CS331: SYSTEM SOFTWARE LAB

LAB MANUAL

Rajasree R

Asst. Professor

Department of Computer Science and Engineering

College of Engineering Trivandrum

Kerala

2018

INDEX

Cycle – I
Operating System Concepts

Sl. No.	List of Programs
1	Simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time: a) FCFS b) SJF c) Round Robin (pre-emptive) d) Priority
2	Simulate the following file organization techniques: a) Single level directory b) Two level directory c) Hierarchical
3	Implement the banker's algorithm for deadlock avoidance.
4	Simulate the following disk scheduling algorithms: a) FCFS b)SCAN c) C-SCAN
5	Implement the producer-consumer problem using semaphores.
6	Write a program to simulate the working of the dining philosopher's problem.

Cycle – II
Assemblers, Loaders and Macroprocessors

Sl. No.	List of Programs
1	Implement pass one of a two pass assembler.
2	Implement pass two of a two pass assembler.
3	Implement a single pass assembler.
4	Implement a two pass macro processor.
5	Implement an absolute loader.
6	Implement a symbol table with suitable hashing.
7	Implement a one pass macro processor.

Cycle I: Operating System Concepts

1) Simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time. (a) FCFS (b) SJF (c) Round Robin (pre-emptive) (d) Priority

ALGORITHMS:

FCFS Scheduling:

1. Start.
2. Input the processes along with their burst time (bt) and arrival time (at).
3. Find the waiting time (wt) for all processes.
    - As process that arrives first need not wait, wt for that process will be 0, wt[0] = 0.
    - Calculate the wt for all other processes as:

    For process i, wt[i] = (bt[0] + bt[1] + …. +bt[i-1]) – at[i]
4. Find turnaround time (tat) for all processes.

    For process i, tat[i] = wt[i] + bt[i]
5. Compute average waiting time as (total_wt / no_of_processes).
6. Compute average turnaround time as (total_tat / no_of_processes).
7. Stop.

SJF Scheduling:

1. Start.
2. Input the processes along with their burst time (bt).
3. Sort the processes in ascending order of their burst times.
4. Find the waiting time (wt) for all processes.
    - As the process with smallest bt is scheduled first, it need not wait, so wt for that process will be 0, wt[0] = 0.

- Calculate the wt for all other processes as:

  For process i, wt[i] = wt[i-1] + bt[i-1]

5. Find turnaround time (tat) for all processes.

   For process i, tat[i] = wt[i] + bt[i]

6. Compute average waiting time as (total_wt / no_of_processes).
7. Compute average turnaround time as (total_tat / no_of_processes).
8. Stop.

Round Robin Scheduling:

1. Start.
2. Input the processes along with their burst time (bt).
3. Input the time quantum, tq.
4. Create an array rem_bt[] to keep track of the remaining burst time of processes. This array is initially a copy of bt[].
5. Create another array wt[] to store the waiting times of processes. Initialize this array as 0.
6. Initialize time t = 0.
7. Keep traversing all the processes while all processes are not done yet. Do the following for the i^{th} process if it is not done yet.

   a) If rem_bt[i] > tq

   - t = t + tq
   - rem_bt[i] -= tq

   b) Else // Last cycle for this process

   - t = t + rem_bt[i]
   - wt[i] = t – bt[i]
   - rem_bt[i] = 0 // This process is over

8. Find turnaround time (tat) for all processes.

   For process i, tat[i] = wt[i] + bt[i]
9. Compute average waiting time as (total_wt / no_of_processes).
10. Compute average turnaround time as (total_tat / no_of_processes).
11. Stop.

Priority Scheduling:

1. Start.
2. Input the processes along with their burst time (bt) and priority.
3. Sort the processes in ascending order of their priorities.
4. Find the waiting time (wt) for all processes.
   - As the process with highest priority is scheduled first, it need not wait, so wt for that process will be 0, wt[0] = 0.
   - Calculate the wt for all other processes as:

   For process i, wt[i] = wt[i-1] + bt[i-1]
5. Find turnaround time (tat) for all processes.

   For process i, tat[i] = wt[i] + bt[i]
6. Compute average waiting time as (total_wt / no_of_processes).
7. Compute average turnaround time as (total_tat / no_of_processes).
8. Stop.

FCFS:

SAMPLE INPUT:

Enter the number of processes -- 3
Enter Burst Time and Arrival Time for Process 0 – 24 0
Enter Burst Time and Arrival Time for Process 1 – 3 0
Enter Burst Time and Arrival Time for Process 2 – 3 0

SAMPLE OUTPUT:

PROCESS	BURST TIME	ARRIVAL TIME	WAITING TIME	TURNAROUND TIME
P0	24	0	0	24
P1	3	0	24	27
P2	3	0	27	30

Average Waiting Time -- 17.000000
Average Turnaround Time -- 27.000000

SJF:

SAMPLE INPUT:

Enter the number of processes -- 4
Enter Burst Time for Process 0 -- 6
Enter Burst Time for Process 1 -- 8
Enter Burst Time for Process 2 -- 7
Enter Burst Time for Process 3 -- 3

SAMPLE OUTPUT:

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
P3	3	0	3
P0	6	3	9
P2	7	9	16
P1	8	16	24

Average Waiting Time -- 7.000000
Average Turnaround Time -- 13.000000

Round Robin:

SAMPLE INPUT:

Enter the no of processes – 3
Enter Burst Time for process 1 – 24
Enter Burst Time for process 2 -- 3
Enter Burst Time for process 3 – 3
Enter the time quantum – 3

SAMPLE OUTPUT:

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
P1	24	6	30
P2	3	4	7
P3	3	7	10

Average Turnaround time – 15.666667
Average Waiting time is -- 5.666667

Priority:

SAMPLE INPUT:

Enter the number of processes -- 5
Enter the Burst Time & Priority of Process 0 --- 10 3
Enter the Burst Time & Priority of Process 1 --- 1 1
Enter the Burst Time & Priority of Process 2 --- 2 4
Enter the Burst Time & Priority of Process 3 --- 1 5
Enter the Burst Time & Priority of Process 4 --- 5 2

