Software Modeling and Design (SMDL02C)

Memory Deallocation

Simulation

Deallocation Simulation for Fixed-Partitioned and Dynamic Partitioned Sytems.

Documentation

Guides, tutorials, and in-depth examples of the functions and libraries

Contact the author

lanz.vencer@lpunetwork.edu.ph

Software Modeling and Design (SMDL02C)

Memory Deallocation

Table of Contents

Contact the author

lanz.vencer@lpunetwork.edu.ph

Memory Deallocation

Tools and Directories

GitHub

GitHub, Inc. is a provider of Internet hosting for software development and version control using Git. It offers the distributed version control and source code management functionality of Git, plus its own features. All the files encompassing the simulation algorithm can be found on the following GitHub Repository: **https://github.com/VinceVence/deallocation-simulation**

Google Colaboratory

Colaboratory, or "Colab" for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. The Colab link containing the source code used for the simulation can be found on: **https://colab.research.google.com/drive/1uVqW8jmV1NY3yie23zOd5iDJL9NSbbyy?usp=sharing**

Streamlit Website

Streamlit is an open source app framework in Python language. It helps us create web apps for data science and machine learning in a short time. It is compatible with major Python libraries such as scikit-learn, Keras, PyTorch, SymPy(latex), NumPy, pandas, Matplotlib etc. Simulations can be done through this streamlit website link: **https://vincevence-deallocation-simulation-web-app-k9jw7d.streamlitapp.com/**

Contact the author

lanz.vencer@lpunetwork.edu.ph

Prerequisites

Memory Deallocation

Installations

Python 3

In order to run the source code for the simulation, a Python version 3 or higher must be available on your system if running locally. Installation detail can be found here: https://www.python.org/downloads/

NumPy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. Installation details can be found here: https://numpy.org/install/

Pandas

Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. Installation details can be found here: https://pandas.pydata.org/docs/getting_started/install.html

Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. Installation details can be found here: https://matplotlib.org/stable/users/installing/index.html

Contact the author

lanz.vencer@lpunetwork.edu.ph

Memory Deallocation

Introduction

What is Deallocation?

Deallocation is simply defined as freeing an allocated memory space. Deallocating varies per system and the approaches may differ depending on the available Job Status of the memory blocks. A process has to be loaded into the RAM for its execution and remains in the RAM until its completion. Finished processes are deallocated or removed from the memory and new processes are allocated again. This is how the OS works with allocation and deallocation.

For fixed-partition system:

- Straightforward process
- When job completes, Memory Manager resets the status of the job's memory block to "free".
- Any code--for example, binary values with 0 indicating free and 1 indicating busy--may be used.

For dynamic-partition system:

- Algorithm tries to combine free areas of memory whenever possible.
- Three cases:
  - Case 1: When the block to be deallocated is adjacent to another free block.
  - Case 2: When the block to be deallocated is between two free blocks.
  - Case 3: When the block to be deallocated is isolated from other free blocks.

Contact the author

lanz.vencer@lpunetwork.edu.ph

Memory Deallocation

Necessary Dependencies

Importing the Libraries

In this simulation, the dependencies that will be utilized are as follows:

- NumPy - Generating randomized numbers.
- Pandas - Structuring data in a tabular format.
- Matplotlib - Visualization and graphs.
- IPython - Displaying tables.
- Time - Time latency and delays.

Please refer to page 2 of this documentation for further installation details.

Sample Importing in Google Colab

```
# Importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
from IPython.display import display

# Ignore pandas chained assignment warning
pd.options.mode.chained_assignment = None  # default='warn'

# Enable inline visualizations
%matplotlib inline
```

Note: All the source code are written using Python. Running the code using a different Integrated Development Environment (IDE) other than Google Colab or Jupyter Notebooks might require some minor adjustments.

Contact the author

lanz.vencer@lpunetwork.edu.ph

Memory Deallocation

General Preliminaries

Default Variables

The simulation utilizes three defaults that are changeable depending on the context at place. The following are configurable defaults:

- NUM_JOBS - Indicates the size of the memory block.
- RANDOM_SEED - Indicates the random seed used in the simulation
- MAX_MEMORY - Indicates the max memory address limit

Time Latency Function

```
time_deallocation(memory_block_size, alpha, beta)
```

Description:

This is a function that sets up the time latency when deallocating a memory block size. The formula is as follows:

$$(memory\_block\_size \times alpha)/1000 + beta$$

Parameters:

- memory_block_size - allocated bytes of memory for a specific block.
- alpha - normalization variable.
- beta - standard deviation variable.

