Name : Ashutosh Ardu

Regno : 20BRS1262

Dynamic Memory Allocation Strategies

Code

(All three Strategies in one code)

```
#include "stdio.h"
#include "stdlib.h"
#define pr printf
#define sc scanf
int remain;

struct block{
  int num,size,occ;
};

void br(){
  pr("_____________________________________________________________________
\n");
}

void swap(struct block *a,struct block *b){
  int t;
  t=a->num;a->num=b->num;b->num=t;
  t=a->size;a->size=b->size;b->size=t;
}

void sort(struct block bloc[],int b){
  int min;
  for(int i=0;i<b-1;++i){
    min=i;
    for(int j=i+1;j<b;++j){
        if(bloc[j].size<bloc[min].size) min=j;
    }swap(&bloc[i],&bloc[min]);
  }
}

void rsort(struct block bloc[],int b){
```

```
  int max;
  for(int i=0;i<b-1;++i){
    max=i;
    for(int j=i+1;j<b;++j){
        if(bloc[j].size>bloc[max].size) max=j;
    }swap(&bloc[i],&bloc[max]);
  }
}

void print(struct block proc[],int p,struct block bloc[],int b){
  br();
  pr("Process no.\tProcess_Size\tBlock no.\tBlock_Size\n");
  for(int i=0;i<p;++i){
    if(proc[i].occ==-1){
      pr("%d\t\t%d\t\t%d\t\t-1\n",proc[i].num,proc[i].size,proc[i].occ);
    }
    else{
      pr("%d\t\t%d\t\t%d\t\t%d\n",proc[i].num,proc[i].size,bloc[proc[i].
occ].num,bloc[proc[i].occ].size);
    }
  }
  br();
  pr("-1 : cannot be allocated\n");
  br();
  pr("Block memory left after allocation : %d\n",remain);
}

void first(struct block proc[],int p,struct block bloc[],int b){
  for(int i=0;i<p;++i){
    for(int j=0;j<b;++j){
      if(bloc[j].size>=proc[i].size && bloc[j].occ!=1){
        remain-=proc[i].size;
        proc[i].occ=j;
        bloc[j].occ=1;
        break;
      }
    }
  }
  print(proc,p,bloc,b);
}

int main(){
  int b,p,ch;remain=0;
  pr("Number of block: ");
  sc("%d",&b);
```

```
  pr("Number of Processes: ");
  sc("%d",&p);
  struct block bk[b],proc[p];
  pr("Enter block size:\n");
  for(int i=0;i<b;++i){
    pr("Size of Block %d: ",i+1);
    sc("%d",&bk[i].size);
    remain+=bk[i].size;
    bk[i].num=i+1;
    bk[i].occ=0;
  }
  pr("Enter the process size:\n");
  for(int i=0;i<p;++i){
    pr("Size of Process %d: ",i+1);
    sc("%d",&proc[i].size);
    proc[i].num=i+1;
    proc[i].occ=-1;
  }
  br();
  pr("1-First Fit 2-Best Fit 3-Worst Fit\n");
  sc("%d",&ch);
  switch(ch){
    case 1:
      first(proc,p,bk,b);
      break;
    case 2:
      sort(bk,b);
      first(proc,p,bk,b);
      break;
    case 3:
      rsort(bk,b);
      first(proc,p,bk,b);
      break;
  }
}
```

Questions

1. Given five memory partitions of 50 KB, 25 KB, 75 KB, 200 KB, & 150 KB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of size 128 KB, 20 KB, 68 KB, 32 KB, and 178 KB (in order)? Rank the algorithms in terms of how efficiently they use memory

Output

First Fit

```
codex@codex:~/Documents/OS/C$ ./tempCodeRunnerFile
Number of block: 5
Number of Processes: 5
Enter block size:
Size of Block 1: 50
Size of Block 2: 25
Size of Block 3: 75
Size of Block 4: 200
Size of Block 5: 150
Enter the process size:
Size of Process 1: 128
Size of Process 2: 20
Size of Process 3: 68
Size of Process 4: 32
Size of Process 5: 178
----------------------------------------------------------------
1-First Fit 2-Best Fit 3-Worst Fit
1
----------------------------------------------------------------
Process no.     Process_Size    Block no.       Block_Size
1               128             4               200
2               20              1               50
3               68              3               75
4               32              5               150
5               178             -1              -1
----------------------------------------------------------------
-1 : cannot be allocated
----------------------------------------------------------------
Block memory left after allocation : 252
codex@codex:~/Documents/OS/C$
```

Best Fit

```
codex@codex:~/Documents/OS/C$ ./tempCodeRunnerFile
Number of block: 5
Number of Processes: 5
Enter block size:
Size of Block 1: 50
Size of Block 2: 25
Size of Block 3: 75
Size of Block 4: 200
Size of Block 5: 150
Enter the process size:
Size of Process 1: 128
Size of Process 2: 20
Size of Process 3: 68
Size of Process 4: 32
Size of Process 5: 178
----------------------------------------------------------------
1-First Fit 2-Best Fit 3-Worst Fit
2
----------------------------------------------------------------
Process no.     Process_Size    Block no.       Block_Size
1               128             5               150
2               20              2               25
3               68              3               75
4               32              1               50
5               178             4               200
----------------------------------------------------------------
-1 : cannot be allocated
----------------------------------------------------------------
Block memory left after allocation : 74
codex@codex:~/Documents/OS/C$
```

