Comp 346 – Fall 2020

Tutorial 2

Shared Data manipulation and Synchronization

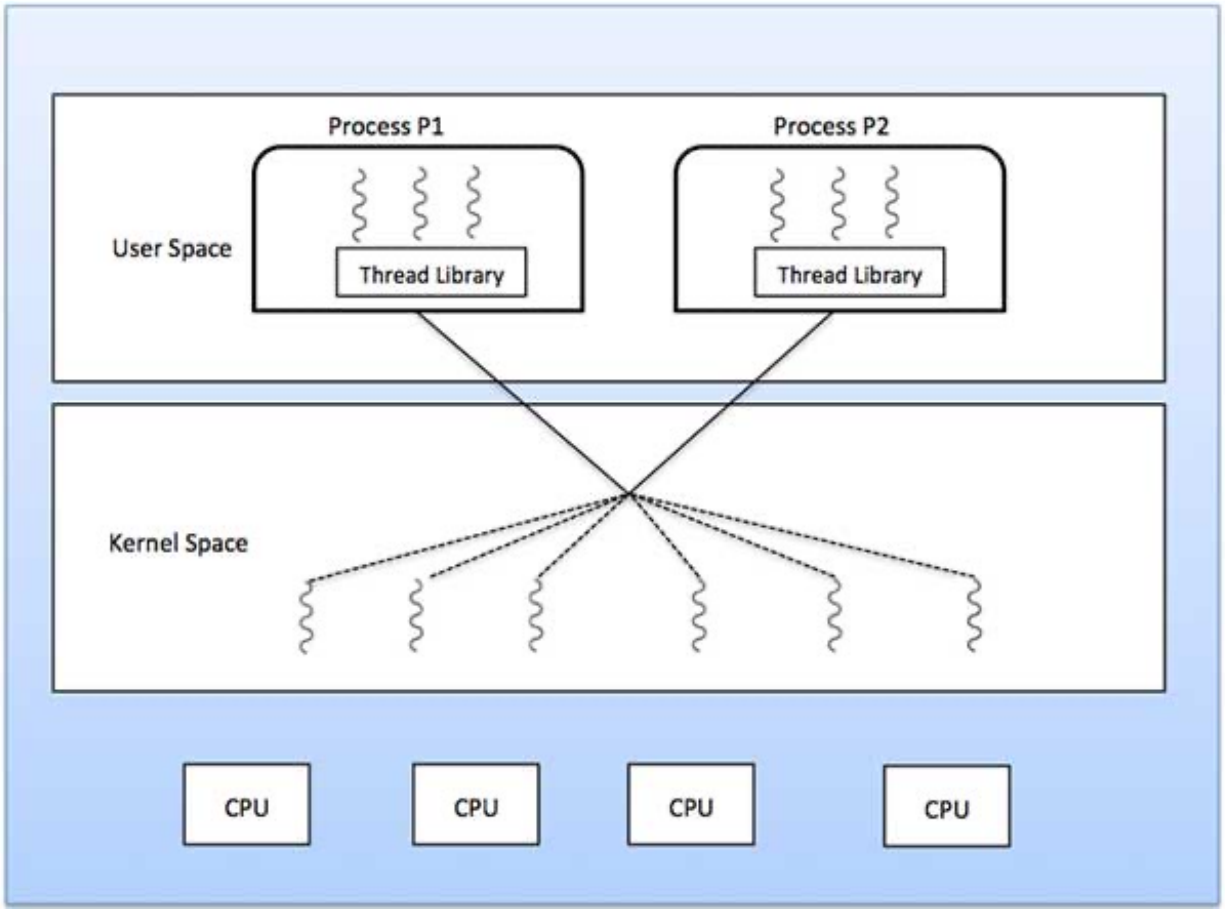
REVIEW - MULTITHREADING MODELS

- Some operating system provide a combined user level thread and Kernel level thread facility.
- Solaris is a good example of this combined approach.
- In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process.
- Multithreading models are three types
  - Many to many relationship.
  - Many to one relationship.
  - One to one relationship.

Many to Many Model

- The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.
- The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads.
- In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine.
- This model provides the best accuracy on concurrency and when a thread performs a blocking system call, the kernel can schedule another thread for execution.