Return value:

Returns a time latency value that is dependent on the block size:

- (memory_block_size * alpha)/1000 + beta

Contact the author

lanz.vencer@lpunetwork.edu.ph

Memory Deallocation

General Preliminaries

Statistics Function

```
statistics(time_latency, df)
```

Description:

This is a function that provides statistical graphs about the necessary information encompassing deallocation. The graphs provided are:

- Memory Block Size Distribution
- Job Status Distribution
- Time Latency Distribution
- Memory Block Size Distribution

It is recommended to run this function after a certain DataFrame has undergone a partition deallocation process.

Parameters:

- time_latency - a list containing the time latencies from a deallocation function.
- df - a pandas DataFrame containing a Memory Address, Memory Block Size, and Job Status. Note that if the DataFrame did not undergo the deallocation process from any partition systems, the function may produce the wrong results or throw an error.

Return value:

- None

Contact the author

lanz.vencer@lpunetwork.edu.ph

Memory Deallocation

Simulation Functions

Fixed-Partition System Deallocation

This is the oldest and simplest technique used to put more than one process in the main memory. In this partitioning, the number of partitions (non-overlapping) in RAM is fixed but the size of each partition may or may not be the same. As it is a contiguous allocation, hence no spanning is allowed. Here partitions are made before execution or during system configure.

Generate Fixed Partition DataFrame Function

```
generate_fixed_partition_dataframe(num_jobs)
```

Description:
This is a function that generates a memory block containing equally distributed memory address and block size, and randomized job status.

Parameters:

- num_jobs - an integer value that indicates the size of the memory block.

Return value:
Returns a pandas DataFrame with three columns (excluding the index):

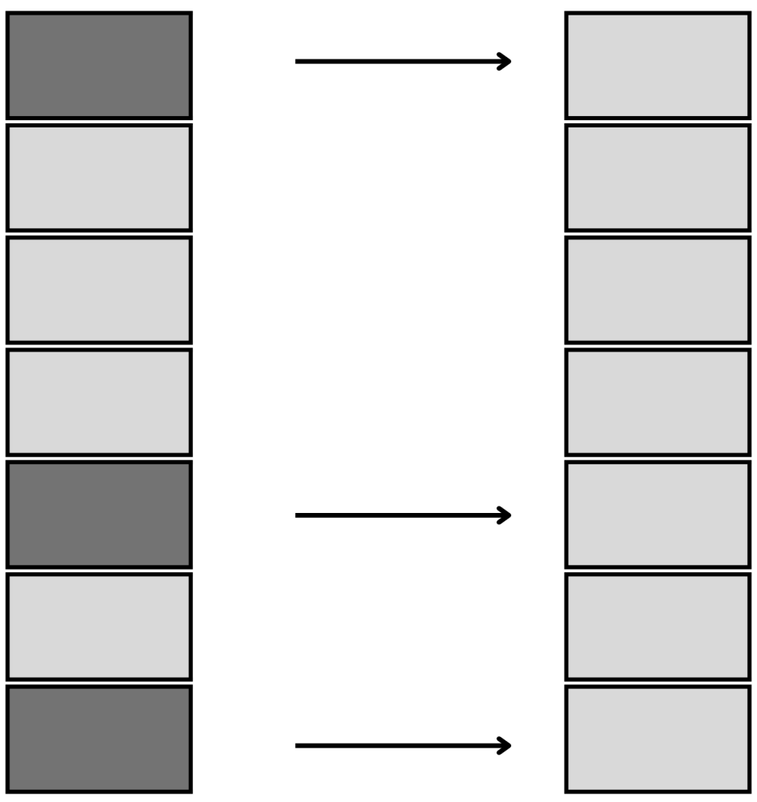
- Memory Address - equally distributed values ranging from 1 to the set MAX_MEMORY.
- Memory Block Size - individual integer blocks with the same value.
- Job Status - randomly designated job status that can either be Free (0) or Busy (1).

