

**Program:**

#include <stdio.h>

#include <limits.h>

struct MemoryBlock {

int size;

int allocated;

int processID;

int remaining;

};

void displayResults(struct MemoryBlock partitions[], int numPartitions, int processes[], int numProcesses) {

printf("\nProcess Allocation:\n");

for (int i = 0; i < numPartitions; ++i) {

if (partitions[i].allocated) {

printf("Partition %d (Size: %d) -> Process %d (Size: %d)\n",

i + 1, partitions[i].size, partitions[i].processID, processes[partitions[i].processID - 1]);

if (partitions[i].remaining > 0) {

printf(" Hole created: %d\n", partitions[i].remaining);

int holeUsed = 0;

for (int j = 0; j < numProcesses; ++j) {

if (processes[j] <= partitions[i].remaining && processes[j] != -1) {

printf(" \* Hole can be utilized by Process %d (Size: %d)\n", j + 1, processes[j]);

holeUsed = 1;

}

}

if (!holeUsed) printf(" \* No process fits in the hole.\n");

}

} else {

printf("Partition %d (Size: %d) -> Unallocated\n", i + 1, partitions[i].size);

}

}

}

void resetAllocations(struct MemoryBlock partitions[], int numPartitions) {

for (int i = 0; i < numPartitions; ++i) {

partitions[i].allocated = 0;

partitions[i].processID = -1;

partitions[i].remaining = 0;

}

}

void firstFit(struct MemoryBlock partitions[], int numPartitions, int processes[], int numProcesses) {

resetAllocations(partitions, numPartitions);

for (int i = 0; i < numProcesses; ++i) {

int allocated = 0;

for (int j = 0; j < numPartitions; ++j) {

if (!partitions[j].allocated && partitions[j].size >= processes[i]) {

partitions[j].allocated = 1;

partitions[j].processID = i + 1;

partitions[j].remaining = partitions[j].size - processes[i];

allocated = 1;

break;

}

}

if (!allocated) {

printf("Process %d (Size: %d) -> Unallocated\n", i + 1, processes[i]);

}

}

displayResults(partitions, numPartitions, processes, numProcesses);

}

void bestFit(struct MemoryBlock partitions[], int numPartitions, int processes[], int numProcesses) {

resetAllocations(partitions, numPartitions);

for (int i = 0; i < numProcesses; ++i) {

int bestIdx = -1;

int minSize = INT\_MAX;

for (int j = 0; j < numPartitions; ++j) {

if (!partitions[j].allocated && partitions[j].size >= processes[i] && partitions[j].size < minSize) {

bestIdx = j;

minSize = partitions[j].size;

}

}

if (bestIdx != -1) {

partitions[bestIdx].allocated = 1;

partitions[bestIdx].processID = i + 1;

partitions[bestIdx].remaining = partitions[bestIdx].size - processes[i];

} else {

printf("Process %d (Size: %d) -> Unallocated\n", i + 1, processes[i]);

}

}

displayResults(partitions, numPartitions, processes, numProcesses);

}

void worstFit(struct MemoryBlock partitions[], int numPartitions, int processes[], int numProcesses) {

resetAllocations(partitions, numPartitions);

for (int i = 0; i < numProcesses; ++i) {

int worstIdx = -1;

int maxSize = -1;

for (int j = 0; j < numPartitions; ++j) {

if (!partitions[j].allocated && partitions[j].size >= processes[i] && partitions[j].size > maxSize) {

worstIdx = j;

maxSize = partitions[j].size;

}

}

if (worstIdx != -1) {

partitions[worstIdx].allocated = 1;

partitions[worstIdx].processID = i + 1;

partitions[worstIdx].remaining = partitions[worstIdx].size - processes[i];

} else {

printf("Process %d (Size: %d) -> Unallocated\n", i + 1, processes[i]);

}

}

displayResults(partitions, numPartitions, processes, numProcesses);

}

void nextFit(struct MemoryBlock partitions[], int numPartitions, int processes[], int numProcesses) {

resetAllocations(partitions, numPartitions);

int lastAllocated = 0;

for (int i = 0; i < numProcesses; ++i) {

int allocated = 0;

for (int j = 0; j < numPartitions; ++j) {

int idx = (lastAllocated + j) % numPartitions;

if (!partitions[idx].allocated && partitions[idx].size >= processes[i]) {

partitions[idx].allocated = 1;

partitions[idx].processID = i + 1;

partitions[idx].remaining = partitions[idx].size - processes[i];

allocated = 1;

lastAllocated = idx;

break;

}

}

if (!allocated) {

printf("Process %d (Size: %d) -> Unallocated\n", i + 1, processes[i]);

}

}

displayResults(partitions, numPartitions, processes, numProcesses);

}

int main() {

int numPartitions, numProcesses;

printf("Enter number of memory partitions: ");

scanf("%d", &numPartitions);

struct MemoryBlock partitions[numPartitions];

printf("Enter sizes of partitions:\n");

for (int i = 0; i < numPartitions; ++i) {

scanf("%d", &partitions[i].size);

partitions[i].allocated = 0;

partitions[i].processID = -1;

partitions[i].remaining = 0;

