UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL
BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO
SISTEMAS OPERACIONAIS

RELATÓRIO: ANÁLISE DE ALGORÍTMO DE PAGINAÇÃO

NOÉ FERNANDES CARVALHO PESSOA
PABLO EMANUELL LOPES TARGINO
RAFAELLA SILVA GOMES

NATAL
NOVEMBRO 2019

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL
BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO
SISTEMAS OPERACIONAIS

RELATÓRIO: ANÁLISE DE ALGORÍTMO DE PAGINAÇÃO

Relatório do projeto único referente à nota integral da terceira unidade da disciplina de Sistemas Operacionais do curso de Bacharelado em Tecnologia da Informação do Instituto Metrópole Digital da Universidade Federal do Rio Grande do Norte.

Grupo formado por:
Noé Fernandes Carvalho Pessoa
Pablo Emanuell Targino
Rafaella Silva Gome

NATAL
NOVEMBRO 2019

Sumário

1. Introdução

Na disciplina de Sistemas Operacionais, nosso projeto da terceira unidade consiste no desenvolvimento de um programa em linguagem Python de um algorítmo de paginação de memória utilizando três métodos distintos e sua análise. O algoritmo simula um gerenciamento de memória virtual em sistemas operacionais de computadores que utilizam a paginação.

A paginação é um método de gerenciamento de memória realizado pelo Sistema Operacional (SO), onde o computador armazena e recupera dados de um armazenamento secundário (HD) para usar na memória principal (RAM).

Na paginação, a memória física é dividida em *frames* e estes são chamados de páginas de memória virtual. Cada página é mapeada em memória, onde é criada uma relação os endereços lineares das páginas e seu correspondente no endereço físico, para que isso funcione corretamente é criado uma tabela que deve ser implementada numa unidade dedicada do *hardware* integrado aos processadores (MMU).

Sobre os métodos, usamos no nosso projeto o algorítmo FIFO, FIFO 2 (segunda chance) e LRU. No FIFO o SO mantém uma fila das páginas, onde a que está no início da fila é a mais antiga e a que está ao final é a mais recente. Quando uma ocorre um evento de *page fault*, a página do início é removida e a nova é inserida ao final da fila.

Quanto ao algorítmo FIFO 2, ele assemelha-se ao FIFO, mas com o acréscimo de um bit de referência, que será inspencionado. Caso o bit de referência seja 0, a página é velha e portanto não foi usada recentemente, logo pode ser alterada. Caso seja 1, o bit é trocado por 0, a página vai para o final da fila e a busca segue.

Por fim, no algorítmo LRU, as páginas que foram muito usadas recentemente serão provavelmente usadas novamente, assim, o algorítmo troca a página que permaneceu em desuso pelo maior tempo.

2. Implementação

Foram implementados três métodos de paginação, como foi dito anteriormente. Foram eles: FIFO, FIFO 2 e LRU.

Nosso programa deve receber inicialmente três parâmetros (Ver Figura 1), sendo eles: a string de referência, o método de paginiação que deve ser aplicado e a quantidade de frames de memória principal. Onde estes são os principais parâmetros de análise, pois a partir deles podemos realizar o cálculo da string de distância e a contagem de *page fault's*.

```
if ( __name__ == "__main__"):
        strstrref = input("digite a string de referência: \n")
        strref = [int(c) for c in strstrref]
        qtdpag = max(strref) - min(strref) + 1
        qtdfrm = int(input("digite a quantidade de frames: "))
        alg = input("digite a politíca desejada: (fifo, lru ou fifo2): ")

        tabela, page_fault = simular(alg, strref, qtdfrm, qtdpag)
        string_dist = calcular_string_distancia(tabela, strref)
        media = calcular_media(string_dist)

        print("string de distância: \n")
        print(string_dist)
        print("\nMédia: ")
        print(media)
        salvar_em_csv("saida.csv", tabela, qtdfrm, page_fault, strref, string_dist)
```

Figura 1: Main do programa com recebimento de parâmetros.