Contact the author
lanz.vencer@lpunetwork.edu.ph

Memory Deallocation

Simulation Functions

Fixed-Partition System Deallocation



Fixed-Partition Deallocation

- In a fixed-partition model, each memory block size has the same value.
- Straightforward process.
- When job completes, Memory Manager resets the status of the job's memory block to "free".
- Any code—for example, binary values with 0 indicating free and 1 indicating busy—may be used.

Deallocate Fixed Partition Function

```
deallocate_fixed_partition(df)
```

Description:

This is a function that alters the DataFrame parameter (df) by performing the fixed-partition deallocation algorithm.

Parameters:

- df - fixed-partitioned DataFrame.

Return value:

- time_latency - list of time latencies for every memory block deallocation.

Contact the author

lanz.vencer@lpunetwork.edu.ph

Memory Deallocation

Simulation Functions

Dynamic-Partition System Deallocation

Dynamic partitioning tries to overcome the problems caused by fixed partitioning. In this technique, the partition size is not declared initially. It is declared at the time of process loading.

Generate Fixed Partition DataFrame Function

```
generate_fixed_partition_dataframe(num_jobs)
```

Description:
This is a function that generates a memory block containing randomized distributed memory address, block size, job status.

Parameters:

- num_jobs - an integer value that indicates the size of the memory block.
- case1 - boolean variable for case 1 dynamic deallocation.
- case2 - boolean variable for case 2 dynamic deallocation.
- case3 - boolean variable for case 3 dynamic deallocation.

Return value:
Returns a pandas DataFrame with three columns (excluding the index):

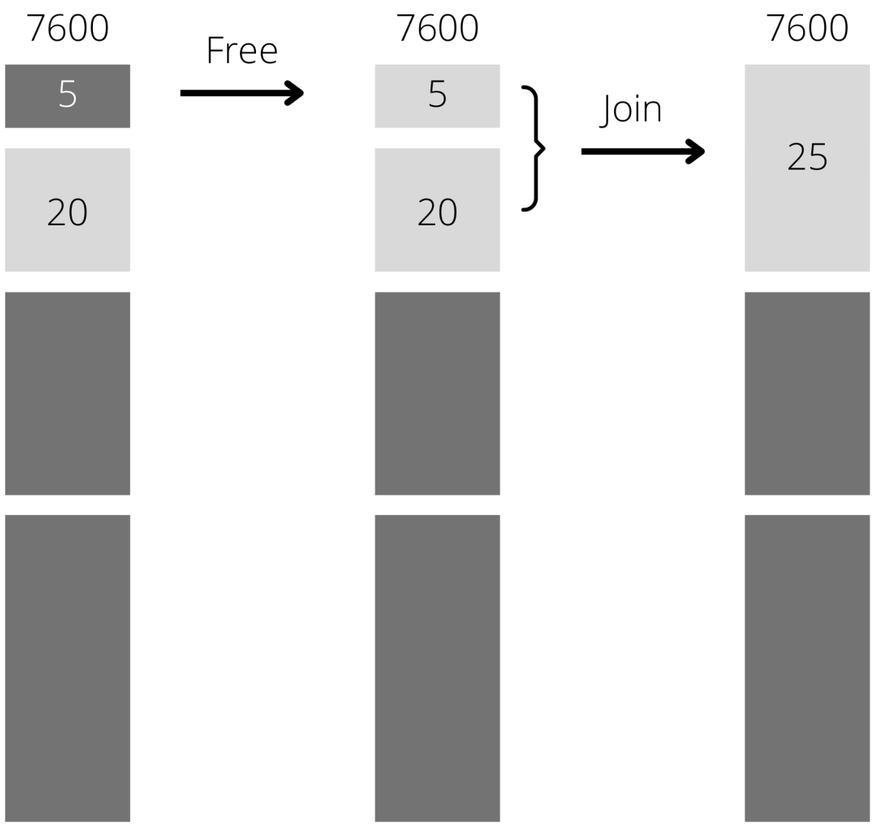
- Memory Address - randomly distributed values ranging from 1 to the set MAX_MEMORY.
- Memory Block Size - Individual integer blocks with randomized values.
- Job Status - randomly designated job status that can either be Free (0) or Busy (1).