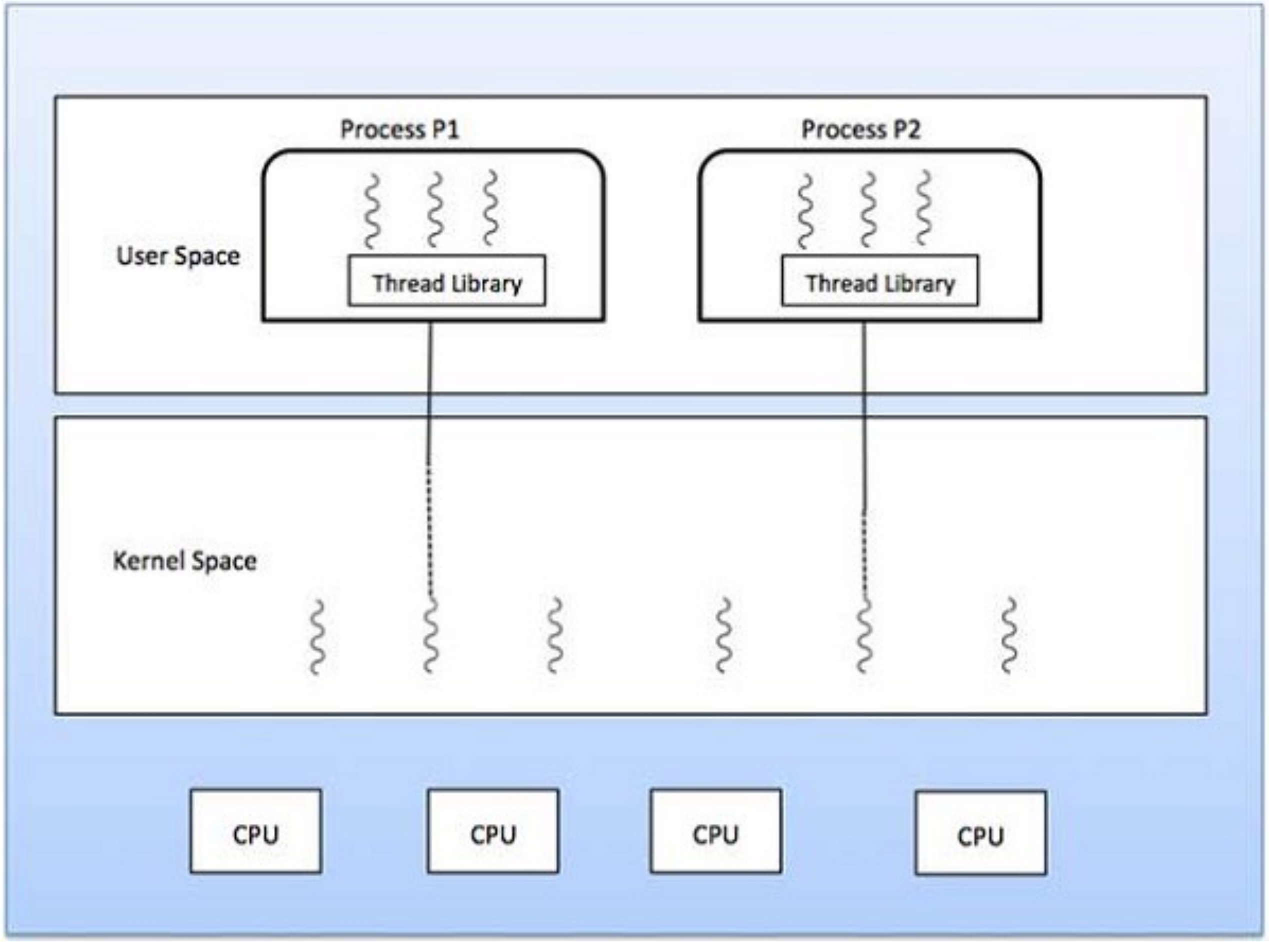
Many to Many Model



Many to One Model

- Many-to-one model maps many user level threads to one Kernel-level thread.
- Thread management is done in user space by the thread library.
- When thread makes a blocking system call, the entire process will be blocked.
- Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.
- If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.

