Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavre

A Lab Report #3

[Course title: COMP 307]

Submitted by:
Bisheshwor Neupane(35)

Submitted to:
Mr. Dhiraj Shrestha
Department of Computer Science and Engineering

Submission Date:

August 09, 2021

1. Simulate the concept of First fit, best fit and worst fit allocation algorithms
[Make necessary assumptions]

Solution:

First fit Allocation:

Algorithm:

- Input memory blocks and processes with sizes.
- Initialize all memory blocks as free.
- Start by picking each process and check if it can be assigend to current block.
- If size-of-process is less than or equals to size-of-block if yes then assign and check for next process.
- If not then keep checking the further process.

Code Implementation in Python:

```python
def firstFit(blockSize, m, processSize, n):

    allocation = [-1] * n

    for i in range(n):
        for j in range(m):
            if blockSize[j] >= processSize[i]:

                # allocate block j to p[i] process
                allocation[i] = j

                # Reduce available memory in this block.
                blockSize[j] -= processSize[i]

                break

    print(" Process No. Process Size     Block no.")
    for i in range(n):
        print(" ", i + 1, "      ", processSize[i],
                        "        ", end = " ")
        if allocation[i] != -1:
            print(allocation[i] + 1)
        else:
            print("Not Allocated")

if __name__ == '__main__':
    blockSize = [200, 300, 100, 300, 600]
    processSize = [313, 417, 120, 420]
    m = len(blockSize)
    n = len(processSize)

    firstFit(blockSize, m, processSize, n)

```

Output:

```
bshesh@pop-os:~/Documents/Programming/OS/OS_lab/lab3$ python3 first.py
 Process No. Process Size        Block no.
  1               313             5
  2               417             Not Allocated
  3               120             1
  4               420             Not Allocated
```

Best Fit Allocation:

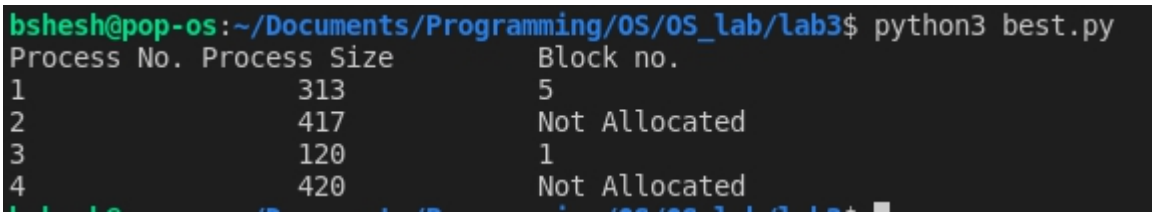
Algorithm:

- Input memory blocks and processes with sizes.
- Initialize all memory blocks as free.
- Start by picking each process and find the minimum block size that can be assigned to current process i.e., find min(blockSize[1], blockSIze[2], ... blockSize[n]) > processSize[current], if found then assign it to the current process.
- If not then leave that process and keep checking the further processes.

Code Implementation in Python:

```python

def bestFit(blockSize, m, processSize, n):

    allocation = [-1] * n

    for i in range(n):

        bestIdx = -1
        for j in range(m):
            if blockSize[j] >= processSize[i]:
                if bestIdx == -1:
                    bestIdx = j
                elif blockSize[bestIdx] > blockSize[j]:
                    bestIdx = j

        if bestIdx != -1:

            allocation[i] = bestIdx
            blockSize[bestIdx] -= processSize[i]

    print("Process No. Process Size  Block no.")
    for i in range(n):
        print(i + 1, "       ", processSize[i],
                                  end = "      ")
        if allocation[i] != -1:
            print(allocation[i] + 1)
        else:
            print("Not Allocated")

if __name__ == '__main__':
    blockSize = [200, 300, 100, 300, 600]
    processSize = [313, 417, 120, 420]
    m = len(blockSize)
    n = len(processSize)

    bestFit(blockSize, m, processSize, n)

```

Output:



Worst Fit Allocation:

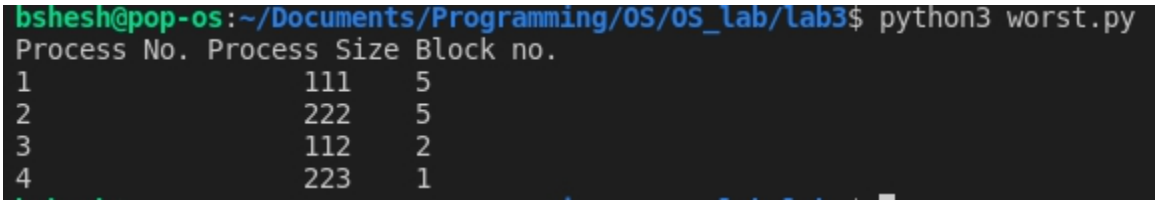
Algorithm:

- Input memory blocks and processes with sizes.
- Initialize all memory blocks as free.
- Start by picking each process and find the maximum block size that can be assigned to current process i.e., find max(bockSize[1], blockSize[2],.....blockSize[n]) > processSize[current], if found then assign it to the current process.
- If not then leave that process and keep checking the further processes.

