Operating Systems Lab Assignment 3 -

Task1- Implement priority and round robin scheduling and calculate turn around time and waiting time.

Sol.- Priority scheduling

```
┌──(root㉿LAPTOP-2SJNMAE1)-[/os_assignment3]
└─# nano task1_priority.py
```

root@LAPTOP-2SJNMAE1: /os

GNU nano 8.1 task1_priority.py *

```
n=int(input("n: "))
bt,at,pr=[],[],[]

for i in range(n):
    b,a,p=map(int,input().split());bt+=[b];at+=[a];pr+=[p]
ct=[0]*n;done=[0]*n;t=0

for _ in range(n):
    r=[i for i in range(n) if not done[i]and at[i]<=t]
    if not r:t=min(at[i]for i in range(n)if not done[i]);continue
    i=min(r,key=lambda x:pr[x]);t+=bt[i];ct[i]=t;done[i]=1

tat=[ct[i]-at[i]for i in range(n)]
wt=[tat[i]-bt[i]for i in range(n)]

print("P\tCT\tTAT\tWT")

for i in range(n):print(f"{i+1}\t{ct[i]}\t{tat[i]}\t{wt[i]}")
print("Avg TAT=",round(sum(tat)/n,2)," Avg WT=",round(sum(wt)/n,2))
```

```
┌──(root㉿LAPTOP-2SJNMAE1)-[/os_assignment3]
└─# python3 task1_priority.py
n: 3
5 0 2
3 1 1
8 2 3
P       CT      TAT     WT
1       5       5       0
2       8       7       4
3       16      14      6
Avg TAT= 8.67  Avg WT= 3.33
```

➔ Round Robin scheduling

```
┌──(root㉿LAPTOP-2SJNMAE1)-[/os_assignment3]
└─# nano task1_roundrobin.py
```

root@LAPTOP-2SJNMAE1: /os

GNU nano 8.1 task1_roundrobin.py *

```
n=int(input("n: "))
bt,at=[],[]

for i in range(n):b,a=map(int,input().split());bt+=[b];at+=[a]
q=int(input("q: "))
r=bt[:];t=0;ct=[0]*n;Q=[]

while 1:
    for i in range(n):
        if at[i]<=t and r[i]>0 and i not in Q:Q+=[i]
    if not Q:
        if all(x==0 for x in r):break
        t+=1;continue
    i=Q.pop(0);x=min(q,r[i]);r[i]-=x;t+=x
    for j in range(n):
        if at[j]<=t and r[j]>0 and j not in Q:Q+=[j]
    if r[i]>0:Q+=[i]
    else:ct[i]=t

tat=[ct[i]-at[i]for i in range(n)]
wt=[tat[i]-bt[i]for i in range(n)]
print("P\tCT\tTAT\tWT")

for i in range(n):print(f"{i+1}\t{ct[i]}\t{tat[i]}\t{wt[i]}")
print("Avg TAT=",round(sum(tat)/n,2)," Avg WT=",round(sum(wt)/n,2))
```

```
┌──(root㉿LAPTOP-2SJNMAE1)-[/os_assignment3]
└─# python3 task1_roundrobin.py
n: 3
5 0
4 1
2 2
q: 2
P       CT      TAT     WT
1       9       9       4
2       11      10      6
3       8       6       4
Avg TAT= 8.33  Avg WT= 4.67
```

Task 2- Simulate worst-fit, best-fit, first-fit memory allocation strategies.