SAMPLE OUTPUT:

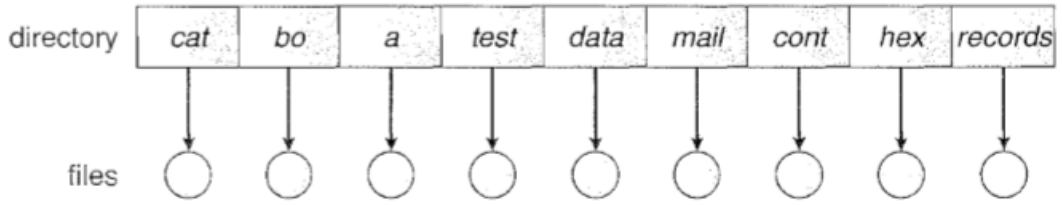
PROCESS	PRIORITY	BURST TIME	WAITING TIME	TURNAROUND TIME
P1	1	1	0	1
P4	2	5	1	6
P0	3	10	6	16
P2	4	2	16	18
P3	5	1	18	19

Average Waiting Time is --- 8.200000
Average Turnaround Time is --- 12.000000

2) Simulate the following file organization techniques (a) Single level directory (b) Two level directory (c) Hierarchical

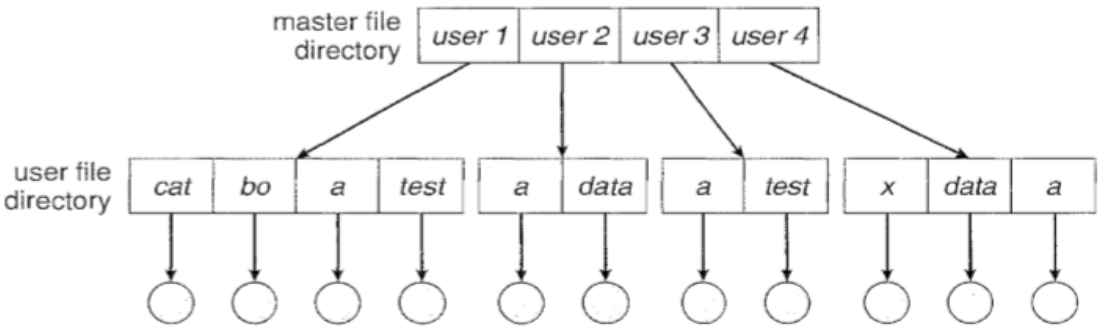
Single-Level Directory System:

In a single-level directory system, all the files are placed in one directory. There is a root directory which has all files. It has a simple architecture and there are no sub directories. Advantage of single-level directory system is that it is easy to find a file in the directory. A single-level directory has significant limitations, however, when the number of files increases or when the system has more than one user. Since all files are in the same directory, they must have unique names.



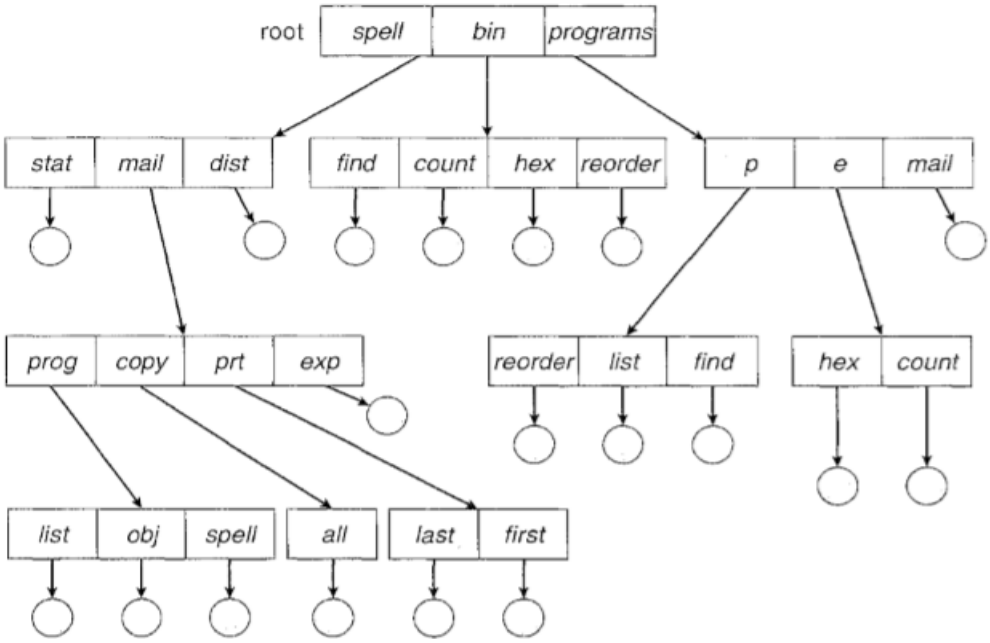
Two-Level Directory System:

In the two-level directory system, each user has own user file directory (UFD). The system maintains a master block that has one entry for each user. This master block contains the addresses of the directory of the users. When a user job starts or a user logs in, the system's master file directory (MFD) is searched. When a user refers to a particular file, only his own UFD is searched. This effectively solves the name collision problem and isolates users from one another. Isolation is an advantage when the users are completely independent but is a disadvantage when the users want to cooperate on some task and to access one another's files.



Hierarchical Directory System:

Hierarchical directory structure, also called a tree-structured directory allows users to create their own subdirectories and to organize their files accordingly. A tree is the most common directory structure. The tree has a root directory, and every file in the system has a unique path name. A directory (or subdirectory) contains a set of files or subdirectories.



