IGNOU MCA MCS-41 Solved Assignment 2011

Course Code : MCS-041
Course Title : Operating System
Assignment Number : MCA(4)/041/Assign/2010-11
Maximum Marks : 100
Weightage : 25%
Last Date of Submission : 15th October, 2010 (for July, 2010 session)
15th April, 2011 (for January, 2011 session)

This assignment has six questions, which carries 80 marks. Answer all questions. Rest 20 marks are for viva voce. You may use illustrations and diagrams to enhance the explanations. Please go through the guidelines regarding assignments given in the Programme Guide for the format of presentation.

Question 1:

(a) Assume you have the following jobs to execute with one processor

Process	Burst Time	Arrival Time
P1	75	0
P2	40	10
P3	25	10
P4	20	80
P5	45	85

Support a system uses RR scheduling with a quantum of 15.

a) Create a Gantt chart illustrating the execution of these processes.
b) What is turn around time for process P3.
c) What is the average wait time for the processes?

(9 Marks)

(b) Consider a swapping system in which memory consists of the following hole size in memory order: 10K, 4K, 20K, 18K, 7K, 9K, 12K and 15K.which hole is taken for successive segment requests of

i) 12K ii) 10K iii) 9K

for first fit, best fit and worst fit.

Answer: (a)
Gantt chart

P1	P2	P3	P4	P5	P1	P2	P3	P4	P5	P1	P2	P5	P1	P1

0 15 30 45 60 75 90 105 120 125 140 155 165 180 195 210

Process	Processing Time	Turnaround Time= Process completed-process Submitted	Waiting Time=Turnaround Time-Processing Time
P1	75	210-0=210	210-75=135
P2	40	165-0=165	165-40=125

P3	25	120-0=120	120-25=95
P4	20	125-0=125	125-20=105
P5	45	180-0=180	180-45=135

(b) The around tine for the process p3 is =120

(c) Average waist time (135+125+95+105+135)/5 = 595 = 119
5

Question No. (b)

Ans: (i) First Fit – 20K
Best fit – 12k
Warst fit – 20k

(ii) First fit – 10 k
Best fit – 10 k
Worst fit – 20k

(iii) First fit – 9 k
Best fit – 9 k
Worst fit – 20k

Question 2:

Write a semaphore based solution to Dining Philosopher problem and explain all the assumptions and the algorithm.

Ans.

Solution written in Java

```
import java.util.concurrent.Semaphore;
import java.util.Random;
import java.util.Vector;

public class Philosopher extends Thread
{
    private static final Random rand = new Random();
    private static int event=0;
    private static String binary="";
    private int id;
    private Semaphore sem;
    private static Vector<Object[]> vector = new Vector<Object[]>();

    public Philosopher(int i, Semaphore s)
    {
        id = i;
        sem = s;
        binary = binary + "0";
    }

    private void busy()
    {
        try
        {
            sleep(rand.nextInt(1000));
        } catch (InterruptedException e){}
    }
```

```
    private void thinking()
    {
        String str = "Philosopher " + id + " is thinking";
        vector.add( this.addToObject(System.currentTimeMillis(), event
, str) );
        event++;
        busy();
    }

    private void eating()
    {
        String str ="Philosopher " + id + " is hungry and is trying to
pick up his chopsticks";
        vector.add( this.addToObject(System.currentTimeMillis(), event
, str) );
        event++;
        busy();
        str = "Philosopher " + id + " is eating";
        this.oneToBinary(id);
        vector.add( this.addToObject(System.currentTimeMillis(), event
, str) );
        event++;
        busy();
        str = "Philosopher " + id + " finished eating, and puts away
his chopsticks";
        this.zeroToBinary(id);
        vector.add( this.addToObject(System.currentTimeMillis(), event
, str) );
        event++;
    }

    private Object[] addToObject(long t, int i,String s ){
        Object[] o = new Object[4];
        o[3] = s;
        o[2] = i;
        o[1] = binary;
        o[0] = t;
        return o;
    }

    private void oneToBinary(int i){
        binary = binary.substring(0,i) + "1" + binary.substring(i+1);
    }

    private void zeroToBinary(int i){
        binary = binary.substring(0,i) + "0" + binary.substring(i+1);
    }

    @Override
    public void run()
    {
        for (int i = 0; i < 10; ++i)
        {
            thinking();
            try
            {
                sem.acquire();
```

```
            } catch (InterruptedException e){}
            eating();
            sem.release();
        }
    }

    public static void main(String[] args)
    {
        final int N = 5;
        Semaphore sem = new Semaphore(N, true);
        Philosopher[] philosopher = new Philosopher[N];

        // Start the philosophers
        for (int i = 0; i < N; i++) {
          philosopher[i] = new Philosopher(i, sem);
          philosopher[i].start();
        }
        // Wait for them to finish
        for (int i = 0; i < N; i++) {
          try {
            philosopher[i].join();
          } catch(InterruptedException ex) {
            return;
          }
        }

        for (int i = 0; i < vector.size(); i++) {
            Object[] o = vector.get(i);
            System.out.printf("%d %d %s %s\n", o[0], o[2], o[1], o[3]);
        }
    }
}
```

Question 3.
a). Discuss how fragmentations manifest itself in each of the following types of virtual storage system.
i) Segmentation
ii). Paging
iii)Combined segmentation and paging
Ans.
i)

memory segmentation is one of the most common ways to achieve memory protection; another common one is paging. In a computer system using segmentation, an instruction operand that refers to a memory location includes a value that identifies a segment and an offset within that segment. A segment has a set of permissions, and a length, associated with it. If the currently running process is allowed by the permissions to make the type of reference to memory that it is attempting to make, and the offset within the segment is within the range specified by the length of the segment, the reference is permitted; otherwise, a hardware exception is raised.

