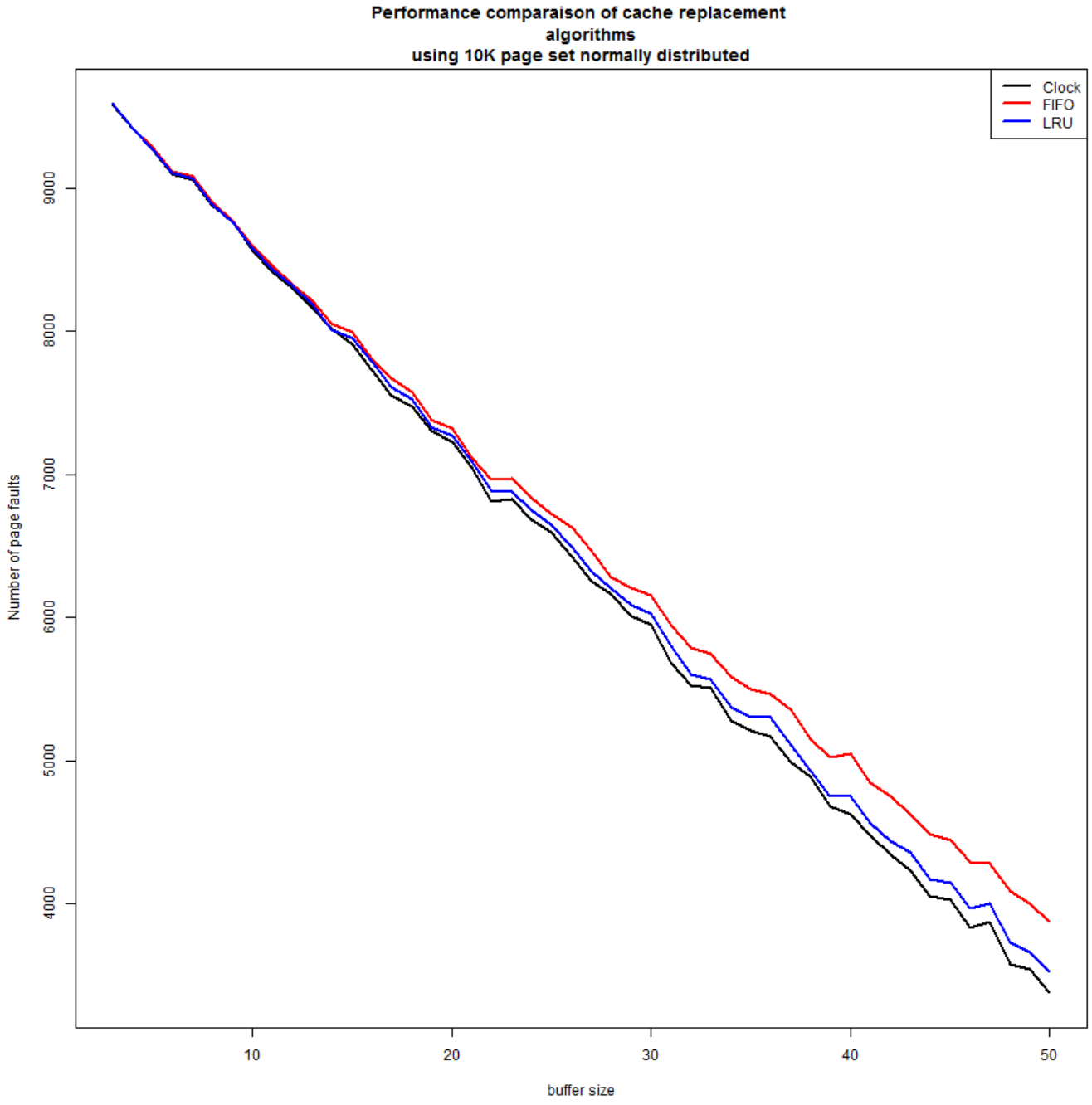

Figure 13 – Evolution fault pages, variance = 20

Conclusion

A travers ce document, nous avons étudié dans un premier temps 3 algorithmes de remplacement de pages de buffer utilisés par le Buffer Manager à savoir FIFO, LRU et Clock algorithm. Dans un second temps nous avons modélisé les structures de données utilisées par ces algorithmes pour proposer une implementation en langage Java de ces derniers. Cette implementation nous a permis de tester et comparer les performances de ces algorithmes à travers une étude expérimentale étudiant le comportement des algorithmes en changeant la distribution des pages utilisées et la taille du buffer.

