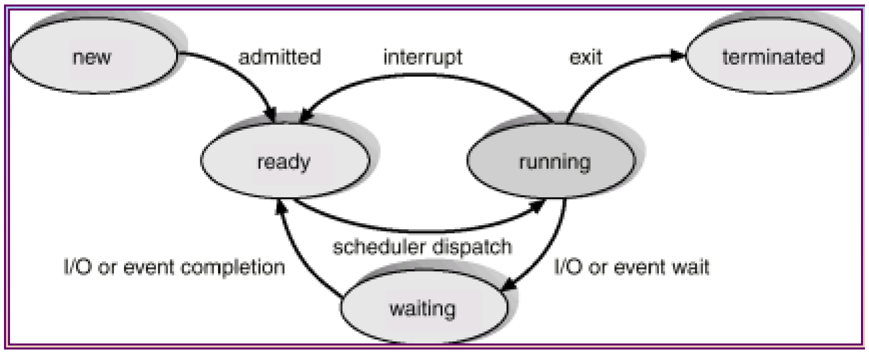
Q2a. What are the various actions an operating system performs when a new process is created? Explain four fundamental states for a process using a state transition diagram.

Ans 2a.

As a process executes, it changes state

- new: The process is being created.
- running: Instructions are being executed.
- waiting: The process is waiting for some event to occur.
- ready: The process is waiting to be assigned to a process.
- terminated: The process has finished execution.



Q3a. Not every unsafe state leads to a deadlock. Give an example to show that the processes in an unsafe state complete their execution without entering a deadlock state.

Ans 3a. Yes, this is true that every unsafe state does not lead to a deadlock. Consider the following snapshot of the system.

	Allocation				**Max**				**Available**			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

By allowing P1 to finish and return its resources to the Available list, available contains (1 5 3 2) we can then service the request of P3.When P3 returns its resources, Available contains (1 11 6 4), thus allowing all of the rest of the processes to finish in any order.

b. Explain the functionality of each of the following and give their differences

(i) Short-term scheduler
(ii) Medium-term scheduler
(iii) Long-term scheduler components

Ans 3b. Short-term (CPU scheduler) -selects from jobs in memory, those jobs which are ready to execute, and allocates the CPU to them. Medium-term -used especially with time-sharing systems as an intermediate scheduling level. A swapping scheme is implemented to remove partially run programs from memory and reinstate them later to continue where they left off. Long-term (job scheduler) determines which jobs are brought into memory for processing.

The primary difference is in the frequency of their execution. The short-term must select a new process quite often. Long-term is used much less often since it handles placing jobs in the system, and may wait a while for a job to finish before it admits another one.

c. What is dispatch latency? How does it affect Real time scheduling? Suggest some solutions to keep dispatch latency low.

Ans 3c. Dispatch latency is the time it takes for the dispatcher to stop one process and start another running. The real time computing is divided into two types:
Hard real-time systems are required to complete a critical task within a guaranteed amount of time.
Soft real-time computing requires that critical processes receive priority over less fortunate ones. It is less restrictive. Implementing soft real-time functionality requires dispatch latency must be small. The smaller the latency, the faster a real time process can start executing once it is runable.
To keep dispatch latency low, we need to allow system calls to be preemptible. This can be done in various ways:

1. Inserting preemption points in long-duration system calls that check whether a high-priority process needs to be run. If so then context switch takes place and when high priority process terminates, the interrupted process continues with the system call.
2. Making entire kernel preemptible as any kernel data being updated are protected from modification by the high-priority process.

Q4a. State Producers-Consumers system with bounded buffer. Write a solution outline for this problem.

Ans 4a. The bounded buffer problem assumes that there is a fixed buffer size. In this case, the consumer must wait if the buffer is empty and the producer must wait if the buffer is full. The buffer may be either provided by the operating system through the use of IPC, or explicitly coded by the application programmer with the use of the shared memory.