Moreover, as well as its set of permissions and length, a segment also has associated with it information indicating where the segment is located in memory. It may also have a flag indicating whether the segment is present in main memory or not; if the segment is not present in main memory, an exception is raised, and the operating system will read the segment into memory from secondary storage. The information indicating where the segment is located in memory might be the address of the first location in the segment, or might be the address of a page table for the segment, if the segmentation is implemented with paging. In the first case, if a reference to a location within a segment is made, the offset within the segment will be

added to address of the first location in the segment to give the address in memory of the referred-to item; in the second case, the offset of the segment is translated to a memory address using the page table.

In most systems in which a segment doesn't have a page table associated with it, the address of the first location in the segment is an address in main memory; in those systems, no paging is done. In the Intel 80386 and later, that address can either be an address in main memory, if paging is not enabled, or an address in a paged "linear" address space, if paging is enabled.

ii).

paging is one of the memory-management schemes by which a computer can store and retrieve data from secondary storage for use in main memory. In the paging memory-management scheme, the operating system retrieves data from secondary storage in same-size blocks called *pages*. The main advantage of paging is that it allows the physical address space of a process to be noncontiguous. Before paging, systems had to fit whole programs into storage contiguously, which caused various storage and fragmentation problems.[1]

Paging is an important part of virtual memory implementation in most contemporary general-purpose operating systems, allowing them to use disk storage for data that does not fit into physical Random-access memory (RAM). Paging is usually implemented as architecture-specific code built into the kernel of the operating system.

iii)
In a combined paging/segmentation system a user address space is broken up into a number of segments at the discretion of the **programmer**
. Each segment is in turn broken up into a number of fixed-size pages which are equal in length to a main memory frame. If a segment is less than a page in length, the segment occupies just one page. From the programmer`s point of view, a logical address still consists of a segment number and a segment offset. From the system`s point of view, the segment offset is viewed as a page number and page offset for a page within the specified segment.

Question 3. b). Compare direct file with indexed sequential file organization.
Ans.
Indexed sequential file organization. In indexed sequential file organization, the records are stored in sequence according to a primary key and an index is created to allow random access of the file. This type of organization also allows the file to be accessed sequentially. Indexed sequential is the most commonly used type of file organization. l Direct file organization. In direct file organization, the records are stored and retrieved using a relative record number, which gives the position of the record in the file. This type of organization also allows the file to be accessed sequentially

Question 4. What are the necessary and sufficient conditions for a deadlock to occur? Explain Banker’s algorithm for deadlock detection with and example. Also suggest 2 mechanisms of recovery from a deadlock.
Ans.

There are four necessary and sufficient conditions for a Coffman deadlock to occur, known as the *Coffman conditions* from their first description in a 1971 article by E. G. Coffman.

1. Mutual exclusion condition: a resource that cannot be used by more than one process at a time
2. Hold and wait condition: processes already holding resources may request new resources
3. No preemption condition: No resource can be forcibly removed from a process holding it, resources can be released only by the explicit action of the process
4. Circular wait condition: two or more processes form a circular chain where each process waits for a resource that the next process in the chain holds

Banker's Algo

The Banker's algorithm is run by the operating system whenever a process requests resources.[2] The algorithm prevents deadlock by denying or postponing the request if it determines that accepting the request could put the system in an unsafe state (one where deadlock could occur). When a new process enters a system, it must declare the maximum number of instances of each resource type that may not exceed the total number of resources in the system. Also, when a process gets all its requested resources it must return them in a finite amount of time.

```
function bankers_algorithm(set of processes P, currently available resources A) {
    while (P not empty) {
        boolean found = false
        for each (process p in P) {
            C_p = current_resource_allocation_for_process(p)
            M_p = maximum_resource_requirement_for_process(p)
            if (M_p − C_p ≤ A) {
                // p can obtain all it needs.
                // Assume it does so, terminates, and releases what it already has.
                A = A + C_p
                remove_element_from_set(p, P)
                found = true
            }
        }
        if (not found) {
            return UNSAFE
        }
    }
    return SAFE
}
```