SAMPLE OUTPUT:

1. Single-level Directory

2. Two-level Directory
3. Hierarchical Directory
4. Exit

Enter your choice: 1

Enter name of directory -- CSE

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice – 1

Enter the name of the file -- A

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice – 1

Enter the name of the file -- B
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice – 1

Enter the name of the file -- C

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice – 4

The Files are -- A B C

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice – 3

Enter the name of the file – ABC
File ABC not found

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice – 2

Enter the name of the file – B

File B is deleted

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice – 5

Enter your choice: 2

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit
Enter your choice -- 1

Enter name of directory -- DIR1
Directory created

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit
Enter your choice -- 1

Enter name of directory -- DIR2
Directory created

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit
Enter your choice -- 2

Enter name of the directory – DIR1
Enter name of the file -- A1
File created

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit
Enter your choice -- 2

Enter name of the directory – DIR1
Enter name of the file -- A2
File created

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit
Enter your choice -- 2

Enter name of the directory – DIR2

Enter name of the file -- B1
File created

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit
Enter your choice -- 5
Directory Files
DIR1 A1 A2
DIR2 B1

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit
Enter your choice -- 4

Enter name of the directory – DIR
Directory not found

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit
Enter your choice -- 3
Enter name of the directory – DIR1
Enter name of the file -- A2

File A2 is deleted

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit
Enter your choice – 6

Enter your choice: 3

Enter Name of dir/file (under root): ROOT
Enter 1 for Dir / 2 For File : 1
No of subdirectories / files (for ROOT) :2

Enter Name of dir/file (under ROOT):USER 1
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for USER 1):1

Enter Name of dir/file (under USER 1):SUBDIR
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for SUBDIR):2

Enter Name of dir/file (under USER 1):JAVA
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for JAVA): 0

Enter Name of dir/file (under SUBDIR):VB
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for VB): 0

Enter Name of dir/file (under ROOT):USER2
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for USER2):2

Enter Name of dir/file (under ROOT):A
Enter 1 for Dir /2 for file:2

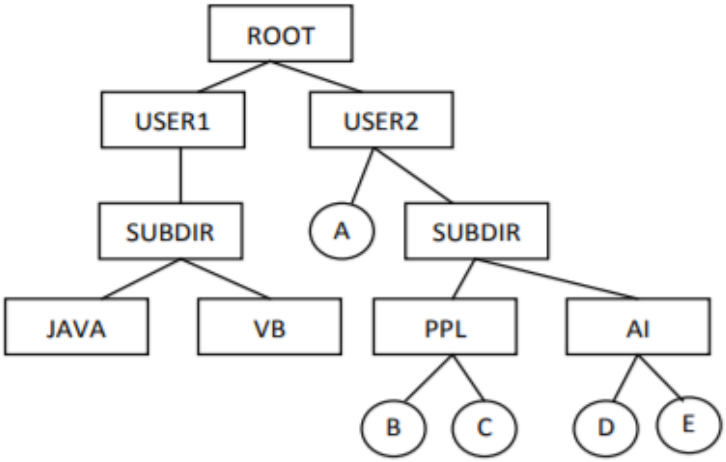
Enter Name of dir/file (under USER2):SUBDIR 2
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for SUBDIR 2):2

Enter Name of dir/file (under SUBDIR2):PPL
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for PPL):2
Enter Name of dir/file (under PPL):B
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under PPL):C
Enter 1 for Dir /2 for file:2

Enter Name of dir/file (under SUBDIR):AI
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for AI): 2

Enter Name of dir/file (under AI):D
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under AI):E

Enter 1 for Dir /2 for file:2


ROOT
USER1
USER2
SUBDIR
A
SUBDIR
JAVA
VB
PPL
AI
B
C
D
E

3. Implement the banker's algorithm for deadlock avoidance.

Bankers algorithm needs the following data structures, where *n* is the number of processes in the system and *m* is the number of resource types:

Available: A vector of length *m* indicates the number of available resources of each type.

Max: An *n* x *m* matrix defines the maximum demand of each process.

Allocation: An *n* x *m* matrix defines the number of resources of each type currently allocated to each process.

Need: An *n* x *m* matrix indicates the remaining resource need of each process.

SAFETY ALGORITHM:

1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively. Initialize *Work = Available* and *Finish[i] =false* for *i* = 0, 1, ... , *n* - 1.

2. Find an index *i* such that both
   a. *Finish[i] ==false*
   b. $Need_i \leq Work$

   If no such *i* exists, go to step 4.

3. *Work = Work + Allocation;*
   Finish[i] = true
   Go to step 2.

4. If *Finish[i] ==true* for all *i*, then the system is in a safe state.

RESOURCE REQUEST ALGORITHM:

Let $Request_i$ be the request vector for process P_i. If $Request_i[j] == k$, then process P_i wants k instances of resource type R_j. When a request for resources is made by process P_i, the following actions are taken:

1. If $Request_i \leq Need_i$, go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.

2. If $Request_i \leq Available$, go to step 3. Otherwise, P_i must wait, since the resources are not available.
3. Have the system pretend to have allocated the requested resources to process , P_i by modify the state as follows:

$$Available = Available - Request_i ;$$
$$Allocation_i = Allocation_i + Request_i ;$$
$$Need_i = Need_i - Request_i ;$$

If the resulting resource-allocation state is safe, the transaction is completed, and process P_i is allocated its resources. However, if the new state is unsafe, then P_i must wait for $Request_i$, and the old resource-allocation state is restored.

SAMPLE INPUT:

Enter the number of resources: 4
Enter the maximum instances of each resources:
a=3
b=14
c=12
d=12
Enter the number of processes: 5
Enter the allocation matrix:

	a	b	c	d
P[0]	0	0	1	2
P[1]	1	0	0	0
P[2]	1	3	5	4
P[3]	0	6	3	2
P[4]	0	0	1	4

Enter the MAX matrix:

	a	b	c	d
P[0]	0	0	1	2
P[1]	1	7	5	0
P[2]	2	3	5	6
P[3]	0	6	5	2
P[4]	0	6	5	6

SAMPLE OUTPUT:

Safe Sequence: <P[0], P[2], P[3], P[4], P[1]>

4) Simulate the following disk scheduling algorithms. (a) FCFS (b) SCAN (c) C-SCAN

a)FCFS

It is the simplest form of disk scheduling algorithms. The I/O requests are served or processes according to their arrival. The request arrives first will be accessed and served first. Since it follows the order of arrival, it causes the wild swings from the innermost to the outermost tracks of the disk and vice versa. The farther the location of the request being serviced by the read/write head from its current location, the higher the seek time will be.

