CSL-240 (OS) - Assignment

```
        - Paras Sharma
        - 2015CSA1126
        - Btech 4th Sem
```

Index:

1. *Unix-Linux Commands*

File Management:

`touch`

```
$ touch hello.txt
```

`nano`

```
$ nano hello.txt
```

```
Hello this is a file!









                                [ New File ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text^T To Spell  ^_ Go To Line
```

`cat`

```
$ cat hello.txt
Hello this is a file!
```

`ls`

```
$ ls -a
.           deadlock_detection.py  fifo_page.py   mem_alloc.py  sjf.py
..          fcfs.py                fork_child.py  priority.py
bankers.py  fifo_disc.py           fragments.py   semophore.py

$ ls -l
total 44
-rw-r--r-- 1 paraz paraz 1862 Apr 15 19:24 bankers.py
-rw-r--r-- 1 paraz paraz  801 Apr 15 16:38 deadlock_detection.py
-rw-r--r-- 1 paraz paraz  668 Apr 15 16:45 fcfs.py
-rw-r--r-- 1 paraz paraz  236 Apr 15 17:21 fifo_disc.py
-rw-r--r-- 1 paraz paraz  605 Apr 12 14:45 fifo_page.py
-rw-r--r-- 1 paraz paraz  435 Apr 12 14:07 fork_child.py
-rw-r--r-- 1 paraz paraz 1169 Apr 15 17:38 fragments.py
-rw-r--r-- 1 paraz paraz 1717 Apr 11 21:55 mem_alloc.py
-rw-r--r-- 1 paraz paraz  683 Apr 15 17:04 priority.py
-rw-r--r-- 1 paraz paraz  467 Apr 15 18:16 semophore.py
-rw-r--r-- 1 paraz paraz  605 Apr 15 17:02 sjf.py
```

`cp`

```
$ cp hello.txt hey.txt
```

`mv`

```
$ mv hello.txt ../
```

`rm`

```
$ rm hello.txt
```

Directory Management:

`mkdir`

```
$ mkdir dm
```

`pwd`

```
$ pwd
/home/paraz/Desktop/
```

`cd`

```
$ cd Desktop
```

`rmdir`

```
$ rmdir dm
```

`rm`

```
$ rm -r -f dm
```

File Permissions:

`chmod`

```
$ chmod +rwx hello.txt
```

```
$ chmod -rwx hello.txt
```

Other:

`ifconfig`

```
$ ifconfig
wlp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.43.95  netmask 255.255.255.0  broadcast 192.168.43.255
```

```
        inet6 2405:205:4209:71f1:2899:d8dd:7a64:1097  prefixlen 64  scopeid
0x0<global>
```

`ps`

```
$ ps
  PID TTY          TIME CMD
 5360 pts/4    00:00:00 bash
 8548 pts/4    00:00:00 ps
```

2. *FCFS Scheduling*

Code:

```
process_queue = []
total_wtime = 0
n = int(input('Enter the total no of processes: '))
for i in range(n):
    process_queue.append([])
    process_queue[i].append(input('Enter p_name: '))
    process_queue[i].append(int(input('Enter p_arrival: ')))
    total_wtime += process_queue[i][1]
    process_queue[i].append(int(input('Enter p_burst: ')))
    print()

process_queue.sort(key = lambda process_queue:process_queue[1])

print('ProcessName\tArrivalTime\tBurstTime')
for i in range(n):
    print(process_queue[i][0],'\t\t',process_queue[i][1],'\t\t',process_queue[i]
[2])

print('Total waiting time: ',total_wtime)
print('Average waiting time: ',(total_wtime/n))
```

Output:

```
$ python3 fcfs.py
Enter the total no of processes: 3
Enter p_name: A
Enter p_arrival: 1
Enter p_burst: 4

Enter p_name: B
Enter p_arrival: 0
Enter p_burst: 6

Enter p_name: C
Enter p_arrival: 4
Enter p_burst: 2
```

```
ProcessName ArrivalTime BurstTime
B        0       6
A        1       4
C        4       2
Total waiting time:  5
Average waiting time:  1.6666666666666667
```