Caso o método desejado seja o FIFO, é implementado o seguinte algorítmo (ver Figura 2):

```
def simular_fifo(strref, qtdfrm, qtdpag):
        fila_frm = deque(qtdfrm * [''])
        fila_prc = deque((qtdpag if qtdpag > qtdfrm else qtdfrm) * [''])
        tabela_prc = [list(fila_prc)]
        page_fault = ['']
        for page in strref:
                if(page not in fila_frm):
                        fila_frm.pop()
                        fila_frm.appendleft(page)
                        fila_prc.pop()
                        fila_prc.appendleft(page)
                        page_fault.append('P')
                else:
                        page_fault.append('')
                tabela_prc.append(list(fila_prc))
        return tabela_prc, page_fault
```

Figura 2: Implementação do Algorítmo FIFO.

Já o algorítmo FIFO 2 (Segunda Chance) foi implementado da seguinte forma (ver Figura 3):

```
def simular_fifo2(strref, qtdfrm, qtdpag):
        refereced = qtdfrm * [False]
        fila_frm = deque(qtdfrm * [''])
        fila_prc = deque((qtdpag-qtdfrm) * ['']) # caso qtdfrm > qtdpag não vai ter substituição de páginas
        tabela_prc = [list(fila_prc + fila_frm)]
        page_fault = ['']
        for page in strref:
                print("\t".join((str(p) for p in fila_frm)))
                print("\t".join((str(r) for r in refereced)))
                if(page not in fila_frm):
                        if(fila_frm[-1] == ''):
                                fila_frm.appendleft(page)
                                fila_frm.pop()
                                page_fault.append('P')
                                refereced[1] = refereced[0]
                                refereced[0] = False
                        else:
                                removed_index = -1
                                for i in range(len(fila_frm)-1, -1, -1):
                                        if(refereced[i]):
                                                refereced[i] = False
                                        else:
                                                refereced[i] = False
                                                removed_index = i
                                                break
                                try:
                                        fila_prc.remove(page)
                                except ValueError:
                                        fila_prc.pop()
                                fila_prc.appendleft(fila_frm[removed_index])
                                fila_frm.remove(fila_frm[removed_index])
                                fila_frm.appendleft(page)
                                page_fault.append('P')
                                for i in range(removed_index, 0, -1):
                                        refereced[removed_index] = refereced[removed_index-1]
                                refereced[0] = False
                else:
                        page_fault.append('')
                        refereced[fila_frm.index(page)] = True
                tabela_prc.append(list(fila_frm + fila_prc))
        return tabela_prc, page_fault
```

Figura 3: Implementação do algorítmo FIFO 2.

Para concluir esta parte dos algorítmos de paginação, temos a implementação do método LRU (ver Figura 4):

```
def simular_lru(strref, qtdfrm, qtdpag):
        fila_frm = deque(qtdfrm * [''])
        fila_prc = deque((qtdpag if qtdpag > qtdfrm else qtdfrm) * [''])
        tabela_prc = [list(fila_prc)]
        page_fault = ['']
        for page in strref:
                if(page not in fila_frm):
                        fila_frm.pop()
                        fila_frm.appendleft(page)
                        if(page in fila_prc):
                                fila_prc.remove(page)
                        else:
                                fila_prc.pop()
                        fila_prc.appendleft(page)
                        page_fault.append('P')
                else:
                        page_fault.append('')
                        fila_prc.remove(page)
                        fila_prc.appendleft(page)
                        fila_frm.remove(page)
                        fila_frm.appendleft(page)
                tabela_prc.append(list(fila_prc))
        return tabela_prc, page_fault
```

Figura 4: Implementação do algorítmo LRU.

Após o programa realizar as ações correspondentes ao método solicitado, é calculado a *string* de distância e em seguida é feita a média desta *string*. A implementação segue abaixo (ver Figura 5):

```
def calcular_string_distancia(tabela, strref):
        lista_dist = [None]*len(strref)
        for i in range(1, len(strref) + 1):
                try:
                        distancia = tabela[i - 1].index(strref[i-1]) + 1
                        lista_dist[i - 1] = distancia
                except ValueError:
                        lista_dist[i - 1] = 0
        return lista_dist

def calcular_media(list_dist):
        return sum(list_dist)/len(list_dist)
```

Figura 5: Cálculo da string de distância e sua média.

