EXPERIMENT-13

Program :

```
#include <stdio.h>
#include <stdlib.h>

struct Block {
    int size;
    int isFree;
    int pid;
    struct Block *prev, *next;
};

struct Block *head = NULL;

// Create a new memory block
struct Block* createBlock(int size) {
    struct Block *newBlock = (struct Block*)malloc(sizeof(struct Block));
    newBlock->size = size;
    newBlock->isFree = 1;
    newBlock->pid = -1;
    newBlock->prev = newBlock->next = NULL;
    return newBlock;
}

// Insert a block at the end
void insertBlock(int size) {
    struct Block *newBlock = createBlock(size);
    if (head == NULL) {
        head = newBlock;
        return;
    }
    struct Block *temp = head;
    while (temp->next != NULL)
        temp = temp->next;
```

```
    temp→next = newBlock;
    newBlock→prev = temp;
}

// Display memory blocks
void displayMemory() {
    struct Block *temp = head;
    printf("\nMemory Blocks:\n");
    while (temp ≠ NULL) {
        if (temp→isFree)
            printf("[ Free | Size: %d ] ⟷ ", temp→size);
        else
            printf("[ P%d   | Size: %d ] ⟷ ", temp→pid, temp→size);
        temp = temp→next;
    }
    printf("NULL\n");
}

// Merge consecutive free blocks
void garbageCollector() {
    struct Block *temp = head;
    while (temp ≠ NULL && temp→next ≠ NULL) {
        if (temp→isFree && temp→next→isFree) {
            temp→size += temp→next→size;
            struct Block *del = temp→next;
            temp→next = del→next;
            if (del→next ≠ NULL)
                del→next→prev = temp;
            free(del);
        } else {
            temp = temp→next;
        }
    }
}

// Allocate memory using chosen method
void allocate(int pid, int size, int choice) {
```

```
struct Block *temp = head, *selected = NULL;

if (choice == 1) { // First Fit
    while (temp != NULL) {
        if (temp->isFree && temp->size >= size) {
            selected = temp;
            break;
        }
        temp = temp->next;
    }
} else if (choice == 2) { // Best Fit
    int bestSize = 1e9;
    while (temp != NULL) {
        if (temp->isFree && temp->size >= size && temp->size < bestSize) {
            bestSize = temp->size;
            selected = temp;
        }
        temp = temp->next;
    }
} else if (choice == 3) { // Worst Fit
    int worstSize = -1;
    while (temp != NULL) {
        if (temp->isFree && temp->size >= size && temp->size > worstSize) {
            worstSize = temp->size;
            selected = temp;
        }
        temp = temp->next;
    }
}

if (selected == NULL) {
    printf("Process %d of size %d cannot be allocated\n", pid, size);
    return;
}

// Split block if necessary
if (selected->size > size) {
```

```
        struct Block *newBlock = createBlock(selected->size - size);
        newBlock->next = selected->next;
        if (selected->next != NULL)
            selected->next->prev = newBlock;
        newBlock->prev = selected;
        selected->next = newBlock;
        selected->size = size;
    }

    selected->isFree = 0;
    selected->pid = pid;
    printf("Process %d allocated %d units\n", pid, size);
}

// Free a process and merge free blocks
void freeProcess(int pid) {
    struct Block *temp = head;
    int found = 0;

    while (temp != NULL) {
        if (temp->pid == pid) {
            temp->isFree = 1;
            temp->pid = -1;
            found = 1;
            printf("Process %d freed\n", pid);
            break;
        }
        temp = temp->next;
    }

    if (!found)
        printf("Process %d not found\n", pid);

    garbageCollector();
}

// Main function
