**Chapter 10 – Week 13 – Exercises**

Exercises #1 – page 364

1. **What does a thread’s run method do?**

The **run** method is called on a Thread object when the thread begins execution. The code in the **run** method will be executed at that time.

1. **What is time slicing?**

Time slicing is dividing the CPU instructions between multiple processes by limiting the amount of time a process can execute before it must pause and allow another process to have its turn. It allows many processes behave as if they had their own processor and run simultaneously in the perception of the users.

1. **What is a synchronization problem?**

When multiple processes want to access or use the same resource at the same time you may have a synchronization problem. When one thread could leave a resource in an inconsistent state if the process is time-sliced out in the middle of its action and the resource is left in an inconsistent or incorrect state a synchronization problem is encountered. Avoid this using locks and critical code sections.

Stated simpler but less precisely, a synchronization problem is when the results of events could depend on the order in which the separate threads/processes act.

1. **What is the difference between a sleeping thread and a waiting thread?**

A sleeping thread is waiting for a certain amount of time to pass while a waiting thread is waiting for any other condition to become true.

1. **Give two real-world examples of the producer-consumer problem.**

Two real world examples of the producer-consumer problem:

1-Restaurant dining. The producer (the COOK) must produce the meal before the consumer (the PATRON) can eat it

2-Dealing cards. The dealer must produce (DEAL) one card to a player who can then pick up and examine (RECEIVE) the card.

Exercises #2 – page 371

1. **Give two real-world examples of the readers and writers problem.**

Two real world examples of the readers and writers problem are:

1-Live TV streaming. The writer must place some properly formatted video onto the stream. Many readers can then read the stream and show the video to their users.

2-An inventory tracker. Any updates to the inventory must be made synchronously to prevent corrupted data. Any number of reports and/or orders can be placed once the data has been updated.

1. **State two ways in which the readers and writers problem is different from the producer-consumer problem.**

Two ways the problems differ is:

1-The producer-consumer problem as described in the text requires immediate consumption of the data or more data cannot be produced. This is not the case in the readers-writers problem.

2-As described, the producer-consumer problem has a single producer and a single consumer while the readers-writers problem can include multiple readers and/or writers.

1. **Describe how you would make the Student class from Chapter 9 thread-safe for readers and writers.**

I would create a new class utilizing the Decorator pattern to implement a ThreadSafeStudent class. It could make use of SharedCell or some similar object in its implementation.

1. **Define a new class called PCCell. This class provides an abstraction of a shared cell for the producer-consumer problem. The design pattern should be similar for the one presented for the shared cell for readers and writers, but it should use the mechanism specific to the producer-consumer situation.**

*"""*  
*File: sharedcell.py*  
*Resource for shared data synchonization for the readers and writers*  
*problem. Guarantees that a writer finishes writing before readers can*  
*read and other writers can write. Also supports concurrent reading.*  
*"""*  
  
from threading import Condition  
  
class PCCell(object):  
 *"""Synchronizes readers and writers around shared data,*  
 *to support thread-safe reading and writing."""*  
  
def \_\_init\_\_(self, data):  
 *"""Sets up the conditions and count of active readers."""*  
self.data = data  
 self.producing = False  
 self.consuming = False  
 self.ot\_to\_consume = Condition()  
 self.ok\_to\_produce = Condition()  
  
 def begin\_consumption(self):  
 *"""Waits until a writer is not writing or the writers*  
 *condition queue is empty. Then increments the reader*  
 *count and notifies the next waiting reader."""*  
self.ok\_to\_consume.acquire()  
 self.ok\_to\_produce.acquire()  
 while self.producing or len(self.ok\_to\_produce.\_waiters) > 0:  
 self.ok\_consume.wait()  
 self.consuming = True  
 self.ok\_to\_consume.notify()  
  
 def end\_consumption(self):  
 *"""Notifies a waiting writer if there are*  
 *no active readers."""*  
self.consuming = False  
 self.ok\_to\_produce.notify()  
 self.ok\_to\_produce.release()  
 self.ok\_to\_consume.release()  
  
 def begin\_produce(self):  
 *"""Can write only when someone else is not*  
 *writing and there are no readers are ready."""*  
self.ok\_to\_produce.acquire()  
 self.ok\_to\_consume.acquire()  
 while self.producing or self.consuming:  
 self.okToWrite.wait()  
 self.producing = True  
  
 def endWrite(self):  
 *"""Notify the next waiting writer if the readers*  
 *condition queue is empty. Otherwise, notify the*  
 *next waiting reader."""*  
self.producing = False  
 if len(self.ok\_to\_consume.\_waiters) > 0:  
 self.ok\_to\_consume.notify()  
 else:  
 self.ok\_to\_produce.notify()  
 self.ok\_to\_consume.release()  
 self.ok\_to\_produce.release()  
  
 def consume(self, readerFunction):  
 *"""Observe the data in the shared cell."""*  
self.beginRead()  
 # Enter reader's critical section  
 result = readerFunction(self.data)  
 # Exit reader's critical section  
 self.endRead()  
 return result  
  
 def produce(self, writerFunction):  
 *"""Modify the data in the shared cell."""*  
self.beginWrite()  
 # Enter writer's critical section  
 result = writerFunction(self.data)  
 # Exit writer's critical section  
 self.endWrite()  
 return result

Exercises #3 – page 380

1. **Explain the role that ports and IP address play in a client/server program.**

The IP Address and port allow a client/server connection to be made. The IP Address specifies the machine to connect to and the port specifies which conversation # to connect to. Together they are similar to an e-mail address or home address in the specify the endpoint of a message.

1. **What is a local host, and how is it used to develop networked applications?**

The local host is the current machine. During development and testing the client and server are often run on the same machine (the “local host”) to remove much of the network environmental complication during development.

1. **Why is it a good idea for a server to create threads to handle clients’ requests?**

It is a good idea for a server to handle each message in a separate thread because it allows a port listener to continue to listen for new connections. If the main thread received and handled a message, all other clients would have to wait for the listener thread to return and handle theirs. Eventually things can back up and connections are lost and customers are sad.

1. **Describe how a menu-driven command processor of the type developed for an ATM application in Chapter 9 could be run on a network.**

In a network, the ATM machine menu would run on a client machine at your local branch or grocery store while the database of information would run on a server machine in a data warehouse somewhere in the cloud keeping track of account balances.

1. **The ATM application discussed in Chapter 9 has a single user. Will there be a synchronization problem if we deploy that application with threads for multiple users? Justify your answer.**

If the ATM app from Chapter 9 is deployed widely in a client-server setup, a synchronization problem becomes possible. The possibility that multiple connections (threads) will want to write an update to an account simultaneously exists. When this happens, a synchronization problem is very possible.

1. **The servers discussed in this section all contain infinite loops. Thus, the applications running them cannot do anything else while the server is waiting for a client’s request, and they cannot even gracefully be shut down. Suggest a way to restructure these applications so that the applications can do other things, including performing a graceful shutdown.**

One way to restructure the simple client-server applications from this chapter to allow them to do “other” things is to create a message that includes an action (or message type) and other information as needed. Each message type could be handled differently. A graceful shutdown can be obtained by locking out all future reads and writes and waiting for the existing activity to complete and then shutting down the process.