CS3500 - Operating System, August 2022

Lab 6: Memory Management

Due Date: Friday, 10th October 11.59PM in Moodle.
Demo Date (if required): 14/10/2022

1. Contiguous Memory Allocations

One of the simplest methods for allocating memory is to divide memory into several fixed-sized partitions. Each partition may contain exactly one process. Thus, the degree of multiprogramming is bound by the number of partitions. In this multiple partition method, when a partition is free, a process is selected from the input queue and is loaded into the free partition.

The first-fit, best-fit, and worst-fit strategies are the ones most commonly used to select a free hole from the set of available holes.

• **First fit.** Allocate the first hole that is big enough. Searching can start at the beginning of the set of holes. We can stop searching as soon as we find a free hole that is large enough. For initial allocation start from the beginning of the fixed holes

• **Best fit.** Allocate the smallest hole that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest leftover hole.

• **Worst fit.** Allocate the largest hole. Again, we must search the entire list, unless it is sorted by size. This strategy produces the largest leftover hole, which may be more useful than the smaller leftover hole from a best-fit approach.

Problem Statement
Given details of memory blocks and process sizes, write a program to implement the above three strategies to assign processes to memory blocks.

Note: Only one process at most can be allocated to a block

Inputs

1. N blocks with their corresponding block sizes (assign block number in ascending order)
2. M process with their process's sizes (assign process number is ascending order)

Output

Table in the format

Process number | Process size | Block size | Block Number

Example

Enter Number of Blocks: 5
Enter 5 block sizes: 200 600 300 400 700

Enter Number of Process: 4
Enter 4 Process Sizes: 330 520 210 550

First-Fit

Process number	Process size	Block size	Block Number
1	330	600	2
2	520	700	5
3	210	300	3
4	550	Not Allocated	

Best-Fit

Process number	Process size	Block size	Block Number
1	330	400	4
2	520	600	2
3	210	300	3
4	550	700	5

Worst-Fit

Process number	Process size	Block size	Block Number
1	330	700	5
2	520	600	2
3	210	400	4
4	550	Not Allocated	

Optional (Bonus Marks)
Create a graphical window to display the occupancy of blocks by processes as per your output

2. Memory usage and Page Faults

Write a program to implement the transpose of a matrix with tracking of memory usage and page faults by the process.
In your program you will be performing transpose of 5 square matrices of n*n(1000<= n <= 5000) dimensions for 10 iterations in 2 different settings. These matrices are allocated dynamically (malloc / calloc) and are filled with random long values. Use Inplace operations for matrix transpose.

Two different settings:

1. Memory allocated only once in 10 iterations.
2. Memory allocated each time in 10 iterations.

Your program must output the memory usage and the number of page faults after each iteration.
Example 1:

Enter n value: 1000
Enter Choice
Memory Allocated once - 1
Memory Allocated each time - 2

1

Output

memory usage: 2588 + 45436, page_faults: 11822
memory usage: 2588 + 45436, page_faults: 11829
memory usage: 2588 + 45436, page_faults: 11829
memory usage: 2588 + 45436, page_faults: 11829
memory usage: 2588 + 45436, page_faults: 11829
memory usage: 2588 + 45436, page_faults: 11829
memory usage: 2588 + 45436, page_faults: 11829
memory usage: 2588 + 45436, page_faults: 11829
memory usage: 2588 + 45436, page_faults: 11829
memory usage: 2588 + 45436, page_faults: 11829

(2588 is the baseline memory before dynamic memory allocation)

Example 2:

Enter n value: 1000

Enter Choice
Memory Allocated once - 1
Memory Allocated each time - 2

2

Output

memory usage: 2588 + 45544, page_faults: 11826
memory usage: 2588 + 92716, page_faults: 23571
memory usage: 2588 + 139708, page_faults: 35313
memory usage: 2588 + 186700, page_faults: 47056
memory usage: 2588 + 233692, page_faults: 58798
memory usage: 2588 + 280420, page_faults: 70540
memory usage: 2588 + 327412, page_faults: 82282
memory usage: 2588 + 374404, page_faults: 94024
memory usage: 2588 + 421396, page_faults: 105767

(2588 is the baseline memory before dynamic memory allocation)

3. Page Replacement

A page fault will happen if a program tries to access a piece of memory that does not exist in physical memory (main memory). The fault specifies the operating system to trace all data into virtual memory management and then relocate it from secondary memory to its primary memory, such as a hard disk.

If the physical memory is full then the process replaces the current page with the new requested page.
FIFO page replacement: The simplest page-replacement algorithm is a first-in, first-out (FIFO) algorithm. When a page must be replaced, the oldest page is chosen.
Optimal Page Replacement: Replace the page that will not be used for the longest period of time.
LRU Page Replacement: Replace the page that has not been used for the longest period of time.

Problem Statement
Write a program to implement the above three strategies for a given sequence of reference strings (strings of memory references) and page frame size (Assume each frame size is 100 bytes). Assume initially everything is in virtual memory.

Input:
Length of the Reference Strings (Number of memory requests)
Order of the sequence (address sequences reduced to reference strings)
Page-frame size (Number of frames in the physical memory)

Output:
Sequence of pages in the frames
Total number of page faults.

Example-1

Sequence Length: 10
Enter Sequence: 2 4 3 4 2 5 6 2 5 3
Page-frame size: 3

FIFO - 1
Optimal - 2
LRU - 3

Enter Page replacement strategy: 1

FIFO

2
2 4
2 4 3
No page fault
No page fault
4 3 5
3 5 6
5 6 2
No page fault
6 2 3

Total number of page faults = 7