CS 405 - Spring 2024
Project 2: Contiguous Memory Allocation

History:

Date	Descriptions
Jan 11, 2024	First released - Project available

1. Objectives

This teamwork project is designed to implement a Java program that demonstrates the continuous memory allocation schemes:

- Apply first-fit, best-fit, and worst-fit strategies for allocating memory contiguously.
- Understand the distinction between internal and external fragmentation.

2. Overview

Your team needs to design and implement a program (in any language, C/C++ or Java) that simulates the continuous memory allocation to processes using best-fit, worst-fit, and first-fit strategies.

This project will involve managing a contiguous region of memory of size MAX where addresses may range from 0 ... MAX−1. Your program must support the following operations of a memory management simulation:

- Allocate a memory partition to a process.
- Free a memory partition allocated to a process, and this process is terminated
- Report the current state of the memory: allocated memory partitions and free holes

3. Requirement Specs

3.1 General requirements

- This is a teamwork project in which you need to form a team of a maximum of 3 persons. You can continue with your team in Project 1 or start a new team.

- You can implement the simulation program in any language of your choice (C/C++, Java, Python)

3.2 Input Format

You will read a configuration file in the following format when your program starts. This file will contain multiple rows, each in the format <key> = <value>. The following table lists the different keys used in this program:

Key	Default value	Description
MEMORY_MAX	1024 KB	The max size of the contiguous memory (default is 1024 KB)
PROC_SIZE_MAX	256 KB	The max size of a process (this parameter controls the randomly generated size of a process)
NUM_PROC	10	The number of processes in the simulation
MAX_PROC_TIME	10000 ms	The maximum lifetime of a process in milliseconds (this parameter controls how long such a process holds its allocated memory partition before releasing it back)

These configuration parameters will be used throughout your program to manage the number of processes and memory boundaries, as well as handle the request/release memory partitions from processes.

3.3 Main program

You can develop your memory allocation simulator program in Console or GUI modes. The main application class should perform the following tasks:

- Load the configuration file - check the previous section (3.2) for the configuration format

- Generate NUM_PROC processes with random process size and lifetime.

- Service the memory request and release for each process or memory compaction (extra credit)

- Display the simulation statistics result.

3.3.1 Generate random processes

After loading the configuration file, your program will generate the list of processes with NUM_PROC size. Each process will need a random size between 0-PROC_SIZE_MAX and a random lifetime between 0-MAX_PROC_TIME in **ms**. The process's lifetime is between the time allocated to a memory partition (matches its request size), and the time the process terminates and releases its memory.

3.3.2 Allocating/Deallocating Memory

Your program will allocate memory using one of the three approaches highlighted in Section 9.2.2: best-fit, worst-fit, or first-fit. It will allocate as many processes as possible in the sequence of the generated processes in step 3.3.1. Any processes whose requests can not be fulfilled will have to wait.

This will require that your program keep track of the different holes representing available memory. When a request for a memory of a process arrives, it will allocate the memory from one of the available holes based on the allocation strategy. If there is insufficient memory to allocate to a request, your program will output an error message and put that request on hold until current running processes terminate and free their memory.

Your program will also need to keep track of which region of memory has been allocated to which process. This is necessary to support the memory visualization function (3.3.3) and is also needed when memory is released when a process terminates. Combine the two holes if a partition being released is adjacent to an existing hole.

3.3.3 The simulation visualization

Your program should keep track of the below simulation status events and display them:

- Your program will display the allocated and unused regions of memory every second after each execution step of all processes.

Below is a sample text representation on the Console mode:

| P1 [15s] (30KB) | P3 [12s] (10 KB) | Free (40 KB) | … |

The above example displays three memory partitions:

a) The size of the first memory partition is 30 KB and is allocated to P1. This process will continue to run for the next 15 seconds before terminating
b) The size of the second memory partition is 10 KB and is allocated to P3. This process will continue to run for the next 12 seconds before terminating
c) The third memory partition is a free (hole) and has a size of 40 KB

Here is another example of a program in the GUI mode:

The above figure displays the memory allocation status for 9 processes from P1 to P9 for three allocation schemes (drawn with light yellow). There is 1 free memory partition (a hole drawn with a light blue background color) with a size of 175KB, which is not enough to allocate to the next process, P10 (request 179KB). The number in the parentheses next to the process ID is the remaining running time of the process in seconds (if it reduces to 0 => the process terminates and releases its memory).

- Display the memory allocation statistics for each execution step (every second):

  + The number of holes (free memory partitions)
  + The average size of the current list of holes
  + The total size of the current list of holes
  + The percentage of total free memory partition over the total memory

Check the above figure for the sample statistics output.

3.4 Testing and Reporting the Results

- After completing the implementation of your program, you may need to test it with different settings in the configuration file.

- Finally, **write a report to discuss your results**.

3.5 Extra Credits

Your team can earn extra credits if you can implement the following extra work:

- (10 pts) Implement the memory compaction function in your program. Your program will compact the set of holes into one larger hole. For example, if you have four separate holes of size 550KB, 375 KB, 1,900 KB, and 4,500 KB, your program will combine these four holes into one large hole of size 7,325 KB. There are several strategies for implementing compaction, one of which is suggested in Section 9.2.3. Be sure to update the beginning address of any processes affected by compaction. You can implement this function by entering a specific key on the Console or clicking a button if you use GUI to develop your application.

4. Project Deliverables

You need to submit the following items:

1. Compress your project's source code, test data (scenario files), test outputs, and comparison results.

2. A DOCX/PDF report (less than ten pages) to summarize the implementation details of your team, discussion, individual contributions of each member, and references.

Submit the two above items to D2L.

5. Grading Rubrics

5.1 Program design and implementation (80 pts)

- 20 pts: Correct implementation of the main program that takes input arguments, starts/pauses/resumes or stops the simulation

- 30 pts: Correct implementation of the three memory allocation algorithms: first-fit, best-fit, and worst-fit

- 20 pts: Correct display of different events in the simulation: memory partitions allocated to a process, free memory from a terminated process, and the memory allocation statistics.

- 10 pts: Good data structures and good coding style and comments

5.2 Report (20 pts)

- 10 pts: Present your simulation results with different configuration settings (at least four)

- 5 pts: Discuss your findings of the simulation and lessons

- 5 pts: Report how your team works together, individual contributions, and references

5.3 Extra Credits (maximum 10 pts if applicable)

- (10 pts) Correctly implement the memory compaction function.

Q&A

None