SAMPLE INPUT:

Enter the size of Queue 8
Enter the Queue 98, 183,37,122, 14,124,65,67
Enter the initial head position 53

SAMPLE OUTPUT:

Move from 53 to 98 with seek 45
Move from 98 to 183 with seek 85
Move from 183 to 37 with seek 146
Move from 37 to 122 with seek 85
Move from 122 to 14 with seek 108
Move from 14 to 124 with seek 110
Move from 124 to 65 with seek 59
Move from 65 to 67 with seek 2
Total seek time is 640
Average seek time is 80.00000

b)SCAN

This algorithm is performed by moving the R/W head back-and-forth to the innermost and outermost track. As it scans the tracks from end to end, it process all the requests found in the direction it is headed. This will ensure that all track requests, whether in the outermost, middle or innermost location, will be traversed by the access arm thereby finding all the requests. This is also known as the Elevator algorithm.

SAMPLE INPUT:

Enter the size of Queue 8
Enter the Queue 98, 183,37,122, 14,124,65,67
Enter the initial head position 53

SAMPLE OUTPUT:

Move from 53 to 37 with seek 16
Move from 37 to 14 with seek 23
Move from 14 to 0 with seek 14
Move from 0 to 65 with seek 65
Move from 65 to 67 with seek 2
Move from 67 to 98 with seek 31
Move from 98 to 122 with seek 24
Move from 122 to 124 with seek 2
Move from 124 to 183 with seek 59

Total seek time is 236
Average seek time is 29.5

c)C-SCAN

This algorithm is a modified version of the SCAN algorithm. C-SCAN sweeps the disk from end-to-end, but as soon it reaches one of the end tracks it then moves to the other end track without servicing any requesting location. As soon as it reaches the other end track it then starts servicing and grants requests headed to its direction. This algorithm improves the unfair situation of the end tracks against the middle tracks.

SAMPLE INPUT:

Enter the size of Queue 8
Enter the Queue 98, 183,37,122, 14,124,65,67
Enter the initial head position 53

SAMPLE OUTPUT:

Move from 53 to 65 with seek 12
Move from 65 to 67 with seek 2
Move from 67 to 98 with seek 31
Move from 98 to 122 with seek 24
Move from 122 to 124 with seek 2
Move from 124 to 183 with seek 59
Move from 183 to 199 with seek 16
Move from 199 to 0 with seek 199
Move from 0 to 14 with seek 14
Move from 14 to 37 with seek 23

Total seek time is 382
Average seek time is 47.75

5) Implement the producer-consumer problem using semaphores.

ALGORITHM - Producer:

```
do
{
        // wait until empty  > 0 and then decrement  'empty'
        wait(empty);
        //acquire lock
        wait(mutex);
        /* perform the insert operation in a slot */
        //release lock
        signal(mutex);
        //increment 'full'
        signal(full);
}
while (TRUE)
```

ALGORITHM – Consumer:

```
do
{
    // wait until full >0 and then decrement 'full'
    wait(full);
    //acquire the lock
    wait(mutex);
    /* perform the remove operation in a slot */
    //release the lock
    signal(mutex);
    //increment 'empty'
    signal(empty);
}
while (TRUE)
```

SAMPLE OUTPUT:

1.Producer

2.Consumer

3.Exit

Enter your choice:1

Producer produces the item 1

Enter your choice:2

Consumer consumes item 1

Enter your choice:2

Buffer is empty!!

Enter your choice:1

Producer produces the item 1

Enter your choice:1

Producer produces the item 2

Enter your choice:1

Producer produces the item 3

Enter your choice:1

Buffer is full!!

Enter your choice:3

6) Write a program to simulate the working of the dining philosopher's problem.

ALGORITHM:

There is a dining room containing a circular table with five chairs. At each chair is a plate, and between each plate is a single chopstick. In the middle of the table is a bowl of spaghetti. Near the room are five philosophers who spend most of their time thinking, but who occasionally get hungry and need to eat so they can think some more.

In order to eat, a philosopher must sit at the table, pick up the two chopsticks to the left and right of a plate, then serve and eat the spaghetti on the plate.

Thus, each philosopher is represented by the following pseudo code:

```
process P[i]
        while true do
          { THINK;
            PICKUP(CHOPSTICK[i], CHOPSTICK[i+1 mod 5]);
            EAT;
            PUTDOWN(CHOPSTICK[i], CHOPSTICK[i+1 mod 5])
           }
```

SAMPLE OUTPUT:

Philosopher 1 is thinking….

Philosopher 1 is eating….

Philosopher 2 is thinking….

Philosopher 3 is thinking….

Philosopher 3 is eating….

Philosopher 4 is thinking….

Philosopher 5 is thinking….

Philosopher 1 finished eating….

Philosopher 3 finished eating….

Philosopher 5 is eating….

Philosopher 2 is eating….

Philosopher 4 is thinking….

Philosopher 5 finished eating….

Philosopher 2 finished eating….

Philosopher 4 is eating….

Philosopher 4 finished eating….