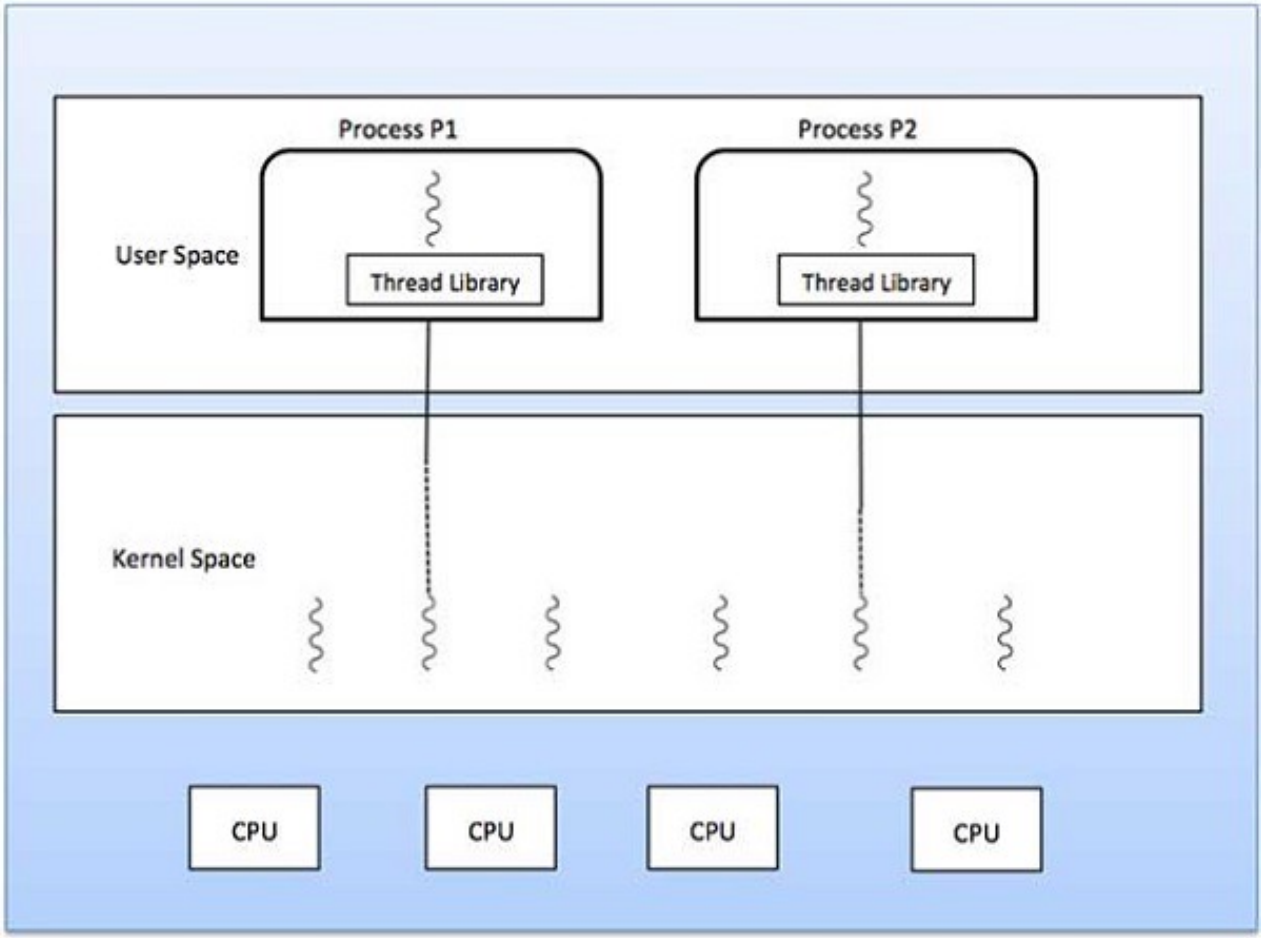
Many to One Model



One to One Model

- There is one-to-one relationship of user-level thread to the kernel-level thread.
- This model provides more concurrency than the many-to-one model.
- It also allows another thread to run when a thread makes a blocking system call.
- It supports multiple threads to execute in parallel on microprocessors.
- **Disadvantage** of this model is that creating user thread requires the corresponding Kernel thread.   windows NT and windows 2000 use one to one relationship model.

One to One Model



Review - Concepts

- There are three very important **concepts** when doing concurrent programming :
- **Atomicity** : An operation is said atomic when it cannot be interrupted. There is almost no atomic operations in Java, the only we've is the assignation a = 5, but a = b++ is not atomic.
- In some cases, you'll have to make atomic some actions with synchronization.
- **Visibility** : This occurs when a thread must watch the actions of the other threads, for example the termination of the thread. This also implies some kind of synchronization.
- **Order of execution** : When you have normal program, all you lines of code run in the same order every time you launch the application.
- This is not the case when you make concurrent programming.
- You first instruction can followed by an instruction of the thread B or by the first instruction. And that can change every time you launch the application. **The order of execution is not guaranteed !**

Shared data manipulation

- A shared data manipulation shows how things are really interleaved and shows their potential problems. Lets create a shared data array and allow our threads to manipulate it.

