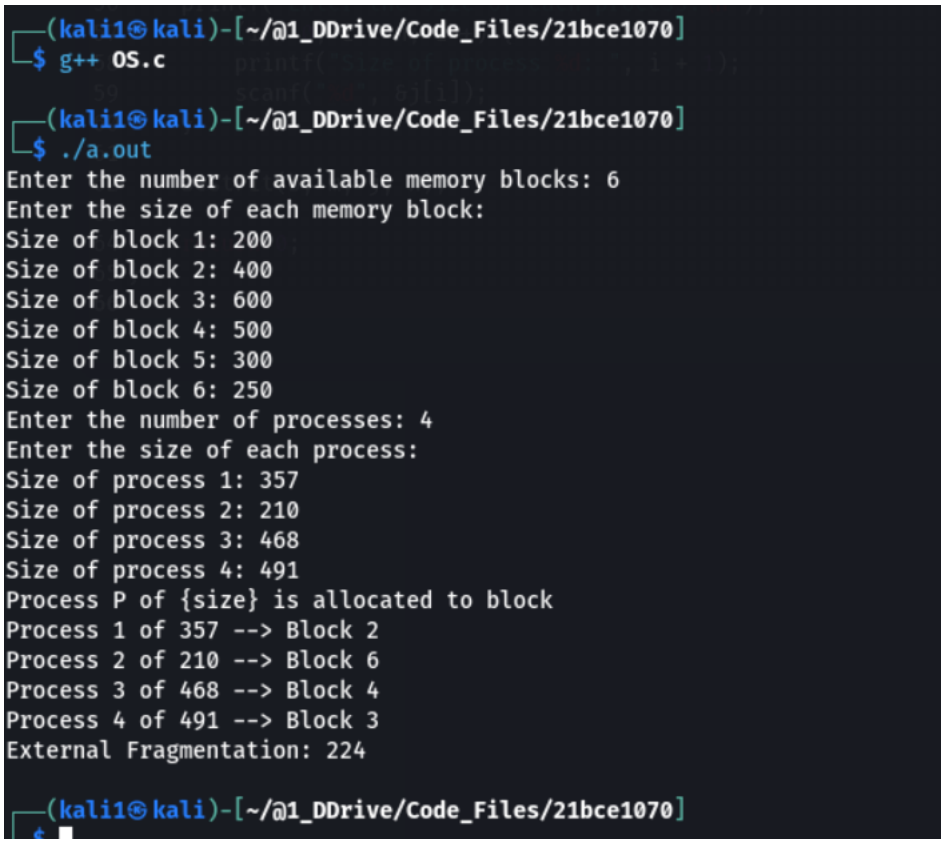
Static Partitioning

Best Fit

```
#include<stdio.h>
#define MAX_BLOCKS 100
#define MAX_PROCESSES 100

void bestFit(int b[], int m, int j[], int n) {
   int a[MAX_PROCESSES];
   int i, k;
   int f = 0;

   for (i = 0; i < m; i++)
      a[i] = -1;

   for (i = 0; i < n; i++) {
      int bf = -1;
      for (k = 0; k < m; k++) {
         if (b[k] >= j[i]) {
            if (bf == -1 || b[k] < b[bf]) {
               bf = k;
            }
         }
      }

      if (bf != -1) {
         a[i] = bf;
         b[bf] -= j[i];
         f += b[bf];
      }
```

```
    }

    printf("Process P of {size} is allocated to block\n");
    for (i = 0; i < n; i++) {
        if (a[i] != -1)
            printf("Process %d of %d --> Block %d\n", i + 1, j[i], a[i] + 1);
        else
            printf("Process %d of %d --> Not allocated\n", i + 1, j[i]);
    }

    printf("External Fragmentation: %d\n", f);
}

int main() {
    int b[MAX_BLOCKS], j[MAX_PROCESSES];
    int n, m, i;

    printf("Enter the number of available memory blocks: ");
    scanf("%d", &m);
    printf("Enter the size of each memory block:\n");
    for (i = 0; i < m; i++) {
        printf("Size of block %d: ", i + 1);
        scanf("%d", &b[i]);
    }

    printf("Enter the number of processes: ");
    scanf("%d", &n);
    printf("Enter the size of each process:\n");
    for (i = 0; i < n; i++) {
        printf("Size of process %d: ", i + 1);
        scanf("%d", &j[i]);
```

```
    }

    bestFit(b, m, j, n);

    return 0;
}
```

