Motivation:

Software is entering more and more aspects of our daily lives today. They bring us convenience by planning our traveling route, bringing us latest news, letting us reach friends and family at our fingertips and helping us create beautiful documents. However, at the same time, software is used in many mission critical situations that can influence life and death. For example, flight control software of commercial airlines, software that controls throttle power and brakes in cars and software that controls medical equipment are all in the mission critical category. We cannot allow malfunctions during the execution of these mission critical softwares. Current methods of developing these software take a very long time and a lot of resources. Furthermore, since most modern software run on an operating system, it is also critical that these mission critical software can have a trusted operating system -- a base that is formally specified and proven to behave within the specification, otherwise, the operating system itself could malfunction and disaster can still happen.

CertiKOS is a research project lead by Prof. Shao and other researchers at Yale University. It is an operating system kernel that is verified to match its specification, which can be used as a trusted computing base. CertiKOS itself is composed of 37 layers of modular code, each layer provides more power with more abstraction over the underlying hardware. It can run on actual Intel and AMD processors, and it is sufficiently powerful and efficient that it can be used as a hypervisor environment to boot other operating systems like Ubuntu.
Project Description:

My project will be focused on the layers that provide the interprocess communication feature. Modern operating systems enforce isolation among user programs to provide a protected environment for each program to execute in. However, user programs may want to communicate with each other to exchange computed information and cooperate with each other. Thus, the operating system need to provide facilities to the user programs that let them communicate with each other with no way to disturb each other's execution environment. This facility is often called interprocess communication. The current interprocess communication in CertiKOS lacks efficiency in that it only allows processes to exchange one 32-bit integer at a time, and it does not allow the receiver to specify a designated origin (which could be of security concern since receivers don’t know the origin of the data won’t know if the data received can be trusted or not). So part of my project would be designing an interprocess communication protocol that provides enough efficiency, power and security.

As a proof of concept, I have already implemented a prototypical synchronous interprocess communication feature that works with the rest of CertiKOS, but it might be completely overthrown for a better design or it might be optimized to reach better efficiency.

Once the design of the new IPC protocol is done, we would formally specify it with the programming language Coq. Coq is a very advanced language that uses Curry-Howard isomorphism to prove mathematical theorems with the help of type theory. In Coq, a proposition is expressed as a type, and a proof of the proposition is a value of that type (this works because of Curry-Howard isomorphism).

---

1 Brief explanation on Curry-Howard Isomorphism
http://en.wikibooks.org/wiki/Haskell/The_Curry%E2%80%93Howard_isomorphism
A formal specification of a program is the set of its possible behaviors over all inputs. Just as the rest of CertiKOS, the specification for my IPC protocol would be divided into many modular and composable layers. Each layer would have its own specification, and they will be separately proven to match the specification (meaning that behavior demonstrated in execution is a subset of what’s described in the specification).

Many modern OS would provide IPC interface through the usage of buffers -- chunks of memory used to store data. User programs would provide read/write buffer for the kernel to read from one buffer and copy data into another. Buffers are mostly represented by pointers -- an abstract index that points to the buffer inside the memory. One of the major challenges of this project would be developing proof and programming techniques that work well with pointers. The difficulty with pointers is that they can often outlive the data they point to, and if a rogue program gets hold of the pointer, they can alter the contents of the pointer. On the other hand, primitive data are concrete information themselves -- whoever has the primitive data knows that it will not be tampered with by other processes. Hence, developing techniques to tackle these complications with pointers would be an important aspect of the project.

**Deliverables:**
- A prototype implemented in C
- Coq specification and proof of the prototype
- A written report on experience gained on this project