3. *Priority Scheduling*

Code:

```
process_queue = []
total_wtime = 0
n = int(input('Enter the total no of processes: '))
for i in range(n):
    process_queue.append([])
    process_queue[i].append(input('Enter p_name: '))
    process_queue[i].append(int(input('Enter p_burst: ')))
    process_queue[i].append(int(input('Enter Priority (<{}): '.format(n))))
    print()

process_queue.sort(key = lambda process_queue:process_queue[2])

total_wtime = 0
for i in process_queue[:n-1]:
    total_wtime+=total_wtime+i[1]

print('ProcessName\tBurstTime')
for i in range(n):
    print(process_queue[i][0],'\t\t',process_queue[i][1])

print('Total waiting time: ',total_wtime)
print('Average waiting time: ',(total_wtime/n))
```

Output:

```
$ python3 priority.py
Enter the total no of processes:3
Enter p_name: A
Enter p_burst: 4
Enter Priority (<3): 3

Enter p_name: B
Enter p_burst: 10
Enter Priority (<3): 1

Enter p_name: C
Enter p_burst: 13
Enter Priority (<3): 2

ProcessName BurstTime
B        10
```

```
C        13
A        4
Total waiting time:  33
Average waiting time:  11.0
```

4. *SJF Scheduling*

Code:

```
process_queue = []
total_wtime = 0
n = int(input('Enter the total no of processes: '))
for i in range(n):
    process_queue.append([])
    process_queue[i].append(input('Enter p_name: '))
    process_queue[i].append(int(input('Enter p_burst: ')))
    print()

process_queue.sort(key = lambda process_queue:process_queue[1])

total_wtime = 0
for i in process_queue[:n-1]:
    total_wtime+=total_wtime+i[1]
print('ProcessName\tBurstTime')
for i in range(n):
    print(process_queue[i][0],'\t\t',process_queue[i][1])

print('Total waiting time: ',total_wtime)
print('Average waiting time: ',(total_wtime/n))
```

Output:

```
$ python3 sjf.py
Enter the total no of processes: 3
Enter p_name: A
Enter p_burst: 3

Enter p_name: B
Enter p_burst: 1

Enter p_name: C
Enter p_burst: 5

ProcessName BurstTime
B        1
A        3
C        5
Total waiting time:  5
Average waiting time:  1.6666666666666667
```

5. Consumer and producer problem using semaphore

Code:

```
import threading
import time


sem = threading.Semaphore()
print("MultiThreading using Semaphore!!")
def fun1():
    while True:
        sem.acquire()
        print("Function 1")
        sem.release()
        time.sleep(0.25)


def fun2():
    while True:
        sem.acquire()
        print("Function 2")
        sem.release()
        time.sleep(0.25)


t1 = threading.Thread(target = fun1)
t1.start()

t2 = threading.Thread(target = fun2)
t2.start()
```

Output:

```
 python3 semophore.py
MultiThreading using Semaphore!!
Function 1
Function 2
Function 1
Function 2
Function 1
Function 2
Function 1
Function 2
Function 1
Function 2
Function 1
Function 2
.
.
.
.
```