Example Questions:

1. Write a program to implement round robin CPU scheduling algorithm for the following given scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Consider the time quantum size for the system processes and user processes to be 5 msec and 2 msec respectively.
2. Write a program to simulate pre-emptive SJF CPU scheduling algorithm.
3. Write a program to simulate a two-level index scheme for file allocation.
4. Write a program to implement compaction technique.
5. Write a program to simulate the following contiguous memory allocation techniques: best-fit, worst-fit, first-fit.
6. Write a program to simulate page replacement algorithms: a) FIFO b) LRU c) LFU
7. Write a program to simulate multi-level queue scheduling algorithm.
8. Implement the following memory management techniques: Paging and Segmentation.
9. Write a program to simulate general graph directory structure.
10. Write a program to implement deadlock detection technique for the following scenarios: (a) Single instance of each resource type (b) Multiple instances of each resource type.
11. Write a program to implement SSTF disk scheduling algorithm.
12. Write a program to simulate LRU-approximation page replacement algorithm: (a) Additional-Reference bits algorithm (b) Second-chance algorithm.
13. Write a program to simulate producer-consumer problem using message-passing system.
14. Write a program to simulate readers-writers problem using monitors.

Viva Questions:

1. Define operating system.
2. What are the different types of operating systems?
3. Define a process.
4. What is CPU Scheduling?
5. Define arrival time, burst time, waiting time, turnaround time.
6. What is the advantage of round robin CPU scheduling algorithm?
7. Which CPU scheduling algorithm is for real-time operating system?
8. In general, which CPU scheduling algorithm works with highest waiting time?
9. Is it possible to use optimal CPU scheduling algorithm in practice?
10. What is the real difficulty with the SJF CPU scheduling algorithm?
11. What is multi-level queue CPU Scheduling?
12. Differentiate between the general CPU scheduling algorithms like FCFS, SJF etc. and multi-level queue CPU Scheduling.
13. What are CPU-bound I/O-bound processes?
14. What are the parameters to be considered for designing a multilevel feedback queue scheduler?
15. Differentiate multi-level queue and multi-level feedback queue CPU scheduling algorithms.
16. Define file.
17. What are the different kinds of files?
18. What is the purpose of file allocation strategies?
19. Identify ideal scenarios where sequential, indexed and linked file allocation strategies are most appropriate.
20. What are the disadvantages of sequential file allocation strategy?
21. What is an index block?
22. What is the file allocation strategy used in UNIX?

23. What is the purpose of memory management unit?
24. Differentiate between logical address and physical address.
25. What are the different types of address binding techniques?
26. What is the basic idea behind contiguous memory allocation?
27. How is dynamic memory allocation useful in multiprogramming operating systems?
28. What is fragmentation? Differentiate internal and external fragmentation.
29. Which of the dynamic contiguous memory allocation strategies suffer with external fragmentation?
30. What are the possible solutions for the problem of external fragmentation?
31. What is 50-percent rule?
32. What is compaction?
33. Which of the memory allocation techniques first-fit, best-fit, worst-fit is efficient? Why?
34. What are the advantages of non-contiguous memory allocation schemes?
35. What is the process of mapping a logical address to physical address with respect to the paging memory management technique?
36. Define the terms – base address, offset.
37. Differentiate between paging and segmentation memory allocation techniques.
38. What is the purpose of page table?
39. What is the effect of paging on the overall context-switching time?
40. Define directory.
41. Which of the directory structures is efficient? Why?
42. Which directory structure does not provide user-level isolation and protection?
43. What is the advantage of hierarchical directory structure?

44. What is deadlock? What are the conditions to be satisfied for the deadlock to occur?
45. How can be the resource allocation graph used to identify a deadlock situation?
46. How is Banker's algorithm useful over resource allocation graph technique?
47. Differentiate between deadlock avoidance and deadlock prevention.
48. What is disk scheduling?
49. List the different disk scheduling algorithms.
50. Define the terms – disk seek time, disk access time and rotational latency.
51. What is the advantage of C-SCAN algorithm over SCAN algorithm?
52. Which disk scheduling algorithm has highest rotational latency? Why?
53. Define the concept of virtual memory.
54. What is the purpose of page replacement?
55. Define the general process of page replacement.
56. List out the various page replacement techniques.
57. What is page fault?
58. Which page replacement algorithm suffers with the problem of Belady's anomaly?
59. Define the concept of thrashing. What is the scenario that leads to the situation of thrashing?
60. What are the benefits of optimal page replacement algorithm over other page replacement algorithms?
61. Why can't the optimal page replacement technique be used in practice?
62. What is the need for process synchronization?
63. Define a semaphore.
64. Define producer-consumer problem.

65. Discuss the consequences of considering bounded and unbounded buffers in producer-consumer problem.
66. Can producer and consumer processes access the shared memory concurrently? If not, which technique provides such a benefit?
67. Differentiate between a monitor, semaphore and a binary semaphore.
68. Define clearly the dining-philosophers problem.
69. Identify the scenarios in the dining-philosophers problem that leads to the deadlock situations.

Cycle II: Assemblers, Loaders and Macro-processors

1) Implement pass one of a two pass assembler.

ALGORITHM:

Pass 1:

```
begin
  read first input line
  if OPCODE = 'START' then
     begin
        save #[OPERAND] as starting address
        initialize LOCCTR to starting address
        write line to intermediate file
        read next input line
     end {if START}
  else
     initialize LOCCTR to 0
  while OPCODE ≠ 'END' do
     begin
        if this is not a comment line then
           begin
              if there is a symbol in the LABEL field then
                 begin
                    search SYMTAB for LABEL
                    if found then
                       set error flag (duplicate symbol)
                    else
                       insert (LABEL,LOCCTR) into SYMTAB
                 end {if symbol}
              search OPTAB for OPCODE
              if found then
                 add 3 {instruction length} to LOCCTR
              else if OPCODE = 'WORD' then
                 add 3 to LOCCTR
              else if OPCODE = 'RESW' then
                 add 3 * #[OPERAND] to LOCCTR
              else if OPCODE = 'RESB' then
                 add #[OPERAND] to LOCCTR
              else if OPCODE  = 'BYTE' then
                 begin
                    find length of constant in bytes
                    add length to LOCCTR
                 end {if BYTE}
              else
                 set error flag (invalid operation code)
           end {if not a comment}
        write line to intermediate file
        read next input line
     end {while not END}
  write last line to intermediate file
  save (LOCCTR - starting address) as program length
end {Pass 1}
```