Bibliographique

[1] Shaify-Mehta. Simulation FIFO LFU and MFU Page Replacement Algorithms. https://fr.scribd.com/doc/73808713/Simulation-FIFO-LFU-and-MFU-Page-Replacement-Algorithms-VB-by-Shaify-Meh (2015). [En ligne ; consulté le 10-Novembre-2017]. 1

[2] Wikimedia. Fifo queue. https://commons.wikimedia.org/wiki/File:Fifo_queue.png, (2015). [En ligne ; consulté le 10-Novembre-2017]. 2

[3] osinfoblog. The Not Recently Used Page Replacement Algorithm. http://www.osinfoblog.com/post/117/the-not-recently-used-page-replacement-algorithm/, (2014). [En ligne ; consulté le 10-Novembre-2017]. 3

[4] techTarget. Buffer. http://whatis.techtarget.com/definition/buffer, (2015). [En ligne ; consulté le 10-Novembre-2017]. 4

Annexe

Listing 1 – Fifo Processing Function Java

```
public int process_FIFO(){
     int count = 0;  // this variable counts the number of fault pages
     point =0;
    this.bufferInit();  // init the buffer

      for(int i=0;i<this.input.size();i++){ // search if element exists in
          buffer

        boolean exist = false;
        if (this.buffer.exist(this.input.get(i).toString())){
         // Do nothing
        }
         else{  // if the element does not exist
            try{
              this.buffer.insert_case(new Case(this.input.get(i).toString
                  (),0),point); // insert the page at the end of queue
                  pointed by "point" variable
              point = (point +1) % this.size; // increment the pointer to
                   point to the oldest element in the queue
              count++;
            } catch(Exception e) {

                System.out.println("Error with inserting new element");
              }
        }
     }
    return count ;}
}
```

Listing 2 – LRU Processing Function Java

```
public int process_LRU(){
        int count = 0;
        point =0;
        this.bufferInit();  // init the buffer
        for(int i=0;i<this.input.size();i++){
            // if the element exists
           // search the index and put the item in the head case and push
              back all the other items
          boolean exist = false;

            if (this.buffer.exist(this.input.get(i).toString())){

             int last_index = this.buffer.existAt(this.input.get(i).toString
                ());
             Case a = new Case();
             a = this.buffer.elementAt(last_index);
              for(int k=last_index;k<(this.size-1);k++){
                  this.buffer.insert_case(this.buffer.elementAt(k+1), k);
              }
              this.buffer.insert_case(a, this.size-1);
             }
            else {
                // if the item doesn't exist in the buffer  we push back all
                   the
                // items and we put the new item in the head

                for(int k=0;k<(this.size-1);k++){
                this.buffer.insert_case(this.buffer.elementAt(k+1), k);

            }
                Case a = new Case(input.elementAt(i).toString(),0);
                this.buffer.insert_case(a, this.size-1);
                count++;
            }
         }
         return count;
    }
```

Listing 3 – Clock Processing Function Java

```
public int Process_CLK(){
        point=0;
        last=-1;
        int count = 0;

        this.bufferInit(); // buffer init

        for(int i=0;i<this.input.size();i++){
           // if the element exists reinsert it with R bit (flag) at 1
           boolean exist = false;
           if (this.buffer.exist(this.input.get(i).toString())){
              try{
               this.buffer.insert_case(new Case(this.input.get(i).toString()
                  ,1),this.buffer.existAt(this.input.get(i).toString()));

              }catch(Exception e){

                  System.out.println("Problem with replacement here frame
                     exists !");
              }
           }
             else{   // if the element does not exist
              point = (last +1) %this.size ;
              count++;
              while(this.buffer.elementAt(point).getFlag()!=0){
                   String a = this.buffer.elementAt(point).getFrame();
                   this.buffer.insert_case(new Case(a,0), point);
                   point = (point +1) % this.size;

              }

              if(this.buffer.elementAt(point).getFlag()==0){
                 try{
              this.buffer.insert_case(new Case(this.input.get(i).toString()
                  ,0),point);
              last = point;
              point = (point +1) % this.size;
```

```
            }catch(Exception e){

                System.out.println("Problem with replacement free space  ")
                    ;
            }
             }
        }
     }
    return count;
     }
```

Listing 4 – Evaluation Function Java

```
public static void main(String[] args) throws FileNotFoundException,
    UnsupportedEncodingException {

         PrintWriter writer = new PrintWriter("D:/Output2.csv", "UTF-8");
         writer.println("size,Clk,FIFO,LRU");

        for(int b= 3; b<=100;b++){


         Random  a = new Random();
         Vector in = new Vector();
         for(int i =0;i<10000;i++){
             in.add(new Double((a.nextGaussian()*20+100)).intValue());
         }
          processing p = new processing(in,b);

          int missing_clk = p.Process_CLK();
          int missing_FIFO = p.process_FIFO();
          int missing_LRU = p.process_LRU();

         System.out.println(b+" "+missing_clk+" "+missing_FIFO+" "+missing_LRU
            );
         String s =b+","+missing_clk+","+missing_FIFO+","+missing_LRU;
         writer.println(s);
       }

        writer.close();
     }

}
```

Listing 5 – Results Plotting R script

```
library(data.table)
png("D:/bd_5.png", width=920, height=920)
tab <- read.table("D:/Output2.csv", header = TRUE, sep = ",")
plot(x = tab$size,y = tab$Clck, type = "l", col = "black",
     ylab = "Number of page faults",  xlab = "buffer size" , lwd= "2",
     main = "Performance comparaison of cache replacement
     algorithms \n using 10K page set normally distributed ")
lines(x = tab$size, y = tab$FIFO, type = "l", col = "red", lwd = "2")
lines(x = tab$size, y = tab$LRU, type = "l", col = "blue", lwd = "2")
legend("topright", c("Clock", "FIFO", "LRU"),lwd = "2" ,
       col = c("black","red", "blue"))
dev.off()
```