Recover from deadlock

Once it is determined that a deadlock exitx, the system needs to recover from the deadlock. For this, one or more transactions are rolled back to break the deadlock. While performing the roll back operation, the following issues need to be addressed :

a. Selection of a victim : in the situation of a deadlock, you first need to determine the transaction (or transaction) that should be rolled back to break the deadlock. Such a transaction is called the victim transaction. The transaction that will lead to minimum loss, in terms of cost, should be chosen for rollback. The following factors determine the cost of a rollback :
How much execution has the transaction completed and for how much more time the transaction will execute to complete its task?♣
How many data items were used by the transaction?♣
How many more data items does the transaction need to complete?♣
How many transaction will be included in the rollback?♣

b. Rollback : after determining the transaction to be rolled back, you need to determine how far the transaction is to be rolled back. The easlest answer to this problem is to do a total rollback, which means that the transaction will be aborted and restarted. However, it is better to roll back the transaction only till the point where the deadlock can be broken. This method requires the DBMS to maintain information about all current transaction.

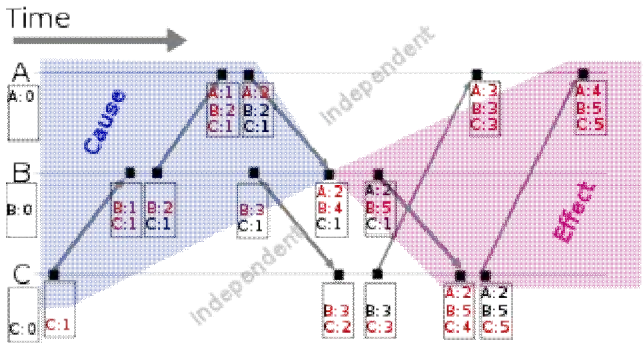
c. Starvation : when the selection of a victim is based on cost factors, it might happen ihat the same transaction is selected as a victim every time a deadlock occurs. Due to this, the transaction might not be able to complete its task. Such a situation is called starvation. To avoid starvation, you need to ensure that a transaction is picked as a victim for only a fixed number of times. To ensure this, you can select a victim based on the number of rollbacks along with the cost factor.

Question 5. a)
Explain take-grant model for operation system security with an example. Also explain the mechanisms of security in WIN 2000 operating system.
Ans.
Modeling and analysis of information system vulnerabilities helps us to predict possible attacks to networks using the network configuration and vulnerabilities information. As a fact, exploiting most of vulnerabilities result in access rights alteration. In this paper, we propose a new vulnerability analysis method based on the Take-Grant protection model. We extend the initial Take-Grant model to address the notion of vulnerabilities and introduce the vulnerabilities rewriting rules to specify how the protection state of the system can be changed by exploiting vulnerabilities. Our analysis is based on a bounded polynomial algorithm, which generates the closure of the Take-Grant graph regarding vulnerabilities. The closure helps to verify whether any subject can obtain an access right over an object. The application of our results have been examined in a case study which reveals how an attacker can gain an unauthorized access right by exploiting chain of vulnerabilities.

Question 5. b) Explain Bell and La-Padula Model for security and protection. Why is security a critical issue in a distributed OS environment?
Ans.
The **Bell-LaPadula Model** (abbreviated **BLP**) is a state machine model used for enforcing access control in government and military applications.[1] It was developed by David Elliott Bell and Leonard J. LaPadula, subsequent to strong guidance from Roger R. Schell to formalize the U.S. Department of Defense (DoD) multilevel security (MLS) policy.[2][3][4] The model is a formal state transition model of computer security policy that describes a set of access control rules which use security labels on objects and clearances for subjects. Security labels range from the most sensitive (e.g."Top Secret"), down to the least sensitive (e.g., "Unclassified" or "Public").
The Bell-LaPadula model focuses on data confidentiality and controlled access to classified information, in contrast to the Biba Integrity Model which describes rules for the protection of data integrity. In this formal model, the entities in an information system are divided into subjects and objects. The notion of a "secure state" is defined, and it is proven that each state transition preserves security by moving from secure state to secure state, thereby inductively proving that the system satisfies the security objectives of the model. The Bell-LaPadula model is built on the concept of a state machine with a set of allowable states in a computer network system. The transition from one state to another state is defined by transition functions.

Question 6). Write and explain an algorithm used for ordering of events in a distributed environment. Implements\ the algorithm with an example and explain?
Ans.

Vector clocks is an algorithm for generating a partial ordering of events in a distributed system and detecting causality violations. Just as in Lamport timestamps, interprocess messages contain the state of the sending process's logical clock. A vector clock of a system of *N* processes is an array/vector of *N* logical clocks, one clock per process; a local "smallest possible values" copy of the global clock-array is kept in each process, with the following rules for clock updates:




Example of a system of vector clocks

- Initially all clocks are zero.
- Each time a process experiences an internal event, it increments its own logical clock in the vector by one.
- Each time a process prepares to send a message, it increments its own logical clock in the vector by one and then sends its entire vector along with the message being sent.
- Each time a process receives a message, it increments its own logical clock in the vector by one and updates each element in its vector by taking the maximum of the value in its own vector clock and the value in the vector in the received message (for every element).

The **vector clocks** algorithm was independently developed by Colin Fidge and Friedemann Mattern in 1988