6. *Bankers Algo*

Code:

```
O_RESOURCES = 4

#WORKS:

RESOURCES = [ 3, 1, 1, 2 ]

ALLOCATED = [[1, 2, 2, 1],
             [1, 0, 3, 3],
             [1, 2, 1, 0]]

MAX = [[3, 3, 2, 2],
       [1, 2, 3, 4],
       [1, 3, 5, 0]]

# DEADLOCK

# RESOURCES = [3, 0, 1, 2]

# ALLOCATED = [[1, 2, 2, 1],
#              [1, 1, 3, 3],
#              [1, 2, 1, 0]]

# MAX = [[3, 3, 2, 2],
#        [1, 2, 3, 4],
#        [ 1, 3, 5, 0]]

def allocate(allocated, maxi, res):
    if len(allocated)==1:
        safe=1
        for i in range(len(res)):
            if(res[i]<(maxi[0][i]-allocated[0][i])):
                safe=0
        if safe:
            return 1
        else:
            return 0
    ###LENGTH is not 1. So you should try all permutations.
    for p in range(len(allocated)):
        safe = 1
        for i in range(len(allocated[p])):
            if res[i]<(maxi[p][i]-allocated[p][i]):
                safe=0
        if safe:
            new_allocated=allocated[:p]+allocated[p+1:]
            new_maxi=maxi[:p]+maxi[p+1:]
            new_res = res[:]
            new_res=[res[i]+allocated[p][i]for i in range(len(res))]
            if allocate(new_allocated, new_maxi, new_res):
                print ('Incoming request: ')
                print ('maximum: \n%s'%('\n'.join([' '.join([str(item) for item in
row]) for row in maxi])))
```

```
                print( 'allocated: \n%s'%('\n'.join([' '.join([str(item) for item
in row]) for row in allocated])))
                print( 'resources in hand: %s'%(res))
                print ('Allocated %s to %s'%([maxi[p][i]-allocated[p][i] for i in
range(len(maxi[p]))], p))
                print ()
                print ()
                return 1
            else:
                return 0
        else:
            continue
    print("Couldnt allocate")
    return 0



if allocate(ALLOCATED, MAX, RESOURCES):
    print("Done")
```

Output:

```
$ python3 bankers.py
Incoming request:
maximum:
1 2 3 4
1 3 5 0
allocated:
1 0 3 3
1 2 1 0
resources in hand: [4, 3, 3, 3]
Allocated [0, 2, 0, 1] to 0


Incoming request:
maximum:
3 3 2 2
1 2 3 4
1 3 5 0
allocated:
1 2 2 1
1 0 3 3
1 2 1 0
resources in hand: [3, 1, 1, 2]
Allocated [2, 1, 0, 1] to 0


Done
```

7. *Memory allocation*

Code:

```
def first_fit(p, pc):
    for n,i in enumerate(pc):
        for e,j in enumerate(p):
            if i<j:
                print("Partition size for process {}: {}".format(n+1,j))
                p.pop(e)
                break

def best_fit(p, pc):
    for n,i in enumerate(pc):
        t = [tp for tp in p if tp>=i]
        t = min(t)
        print("Partition size for process {}: {}".format(n+1,t))
        p.pop(p.index(t))

def worst_fit(p, pc):
    for n,i in enumerate(pc):
                t = [tp for tp in p if tp>=i]
                t = max(t)
                print("Partition size for process {}: {}".format(n+1,t))
                p.pop(p.index(t))




def cont():
    print("Do you want to continue: (y or n):", end="")
    c = input()
    if c=='n' or c=="N":
        exit()
    elif c=="y" or c=="Y":
        pass
    else:
        print("Please Enter y or n:", end="")
        cont()

def choose(i,p,pc):
    if i==1:
        first_fit(p,pc)
    elif i==2:
        best_fit(p,pc)
    elif i==3:
        worst_fit(p,pc)
    else :
        print("Enter 1,2 or 3")

def main():
    partition_size = []
    process_size = []
    print("No. of memory partition:", end="")
    n_partition = int(input())
    print("--"*5)
    print ("Enter size for each partition:")
    for i in range(n_partition):
        print("Size for partition "+str(i+1)+":", end="")
        partition_size.append(int(input()))
    print("--"*5)
    print ("Enter size for each process:")
    for i in range(n_partition):
```

```
            print("Size for Process "+str(i+1)+":", end="")
            process_size.append(int(input()))
        print("--"*5)
        print("""Select algo for partition:
        1. First Fit
        2. Best Fit
        3. Worst Fit""")
        algo = int(input())
        print("--"*5)
        choose(algo, partition_size, process_size)
        print("--"*5)
        cont()
        print("--"*5)

    if __name__=="__main__":
        while(True):
            try:
                main()
            except Exception as e:
                print("Error: {}".format(e))
```