SAMPLE INPUT:

In SAMPLE.TXT

```
          START  1000
FIRST    LDA      ALPHA
-          ADD    BETA
-          STA     GAMA
ALPHA  WORD 5
BETA     WORD 10
ALPHA  WORD 2
BETA     RESW  4
GAMA   RESW  1
-          END
```

SAMPLE OUTPUT:

SYMBOL TABLE

label	address
FIRST	1000
ALPHA	1009
BETA	1012
GAMA	1030

In OUTPUT.TXT

```
1000      START 1000
1000      FIRST   LDA       ALPHA
1003      -          ADD      BETA
1006      -          STA       GAMA
1009      ALPHA WORD   5
1012      BETA              WORD 10
```

duplicate symbol
ALPHA

duplicate symbol
BETA

```
1030      GAMA RESW 1
          -          END    SYMBOL
```

Program length is 36.

2) Implement pass two of a two pass assembler.

ALGORITHM:

Pass 2:

```
begin
  read first input line (from intermediate file)
  if OPCODE = 'START' then
     begin
        write listing line
        read next input line
     end (if START)
  write Header record to object program
  initialize first Text record
  while OPCODE ≠ 'END' do
     begin
        if this is not a comment line then
           begin
              search OPTAB for OPCODE
              if found then
                 begin
                    if there is a symbol in OPERAND field then
                       begin
                          search SYMTAB for OPERAND
                          if found then
                             store symbol value as operand address
                          else
                             begin
                                store 0 as operand address
                                set error flag (undefined symbol)
                             end
                       end (if symbol)
                    else
                       store 0 as operand address
                    assemble the object code instruction
                 end (if opcode found)
              else if OPCODE = 'BYTE' or 'WORD' then
                 convert constant to object code
              if object code will not fit into the current Text record then
                 begin
                    write Text record to object program
                    initialize new Text record
                 end
              add object code to Text record
           end (if not comment)
        write listing line
        read next input line
     end (while not END)
  write last Text record to object program
  write End record to object program
  write last listing line
```

SAMPLE INPUT:

In INNER.TXT:

```
1000    COPY    START   1000
1000    -       LDA     ALPHA
1003    -       ADD     BETA
1006    -       STA     GAMMA
1009    ALPHA   WORD    15
100A    BETA    WORD    15
100C    GAMMA   RESW    1
-       -       END     -
```

SAMPLE OUTPUT:

SYMTAB.TXT

```
ALPHA 1009
BETA    100A
GAMMA   100C
```

OUTPUT:

H^COPY^00%^00T^001000^F^031009^16100a^14100c^ADDf^ADDf^ADDf
E^00100

3) Implement a single pass assembler**.**

ALGORITHM:

begin

```
read first input line
if OPCODE = 'START' the
        begin
                save #[OPERAND] as starting address
                initialize LOCCTR as starting address
                read next input line
        end {if START}
else
        initialize LOCCTR to 0
while OPCODE != 'END' do
        begin
                if this is not a comment line then
                        begin
                                if there is a symbol in the LABEL field then
                                        begin
                                                search SYMTAB for LABEL
                                                if found then
                                                        begin
                                                                if symbol value is null
                                                                        set symbol value
as LOCCTR and search the linked list with the corresponding operand PTR addresses and
generate operand addressses as corresponding symbol values. Set symbol value as LOCCTR
in symbol table and delete the linked list
                                                        end {if found}
                                                else
                                                        insert (LABEL,LOCCTR) into
SYMTAB
                                        end {if SYMBOL}
                                search OPTAB for OPCODE
                                if found then
                                        begin
                                                search SYMTAB for OPERAND
address
                                                if found then
                                                        if symbol value not equal to null
then
                                                                store symbol value as
OPERAND address
                                                        else
                                                                insert at the end of the
linked list with a node with address as LOCCTR
                                                else
                                                        insert (symbol name, null)
                                                add 3 to LOCCTR
                                        end {if found OPCODE}
                                else if OPCODE = 'WORD' then
                                        add 3 to LOCCTR and convert constant to
object code
                                else if OPCODE ='RESW' then
                                        add 3 * #[OPERAND] to LOCCTR
```

```
                                    else if OPCODE = 'RESB' then
                                            add #[OPERAND] to LOCCTR
                                    else if OPCODE = 'BYTE' then
                                            begin
                                                    find length of constant in bytes
                                                    add length to LOCCTR
                                                    convert constant to object code
                                            end
                                    if object code will not fit into current Text record then
                                            begin
                                                    write text record to object program
                                                    initialize new text record
                                            end
                                    add object code to Text record
                            end {if not comment}
                    write listing line
                    read next input line
            end {if not END}
        write last Text record to object program
        write End record to object program
        write last listing line
end {PASS 1}
```

SAMPLE INPUT:

```
COPY START 1000
LDA ALPHA
STA ALPHA
ADD BETA
STA GAMMA
```

```
EOF BYTE C'EOF'
. this is a comment
HEX BYTE X'F0'
ALPHA WORD 1
BETA WORD 2
GAMMA RESW 4096
END COPY
```

SAMPLE OUTPUT:

```
H^COPY^001000^003016
T^001000^10^000000^500000^6B0000^500000^454f46^F0
T^001001^02^1010
T^001004^02^1010
T^001010^03^000001
T^001007^02^1013
T^001013^03^000002
T^00100a^02^1016
E^001000
```

4) Implement a two pass macro processor**.**

ALGORITHM:

Pass 1

- pass 1 of macro processor makes a line by line scan over its input.
- Set MDTC=1 as well as MNTC=1.
- Read next line from input program.
- If it is a MACRO pseudo-op, the entire macro definition except this (MACRO) line is stored in MDT.
- The name is entered into Macro Name Table along with a pointer to the first location of MDT entry of the definition.
- When the END pseudo-op is encountered all the macro-definitions have been processed, so control is transferred to pass2.

