CONTIGUOUS MEMORY ALLOCATION – VARIABLE SIZE PARTITIONING ALGORITHM

MINI PROJECT REPORT

Submitted by

M. Ragavan(202009034)
S. Siddarth(202009038)
J. Vishnu Sankar(202009046)

in

19AD481 – OPERATING SYSTEM PRINCIPLES

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

MEPCO SCHLENK ENGINEERING COLLEGE

SIVAKASI

MAY 2022

MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

AUTONOMOUS

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

BONAFIDE CERTIFICATE

This is to certify that it is the bonafide work of **"M.Ragavan (202009034), S. Siddarth (202009038) AND J. Vishnu Sankar (202009046)"** for the mini project titled "C**ONTIGUOUS MEMORY ALLOCATION – VARIABLE SIZE PARTITIONING ALGORITHM**" in 19AD481 – Operating System Principles during the fourth semester January 2022 – May 2022 under my supervision.

SIGNATURE

Mrs. P. Swathika,
Asst. Professor (Senior Grade),
AI&DS Department,
Mepco Schlenk Engg. College, Sivakasi

SIGNATURE

Dr. J. Angela Jennifa Sujana,
Asso. Professor (Senior Grade)& Head,
AI&DS Department
Mepco Schlenk Engg. College, Sivakasi

ABSTRACT

Our project is to determine memory management using bestfit, worstfit, firstfit using gui in python. The objective is to show the memory allocation of the major three techniques like bestfit, worstfit, bestfit.

TABLE OF CONTENTS

Introduction:

Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status. The process address space is the set of logical addresses that a process references in its code. For example, when 32-bit addressing is in use, addresses can range from 0 to 0x7fffffff; that is, 2^31 possible numbers, for a total theoretical size of 2 gigabytes. Virtual and physical addresses are the same in compile-time and load-time address-binding schemes. Virtual and physical addresses differ in execution-time address-binding scheme. The set of all logical addresses generated by a program is referred to as a logical address space. The set of all physical addresses corresponding to these logical addresses is referred to as a physical address space.

Objective:

The term Memory can be defined as a collection of data in a specific format. It is used to store instructions and processed data. The memory comprises a large array or group of words or bytes, each with its own location. The primary motive of a computer system is to execute programs. These programs, along with the information they access, should be in the main memory during execution. The CPU fetches instructions from memory according to the value of the program counter. To achieve a degree of multiprogramming and proper utilization of memory, memory management is important. Many memory management methods exist, reflecting various approaches, and the effectiveness of each algorithm depends on the situation. In a multiprogramming computer, the operating system resides in a part of memory and the rest is used by multiple processes. The task of subdividing the memory among different processes is called memory management. Memory management is a method in the operating system to manage operations between main memory and disk during process execution. The main aim of memory management is to achieve efficient utilization of memory.

Scope of the project:

- It allows you to check how much memory needs to be allocated to processes that decide which processor should get memory at what time.
- Tracks whenever inventory gets freed or unallocated. According to it will update the status.
- It allocates the space to application routines.
- It also make sure that these applications do not interfere with each other.
- Helps protect different processes from each other
- It places the programs in memory so that memory is utilized to its full extent.

Best-Fit:

This method keeps the free/busy list in order by size – smallest to largest. In this method, the operating system first searches the whole of the memory according to the size of the given job and allocates it to the closest-fitting free partition in the memory, making it able to use memory efficiently. Here the jobs are in the order from smallest job to largest job.

Job Number	Memory Requested
J1	20 K
J2	200 K
J3	500 K
J4	50 K

Memory location	Memory block size	Job number	Job size
10567	30 K	J1	20 K
30457	50 K	J4	50 K
300875	200 K	J2	200 K
809567	700 K	J3	500 K
Total available :	980 K	Total used :	770 K

Fig. 1.1 (Best Fit)

Worst-Fit:

In this allocation technique, the process traverses the whole memory and always search for the largest hole/partition, and then the process is placed in that hole/partition. It is a slow process because it has to traverse the entire memory to search the largest hole.

Process Number	Process Size
P1	30K
P2	100K
P3	45K

MEMORY LOCATION	MEMORY BLOCK SIZE	PROCESS NUMBER	PROCESS SIZE	STATUS
12345	50K	P3	45K	Busy
45871	100K	P2	100K	Busy
1245	400K	P1	30K	Busy
TOTAL AVAILABLE:	550K	TOTAL USED:	175K	

Fig. 1.2 (Worst Fit)

First-Fit:

This method keeps the free/busy list of jobs organized by memory location, low-ordered to high-ordered memory. In this method, first job claims the first available memory with space more than or equal to it's size. The operating system doesn't search for appropriate partition but just allocate the job to the nearest memory partition available with sufficient size.