}

printf("Enter number of processes: ");

scanf("%d", &numProcesses);

int processes[numProcesses];

printf("Enter sizes of processes:\n");

for (int i = 0; i < numProcesses; ++i) {

scanf("%d", &processes[i]);

}

int choice;

do {

printf("\nChoose Memory Allocation Algorithm:\n"

"1. First Fit\n"

"2. Best Fit\n"

"3. Worst Fit\n"

"4. Next Fit\n"

"5. Exit\n"

"Enter choice (1-5): ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("\n--- First Fit Allocation ---\n");

firstFit(partitions, numPartitions, processes, numProcesses);

break;

case 2:

printf("\n--- Best Fit Allocation ---\n");

bestFit(partitions, numPartitions, processes, numProcesses);

break;

case 3:

printf("\n--- Worst Fit Allocation ---\n");

worstFit(partitions, numPartitions, processes, numProcesses);

break;

case 4:

printf("\n--- Next Fit Allocation ---\n");

nextFit(partitions, numPartitions, processes, numProcesses);

break;

case 5:

printf("\nExiting program...\n");

break;

default:

printf("Invalid choice! Please try again.\n");

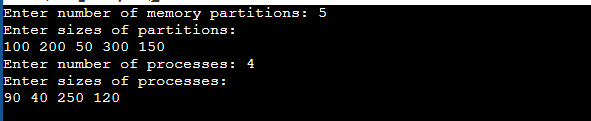
}

} while (choice != 5);

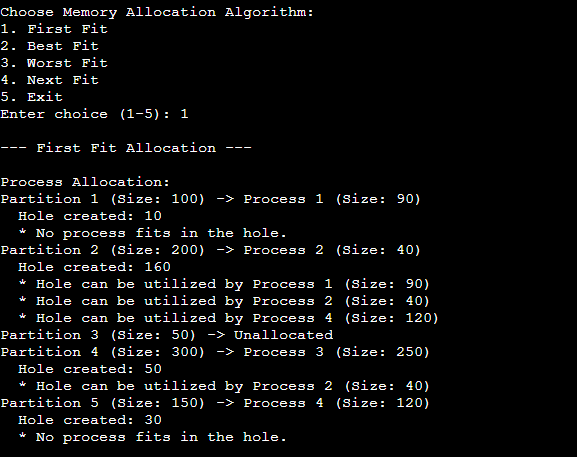
return 0;

}

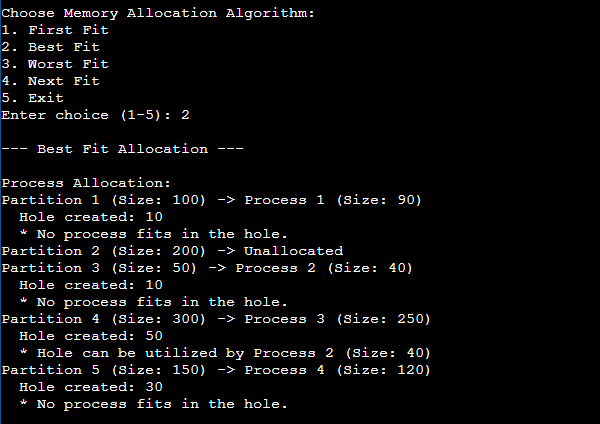
**Output:**



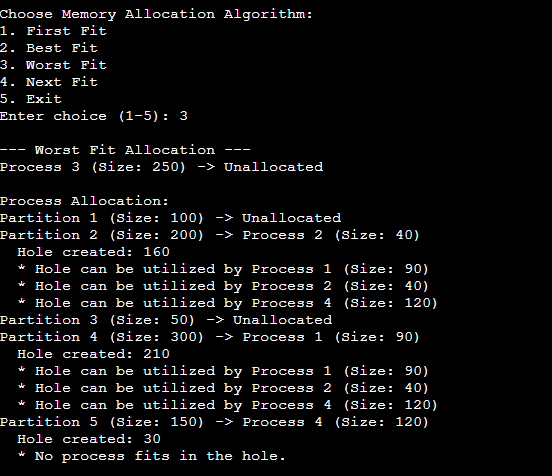
1. **First Fit:**



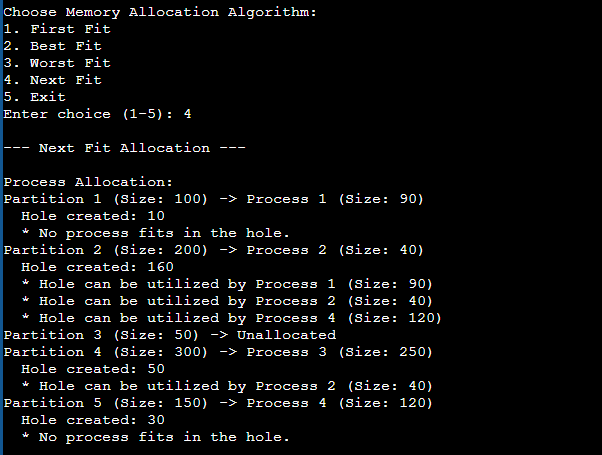
1. **Best Fit:**



1. **Worst Fit:**



1. **Next Fit:**



1. **Exit:**

