**Assignment Number 06**

**Name:** Mihir Unmesh Patil

**Roll NO**: TYCOC213

**Batch:** C/C-3

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

typedef *struct* MemoryBlock {

*int* start\_address;

*int* size;

    bool is\_free;

*struct* MemoryBlock\* next;

} MemoryBlock;

MemoryBlock\* next\_fit\_pointer = NULL;

MemoryBlock\* createMemoryBlock(*int* *start\_address*, *int* *size*, bool *is\_free*) {

    MemoryBlock\* block = (MemoryBlock\*)malloc(sizeof(MemoryBlock));

    block->start\_address = *start\_address*;

    block->size = *size*;

    block->is\_free = *is\_free*;

    block->next = NULL;

    return block;

}

*void* updateStartAddresses(MemoryBlock\* *head*) {

*int* address = 0;

    while (*head*) {

*head*->start\_address = address;

        address += *head*->size;

*head* = *head*->next;

    }

}

*void* displayMemory(MemoryBlock\* *head*) {

    printf("\n=== Memory Status ===\n");

    while (*head*) {

        printf("Start: %d | Size: %d | Free: %s\n",

*head*->start\_address, *head*->size,

*head*->is\_free ? "Yes" : "No");

*head* = *head*->next;

    }

    printf("=====================\n");

}

*int* allocateMemory(MemoryBlock\*\* *head*, *int* *process\_size*, *char* *method*) {

    MemoryBlock \*selected = NULL, \*current = \**head*, \*start = NULL;

*int* selected\_size = (*method* == 'B') ? \_\_INT\_MAX\_\_ : -1;

    if (*method* == 'N') {

        if (!next\_fit\_pointer) next\_fit\_pointer = \**head*;

        start = current = next\_fit\_pointer;

    }

    bool looped = false;

    do {

        if (current->is\_free && current->size >= *process\_size*) {

            if (*method* == 'F') {

                selected = current;

                break;

            } else if (*method* == 'B' && current->size < selected\_size) {

                selected = current;

                selected\_size = current->size;

            } else if (*method* == 'W' && current->size > selected\_size) {

                selected = current;

                selected\_size = current->size;

            } else if (*method* == 'N') {

                selected = current;

                break;

            }

        }

        current = (*method* == 'N') ?

                 (current->next ? current->next : \**head*) :

                 current->next;

        if (*method* == 'N' && current == start) looped = true;

    } while ((*method* != 'N' && current) || (*method* == 'N' && !looped));

    if (!selected) return -1;

*int* start\_address = selected->start\_address;

    if (selected->size == *process\_size*) {

        selected->is\_free = false;

    } else {

        MemoryBlock\* new\_block = createMemoryBlock(0, selected->size - *process\_size*, true);

        new\_block->next = selected->next;

        selected->next = new\_block;

        selected->size = *process\_size*;

        selected->is\_free = false;

        updateStartAddresses(\**head*);

    }

    if (*method* == 'N') {

        next\_fit\_pointer = selected->next ? selected->next : \**head*;

    }

    return start\_address;

}

bool freeBlock(MemoryBlock\*\* *head*, *int* *start\_address*) {

    MemoryBlock \*current = \**head*, \*prev = NULL;

    while (current) {

        if (current->start\_address == *start\_address*) {

            current->is\_free = true;

            if (current->next && current->next->is\_free) {

                MemoryBlock\* temp = current->next;

                current->size += temp->size;

                current->next = temp->next;

                free(temp);

            }

            if (prev && prev->is\_free) {

                prev->size += current->size;

                prev->next = current->next;

                free(current);

                current = prev;

            }

            updateStartAddresses(\**head*);

            return true;

        }

        prev = current;

        current = current->next;

    }

    return false;

}

*void* deallocateAll(MemoryBlock\* *head*) {

    while (*head*) {

        MemoryBlock\* temp = *head*;

*head* = *head*->next;

        free(temp);

    }

}

*int* main() {

*int* total\_memory, num\_blocks, method\_choice;

*char* method\_char;

    printf("Enter total memory size: ");

    scanf("%d", &total\_memory);

    printf("Enter number of free blocks: ");

    scanf("%d", &num\_blocks);

*int*\* block\_sizes = (*int*\*)malloc(num\_blocks \* sizeof(*int*));

    printf("Enter sizes of the %d free blocks:\n", num\_blocks);

    for (*int* i = 0; i < num\_blocks; i++) {

        printf("Block %d: ", i + 1);

        scanf("%d", &block\_sizes[i]);

    }

    MemoryBlock \*memory = NULL, \*last = NULL;

    for (*int* i = 0; i < num\_blocks; i++) {

        MemoryBlock\* block = createMemoryBlock(0, block\_sizes[i], true);

        if (!memory) {

            memory = block;

            last = block;