SHARED DATA EXAMPLE DESCRIPTION

- First lets create a static integer array, called `a`.
- This single array will be shared among the two threads in our program.
- Each thread will deposit its id on a slot on the array.
- We will have our array size set to 2*NI, and let each thread deposit its id on a slot NI times.
- Ideally, each thread should deposit on an empty slot.

Shared Data Example Implementation

- To implement this, we will also maintain a shared index.
- Initially the shared index will be set to 0, indicating that slot 0 is empty.
- As a thread deposits its id on a slot, it will increment the index

Thread Class

```
public class SharedArrayGame extends Thread{
  private static final int NI = 100000;
  private static int a[] = new int[2*NI];
  private static int index = 0;
  private int tid;
  public SharedArrayGame(int tid){
     this.tid = tid;
  }
  public void run() {
     for (int i=0;I < NI; i++) {
        a[index] = tid;
         index = index + 1;
     }
  }
```

Main Method

```
public static void main(String[] args){
   System.out.println("SharedArrayGame starts");
   SharedArrayGame t1 = new SharedArrayGame(1);
   SharedArrayGame t2 = new SharedArrayGame(2);
   t1.start();
   t2.start();
   try{
      t1.join();
      t2.join();
   } catch(InterruptedException e){}
   System.out.println("SharedArrayGame done");
   for(int i=0;i<2*NI;i++){
      if(a[i]==1) System.out.println("000");
      else if (a[i]==2) System.out.println("-----");
      else System.out.println("EEEEEEEE");
   }
}
```

SHARED DATA MANIPULATION

- If you execute the program, you should see mix of 000 and ------ being printed, which is expected.
- But you would also see some EEEEEEEE being printed, specially at the end.
- We might not have expected this, but this is all logical.
- What happened is a memory inconsistency.
- Lets go over a scenario.

Shared data manipulation

- We have two threads and both are running the same code.
- The problem is they are reading and updating a shared variable. Because we have no protection, reading and updating a shared variable will lead to problem.
- Consider the code of run:

```
    for (int i= 0; i < NI; i++){
        a[index] = tid;
        index = index+1;
    }
```

Shared Data Problem [1]

- Lets say, thread corresponding the object t1 just finished executing the instruction `a[index]=tid`; where index is, say 15.
- But before it can update the value of index, to make it 16, a context switch occurred and thread corresponding the object t2 gets to execute.
- Because t1 could not update the index, t2 will use the same slot as t1, namely slot 15.
- As a result, deposit of t1 will be lost.
- Now, thread t2 will probably execute for a while, say it deposits its tid 10 times.

Shared Data Problem [2]

- Value of index becomes 25. Which means, next slot to be filled is 25.
- A context switch occurs again and t1 gets to execute.
- But t1 did not finish updating index, last time. So as it comes back to the instruction `index=index+1`, index becomes 26.
- Slot corresponding to index 25 never gets filled! and thus we have a printout of EEEEEEEE for that slot.

Shared data manipulation

- Its good that multiple threads gets a share of the processing unit but we may risk correctness of the computation when sharing is involved.
- The above problem only occurs because threads are being kicked out of the running state to ready state time to time to give other threads a chance to run. We will see how we can bring order to the house.

SYNCHRONIZING WITH JAVA

- If the body of the for loop in the run() method, were executed uninterruptedly, then we would not get any inconsistency with our output.
- One easy way to do this is with ***synchronized*** statement.
- Any object in java has an implicit intrinsic lock.
- When a synchronized statement is executed based on an object, its intrinsic lock is acquired by the calling thread. **Another thread cannot execute a synchronized statement based on the same object as long as its intrinsic lock is owned by some other thread.**

JAVA SYNCRONIZATION

```
for (int i= 0; i < NI; i++){
      synchronized(o) {
            a[index] = tid;
            index = index+1;
      }
}
```

- where o is a shared object between thread t1 and t2. We can declare it at the top of our file as:
- `private static Object o = new Object();`

Synchronized method Vs Synchronized block

- Java programming language provides two basic synchronization idioms:
  - synchronized methods
  - synchronized statements or Block
- The synchronized keyword helps to build an atomic action, this action "cannot stop in the middle".