Job Number	Memory Requested
J1	20 K
J2	200 K
J3	500 K
J4	50 K

Memory location	Memory block size	Job number	Job size
10567	200 K	J1	20 K
30457	30 K		
300875	700 K	J2	200 K
809567	50 K	J4	50 K
Total available :	980 K	Total used :	270 K

Fig. 1.3 (First Fit)

Algorithm:

Best Fit:

1- Input memory blocks and processes with sizes.

2- Initialize all memory blocks as free.

3- Start by picking each process and find the
minimum block size that can be assigned to
current process i.e., find min(bockSize[1],
blockSize[2],.....blockSize[n]) >
processSize[current], if found then assign
it to the current process.

5- If not then leave that process and keep checking
the further processes.

Worst Fit:

1- Input memory blocks and processes with sizes.

2- Initialize all memory blocks as free.

3- 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.

5- If not then leave that process and keep checking
the further processes.

First Fit:

1- Input memory blocks with size and processes with size.

2- Initialize all memory blocks as free.

3- Start by picking each process and check if it can

be assigned to current block.

4- If size-of-process <= size-of-block if yes then

assign and check for next process.

5- If not then keep checking the further blocks.

Program Coding:

```
import tkinter as tk
from tkinter import *
import tkinter.ttk as ttk
import tkinter.messagebox as tkMessageBox
global a,b,c
global new
global new1
global new2
global processSize
global processNo
processNo=[1,2,3,4]
processSize = [212, 417, 112, 426]
new=[]
new1=[]
new2=[]
newline="\n"
def DisplayForm(new):
  root=Tk()
  root.geometry("250x170")
  T = Text(root, height = 500, width = 500)
  l = Label(root, text ="Best Fit")
  T.insert(tk.END,processNo)
  T.insert(tk.END,newline)
  T.insert(tk.END,processSize)
  T.insert(tk.END,newline)
  T.insert(tk.END,new)
  l.config(font =("Courier", 14))
  b2 = Button(root, text = "Exit",command = root.destroy)
  l.pack()
  T.pack()
```

```
def DisplayForm1(new1):
  root=Tk()
  root.geometry("250x170")
  T = Text(root, height = 500, width = 500)
  l = Label(root, text ="Worst Fit")
  T.insert(tk.END,processNo)
  T.insert(tk.END,newline)
  T.insert(tk.END,processSize)
  T.insert(tk.END,newline)
  T.insert(tk.END,new1)
  l.config(font =("Courier", 14))
  b2 = Button(root, text = "Exit",command = root.destroy)
  l.pack()
  T.pack()
def DisplayForm2(new2):
  root=Tk()
  root.geometry("250x170")
  T = Text(root, height = 500, width = 500)
  l = Label(root, text ="First Fit")
  T.insert(tk.END,processNo)
  T.insert(tk.END,newline)
  T.insert(tk.END,processSize)
  T.insert(tk.END,newline)
  T.insert(tk.END,new2)
  l.config(font =("Courier", 14))
  b2 = Button(root, text = "Exit",command = root.destroy)
  l.pack()
  T.pack()
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)
                a=allocation[i] + 1
                new.append(a)
            else:
                print("Not Allocated")
                new.append("Not Allocated")
        DisplayForm(new)
def BestFit():
    blockSize = [100, 500, 200, 300, 600]
    processSize = [212, 417, 112, 426]
    m = len(blockSize)
    n = len(processSize)
    bestFit(blockSize, m, processSize, n)
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)
                        b=allocation[i] + 1
                        new1.append(b)
                else:
                        print("Not Allocated")
                        new1.append("Not Allocated")
        DisplayForm1(new1)
def WorstFit():
   blockSize = [100, 500, 200, 300, 600]
   processSize = [212, 417, 112, 426]
   m = len(blockSize)
   n = len(processSize)
   worstFit(blockSize, m, processSize, n)
def firstFit(blockSize, m, processSize, n):
        allocation = [-1] * n
        for i in range(n):
                for j in range(m):
```

```
                    if blockSize[j] >= processSize[i]:
                        allocation[i] = j
                        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)
                c=allocation[i] + 1
                new2.append(c)
            else:
                print("Not Allocated")
                new2.append("Not Allocated")
        DisplayForm2(new2)
def FirstFit():
    blockSize = [100, 500, 200, 300, 600]
    processSize = [212, 417, 112, 426]
    m = len(blockSize)
    n = len(processSize)
    firstFit(blockSize, m, processSize, n)
def open():
    login_screen=Tk()
    login_screen.title("Select a partition")
    login_screen.geometry("350x300")
    Label(login_screen, text="").pack()
    Button(login_screen, text="BestFit", width=10, height=1, command=BestFit).pack()
    Label(login_screen, text="").pack()
    Button(login_screen, text="WorstFit", width=10, height=1, command=WorstFit).pack()
    Label(login_screen, text="").pack()
    Button(login_screen, text="FirstFit", width=10, height=1, command=FirstFit).pack()
```

open()

mainloop()

