GOA COLLEGE OF ENGINEERING

"Bhausaheb Bandodkar Technical Education Complex"

Experiment No: 8

Aim: To implement Group Sweeping Algorithm

Theory:

With Group Sweeping Scheduling (GSS), requests are served in cycles, in round robin man- ner. To reduce disk arm movements, the set of n streams is divided into g groups. Groups are served in fixed order. Individual streams within a group are served according to SCAN; therefore, it is not fixed at which time or order individual streams within a group are served. In one cycle, a specific stream may be the first to be served; in another cycle, it may be the last in the same group. A smoothing buffer which is sized according to the cycle time and data rate of the stream assures continuity. If the SCAN scheduling strategy is applied to all streams of a cycle without any grouping, the playout of a stream cannot be started until the end of the cycle of its first retrieval (where all requests are served once) because the next service may be in the last slot of the following cycle. As the data must be buffered in GSS, the playout can be started at the end of the group in which the first retrieval takes place. Whereas SCAN requires buffers for all streams, in GSS, the buffer can be reused for each group. Further optimizations of this scheme are proposed in. In this method, it is ensured that each stream is served once in each cycle. GSS is a trade-off between the optimization of buffer space and arm movements. To provide the requested guarantees for continuous media data, we propose here to introduce a "joint deadline" mechanism: we assign to each group of streams one deadline, the "joint deadline." This deadline is specified as being the earliest one out of the deadlines of all streams in the respective