Synchronized Counter Example

```
class Synchronized Counter {
   private int c = 0;

   public synchronized void increment() {
     c++;
   }
    public synchronized void decrement() {
     c--;
   }
    public synchronized int value() {
     return c;
   }
 }
```

Synchronized Statements

- Another way to create synchronized code is with *synchronized statements*. Unlike synchronized methods, synchronized statements must specify the object that provides the intrinsic lock:

```
public void addName(String name) {
    synchronized(this) {
        lastName = name;
        nameCount++;
    }
    nameList.add(name);
}
```

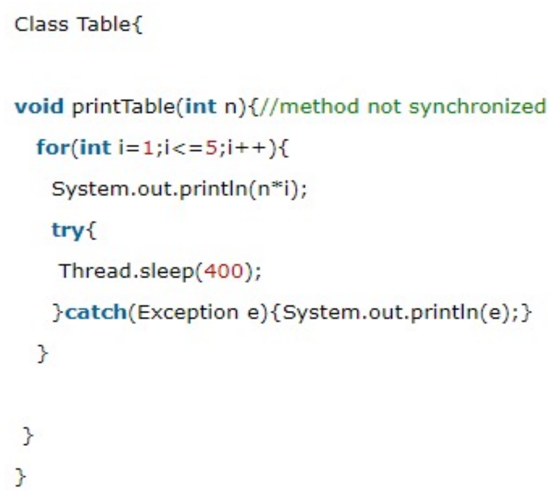
Difference between synchronized block and method in Java

- Two ways to use synchronized keyword in Java and implement mutual exclusion on critical section of code
- Java provides inbuilt synchronized keyword to achieve synchronization in Java
- Main difference between synchronized method and synchronized block is selection of lock on which critical section is locked.
- Though both block and method can be used to provide highest degree of synchronization in Java, use of synchronized block over method is considered as better Java coding practices.

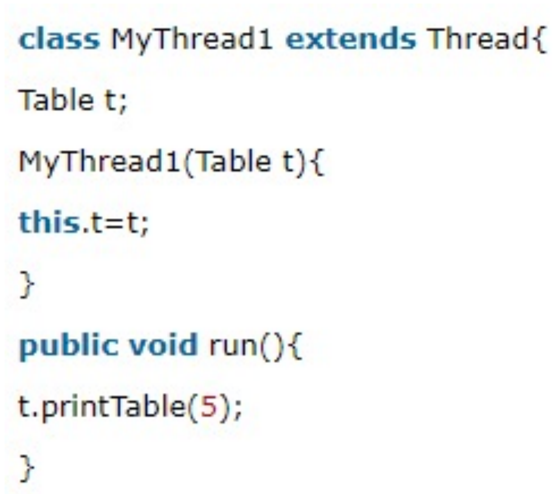
Difference between synchronized block and method in Java

- 1.One significant difference between synchronized method and block is that, **Synchronized block generally reduce scope of lock**. As scope of lock is inversely proportional to performance, its always better to **lock only critical section of code**. This improves performance drastically because locking is only required one or two times.
- 2. **Synchronized block provide granular control over lock**, as you can use arbitrary any lock to provide mutual exclusion to critical section code. On the other hand synchronized method **always lock either on current object represented by this keyword or class level lock**, if its static synchronized method.
- 3. In case of synchronized method, lock is **acquired** by thread when it **enter** method and **released** when it **leaves** method, either normally or by throwing Exception. On the other hand in case of synchronized block, thread acquires lock when they enter synchronized block and release when they leave synchronized block.

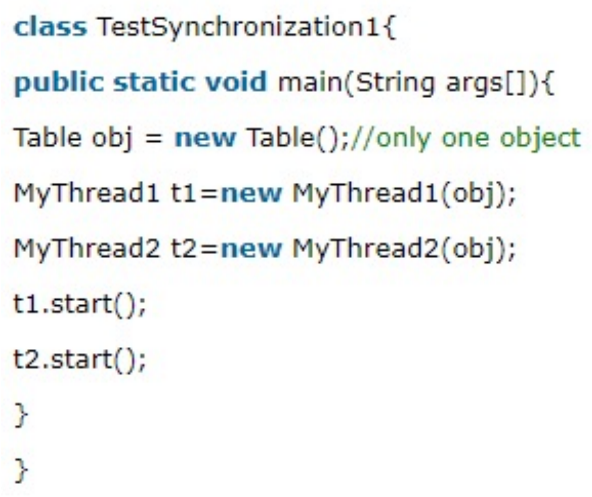
What is the problem?