Code Implementation in Python:

```python

def worstFit(blockSize, m, processSize, n):

    allocation = [-1] * n

    for i in range(n):

        wstIdx = -1
        for j in range(m):
            if blockSize[j] >= processSize[i]:
                if wstIdx == -1:
                    wstIdx = j
                elif blockSize[wstIdx] < blockSize[j]:
                    wstIdx = j

        if wstIdx != -1:

            allocation[i] = wstIdx
            blockSize[wstIdx] -= processSize[i]

    print("Process No. Process Size Block no.")
    for i in range(n):
        print(i + 1, "       ",
            processSize[i], end = "  ")
        if allocation[i] != -1:
            print(allocation[i] + 1)
        else:
            print("Not Allocated")

if __name__ == '__main__':
    blockSize = [300, 400, 200, 100, 600]
    processSize = [111, 222, 112, 223]
    m = len(blockSize)
    n = len(processSize)

    worstFit(blockSize, m, processSize, n)


```

Output:


bshesh@pop-os:~/Documents/Programming/OS/OS_lab/lab3$ python3 worst.py
Process No. Process Size Block no.
1 111 5
2 222 5
3 112 2
4 223 1

2. Simulate the concept of following page replacement algorithms: [Make necessary assumptions]

a. **FIFO**
b. **Optimal**
c. **LRU**
d. **Second Chance page replacement algorithm**

Solutions:

A. FIFO

Algorithm:

1- Start traversing the pages.
i) If set holds less pages than capacity.
a) Insert page into the set one by one until the size of set reaches capacity or all page requests are processed.
b) Simultaneously maintain the pages in the queue to perform FIFO.
c) Increment page fault
ii) Else
If current page is present in set, do nothing.
Else
a) Remove the first page from the queue as it was the first to be entered in the memory
b) Replace the first page in the queue with the current page in the string.
c) Store current page in the queue.
d) Increment page faults.

2. Return page faults.

Code Implementation in Python:

```
q = []
for i in range(6):
    q.append(i)
print("Elements of queue-" , q)
removedele = q.pop(0)
print("removed element-" , removedele)

print(q)
head = q[0]
print("head of queue-" , head)

size = len(q)
print("Size of queue-" , size)
```

Output:

```
bshesh@pop-os:~/Documents/Programming/OS/OS_lab/lab3$ python3 fifo.py
Elements of queue- [0, 1, 2, 3, 4, 5]
removed element- 0
[1, 2, 3, 4, 5]
head of queue- 1
Size of queue- 5
```

B. Optimal

Algorithm:

1. If referred page is already present, increment hit count.
2. If not present, find if a page that is never referenced in future. If such a page exists, replace this page with new page. If no such page exists, find a page that is referenced farthest in future. Replace this page with new page.

Code Implementation in Python:

```python
print("Enter the number of frames: ",end="")
capacity = int(input())
f,fault,pf = [],0,'No'
print("Enter the reference string: ",end="")
s = list(map(int,input().strip().split()))
print("\nString|Frame →\t",end='')
for i in range(capacity):
    print(i,end=' ')
print("Fault\n   ↓\n")
occurance = [None for i in range(capacity)]
for i in range(len(s)):
    if s[i] not in f:
        if len(f)<capacity:
            f.append(s[i])
        else:
            for x in range(len(f)):
                if f[x] not in s[i+1:]:
                    f[x] = s[i]
                    break
                else:
                    occurance[x] = s[i+1:].index(f[x])
            else:
                f[occurance.index(max(occurance))] = s[i]
        fault += 1
        pf = 'Yes'
    else:
        pf = 'No'
    print("   %d\t\t"%s[i],end='')
    for x in f:
        print(x,end=' ')
    for x in range(capacity-len(f)):
        print(' ',end=' ')
    print(" %s"%pf)
print(
"\nTotal requests: %d\nTotal Page Faults: %d\nFault Rate: %0.2f%
%"
%(len(s),fault,(fault/len(s))*100))

```

Output:

```
bshesh@pop-os:~/Documents/Programming/OS/OS_lab/lab3$ python3 optimal.py
Enter the number of frames: 6
Enter the reference string: 7

String|Frame →  0 1 2 3 4 5 Fault
   ↓

   7            7            Yes

Total requests: 1
Total Page Faults: 1
Fault Rate: 100.00%
```

C. LRU

Algorithm:

Let capacity be the number of pages that memory can hold. Let set be the current set of pages in memory.