Contact the author
lanz.vencer@lpunetwork.edu.ph

Memory Deallocation

Simulation Functions

Dynamic-Partition System Deallocation Case 1



Case 1: Joining Two Free Blocks

- Change list must reflect starting address of the new free block
  - In the example, 7600—which was the address of the first instruction of the job that just released this block
- Memory block size for the new free space must be changed to show its new size—that is, the combined total of the two free partitions
  - In the example, (20 + 5)

Deallocate Dynamic-Partition Function for Case 1

```
deallocate_dynamic_case_1(df, one_iter)
```

Description:

This function takes a memory block DataFrame as an input and simulates a dynamic memory deallocation based on case 1 parameters.

Parameters:

- df - dynamic-partitioned DataFrame.
- one_iter - boolean variable to run the deallocation in just one iteration.

Return value:

- time_latency - list of time latencies for every memory block deallocation.

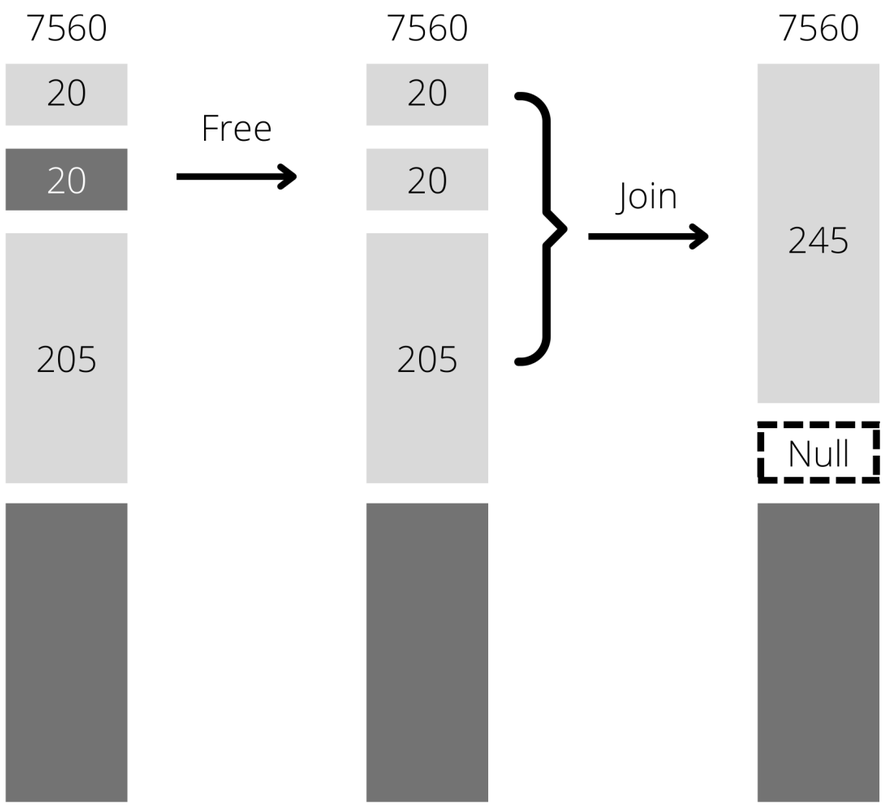
Contact the author

lanz.vencer@lpunetwork.edu.ph

Memory Deallocation

Simulation Functions

Dynamic-Partition System Deallocation Case 2



Case 2: Joining Three Free Blocks

- Deallocated memory space is between two free memory blocks.
- Change list to reflect the starting address of the new free block.
  - In the example, 7560— which was the smallest beginning address.
- Sizes of the three free partitions must be combined.
  - In the example, (20 + 20 + 205)
- Combined entry is given the status of null entry.
  - In the example, any block after 7560

Deallocate Dynamic-Partition Function for Case 2

```
deallocate_dynamic_case_2(df, remove_null)
```

Description:

This function takes a memory block DataFrame as an input and simulates a dynamic memory deallocation based on case 2 parameters.

Parameters:

- df - dynamic-partitioned DataFrame.
- remove_null - boolean variable to remove the null entries.

Return value:

- time_latency - list of time latencies for every memory block deallocation.