Output:

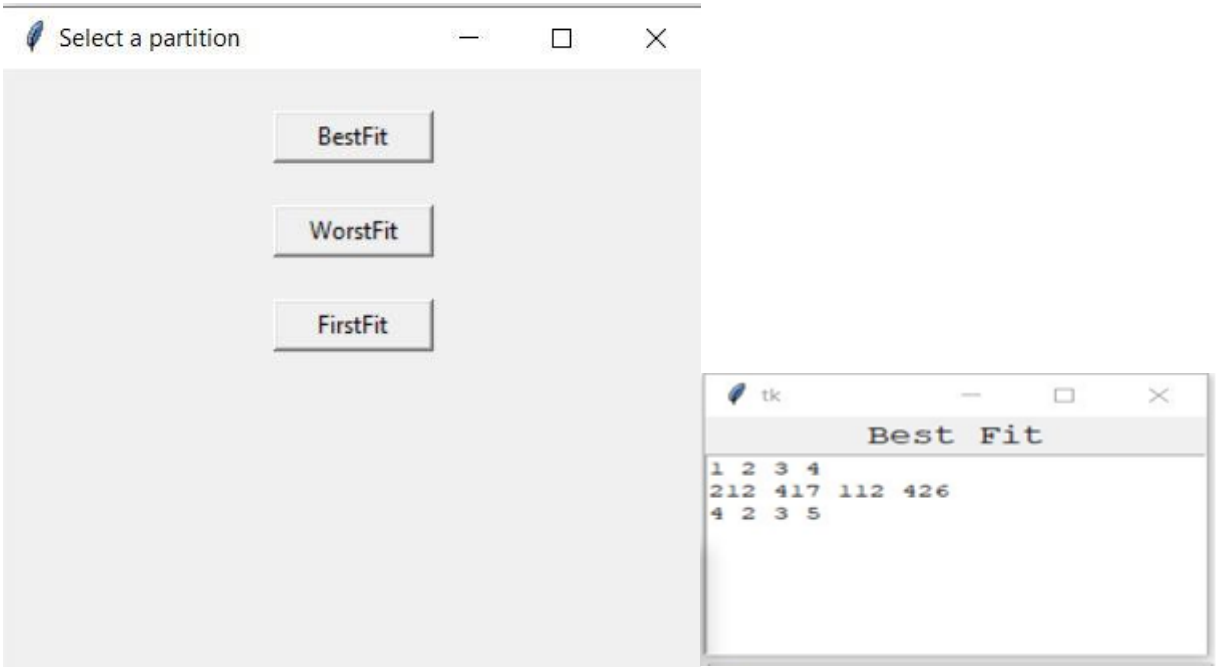

Fig. 1.4 (Best Fit)