A general structure of the scheme is given below. We assume that the pool consists of n buffers, each capable of holding one item. The mutex semaphore provides mutual exclusion for access to the buffer pool and is initialized to the value 1. The empty and full semaphores count the number of empty and full buffers respectively. The semaphore empty is initialized to the value n, the semaphore full is initialized to the value 0. The code for the producer and consumer process is as shown below:

Bounded-Buffer Problem Producer Process

```
do {
    …
        produce an item in nextp
    …
    wait(empty);
    wait(mutex);
      …
    add nextp to buffer
      …
    signal(mutex);
    signal(full);
} while (1);
```

Bounded-Buffer Problem Consumer Process

```
do {
    wait(full)
    wait(mutex);
     …
    remove an item from buffer to nextc
     …
    signal(mutex);
    signal(empty);
     …
    consume the item in nextc
    …
    } while (1);
```

Q5a. Consider a system which has 170 K bytes available for user programs. Let the following programs await memory allocation:

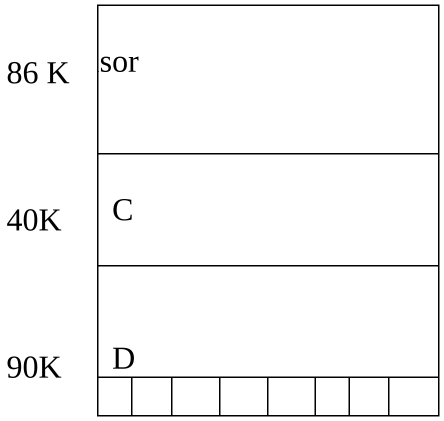
Program name	Size
C	40K
D	90K
E	55K
F	70K

How much total fragmentation would be there when using

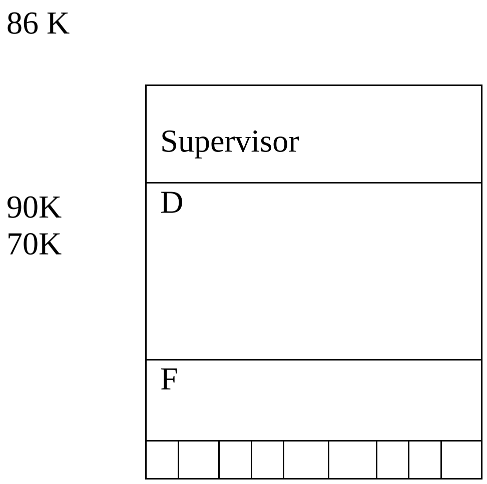
(i) first-fit

(ii) best-fit criterion for memory allocation.

Ans 5a.



(a)