Pass 2

- This algorithm reads one line input program at a time.
- For each Line it checks if op-code of that line matches any of the MNT entry.
- When match is found (i.e. when call is pointer called MDTF to corresponding macro definition stored in MDT.
- The initial value of MDTP is obtained from MDT index field on MNT entry.
- The macro expander prepares the ALA consisting of a table of dummy argument indices & corresponding arguments to the call.
- Reading proceeds from the MDT, as each successive line is read, the values from the argument list one substituted for dummy arguments indices in the macro definition.
- Reading MEND line in MDT terminates expansion of macro & scanning continues from the input file.
- When END pseudo-op encountered, the expanded source program is given to the assembler.

Pass 1:

SAMPLE INPUT:

In minp2.txt (input file)

```
EX1     MACRO   &A,&B
-       LDA     &A
-       STA     &B
-       MEND    -
```

```
SAMPLE  START   1000
-       EX1     N1,N2
N1      RESW    1
N2      RESW    1
-       END     -
```

SAMPLE OUTPUT:

In dtab2.txt

```
EX1     &A,&B
LDA     &A
STA     &B
MEND
```

ntab2.txt

```
EX1
```

Pass 2:

SAMPLE INPUT:

minp2.txt

```
EX1     MACRO   &A,&B
-       LDA     &A
-       STA     &B
-       MEND    -
SAMPLE  START   1000
-       EX1     N1,N2
N1      RESW    1
N2      RESW    1
-       END     -
```

dtab2.txt

```
EX1     &A,&B
LDA     &A
```

```
STA     &B
MEND
```

ntab2.txt

```
EX1
```

SAMPLE OUTPUT:

atab2.txt

```
N1
N2
```

op2.txt

```
SAMPLE  START   1000
.       EX1     N1,N2
-       LDA     N1
-       STA     N2
N1      RESW    1
N2      RESW    1
-       END     -
```

5) Implement an absolute loader.

ALGORITHM:

```
begin
   read Header record
   verify program name and length
   read first Text record
   while record type ≠ 'E' do
     begin
       {if object code is in character form, convert into
          internal representation}
       move object code to specified location in memory
       read next object program record
     end
   jump to address specified in End record
end
```

INPUT:

In text file:

H^SAMPLE^001000^0C

T^001000^09^001003^181006^0C1009^$

T^002000^03^001010^$

E^001000

OUTPUT:

Enter the program name: SAMPLE

001000 00

001001 10

001002 03

001003 18

001004 10

001005 06

001006 0C

001007 10

001008 09

002000 00

002001 10

002002 10

6) Implement a symbol table with suitable hashing.

Symbol table is an important data structure created and maintained by compilers in order to store information about the occurrence of various entities such as variable names, function names, objects, classes, interfaces, etc. Symbol table is used by both the analysis and the synthesis parts of a compiler. Each entry in symbol table is associated with attributes that support compiler in different phases.

A symbol table may serve the following purposes depending upon the language in hand:

- To store the names of all entities in a structured form at one place.
- To verify if a variable has been declared.
- To implement type checking, by verifying assignments and expressions in the source code are semantically correct.
- To determine the scope of a name (scope resolution).

Items stored in the symbol table:

- Variable names and constants
- Procedure and function names
- Literal constants and strings
- Compiler generated temporaries
- Labels in source languages

A symbol table is simply a table which can be either linear or a hash table. It maintains an entry for each name in the following format:

< symbol name, type, attribute >

The basic operations defined on a symbol table include:

- allocate – to allocate a new empty symbol table
- free – to remove all entries and free the storage of a symbol table
- insert – to insert a name in a symbol table and return a pointer to its entry
- lookup – to search for a name and return a pointer to its entry
- set attribute – to associate an attribute with a given entry
- get attribute – to get an attribute associated with a given entry

Implementing Symbol Tables with hashing:

Symbol tables are mostly implemented as hash tables, where the source code symbol itself is treated as a key for the hash function and the return value is the information about the symbol.

A hash table is an array with index range from 0 to tablesize – 1.These entries are pointer pointing to names of symbol table. To search for a name we use hash function that will result in any integer between 0 to tablesize – 1. With hashing, insertion and lookup can be done in O(1) time. Advantage is quick search is possible and disadvantage is that hashing is complicated to implement.

ALGORITHM:

1. Start.
2. Define the structure of the Symbol Table
3. Enter the choice for performing the operations in the Symbol Table
4. If the entered choice is 1, search the symbol table for the symbol to be inserted. If the symbol is already present, it displays "Duplicate Symbol", else, insert the symbol and the corresponding address in the symbol table.
5. If the entered choice is 2, delete the particular symbol.
6. If the entered choice is 3, the symbols present in the symbol table are displayed.

7. If the entered choice is 4, the symbol is searched in the symbol table. If it is not found in the symbol table it displays "Not found".
8. If the entered choice is 5, the symbol to be modified is searched in the symbol table. The address of the label can be modified.
9. Enter choice 6 to exit the program.
10. Stop.

OUTPUT:

1.Insert

2.Delete

3.Search

4.Modify

5.Display

6.Exit

Enter the choice: 1

Enter the symbol to be inserted: abc

Enter the datatype: int

Enter the value: 43

1.Insert

2.Delete

3.Search

4.Modify

5.Display

6.Exit

Enter the choice: 5

0

1 [main, int, 2,]

2 [f, int, 2,]

3 [ch, char,1,]

4 [abc, int, 2, 43]

5

6

7 [a, int, 2,][add, float, 4,]

8 [b, int, 2, 340]

9 [c, float, 4,]

1.Insert

2.Delete

3.Search

4.Modify

5.Display

6.Exit

Enter the choice: 2

Enter the symbol to be deleted: add

1.Insert

2.Delete

3.Search

4.Modify

5.Display

6.Exit

Enter the choice: 5

0

1 [main, int, 2,]

2 [f, int, 2,]

3 [ch, char,1,]

4 [abc, int, 2, 43]

5

6

7 [a, int, 2,]

8 [b, int, 2, 340]

9 [c, float, 4,]

1.Insert

2.Delete

3.Search

4.Modify

5.Display

6.Exit

Enter the choice: 2

Enter the symbol to be deleted: s

Symbol Not Found!