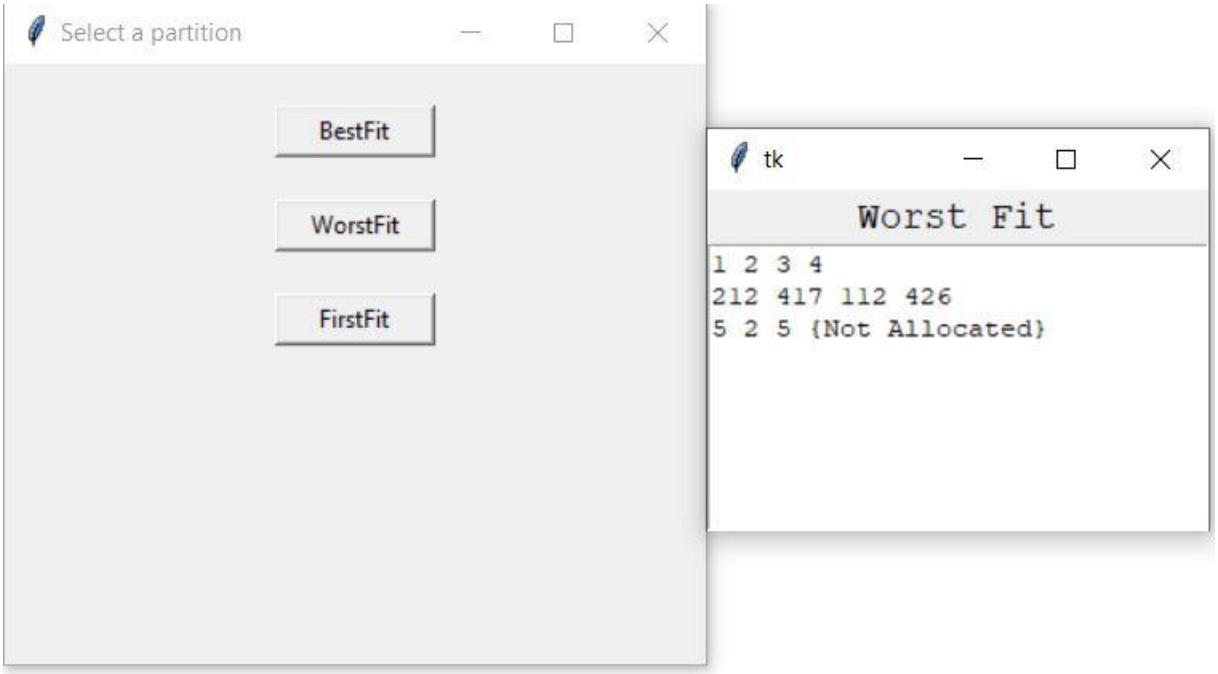

Fig. 1.5 (Worst Fit)

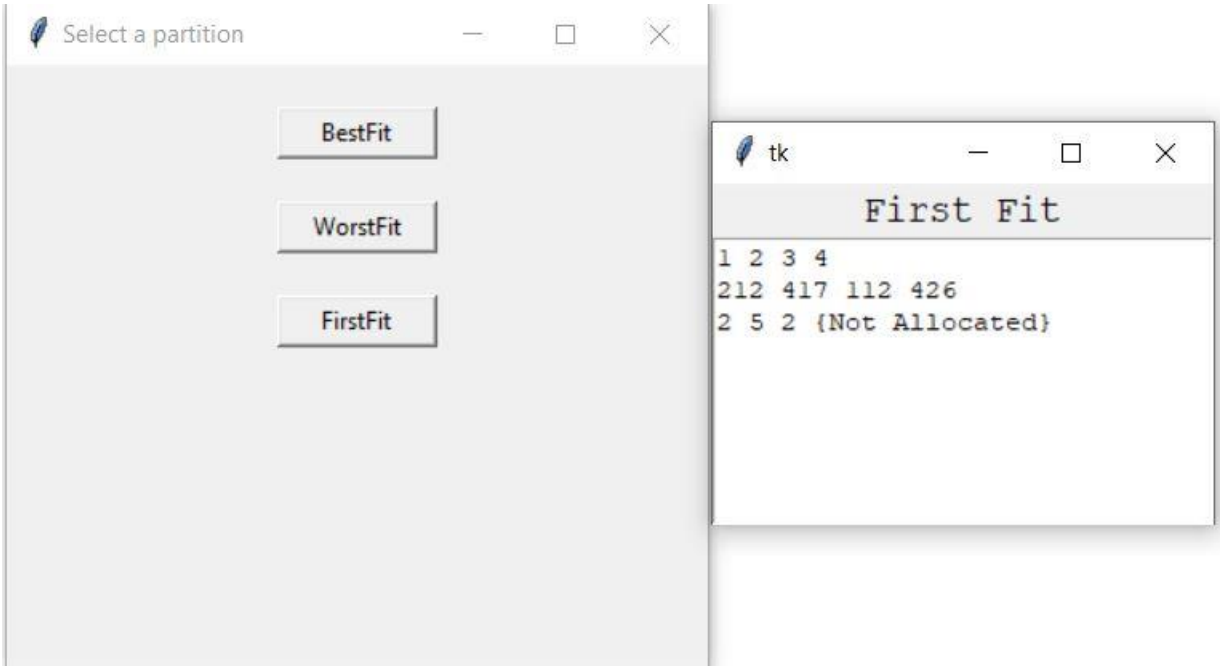

Fig. 1.6 (First Fit)

Conclusion:

Firstly we created a gui and assigned button for each partitions .While executing each fit we have passed the size of process and blocks. Then we got the final output in a text area in a new window.

References:

https://www.geeksforgeeks.org/memory-management-in-operating-system/

https://www.tutorialspoint.com/operating_system/os_memory_management.htm

https://www.javatpoint.com/memory-management-operating-system

https://www.guru99.com/os-memory-management.html