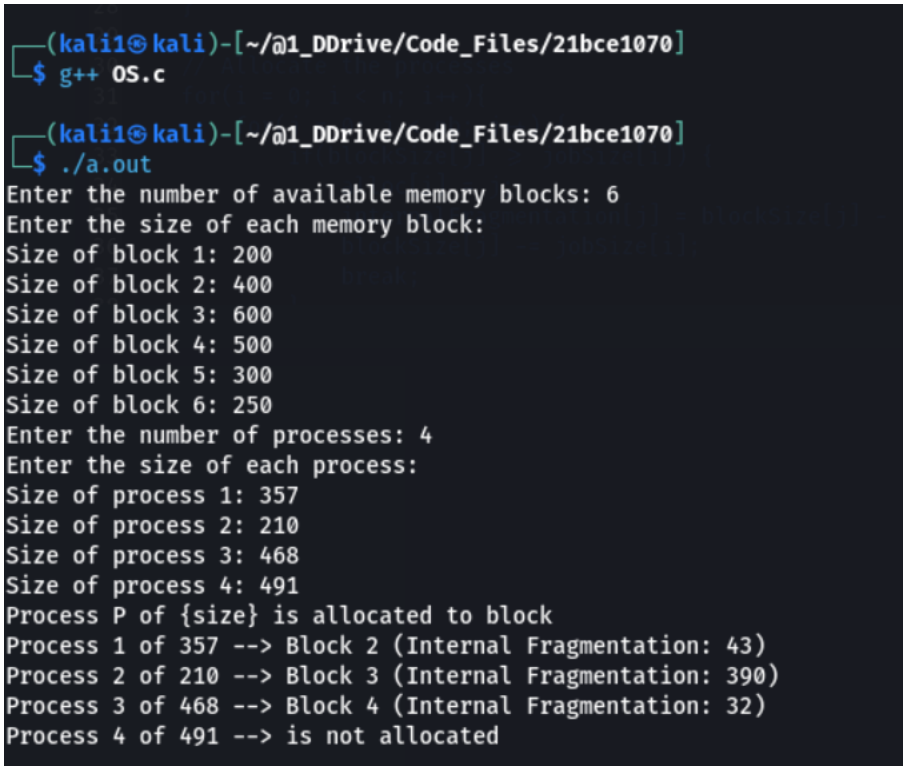


First Fit

```
#include <stdio.h>

int main() {
```

```
int a, b[100], c, d[100], e, f, g[100];
int h[100] = {0};

printf("Enter the number of available memory blocks: ");
scanf("%d", &a);
printf("Enter the size of each memory block: \n");
for (e = 0; e < a; e++) {
   printf("Size of block %d: ", e + 1);
   scanf("%d", &b[e]);
}

printf("Enter the number of processes: ");
scanf("%d", &c);
printf("Enter the size of each process: \n");
for (e = 0; e < c; e++) {
   printf("Size of process %d: ", e + 1);
   scanf("%d", &d[e]);
}

for (e = 0; e < c; e++) {
   g[e] = -1;
}

for (e = 0; e < c; e++) {
   for (f = 0; f < a; f++) {
      if (b[f] >= d[e]) {
         g[e] = f;
         h[f] = b[f] - d[e];
         b[f] -= d[e];
         break;
      }
```

```
    }
  }

  printf("Process P of {size} is allocated to block \n");
  for (e = 0; e < c; e++) {
    if (g[e] != -1)
      printf("Process %d of %d --> Block %d (Internal Fragmentation: %d)\n", e + 1, d[e], g[e] + 1,
h[g[e]]);
    else
      printf("Process %d of %d --> is not allocated \n", e + 1, d[e]);
  }

  return 0;
}
```



Worst Fit

```
#include <stdio.h>

int main() {
   int nb, bs[100], n, js[100], i, j, a[100], av[100], m;
   int ifrag[100] = {0}; // Array to store internal fragmentation for each block

   printf("Enter the number of available memory blocks: ");
   scanf("%d", &nb);
   printf("Enter the size of each memory block:\n");
   for (i = 0; i < nb; i++) {
      printf("Size of block %d: ", i + 1);
      scanf("%d", &bs[i]);
   }

   printf("Enter the number of processes: ");
   scanf("%d", &n);
   printf("Enter the size of each process:\n");
   for (i = 0; i < n; i++) {
      printf("Size of process %d: ", i + 1);
      scanf("%d", &js[i]);
   }

   for (i = 0; i < n; i++) {
      a[i] = -1;
   }
   for (i = 0; i < nb; i++) {
      av[i] = -1;
   }
```

```
    for (i = 0; i < n; i++) {
        for (j = 0; j < nb; j++) {
            if (bs[j] >= js[i]) {
                av[j] = bs[j] - js[i];
            }
        }
        m = 0;
        for (j = 0; j < nb; j++) {
            if (av[m] < av[j]) {
                m = j;
            }
        }
        a[i] = m;
        if (av[m] <= 50) {
            a[i] = -1;
        }
        if (a[i] != -1) {
            ifrag[a[i]] = bs[a[i]] - js[i];
            bs[a[i]] -= js[i];
        }

        for (j = 0; j < nb; j++) {
            av[j] = -1;
        }
    }

    printf("Process P of {size} is allocated to block \n");
    for (i = 0; i < n; i++) {
        if (a[i] != -1)
            printf("Process %d of %d --> Block %d (Internal Fragmentation: %d)\n", i + 1, js[i], a[i] + 1,
ifrag[a[i]]);
```

```
        else
            printf("Process %d of %d --> is not allocated\n", i + 1, js[i]);
    }

    return 0;
}
```