Para finalizar, é gerado um arquivo de saída contendo a execução do algorítmo correspondente apresentando a *string* de referência no topo, a divisão dos frames e da memória secundária, por fim as *page fault's* e a *string* de distância, conforme mostraremos logo abaixo (ver Figura 6):

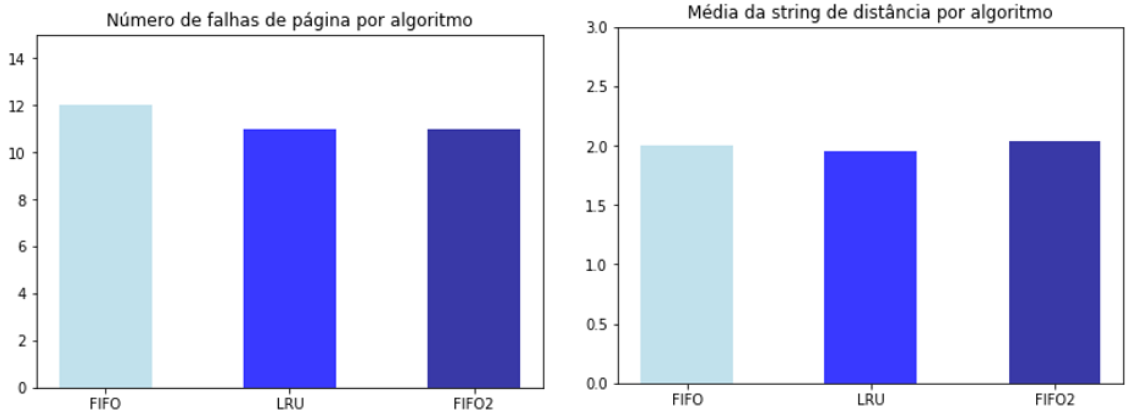
	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		1	2	3	4	3	2	5	4	6	3	4	6	7
2	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3		1	2	3	4	3	2	5	4	6	3	4	6	7
4			1	2	3	4	3	2	5	4	6	3	4	6
5				1	2	2	4	3	2	5	4	6	3	4
6					1	1	1	4	3	2	5	5	5	3
7	-	-	-	-	-	-	-	-	-	-	-	-	-	-
8								1	1	3	2	2	2	5
9										1	1	1	1	2
10														1
11		P	P	P	P			P		P	P			P
12		0	0	0	0	2	3	0	4	0	5	3	3	0

Figura 6: Arquivo de saída de teste para o exemplo com a string: 1234325463467, com 4 frames e modelo LRU.