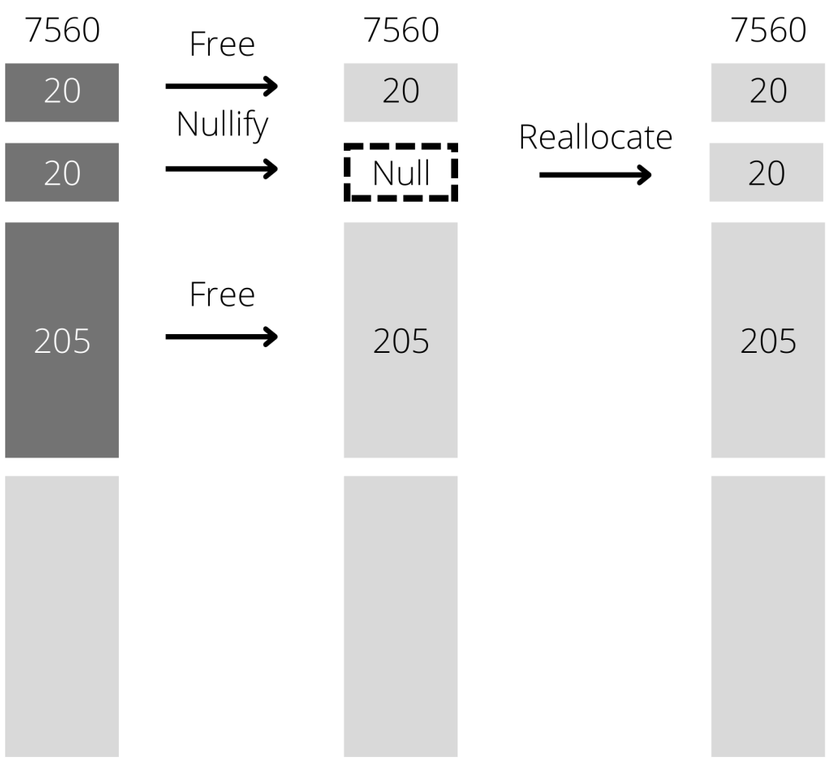
Contact the author

lanz.vencer@lpunetwork.edu.ph

Memory Deallocation

Simulation Functions

Dynamic-Partition System Deallocation Case 3



Case 3: Deallocating an Isolated Block

- Space to be deallocated is isolated from other free areas.
- System learns that the memory block to be released is not adjacent to any free blocks of memory, it is between two other busy areas.
- Must search the table for a null entry.
- Null entry in the busy list occurs when a memory block between two other busy memory blocks is returnedto the free list.

Deallocate Dynamic-Partition Function for Case 3

```
deallocate_dynamic_case_3(df, freeing_latency)
```

Description:

This function takes a memory block DataFrame as an input and simulates a dynamic memory deallocation based on case 3 parameters.

Parameters:

- df - dynamic-partitioned DataFrame.
- freeing_latency - integer that determines the time to free null entries.

Return value:

- time_latency - list of time latencies for every memory block deallocation.

Contact the author

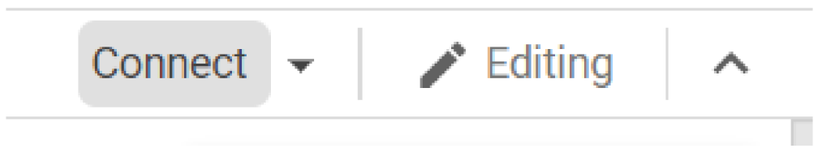
lanz.vencer@lpunetwork.edu.ph

Memory Deallocation

Simulation Examples

Connect to a Google Colab instance

1. You need to have a Google account to access the Colab Notebooks when running the simulation.
2. Go to the following link: https://colab.research.google.com/drive/1uVqW8jmV1NY3yie23zOd5iDJL9NSbbyy?usp=sharing.
3. Upon accessing the link, you should be able to see a connect button in the *upper right corner* of your screen. Click the connect button. If a sign in option prompts, you should sign in using your Google account.



You should now be connected to a Central Processing Unity (CPU) instance within Google Colab. To verify that you have successfully connected you should be able to see the following on the upper right corner of your screen:

Running code within Google Colab

To run code within Google Colab, you have two options: You could run code individually per cell or you could run all cells at once.

1. To run code per cell, hold Shift then press Enter - **Shift + Enter**
2. To run all cells at once, within the menu bar go to "**Runtime**" then click "**run all**". You could also press Ctrl + F9 as a keyboard shortcut.

Note: When running cells individually, it is recommended to run from top to bottom to avoid undeclared variables and functions.