Output:

```
$ python3 mem_alloc.py
No. of memory partition:3
----------
Enter size for each partition:
Size for partition 1:200
Size for partition 2:250
Size for partition 3:180
----------
Enter size for each process:
Size for Process 1:160
Size for Process 2:200
Size for Process 3:120
----------
Select algo for partition:
    1. First Fit
    2. Best Fit
    3. Worst Fit
2
----------
Partition size for process 1: 180
Partition size for process 2: 200
Partition size for process 3: 250
----------
```

8. *Fragmentation and Compaction*

Code:

```
def main():
    print("Enter no. of partitions:", end="")
    n_partition = int(input())
    partition_size = []
    print ("Enter size for each partition:")
    for i in range(n_partition):
        print("Size for partition "+str(i+1)+":", end="")
        partition_size.append(int(input()))
    print("--"*5)
    print("Enter no. of processes (<{}):".format(n_partition), end="")
    n_proc = int(input())
    proc_size = []
    for i in range(n_proc):
        print("Size for process "+str(i+1)+":", end="")
        proc_size.append(int(input()))

    print("--"*5)
    exf = []
    print("Allocating processes with best-fit algo.....")
    size = sum(partition_size)
    for n,i in enumerate(proc_size):
        t = [tp for tp in partition_size if tp>=i]
        t = min(t)
        size-=i
        print("Partition size for process {}: {}".format(n+1,t))
        partition_size.pop(partition_size.index(t))
        ef = t-i
        if ef>0:
            print("Internal Fragment size: {}".format(ef))
            exf.append(ef)

    print("--"*5)
    print("External Fragments:", exf)
    print("--"*5)
    print("Applying Compaction.....")
    print("Free space available after compaction: {}".format(size))



if __name__=="__main__":
    try:
        main()
    except Exception as e:
        print("Error: {}".format(e))
```

Output:

```
$ python3 fragments.py
Enter no. of partitions:4
Enter the size for each partition:
Size for partition 1:200
Size for partition 2:300
Size for partition 3:180
Size for partition 4:250
----------
Enter no. of processes (<4):2
Size for process 1:150
```

```
Size for process 2:200
----------
Allocating processes with best-fit algo.....
Partition size for process 1: 180
Internal Fragment size: 30
Partition size for process 2: 200
----------
External Fragments: [30]
----------
Applying Compaction......
Free space available after compaction: 580
```

9. *Page Replacement (FIFO)*

Code:

```
def revrs(l):
    return l[::-1]

n_part = int(input("Enter no. of partitions:"))
pages = list(map(int, input("Enter pages string:").split()))

frame = []

page_fault = 0

frame = revrs(list(set(pages)))[-n_part:]
pages = revrs(revrs(pages)[:-n_part])


print("--"*5)
print("Initial frame->", end="")
print(frame)
print("--"*5)
for i,p in enumerate(pages):
    if p not in frame:
        page_fault+=1
        frame.pop()
        frame = revrs(frame)
        frame.append(p)
        frame = revrs(frame)
        print("Frame after PageFault "+str(page_fault)+"->", frame)
        print("--"*5)


print("Total No. of page-faults: {}".format(page_fault))
```

Output:

```
$ python3 fifo_page.py
Enter no. of partitions:3
Enter pages string:6 7 7 3 2 1 2 3 2 2 1 1 2 4 4 3 3 2 2 3 1 7
```

```
----------
Initial frame->[3, 2, 1]
----------
Frame after PageFault 1-> [4, 3, 2]
----------
Frame after PageFault 2-> [1, 4, 3]
----------
Frame after PageFault 3-> [7, 1, 4]
----------
Total No. of page-faults: 3
```

10. *Disc Scheduling (FIFO)*

Code:

```
init = int(input("Enter initial position of head:"))
queue = list(map(int, input("Enter requests queue data:").split()))

seek_time = 0

for i in queue:
    seek_time+=abs(init-i)
    init = i


print("Total Seek Time: {}".format(seek_time))
```

Output:

```
$ python3 fifo_disc.py
Enter initial position of head:50
Enter requests queue data:95 34 57 60 102 134 12 14 87
Total Seek Time: 403
```

Thanks....