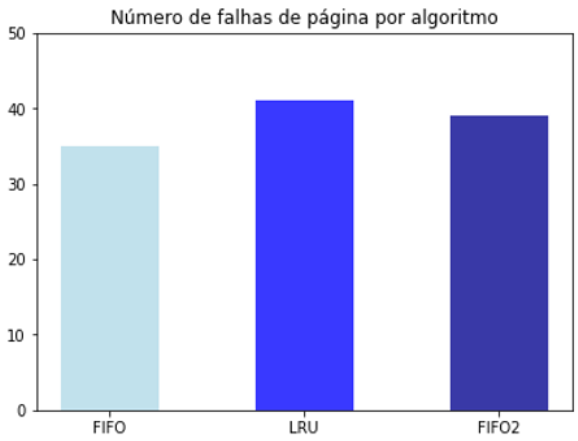
3. Simulações

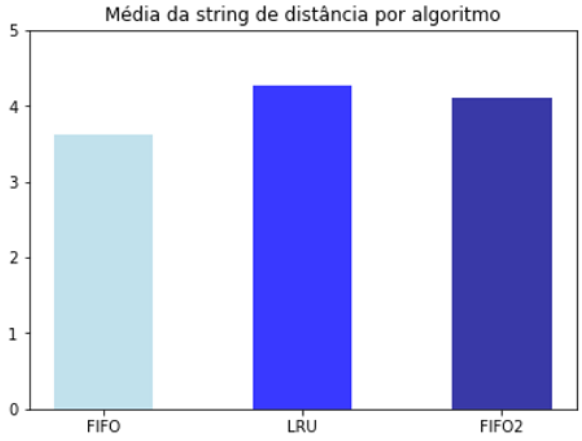
Quanto as simulações realizadas, nós realizamos os seguintes testes:

- String: 02135463747335531117134 1
- Frames: 4

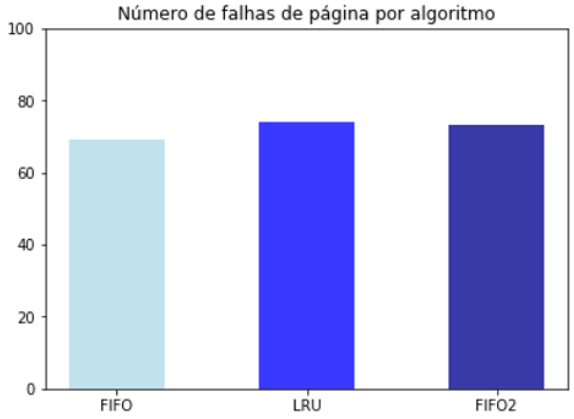


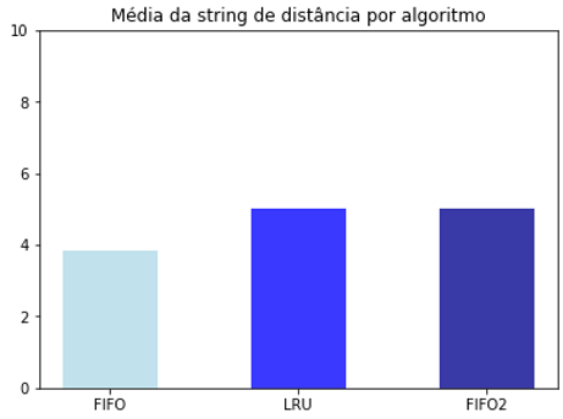
- String: 0213322415312457698213415346134776497614534612356123541 32
- Frames: 4





- String: 08768234628615383854896254892436513456130947566986742765483579567762454312866737896871241846634783812963 28
- Frames: 4





Outro teste que realizamos foi com relação à anomalia de Belady, onde utilizamos a seguinte *string*: 012301401234, com 3 e 4 frames.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		0	1	2	3	0	1	4	0	1	2	3	4
2	-	-	-	-	-	-	-	-	-	-	-	-	-
3		0	1	2	3	0	1	4	4	4	2	3	3
4			0	1	2	3	0	1	1	1	4	2	2
5				0	1	2	3	0	0	0	1	4	4
6	-	-	-	-	-	-	-	-	-	-	-	-	-
7					0	1	2	3	3	3	0	1	1
8						0	1	2	2	2	3	0	0
9		P	P	P	P	P	P	P			P	P	
10		0	0	0	0	4	4	0	3	2	5	5	3

Figura 8: Teste da anomalia de Belady com 3 frames.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		0	1	2	3	0	1	4	0	1	2	3	4
2	-	-	-	-	-	-	-	-	-	-	-	-	-
3		0	1	2	3	3	3	4	0	1	2	3	4
4			0	1	2	2	2	3	4	0	1	2	3
5				0	1	1	1	2	3	4	0	1	2
6					0	0	0	1	2	3	4	0	1
7	-	-	-	-	-	-	-	-	-	-	-	-	-
8								0	1	2	3	4	0
9		P	P	P	P			P	P	P	P	P	P
10		0	0	0	0	4	3	0	5	5	5	5	5

Figura 7: Teste da anomalia de Belady com 4 frames.