        } else {

            last->next = block;

            last = block;

        }

    }

    updateStartAddresses(memory);

    free(block\_sizes);

    printf("\nChoose allocation method:\n");

    printf("1. First Fit\n2. Best Fit\n3. Worst Fit\n4. Next Fit\n");

    printf("Enter choice: ");

    scanf("%d", &method\_choice);

    switch (method\_choice) {

        case 1: method\_char = 'F'; break;

        case 2: method\_char = 'B'; break;

        case 3: method\_char = 'W'; break;

        case 4: method\_char = 'N'; break;

        default: printf("Invalid method\n"); return 1;

    }

*int* choice;

    while (1) {

        printf("\n=== Menu ===\n");

        printf("1. Allocate process\n");

        printf("2. Free memory block\n");

        printf("3. Display memory\n");

        printf("0. Exit\n");

        printf("Enter choice: ");

        scanf("%d", &choice);

        if (choice == 0) break;

*int* size, addr;

        switch (choice) {

            case 1:

                printf("Enter process size to allocate: ");

                scanf("%d", &size);

                addr = allocateMemory(&memory, size, method\_char);

                if (addr == -1)

                    printf("Allocation failed: Not enough memory.\n");

                else

                    printf("Process allocated at address %d\n", addr);

                break;

            case 2:

                printf("Enter start address of block to free: ");

                scanf("%d", &addr);

                if (freeBlock(&memory, addr))

                    printf("Freed memory block at address %d\n", addr);

                else

                    printf("Invalid address or already free.\n");

                break;

            case 3:

                displayMemory(memory);

                break;

            default:

                printf("Invalid option.\n");

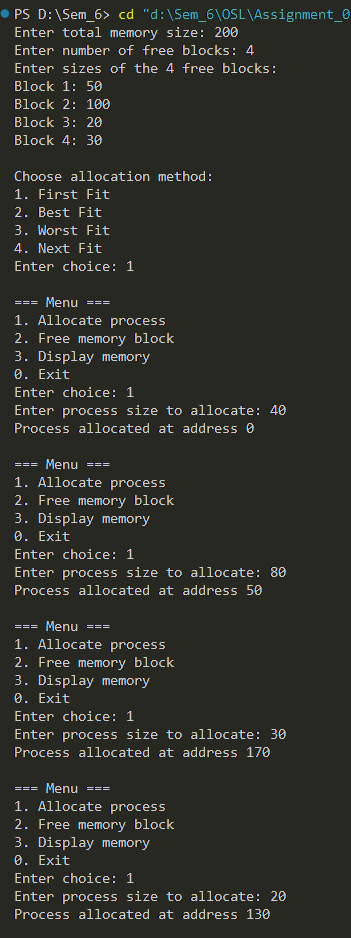
        }

    }

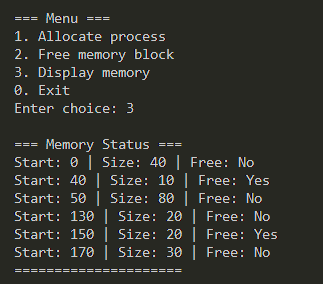
    deallocateAll(memory);

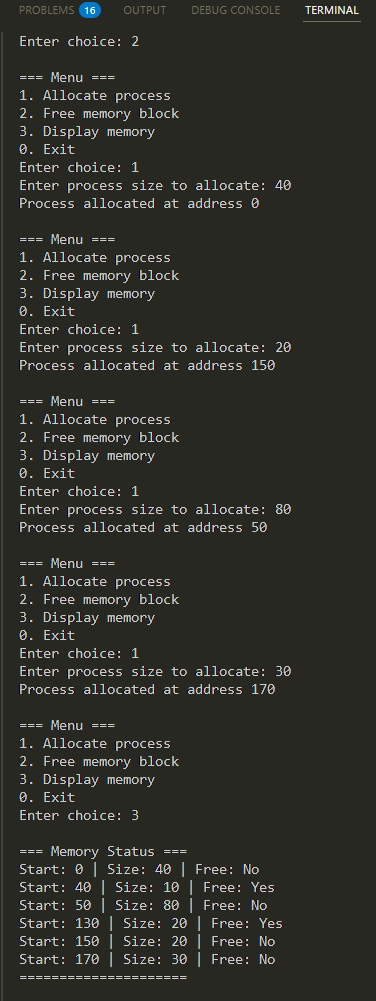
    return 0;

}

OUTPUT:

First Fit:



Best Fit: Worst Fit: Next Fit:

