Ex. No.10 a

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 <stdlib.h>
#include <pthread.h>
#include <semaphore.h>

#define MAX_ITEMS 10

sem_t empty, full, mutex;
int buffer[MAX_ITEMS];
int in = 0, out = 0;

void *producer(void *param) {
    int item;
    for (int i = 0; i < 10; i++) {
        item = rand() % 100;  // Produce a random item
        sem_wait(&empty);  // Decrease empty semaphore
        sem_wait(&mutex);  // Enter critical section

        // Add the item to the buffer
        buffer[in] = item;
        printf("Producer produced item: %d\n", item);
        in = (in + 1) % MAX_ITEMS;  // Circular buffer

        sem_post(&mutex);  // Exit critical section
        sem_post(&full);   // Increase full semaphore
    }
    return NULL;
}
```

\

```
int main() {
    pthread_t prod_thread, cons_thread;

    // Initialize semaphores
    sem_init(&empty, 0, MAX_ITEMS);  // Initially, buffer is empty
    sem_init(&full, 0, 0);           // Initially, no items are full
    sem_init(&mutex, 0, 1);          // Mutex for critical section (1 means unlocked)

    // Create producer and consumer threads
    pthread_create(&prod_thread, NULL, producer, NULL);
    pthread_create(&cons_thread, NULL, consumer, NULL);

    // Wait for threads to finish
    pthread_join(prod_thread, NULL);
    pthread_join(cons_thread, NULL);

    // Destroy semaphores
    sem_destroy(&empty);
    sem_destroy(&full);
    sem_destroy(&mutex);

    return 0;
}
[cse46@localhost ~]$ vi best_fit.py
[cse46@localhost ~]$ python3 best_fit.py
Process No.     Process Size    Block no.
1               212             4
2               417             2
3               112             3
4               426             5
[cse46@localhost ~]$ cat best_fit.py
def best_fit(block_size, process_size):
    m = len(block_size)  # Number of blocks
    n = len(process_size)  # Number of processes

    allocation = [-1] * n  # Stores block index assigned to each process
    for i in range(n):
        best_index = -1
        for j in range(m):
            if block_size[j] >= process_size[i]:
                if best_index == -1 or block_size[j] < block_size[best_index]:
                    best_index = j

        # If found, allocate the block
        if best_index != -1:
            allocation[i] = best_index
            block_size[best_index] -= process_size[i]

    # Display the result
    print("Process No.\tProcess Size\tBlock no.")
    for i in range(n):
        print(f"{i + 1}\t\t{process_size[i]}\t\t{allocation[i] + 1 if allocation[i] != -1 else 'Not Allocated'}")


# Example Data
block_size = [100, 500, 200, 300, 600]
process_size = [212, 417, 112, 426]

best_fit(block_size, process_size)
```

Output:

```
[cse46@localhost ~]$ vi best_fit.py
[cse46@localhost ~]$ python3 best_fit.py
Process No.     Process Size    Block no.
1               212             4
2               417             2
3               112             3
4               426             5
```

Ex. No. 10 b

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:

```
[cse46@localhost ~]$ vi first_fit.c
[cse46@localhost ~]$ cat first_fit.c
#include <stdio.h>

#define MAX 25

int main() {
    int frag[MAX], b[MAX], f[MAX], i, j, nb, nf, temp;
    static int bf[MAX], ff[MAX];

    printf("Enter 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 = 0; i < nb; i++) {
        printf("Block %d: ", i + 1);
        scanf("%d", &b[i]);
        bf[i] = 0; // Initially mark block as free
    }

    printf("\nEnter the size of the files:-\n");
    for(i = 0; i < nf; i++) {
        printf("File %d: ", i + 1);
        scanf("%d", &f[i]);
    }
```

```
    // First Fit Allocation
    for(i = 0; i < nf; i++) {
        for(j = 0; j < nb; j++) {
            if(bf[j] == 0 && b[j] >= f[i]) {
                ff[i] = j; // allocate block j to file i
                frag[i] = b[j] - f[i];
                bf[j] = 1; // mark block as allocated
                break;
            }
        }
    }

    // Displaying Output
    printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment\n");
    for(i = 0; i < nf; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
            i + 1,
            f[i],
            ff[i] + 1,
            b[ff[i]],
            frag[i]
        );
    }

    return 0;
}
```

Output:

```
[cse46@localhost ~]$ gcc first_fit.c -o first_fit
[cse46@localhost ~]$ ./first_fit
Enter the number of blocks: 7
Enter the number of files: 6

Enter the size of the blocks:-
Block 1: 5
Block 2: 5
Block 3: 5
Block 4: 5
Block 5: 5
Block 6: 5
Block 7: 5

Enter the size of the files:-
File 1: 9
File 2: 3
File 3: 5
File 4: 23
File 5: 7
File 6: 5

File_no:        File_size:      Block_no:       Block_size:     Fragment
1               9               1               5               0
2               3               1               5               2
3               5               2               5               0
4               23              1               5               -1075737014
5               7               1               5               1
6               5               3               5               0
```