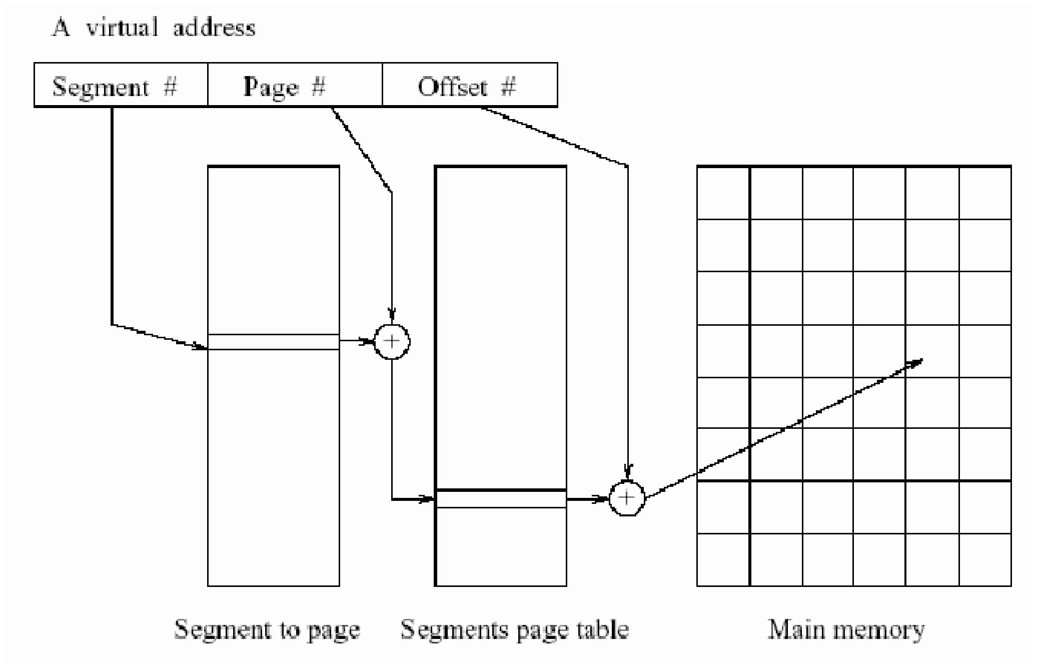


The first fit policy would initiate programs C and D, leading to total fragmentation of 40 K bytes. The best fit policy would initiate programs D and F, leading to total fragmentation of 10 K bytes.

b. Differentiate between Paging and Segmentation. What is need of paged segmentation?

Ans 5b. Differences between Paging and segmentation

Paging can be superimposed on a segment oriented addressing mechanism to obtain efficient utilization of the memory. This is a clever scheme with advantages of both paging as well as segmentation. In such a scheme each segment would have a descriptor with its pages identified. So we have to now use three sets of offsets. First, a segment offset helps to identify the set of pages. Next, within the corresponding page table (for the segment), we need to identify the exact page table. This is done by using the page table part of the virtual address. Once the exact page has been identified, the offset is used to obtain main memory address reference. The final address resolution is exactly same as in paging. The different pieces of virtual address in a segmented paging is as shown below:



c. Explain LRU page replacement algorithm. How many page faults are there if LRU is used with reference string as 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 and with 4 physical pages?

Ans 5c.LRU page replacement is approximate optimal algorithm expands to least recently used. This policy suggests that we re- move a page whose last usage is farthest from current time.

For example—reference string 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 and with 4 physical pages, we have 8 page faults.

Q6a. List the properties which a hashing function should possess to ensure a good search performance. What approaches are adopted to handle collision?

Ans 6a. A hashing function h should possess the following properties to ensure good search performance:

1. The hashing function should not be sensitive to the symbols used in some source program. That is it should perform equally well for different source programs.

2. The hashing function h should execute reasonably fast.

b. Give the Schematic of Interpretation of HLL program and execution of a machine language program by the CPU.

Ans. The CPU uses a program counter (PC) to note the address of next instruction to be executed. This instruction is subjected to the instruction execution cycle consisting of the following steps:

1. Fetch the instruction.
2. Decode the instruction to determine the operation to be performed, and also its operands.
3. Execute the instruction.

At the end of the cycle, the instruction address in PC is updated and the cycle is repeated for the next instruction. Program interpretation can proceed in a similar manner. The PC can indicate which statement of the source program is to be interpreted next. This statement would be subjected to the interpretation cycle, which consists of the following steps:

1. Fetch the statement.
2. Analyse the statement and determine its meaning, viz. the computation to be performed and its operands.
3. Execute the meaning of the statement.

Q7a. What is parsing? Write down the drawback of top down parsing of backtracking.

Ans7a. Parsing is the process of analyzing a text, made of a sequence of tokens, to determine its grammatical structure with respect to a given formal grammer. Parsing is also known as syntactic analysis and parser is used for analyzing a text. The task of the parser is essentially to determine if and how the input can be derived from the start symbol of the grammar.

Following are drawbacks of top down parsing of backtracking:

(i) Semantic actions cannot be performed while making a prediction. The actions must be delayed until the prediction is known to be a part of a successful parse.
(ii) Precise error reporting is not possible. A mismatch merely triggers backtracking. A source string is known to be erroneous only after all predictions have failed.

b. Differentiate between non-relocatable, relocatable and self-relocatable programs.

Ans 7b. A non-relocatable program is the one that cannot be executed in any memory area other than the area starting on its translated origin. For example a hand coded machine language program.

A reloactable program is the one that can be processed to relocate it to a desired area of memory. For example an object module.The difference between a reloactable and a non-reloactable program is the availability of information concerning the address sensitive instructions in it.