Sol.-

```
┌──(root㉿LAPTOP-2SJNMAE1)-[~/os_assignment3]
└─# nano task2.py
```

root@LAPTOP-2SJNMAE1: ~/

GNU nano 8.1 task2.py *

```
def allocate(blocks, procs, strat):
    b_used=[False]*len(blocks)
    for i,p in enumerate(procs):
        idx=None
        if strat=='first':
            for j,b in enumerate(blocks):
                if not b_used[j] and b>=p: idx=j; break
        elif strat=='best':
            idx=min((j for j in range(len(blocks)) if not b_used[j] and blocks[j]>=p),key=lambda j:blocks[j],default=None)
        else:
            idx=max((j for j in range(len(blocks)) if not b_used[j] and blocks[j]>=p),key=lambda j:blocks[j],default=None)
        if idx is not None:
            b_used[idx]=True
            print(f"P{i+1}({p}) → Block{idx+1} ({blocks[idx]-p} internal frag)")
        else:
            print(f"P{i+1}({p}) → Not Allocated")

blocks=list(map(int,input("Enter memory block sizes: ").split()))

procs=list(map(int,input("Enter process sizes: ").split()))

s=input("Strategy (first/best/worst): ").lower()

allocate(blocks,procs,s)
```

```
┌──(root㉿LAPTOP-2SJNMAE1)-[~/os_assignment3]
└─# python3 task2.py
Enter memory block sizes: 120 250 300 180 100
Enter process sizes: 115 220 95 275 50
Strategy (first/best/worst): first
P1(115) → Block1 (5 internal frag)
P2(220) → Block2 (30 internal frag)
P3(95) → Block3 (205 internal frag)
P4(275) → Not Allocated
P5(50) → Block4 (130 internal frag)

┌──(root㉿LAPTOP-2SJNMAE1)-[~/os_assignment3]
└─# python3 task2.py
Enter memory block sizes: 120 250 300 180 100
Enter process sizes: 115 220 95 275 50
Strategy (first/best/worst): best
P1(115) → Block1 (5 internal frag)
P2(220) → Block2 (30 internal frag)
P3(95) → Block5 (5 internal frag)
P4(275) → Block3 (25 internal frag)
P5(50) → Block4 (130 internal frag)

┌──(root㉿LAPTOP-2SJNMAE1)-[~/os_assignment3]
└─# python3 task2.py
Enter memory block sizes: 120 250 300 180 100
Enter process sizes: 115 220 95 275 50
Strategy (first/best/worst): worst
P1(115) → Block3 (185 internal frag)
P2(220) → Block2 (30 internal frag)
P3(95) → Block4 (85 internal frag)
P4(275) → Not Allocated
P5(50) → Block1 (70 internal frag)
```

Task 3 - Implement MFT and MVT Memory Management

Sol.- MFT (fixed partition)

```
┌──(root㉿LAPTOP-2SJNMAE1)-[~/os_assignment3]
└─# nano task3_mft.py
```

```
root@LAPTOP-2SJNMAE1: ~/(
  GNU nano 8.1                                                task3_mft.py *
parts=list(map(int,input("Enter partition sizes: ").split()))
procs=list(map(int,input("Enter process sizes: ").split()))
used=[0]*len(parts)
for i,p in enumerate(procs):
    idx=None
    for j in range(len(parts)):
        if not used[j] and parts[j]>=p:
            idx=j; used[j]=1; break
    if idx is not None:
        print(f"P{i+1}({p}) → Partition{idx+1} (Internal Frag={parts[idx]-p})")
    else:
        print(f"P{i+1}({p}) → Not Allocated")
```

```
┌──(root㉿LAPTOP-2SJNMAE1)-[~/os_assignment3]
└─# python3 task3_mft.py
Enter partition sizes: 120 250 300 180 100
Enter process sizes: 115 220 95 275 50
P1(115) → Partition1 (Internal Frag=5)
P2(220) → Partition2 (Internal Frag=30)
P3(95) → Partition3 (Internal Frag=205)
P4(275) → Not Allocated
P5(50) → Partition4 (Internal Frag=130)
```

➔ MVT (variable partition)

```
┌──(root㉿LAPTOP-2SJNMAE1)-[~/os_assignment3]
└─# nano task3_mvt.py
```

root@LAPTOP-2SJNMAE1: ~/

GNU nano 8.1 task3_mvt.py *

```
mem=int(input("Enter total memory size: "))
free=[(0,mem)]
allocs={}
while True:
    cmd=input("cmd (alloc/free/state/exit): ").split()
    if not cmd: continue
    if cmd[0]=='alloc':
        name,sz=cmd[1],int(cmd[2])
        for i,(s,l) in enumerate(free):
            if l>=sz:
                allocs[name]=(s,sz)
                del free[i]
                if l>sz: free.insert(i,(s+sz,l-sz))
                print(f"{name} allocated at {s}")
                break
        else: print("Not enough space")
    elif cmd[0]=='free':
        if cmd[1] in allocs:
            s,sz=allocs.pop(cmd[1])
            free.append((s,sz))
            free=sorted(free,key=lambda x:x[0])
            merged=[]
            for a,b in free:
                if not merged: merged.append((a,b))
                else:
                    pa,pb=merged[-1]
                    if pa+pb==a: merged[-1]=(pa,pb+b)
                    else: merged.append((a,b))
            free=merged
            print(f"{cmd[1]} freed")
        else: print("Process not found")
    elif cmd[0]=='state':
        print("Allocated:",allocs)
        print("Free:",free)
    elif cmd[0]=='exit': break
```

```
┌──(root㉿LAPTOP-2SJNMAE1)-[~/os_assignment3]
└─# python3 task3_mvt.py
Enter total memory size: 1000
cmd (alloc/free/state/exit): alloc P1 250
P1 allocated at 0
cmd (alloc/free/state/exit): alloc P2 400
P2 allocated at 250
cmd (alloc/free/state/exit): state
Allocated: {'P1': (0, 250), 'P2': (250, 400)}
Free: [(650, 350)]
cmd (alloc/free/state/exit): free P2
P2 freed
cmd (alloc/free/state/exit): state
Allocated: {'P1': (0, 250)}
Free: [(250, 750)]
cmd (alloc/free/state/exit): exit
```