Worst fit

```
codex@codex:~/Documents/OS/C$ ./tempCodeRunnerFile
Number of block: 5
Number of Processes: 5
Enter block size:
Size of Block 1: 50
Size of Block 2: 25
Size of Block 3: 75
Size of Block 4: 200
Size of Block 5: 150
Enter the process size:
Size of Process 1: 128
Size of Process 2: 20
Size of Process 3: 68
Size of Process 4: 32
Size of Process 5: 178
----------------------------------------------------------------
1-First Fit 2-Best Fit 3-Worst Fit
3
----------------------------------------------------------------
Process no.     Process_Size    Block no.       Block_Size
1               128             4               200
2               20              5               150
3               68              3               75
4               32              1               50
5               178             -1              -1
----------------------------------------------------------------
-1 : cannot be allocated
----------------------------------------------------------------
Block memory left after allocation : 252
codex@codex:~/Documents/OS/C$
```

2. Given six memory partitions of 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, and 125 KB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of size 115 KB, 500 KB, 358 KB, 200 KB, and 375 KB (in order)? Rank the algorithms in terms of how efficiently they use memory

Output

First Fit

```
codex@codex:~/Documents/OS/C$ ./tempCodeRunnerFile
Number of block: 6
Number of Processes: 5
Enter block size:
Size of Block 1: 300
Size of Block 2: 600
Size of Block 3: 350
Size of Block 4: 200
Size of Block 5: 750
Size of Block 6: 125
Enter the process size:
Size of Process 1: 115
Size of Process 2: 500
Size of Process 3: 358
Size of Process 4: 200
Size of Process 5: 375
-----------------------------------------------------------------
1-First Fit 2-Best Fit 3-Worst Fit
1
-----------------------------------------------------------------
Process no.     Process_Size    Block no.       Block_Size
1               115             1               300
2               500             2               600
3               358             5               750
4               200             3               350
5               375             -1              -1
-----------------------------------------------------------------
-1 : cannot be allocated
-----------------------------------------------------------------
Block memory left after allocation : 1152
codex@codex:~/Documents/OS/C$
```

Best Fit

```
codex@codex:~/Documents/OS/C$ ./tempCodeRunnerFile
Number of block: 6
Number of Processes: 5
Enter block size:
Size of Block 1: 300
Size of Block 2: 600
Size of Block 3: 350
Size of Block 4: 200
Size of Block 5: 750
Size of Block 6: 125
Enter the process size:
Size of Process 1: 115
Size of Process 2: 500
Size of Process 3: 358
Size of Process 4: 200
Size of Process 5: 375
----------------------------------------------------------------
1-First Fit 2-Best Fit 3-Worst Fit
2
----------------------------------------------------------------
Process no.     Process_Size    Block no.       Block_Size
1               115             6               125
2               500             2               600
3               358             5               750
4               200             4               200
5               375             -1              -1
----------------------------------------------------------------
-1 : cannot be allocated
----------------------------------------------------------------
Block memory left after allocation : 1152
codex@codex:~/Documents/OS/C$
```

Worst Fit

```
codex@codex:~/Documents/OS/C$ gcc tempCodeRunnerFile.c -o tempCodeRunnerFile
codex@codex:~/Documents/OS/C$ ./tempCodeRunnerFile
Number of block: 6
Number of Processes: 5
Enter block size:
Size of Block 1: 300
Size of Block 2: 600
Size of Block 3: 350
Size of Block 4: 200
Size of Block 5: 750
Size of Block 6: 125
Enter the process size:
Size of Process 1: 115
Size of Process 2: 500
Size of Process 3: 358
Size of Process 4: 200
Size of Process 5: 375
----------------------------------------------------------------
1-First Fit 2-Best Fit 3-Worst Fit
3
----------------------------------------------------------------
Process no.     Process_Size    Block no.       Block_Size
1               115             5               750
2               500             2               600
3               358             -1              -1
4               200             3               350
5               375             -1              -1
----------------------------------------------------------------
-1 : cannot be allocated
----------------------------------------------------------------
Block memory left after allocation : 1510
codex@codex:~/Documents/OS/C$
```

Conclusion

Generally

Rank	Strategy
1	Best Fit
2	First Fit
3	Worst Fit

Q1. Memory Wise

Rank	Strategy	Memory Left after completion
1	Best Fit	74
2	First Fit	252
3	Worst Fit	252

Here First fit is given better rank than worst fit even though they have similar memory left value is because the time taken for execution is less for first fit as compared to that for worst fit.(because in worst fit you are searching for the next biggest block available, which is not the case in first fit)

Q2. Memory Wise

Rank	Strategy	Memory Left after completion
1	First Fit	1152
2	Best Fit	1152
3	Worst Fit	1510

Here First fit is given better rank than best fit even though they have similar memory left value is because the time taken for execution is less for first fit as compared to that for best fit.(because in best fit you are searching for the a particular block which results in minimum memory fragementation and also that particular block should be available at the same time, which is not the case in first fit)