Contact the author

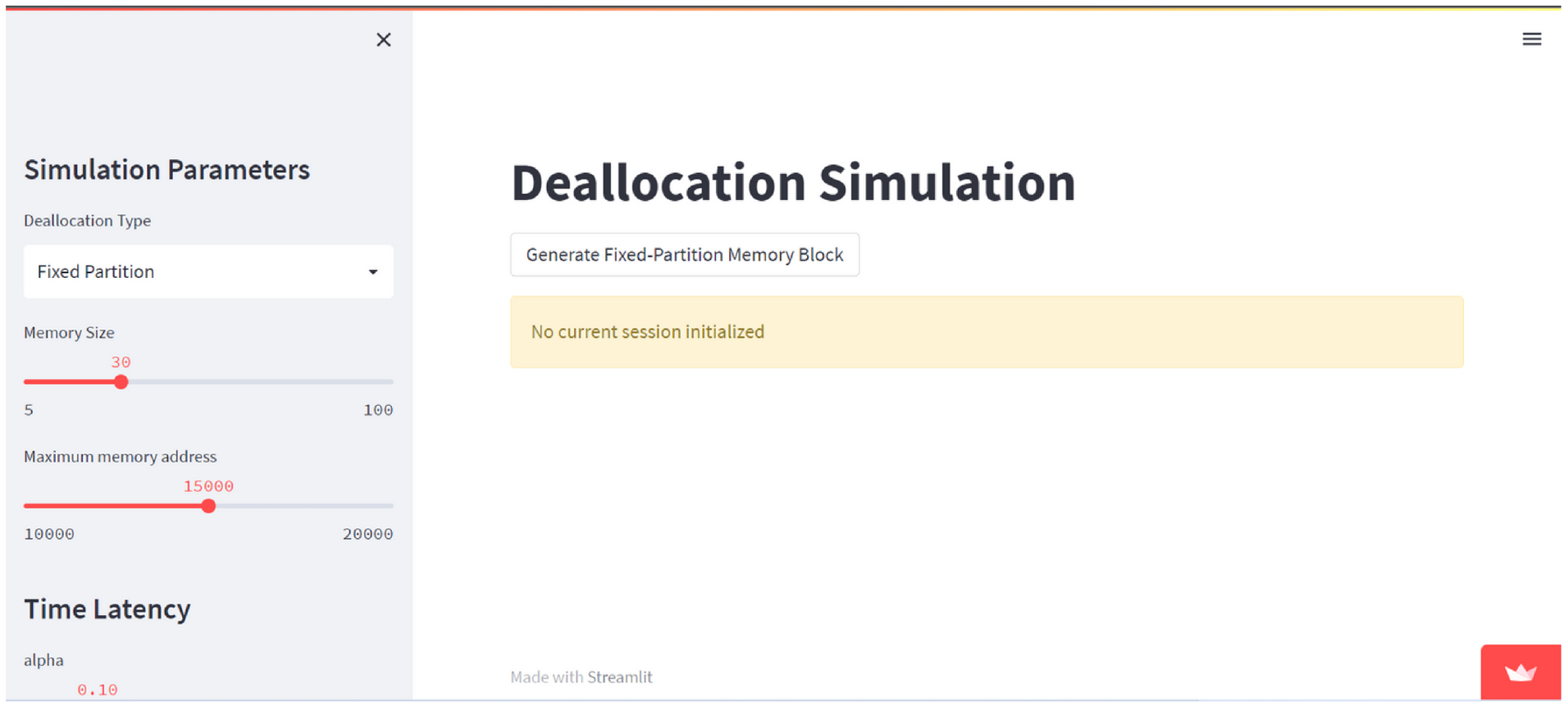
lanz.vencer@lpunetwork.edu.ph

Memory Deallocation

Simulation Examples

Simulate through Streamlit

1. Go the following link: https://vincevence-deallocation-simulation-web-app-k9jw7d.streamlitapp.com/
2. Upon clicking the link, you should be able to see the following user interface:



3. You may adjust the deallocation type of your choice or generate a new memory block by clicking the "Generate Fixed-Partition Memory Block" button. Other simulation parameters that you could adjust are:

- Memory Size
- Maximum memory address
- Time latency
  - alpha value
  - beta value

Contact the author

lanz.vencer@lpunetwork.edu.ph