1.Insert

2.Delete

3.Search

4.Modify

5.Display

6.Exit

Enter the choice: 3

Enter the symbol to be searched: abc

The symbol is present:

name: abc

value: 43

type: int

size: 2

1.Insert

2.Delete

3.Search

4.Modify

5.Display

6.Exit

Enter the choice: 3

Enter the symbol to be searched: d

Symbol doesn’t exist!

1.Insert

2.Delete

3.Search

4.Modify

5.Display

6.Exit

Enter the choice: 4

Enter the symbol to be searched: a

Enter the new value: 10

1.Insert

2.Delete

3.Search

4.Modify

5.Display

6.Exit

Enter the choice: 5

0

1 [main, int, 2,]

2 [f, int, 2,]

3 [ch, char,1,]

4 [abc, int, 2, 43]

5

6

7 [a, int, 2, 10]

8 [b, int, 2, 340]

9 [c, float, 4,]

1.Insert

2.Delete

3.Search

4.Modify

5.Display

6.Exit

Enter the choice: 6

7) Implement a one pass macro processor.

ALGORITHM:

```
begin {macro processor}
        EXPANDING : = FALSE
            while OPCODE ≠ 'END' do
                begin
                    GETLINE
                    PROCESSLINE
                    end  {while}
            end {macro processor}
        procedure PROCESSLINE
            begin
                search NAMTAB for OPCODE
                if found then
                    EXPAND
                else if OPCODE = 'MACRO' then
                    DEFINE
                else write source line to expanded file
                end {PROCESSLINE}

procedure EXPAND
    begin
        EXPANDING : = TRUE
        get first line of macro definition {prototype} from DEFTAB
        set up arguments from macro invocation in ARGTAB
        write macro invocation to expanded file as a comment
        while not end of macro definition do
            begin
                GETLINE
                PROCESSLINE
                end {while}
            EXPANDING : = FALSE
    end  {EXPAND}

procedure GETLINE
    begin
        if EXPANDING then
            begin                    get next line of macro definition from DEFTAB
                substitute arguments from ARGTAB for positional notation
            end {if}
    else
        read next line from input file
  end {GETLINE}
```

SAMPLE INPUT:

In Input.txt:

```
EX1 MACRO &A,&B
- LDA &A
- STA &B
- MEND -
SAMPLE START 1000
- EX1 N1,N2
N1 RESW 1
N2 RESW 1
- END -
```

SAMPLE OUTPUT:

In Argtab.txt:s

```
N1
N2
```

In Op.txt:

```
SAMPLE START 1000
. EX1 N1,N2
- LDA ?1
- STA ?2
N1 RESW 1
N2 RESW 1
- END -
```

In Deftab.txt:

```
EX1 &A,&B
LDA ?1
STA ?2
```

MEND

In Namtab.txt:

EX1

Example questions:

1. Implement a one-pass macro processor.
2. Implement a relocatable loader.

Viva questions:

1. Define system software.
2. What is an Assembler?
3. What is instruction set?
4. What is direct addressing mode and indirect addressing mode?
5. Differentiate between Assembler and Interpreter.
6. List the types of registers used in a system.
7. What are the instruction formats of SIC/SC?
8. What are Assembler directives or pseudo-instructions?
9. Give some examples for assembler directives.
10. What are functions required in translation of source program to object code.
11. What is forward reference?
12. What are the tree types of records in a simple object program format?
13. What is the information present in a Header record?
14. What is the information present in a Modification record?
15. What is the information present in a Define record?
16. What is the information present in a Refer record?
17. What are functions performed in Pass 1 by a two pass assembler?
18. What are functions performed in Pass 2 by a two pass assembler?
19. Name the data structures used by an assembler.
20. What is OPTAB?
21. What is SYMTAB?
22. What is LOCCTR?
23. Name the addressing modes used for assembling register-to-memory instructions.

24. What is the advantage of register-to-register instructions?
25. What is a re-locatable program?
26. Name the two methods of performing relocation?
27. What are the machine independent assembler features?
28. What does an assembler perform when it encounters LTORG assembler directive?
29. What is LITTAB or What is basic data structure needed to handle literal?
30. Name the symbol defining statements.
31. What is the use of the symbol defining statement EQU?
32. What is the use of the symbol defining statement ORG?
33. What is relative expression?
34. What is absolute expression?
35. List the types of Assemblers.
36. How assemblers handle forward reference instructions?
37. List the types of one pass Assemblers.
38. What is load-and-go assembler?
39. What is multi-pass assembler?
40. What is MASM assembler?
41. What is near jump and far jump?
42. What is a bootstrap loader?
43. What is an absolute loader?
44. What are the disadvantages of an absolute loader or machine dependent loader?
45. What is a bit mask?
46. What is the purpose of the relocation bit in object code of relocation loader or what is a relocation bit?
47. What is external reference?
48. What is EXTDEF?
49. What is EXTREF?

50. What are data structures needed for linking loader?
51. What is the use ESTAB?
52. What is a macro?
53. How does the macro processor help the programmer?
54. What are the two main assembler directives use with macro definitions?
55. What is the logic behind the two-pass macro processor?
56. What are the three main data structures involved in a macro processor?
57. What does the macro definition table contain?
58. What is the purpose of the ARGTAB?
59. How are the ambiguities in parameters avoided in macro processor?
60. What is meant by conditional macro expansion?
61. Define positional parameters.
62. What should be done for recursive macro expansion if the chosen programming language does not support recursion?
63. What is a pre-Processor?
64. What is a line-by-line macro processor?
65. Give any two examples of macro definitions in ANSI C.