Jacob Pledger 001136676

Assignment 1

1. Access Transparency:

   All resources must be accessed as though they were local. This is important to make life easy for programmers and reduces complications in implementations of higher level services.

   Concurrency Transparency:

   Several processes must be able to run concurrently and smoothly. This is crucial since multiple users will be logged in to the system at once, which in itself will require multiple processes, as well as running the applications the users will want to run, and operating system tasks.

   Scaling Transparency:

   The system must be able to grow without having to change any of the current implementation for compatibility. For example, the addition of a new processor bank of perhaps a different architecture should not require a rewrite of any code in the system. A translation component may need to be added to the software, but the existing software should remain untouched. This is crucial for a distributed system, since it is meant to grow and improve over time, and having to restructure the system for every new component is obviously not a good idea.

   Mobility Transparency:

   Similar to location transparency, resources must be able to be relocated without causing problems. With so many users working on such a large system, the system must be able to to gracefully optimize file locations and memory allocation to improve efficiency.

   Performance Transparency

   The system must be able to balance its workload appropriately, so that all users have a similar experience and are able to use the system effectively. This may require file or memory optimization, balancing or rebalancing processor loads among all processors in the system.

   Location Transparency:

   Users should be unaware of where files are located, both in terms of which disk or logical partition, but also the actual location of the disk itself. Many users cannot keep track of their files on one hard disk, let alone many spread across multiple machines. It is more convenient for the user for this to be transparent. It would be a nightmare for inputting file paths to a program if this were not.

   Replication Transparency:

   Multiple copies of files and resources must be created as backup in a separate storage location in case the system containing the original goes down. This is particularly important for system files, since the sudden loss of such a file could cause the entire system to crash. This could also speed up file access times, depending on the implementation of the system, since files could be copied to a location closer to the terminal that accesses it, or it could be “cached” on a terminal disk.

Jacob Pledger 001136676

Assignment 1

Failure Transparency:

With so many computers networked together and so many simultaneous users, the likelihood of one component of the system going down becomes increased. Therefore, the system must be able to handle the loss of these components gracefully, or else the system would be risky to use. Also, if a distributed system can't handle the loss of one computer, then all the users on the system are affected, as opposed to current systems where the loss of one computer only affects a single user. This is particularly problematic for environments likely to use these systems, such as a company.

2. Scalability is important in distributed systems because as the system grows, the more computing power it gets. Depending on the implementation, as more users are added to the system, the system will either require more resources to support them and require additional components, or will have the added power of the new user's terminal machines. Either way, the system must grow to support its users. The internet works similarly, in that, as new users set up connections and sites require more bandwidth, new infrastructure must be added or the current model must be streamlined.

3. a. The following procedure is called by either the main process in single thread execution, or by the thread that is attempting to receive the message. Multithreading should not be any different, since the process must block anyway.

```
message sim_blockReceive() //returns a message
{
 fork();
 if(in new process)
 {
   message = nbreceive(source_mailbox, &buffer);
   //nbreceive returns a message
 }
 else //in parent process
 wait(); //waits for child process to complete
 }
```

b. You should be able to simulate non-blocking send with a blocking send if you provide immediate artificial feedback. Simply spawn a separate process to generate dummy responses to prevent processes from blocking. The process may block briefly while receiving the dummy reply, but not as long as it would if it were waiting for a real reply. It cannot be simulated in a single process since the process will block after calling blocking send, unless the dummy response could be sent beforehand and reception was delayed until after send was called. The use of multithreading would not particularly help, since if the process blocks, so will the threads.

Jacob Pledger   001136676

Assignment 1

```
  int main()
  {
   fork();
   if (in new process) {
      while (true) {
         if (there is an outgoing message) {
               dummy_reply(sender, ACK);
         //sends an acknowledgement of transmission/reception immediately
         //to the sender
         }
   }
  }
  //run process as usual
  blocking_send(destination, message);
  //now whenever this is called, the auto-reply process sends an ack to
  //keep the process from blocking
  //continue processing
}
```

4. a. If multiple tasks had the receive rights on a port it necessitate a priority handler to prevent the tasks from competing with one another for the data. It also prevents the overhead of having to handle removing receive rights on a port if the information going to that port is not meant for all tasks that are allowed to receive on that port. In addition, it requires less overhead when transferring the rights to another task.

   b. The transfer of receive rights is more complicated than transfer of send rights because whatever was sending to the previous owner of the receive right must also be notified of the change so that it does not send information to the new owner in error.