Ex. No.: 10 a
Date: 23.04.2024

BEST FIT

Aim:

To implement Best Fit memory allocation technique using Python.

Algorithm:

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
4. If found then assign it to the current process.
5. If not found then leave that process and keep checking the further

processes. **Program Code:**

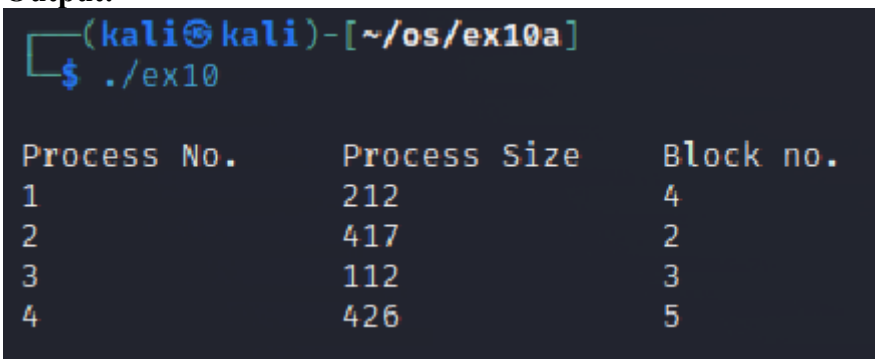
```
#include<stdio.h>
#include<string.h>

void bestFit(int blockSize[], int m, int processSize[], int n) {
   int allocation[n];
   memset(allocation, -1, sizeof(allocation));
   for (int i = 0; i < n; i++) {
      int bestIdx = -1;
      for (int j = 0; j < m; j++) {
         if (blockSize[j] >= processSize[i]) {
            if (bestIdx == -1)
               bestIdx = j;
            else if (blockSize[bestIdx] > blockSize[j])
               bestIdx = j;
         }
      }
      if (bestIdx != -1) {
         allocation[i] = bestIdx;
         blockSize[bestIdx] -= processSize[i];
      }
   }
   printf("\nProcess No.\tProcess Size\tBlock no.\n");
   for (int i = 0; i < n; i++) {
      printf("%d\t\t%d", i + 1, processSize[i]);
      if (allocation[i] != -1)
         printf("\t\t%d", allocation[i] + 1);
```

```
            else
                printf("\nNot Allocated");
            printf("\n");
        }
    }

    int main() {
        int blockSize[] = {100, 500, 200, 300, 600};
        int processSize[] = {212, 417, 112, 426};
        int m = sizeof(blockSize) / sizeof(blockSize[0]);
        int n = sizeof(processSize) / sizeof(processSize[0]);
        bestFit(blockSize, m, processSize, n);
        return 0;
    }
```

Output:



Result:

The above program executed successfully and output got verified.

Ex. No.: 10 b
Date: 27.04.2024

FIRST FIT

Aim:

To write a C program for implementation memory allocation methods for fixed partition using first fit.

Algorithm:

1. Define the max as 25.
2: Declare the variable frag[max],b[max],f[max],i,j,nb,nf,temp, highest=0, bf[max],ff[max]. 3: Get the number of blocks,files,size of the blocks using for loop.
4: In for loop check bf[j]!=1, if so temp=b[j]-f[i]
5: Check highest

Program Code:

```
#include<stdio.h>
#define max 25

void main() {
   int frag[max], b[max], f[max], i, j, nb, nf, temp;
   static int bf[max], ff[max];
   printf("\nEnter the number of blocks:");
   scanf("%d", &nb);
   printf("Enter the number of files:");
   scanf("%d", &nf);
   printf("\nEnter the size of the blocks:-\n");
   for (i = 1; i <= nb; i++) {
      printf("Block %d:", i);
      scanf("%d", &b[i]);
   }
   printf("Enter the size of the files:-\n");
   for (i = 1; i <= nf; i++) {
      printf("File %d:", i);
      scanf("%d", &f[i]);
   }
   for (i = 1; i <= nf; i++) {
      for (j = 1; j <= nb; j++) {
         if (bf[j] != 1) {
            temp = b[j] - f[i];
            if (temp >= 0) {
               ff[i] = j;
               break;
            }
         }
      }
      frag[i] = temp;
      bf[ff[i]] = 1;
```

```
    }
    printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragment");
    for (i = 1; i <= nf; i++)
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}
```

Output:

```
┌──(kali㉿kali)-[~/os/ex10b]
└─$ ./ex10b

Enter the number of blocks:4
Enter the number of files:3

Enter the size of the blocks:-
Block 1:10
Block 2:20
Block 3:33
Block 4:2
Enter the size of the files:-
File 1:1
File 2:2
File 3:3

File_no:        File_size :     Block_no:       Block_size:     Fragment
1               1               1               10              9
2               2               2               20              18
3               3               3               33              30
```

Result:

The above program executed successfully and output got verified.