1- Start traversing the pages.
i) If set holds less pages than capacity.
a) Insert page into the set one by one until the size of set reaches capacity or all page requests are processed.
b) Simultaneously maintain the recent occurred index of each page in a map called indexes.
c) Increment page fault
ii) Else
If current page is present in set, do nothing.
Else
a) Find the page in the set that was least recently used. We find it using index array. We basically need to replace the page with minimum index.
b) Replace the found page with current page.
c) Increment page faults.
d) Update index of current page.

2. Return page faults.

Code Implementation in Python:

```
print("Enter the number of frames: ",end="")
capacity = int(input())
f,st,fault,pf = [],[],0,'No'
print("Enter the reference string: ",end="")
s = list(map(int,input().strip().split()))
print("\nString|Frame →\t",end='')
for i in range(capacity):
    print(i,end=' ')
print("Fault\n   ↓\n")
for i in s:
    if i not in f:
        if len(f)<capacity:
            f.append(i)
            st.append(len(f)-1)
        else:
            ind = st.pop(0)
            f[ind] = i
            st.append(ind)
        pf = 'Yes'
        fault += 1
    else:
        st.append(st.pop(st.index(f.index(i))))
        pf = 'No'
    print("   %d\t\t"%i,end='')
    for x in f:
        print(x,end=' ')
    for x in range(capacity-len(f)):
        print(' ',end=' ')
    print(" %s"%pf)
print(
"\nTotal Requests: %d\nTotal Page Faults: %d\nFault Rate: %0.2f%
%"
%(len(s),fault,(fault/len(s))*100))

```

Output:

```
bshesh@pop-os:~/Documents/Programming/OS/OS_lab/lab3$ python3 lru.py
Enter the number of frames: 5
Enter the reference string: 3

String|Frame →  0 1 2 3 4 Fault
   ↓

   3            3          Yes

Total Requests: 1
Total Page Faults: 1
Fault Rate: 100.00%
bshesh@pop-os:~/Documents/Programming/OS/OS_lab/lab3$
```

D.Second Chance page replacement algorithm

Algorithm:

Create an array frames to track the pages currently in memory and another Boolean array second_chance to track whether that page has been accessed since it's last replacement (that is if it deserves a second chance or not) and a variable pointer to track the target for replacement.

1. Start traversing the array arr. If the page already exists, simply set its corresponding element in second_chance to true and return.

2. If the page doesn't exist, check whether the space pointed to by pointer is empty (indicating cache isn't full yet) – if so, we will put the element there and return, else we'll traverse the array arr one by one (cyclically using the value of pointer), marking all corresponding second_chance elements as false, till we find a one that's already false. That is the most suitable page for replacement, so we do so and return.

3. Finally, we report the page fault count.

Code Implementation in Python:

```
1
2  def findAndUpdate(x, ref_frames, second_chance, frames):
3      for i in range(frames):
4          if(ref_frames[i]==x):
5              second_chance[i] =True
6              return True;
7      return False
8
9  def
    replaceAndUpdate(x, ref_frames, second_chance, frames, pointer):
10     while True:
11         if not second_chance[pointer]:
12             ref_frames[pointer] = x
13             return ((pointer+1)%frames)
14         second_chance[pointer]=False
15         pointer = (pointer+1)%frames
16
17 def printHitsAndFaults( ref_string, frames):
18     pointer, pf =0,0
19     ref_frames = [-1 for i in range(frames)]
20     second_chance = [False for i in range(frames)]
21     ref_string_list = ref_string.split()
22     length = len(ref_string_list)
23     for i in range(length):
24         x = int(ref_string_list[i])
25         if not
    findAndUpdate(x, ref_frames, second_chance, frames):
26             pointer = replaceAndUpdate(x, ref_frames, second_chan
   ce, frames, pointer)
27             pf+=1
28
29     print(f'Total page faults: {pf}')
30
31
32 def main():
33     ref_string = input("Enter the reference string:\t")
34     frames= input("Enter number of frames: \t")
35     printHitsAndFaults(ref_string, int(frames))
36
37 main()
```

Output:

```
bshesh@pop-os:~/Documents/Programming/OS/OS_lab/lab3$ python3 secondchance.py
Enter the reference string:     7
Enter number of frames:         4
Total page faults: 1
```