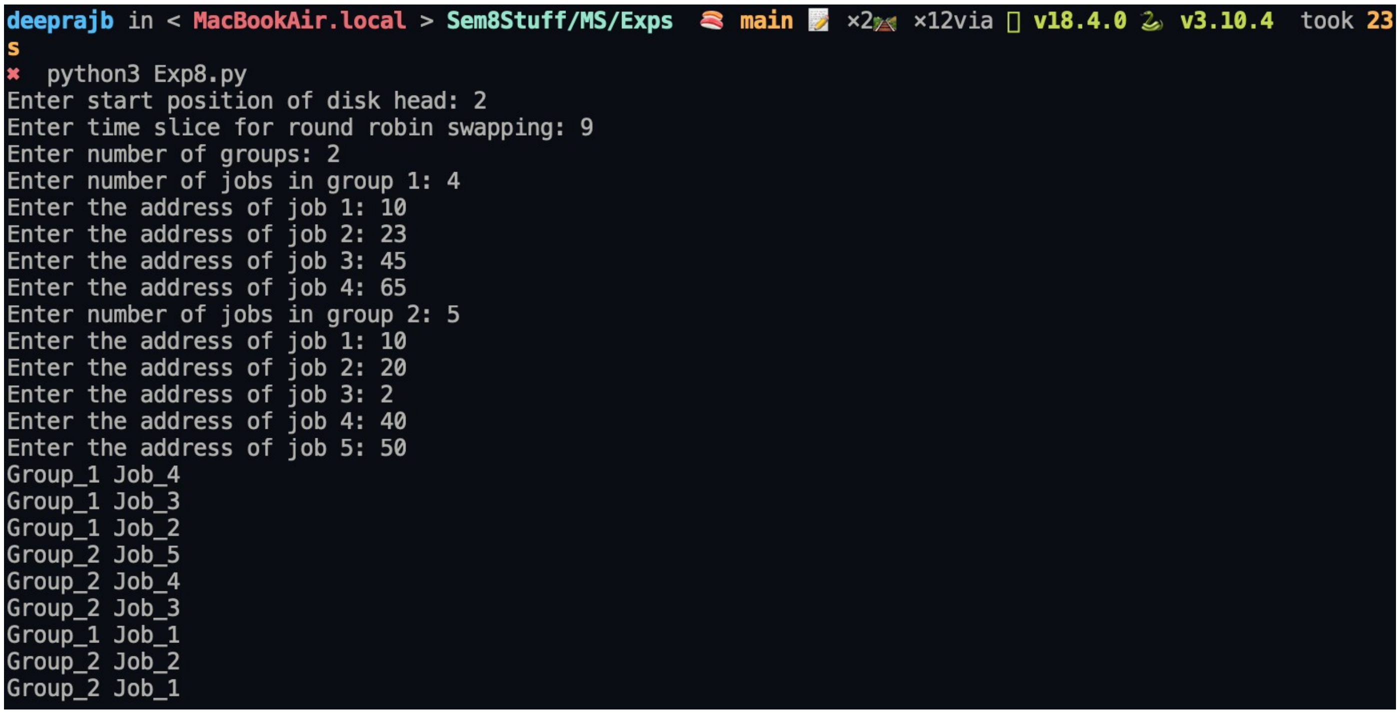
Code:

```
# Assume a seek operation takes 3 time units
class Group:
def __init__(self, name, start):
self.name = name
self.jobList = []
self.start = start

def __repr__(self):
return f"Group_{self.name}"

def __str__(self):
return f"Group_{self.name}"


class Job:
def __init__(self, name, addr):
self.name = name
self.addr = addr

def __str__(self):
```

```
return f"Job_{self.name}"

def __repr__(self):
return f"Job_{self.name}"


start = int(input("Enter start position of disk head: "))
tSlice = int(input("Enter time slice for round robin swapping: "))
n = int(input("Enter number of groups: "))
direction = 1


def init():
groupList = []
for i in range(1, n+1):
group = Group(i, start)
n_jobs = int(input(f"Enter number of jobs in group {i}: "))
for j in range(1, n_jobs+1):
addr = int(input(f"Enter the address of job {j}: "))
group.jobList.append(Job(j, addr))
groupList.append(group)
return groupList


def seekTime(currPos, dest):
return 3


def SCAN(jobList, currPos):
global direction
shortestDis = 100000
closestJob = -1
for idx, job in enumerate(jobList):
dist = abs(currPos-job.addr)
if direction > 0:
if job.addr > currPos and dist < shortestDis:
shortestDis = dist
closestJob = idx
else:
if job.addr < currPos and dist < shortestDis:
shortestDis = dist
closestJob = idx
if idx == -1:
direction *= -1
idx = SCAN(jobList, currPos)
```

```
return idx


def schedule(groupList):
jobOrder = []
currTime = 0
currPos = start
while(len(groupList) > 0):
for idx, group in enumerate(groupList):
if len(group.jobList) == 0:
del groupList[idx]
else:
elapsedTime = 0
while elapsedTime < tSlice:
if len(group.jobList) == 0:
break
idx = SCAN(group.jobList, currPos)
job = group.jobList[idx]
elapsedTime += seekTime(currPos, job)
jobOrder.append(f"{group} {job}")
del group.jobList[idx]
if elapsedTime >= tSlice:
break
for job in jobOrder:
print(job)


groupList = init()
schedule(groupList)
```

Output:



```
deeprajb in < MacBookAir.local > Sem8Stuff/MS/Exps  main  ×2 ×12via  v18.4.0  v3.10.4  took 23
s
✖  python3 Exp8.py
Enter start position of disk head: 2
Enter time slice for round robin swapping: 9
Enter number of groups: 2
Enter number of jobs in group 1: 4
Enter the address of job 1: 10
Enter the address of job 2: 23
Enter the address of job 3: 45
Enter the address of job 4: 65
Enter number of jobs in group 2: 5
Enter the address of job 1: 10
Enter the address of job 2: 20
Enter the address of job 3: 2
Enter the address of job 4: 40
Enter the address of job 5: 50
Group_1 Job_4
Group_1 Job_3
Group_1 Job_2
Group_2 Job_5
Group_2 Job_4
Group_2 Job_3
Group_1 Job_1
Group_2 Job_2
Group_2 Job_1
```

Conclusion: A program implementing Group Sweeping Algorithm was successfully implemented and executed.