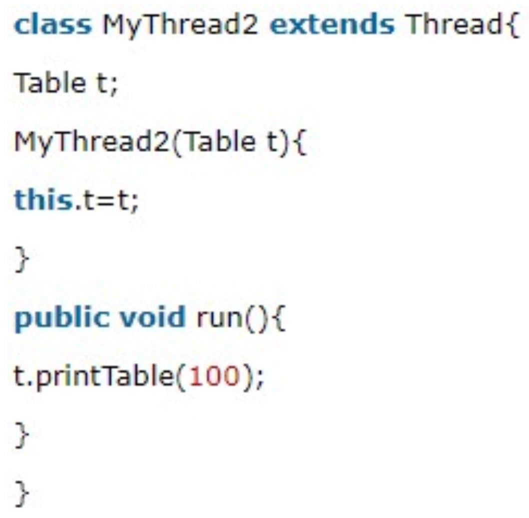
```
Class Table{

void printTable(int n){//method not synchronized
  for(int i=1;i<=5;i++){
    System.out.println(n*i);
    try{
     Thread.sleep(400);
    }catch(Exception e){System.out.println(e);}
  }

 }
}
```



```
class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}
```

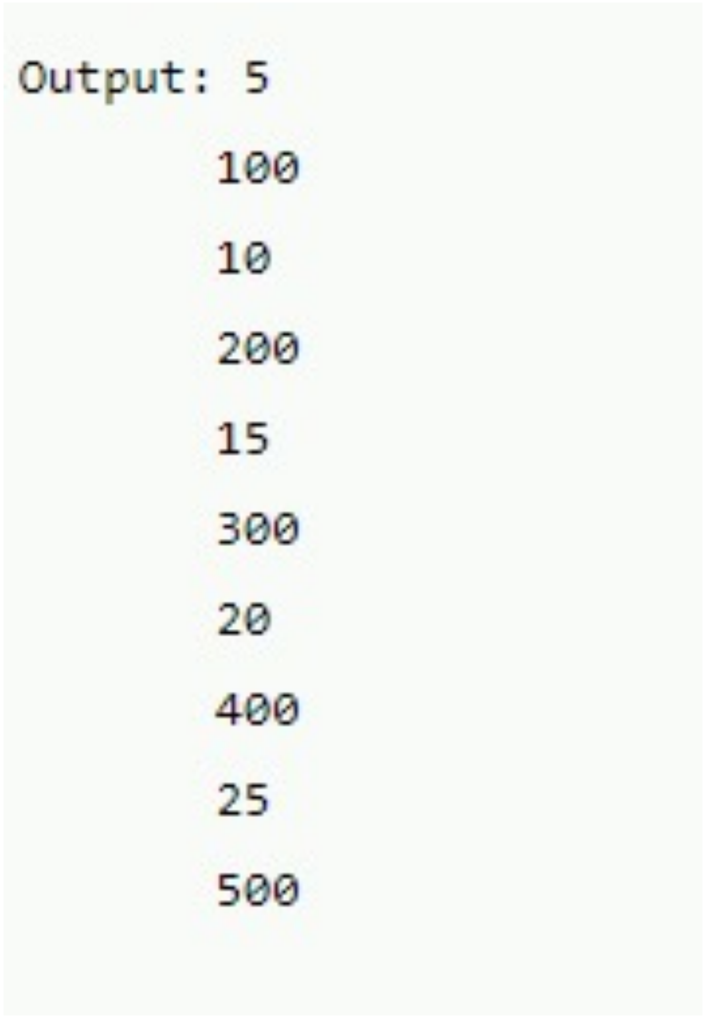


```
class TestSynchronization1{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```