A self-relocating program is the one that can perform the relocation of its own address sensitive instructions. A self-relocating program can execute in any area of memory. This is very important in time-sharing operating system where load address of a program is likely to be different for different executions

c. What is macro-expansion? List the key notions concerning macro expansion. Write an algorithm to outline the macro-expansion using macro-expansion counter.

Ans 7c. A macro call leads to macro expansion. During macro expansion, the macro call statement is replaced by a sequence of assembly statements. Two key notions concerning macro expansion are:

1. Expansion time control flow- this determines the order in which model statements are visited during macro expansion.
2. Lexical substitution: Lexical substitution is used to generate an assembly statement from a modal statement.

The flow of control during macro expansion can be implemented using a macro-expansion counter (MEC). The outline of algorithm is as follows:

```
1.      MEC:=statement number of first statement following the prototype statement;
2.      While statement pointed by MEC is not a MEND statement
(a)             If a model statement then
(i)                 expand the statement.
(ii) MEC:=MEC+1;
(b)             Else (i.e. a preprocessor statement)
(i)                 MEC:=new value specified in the statement;
 3.  Exit from macro expansion.
```

Q8 b. List the tasks performed by the analysis and synthesis phases of an assembler.

Ans8b. The tasks list performed by the analysis and synthesis phases of an assembler is:

Analysis phase-

1. Isolate the label, mnemonic opcode and operand fields of a statement.
2. If a label is present, enter the pair (symbol,<LC contents>) in symbol table.

3. Check validity of the mnemonic opcode through a look-up in the Mnemonics table.
4. Update the value contained in LC by considering the opcode and operands of the statement.

Synthesis phase-

1. Obtain the machine code corresponding to the mnemonic from the Mnemonics table.
2. Obtain address of a memory operand from the Symbol.
3. Synthesize a machine instruction or the machine form of a constant as the case may be.

Q9a. Differentiate between:

(i) Pure and impure interpreters

Ans9a. (i) In a pure interpreter, the source program is retained in the source form all through its interpretation. This arrangement incurs substantial analysis overheads while interpreting a statement.
An impure interpreter performs some preliminary processing of the source program to reduce the analysis overheads during interpretation. The preprocessor converts the program to an intermediate representation (IR), which is used during interpretation. This speeds up interpretation as the code component of the IR i.e the IC, can be analyzed more efficiently than the source form of the program.

(ii) Static and Dynamic binding

Ans(ii) Dynamic and static binding: A dynamic binding is a binding performed after the execution of a program has just begun while static binding is a binding performed before the execution of a program begins. Static bindings lead to more efficient execution of a program than dynamic bindings.

b. Explain the following optimizing transformations used in compilers by giving suitable example for each:

(i) Frequency reduction

Ans9b. Frequency Reduction:

Execution time of a program can be reduced by moving code from a high execution frequency region of the program to low execution frequency region(s). For example, the transformation of loop optimization moves loop invariant code out of loop and places it prior to loop entry.

Example:

```
                                          X: 25*a;
for i:=1 to100 do                         for i:= 1 to 100 do
begin                                     begin
  z:=i;                       ⇨              z:= i;
  x:=25*a;
  y:=x + z;                                  y:= x+z;
end;                                      end;
```

Here x:=25*a; is loop invariant. Hence it is computed only once before entering the for loop. y:=x+z; is not loop invariant. Hence it cannot be moved to achieve frequency reduction.

(ii) Strength reduction

Ans9b. Strength Reduction:

The strength reduction optimization replaces the occurrence of a time consuming operation (a 'high strength' operation) by an occurrence of a faster operation(a 'low strength' operation), e.g. replacement of a multiplication by an addition.

Example:
In Figure, the 'high operation' operator '*' in i*5 inside the loop is replaced by a low strength operator '+' in i temp+5.

```
                                        i temp :=5;
for i:=1 to 10 do                       for i:= 1 to 10 do
begin                                    begin
   ---                                     ---
   k:= i*5;                                 k:= i temp;
   ---                        ⇨            ---
                                           i temp:= i temp+5;
 end;                                    end;
```

<u>TEXTBOOK</u>

Systems Programming and Operating Systems, D. M. Dhamdhere, Tata McGraw-Hill, Second Revised Edition, 2005