```

```
int main() {
    int n, choice, size, pid = 1;

    printf("Enter number of memory blocks: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        printf("Enter size of block %d: ", i + 1);
        scanf("%d", &size);
        insertBlock(size);
    }

    while (1) {
        printf("\nMenu:\n");
        printf("1. Display Memory\n");
        printf("2. Allocate Process\n");
        printf("3. Free Process\n");
        printf("4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            displayMemory();
        } else if (choice == 2) {
            printf("Enter process size: ");
            scanf("%d", &size);
            printf("Choose allocation method: 1.First Fit 2.Best Fit 3.Worst Fit :
");
            int method;
            scanf("%d", &method);
            allocate(pid++, size, method);
        } else if (choice == 3) {
            printf("Enter process ID to free: ");
            scanf("%d", &size);
            freeProcess(size);
        } else if (choice == 4) {
            break;
        } else {
```

```
            printf("Invalid choice\n");
        }
    }

    return 0;
}
```

Output :

```
PS C:\Users\there> cd downloads/Alwin
PS C:\Users\there\downloads\Alwin> gcc allocation.c
PS C:\Users\there\downloads\Alwin> ./a.exe
Enter number of memory blocks: 3
Enter size of block 1: 100
Enter size of block 2: 200
Enter size of block 3: 300

Menu:
1. Display Memory
2. Allocate Process
3. Free Process
4. Exit
Enter choice: 1

Memory Blocks:
[ Free | Size: 100 ] <-> [ Free | Size: 200 ] <-> [ Free | Size: 300 ] <-> NULL

Menu:
1. Display Memory
2. Allocate Process
3. Free Process
4. Exit
Enter choice: 2
Enter process size: 3
Choose allocation method: 1.First Fit 2.Best Fit 3.Worst Fit : 2
Process 1 allocated 3 units


Menu:
1. Display Memory
2. Allocate Process
3. Free Process
4. Exit
Enter choice: 1

Memory Blocks:
[ P1   | Size: 3 ] <-> [ Free | Size: 97 ] <-> [ Free | Size: 200 ] <-> [ Free | Size: 300 ] <-> NULL

Menu:
1. Display Memory
2. Allocate Process
3. Free Process
4. Exit
Enter choice: 1

Memory Blocks:
[ P1   | Size: 3 ] <-> [ Free | Size: 97 ] <-> [ Free | Size: 200 ] <-> [ Free | Size: 300 ] <-> NULL

Menu:
1. Display Memory
2. Allocate Process
3. Free Process
4. Exit
Enter choice: 2
Enter process size: 1
Choose allocation method: 1.First Fit 2.Best Fit 3.Worst Fit : 3
Process 2 allocated 1 units
```

```
Menu:
1. Display Memory
2. Allocate Process
3. Free Process
4. Exit
Enter choice: 1

Memory Blocks:
[ P1   | Size: 3 ] <-> [ Free | Size: 97 ] <-> [ Free | Size: 200 ] <-> [ P2   | Size: 1 ] <-> [ Free | Size: 299 ] <-> NULL

Menu:
1. Display Memory
2. Allocate Process
3. Free Process
4. Exit
Enter choice: 3
Enter process ID to free: 2
Process 2 freed

Menu:
1. Display Memory
2. Allocate Process
3. Free Process
4. Exit
Enter choice: 3
Enter process ID to free: 4
Process 4 not found

Menu:
1. Display Memory
2. Allocate Process
3. Free Process
4. Exit


Enter choice: 3
Enter process ID to free: 4
Process 4 not found

Menu:
1. Display Memory
2. Allocate Process
3. Free Process
4. Exit
Enter choice: 1

Memory Blocks:
[ P1   | Size: 3 ] <-> [ Free | Size: 597 ] <-> NULL

Menu:
1. Display Memory
2. Allocate Process
3. Free Process
4. Exit
Enter choice: 3
Enter process ID to free: 3
Process 3 not found

Menu:
1. Display Memory
2. Allocate Process
3. Free Process
4. Exit
Enter choice: 1

Memory Blocks:
[ P1   | Size: 3 ] <-> [ Free | Size: 597 ] <-> NULL


Menu:
1. Display Memory
2. Allocate Process
3. Free Process
4. Exit
Enter choice: 4
PS C:\Users\there\downloads\Alwin>
```