```
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}
```

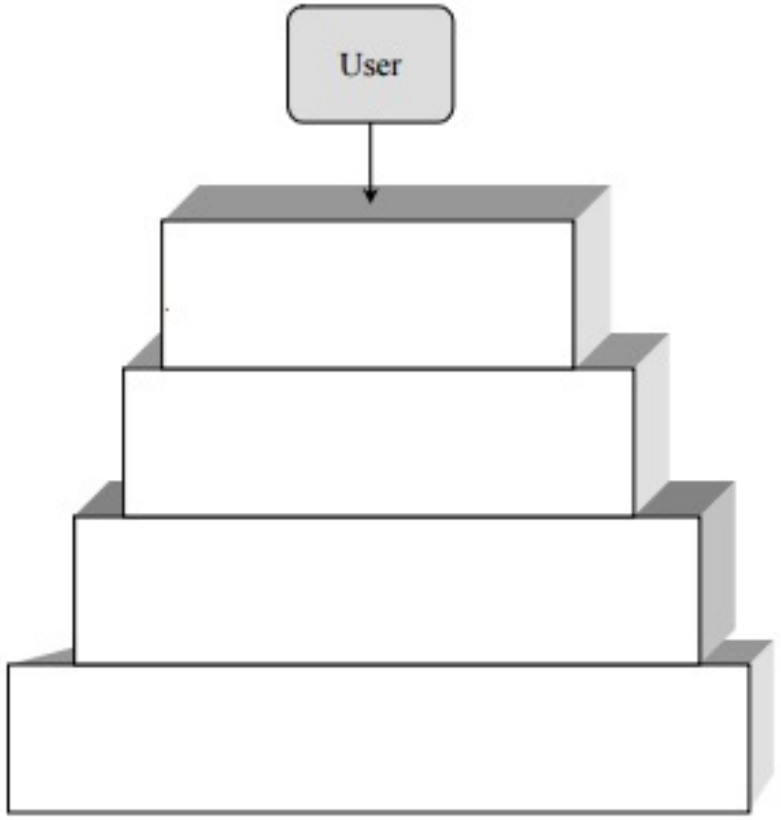
OUTPUT



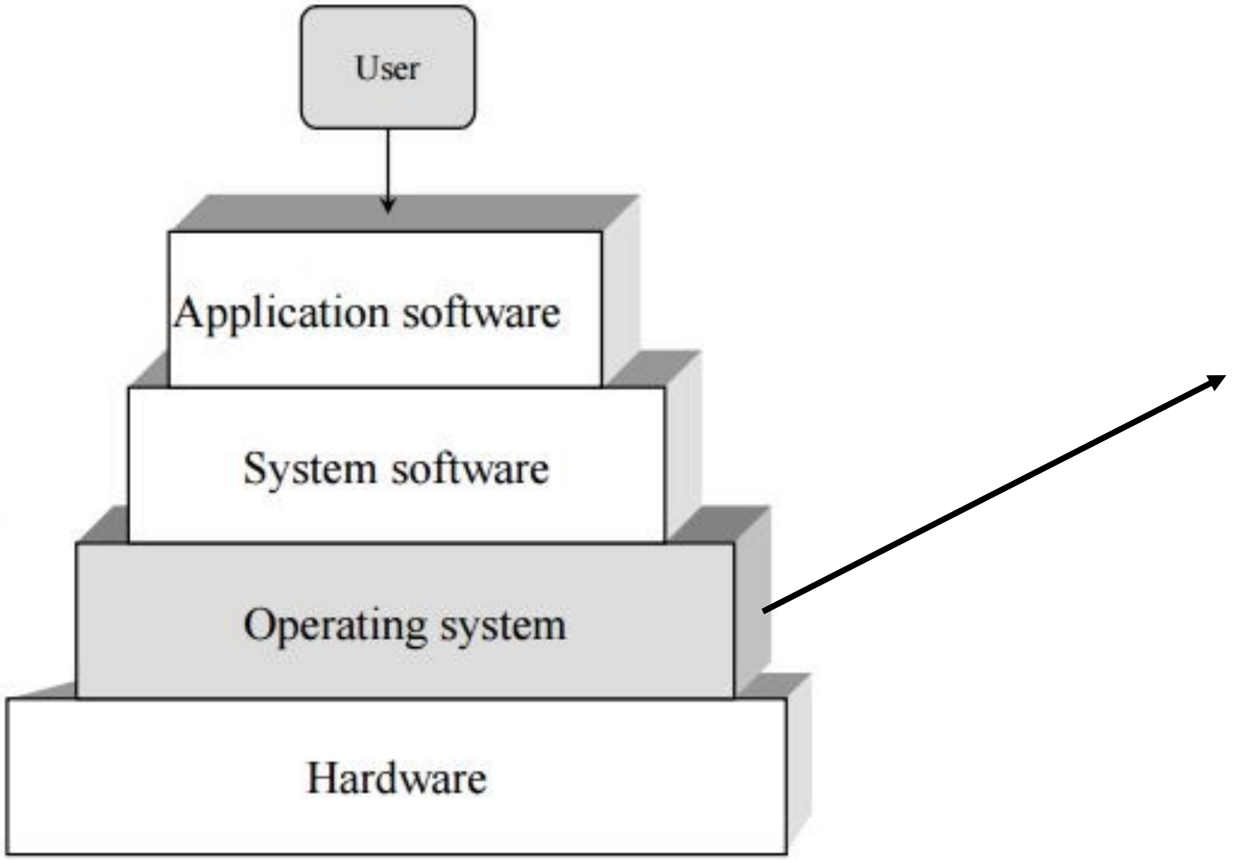
```
Output: 5
        100
        10
        200
        15
        300
        20
        400
        25
        500
```

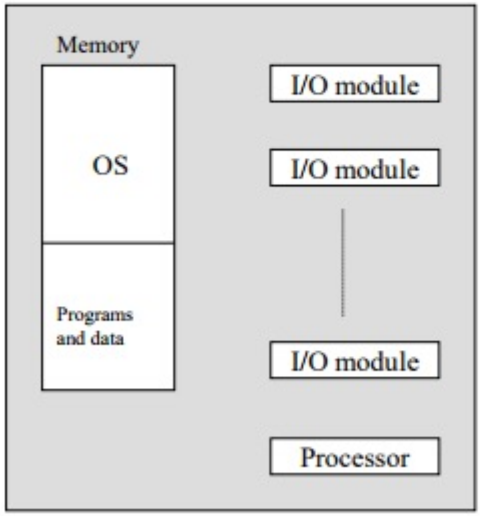
Question 1 – OS Components

- Give the name of each layer of the following schema.
- Provide a short description for each layers.



SOLUTION 1





• A uniprocessor computer system

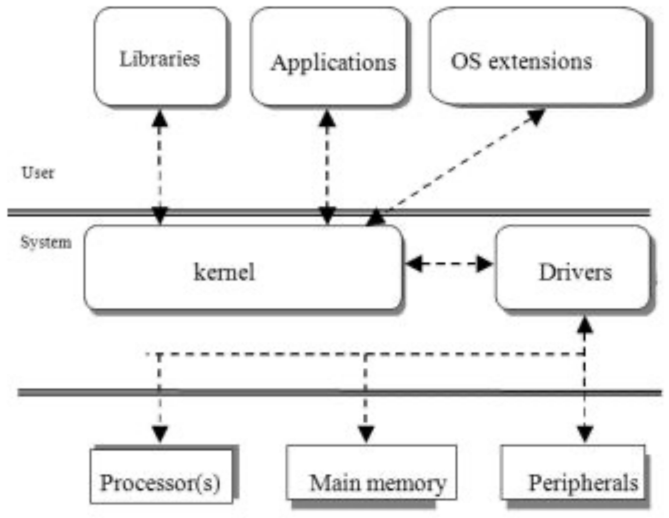
Source: Course slides – Lesson 1

Solution 1

- **Hardware Layer:** The actual electronic hardware of the computer, including and not restricted to: buses, registers, clock, CPU's…
- **Operating System Layer:** Manages and controls resources, as well as program execution. Serves as an interface between the user programs and the hardware.
- **System Software Layer:** Software that is not part of the operating system, but which is needed for the proper functioning of other software. For example, a graphic API (DirectX) needed to run a specific game on your computer.
- **Application software Layer:** Combines the end user programs and applications. Examples: Microsoft Word, Mozilla Firefox, Temple Run…

Question 3 – OS Components

- What are the resources in this schema?
  - Can an application work with the processor directly?
  - What is the programmer's interface to the resources?
  - What would drive the resources?



Source: Course slides – Lesson 1

SOLUTION 3

- Resources: Processor(s), main memory, peripherals
- o An application can NOT work directly with the processor or any resource
- o All applications use the operating system as an interface to the resources
- o The drivers directly control the resources

Question 4 – Context Switch

- Define each step of a context switch
- Specify the overheads that occur during a context switch and that are not considered in the context switch time.

SOLUTION 4

- Step 1: Interruption (through a trap)
- Step 2: OS scheduling is woke-up
- Step 3: Scheduler store registry (in main memory) associated to the stopped process
- Step 4: Scheduler look at the ready-queue for the next process to run
- Step 5: Scheduler restore registry of the next process to run
- Step 6: Scheduler set the IR registry (next instruction to execute) so the next process will restart exactly where he left
- Step 7: Scheduler interrupt himself

Solution 4

- The time that the dispatcher executes to handle the context switch
- The context switch time between the dispatcher and the other two processes

REFERENCES

- [1] http://download.oracle.com/docs/cd/E17409_01/javase/tutorial/essential/concurrency/syncmeth.html
- [2] Questions are prepared by:Alimohammad Firooz,François Gingras, Tamara Finide Dittimi,