Exploring Fast and Secure System Calls in Microkernels

Mina Soltani Siapoush, Jim Alves-Foss
Center for Secure and Dependable Systems

Introduction

System calls (syscalls) are a critical mechanisms for modern Operating Systems (OS), including not only microkernels such as seL4, QNX, and Fuchsia where system functionalities are deployed in the user-level workflow but also monolithic kernels like Android where apps often communicate with most of the user-level services. However, existing syscall designs still suffer from long latency and less protection. To address the problem, we are exploring mechanisms for a new approach that is fast and secure.

Research Goals

Security

- Secure isolation.
- Prevention of race condition vulnerabilities.
- Prevention of other common vulnerabilities.

Efficiency

- Support long messages along the call chain.
- Fine-grained message granularity.
- Limited copying times (fast syscall).

Flexibility

- Independent of specific kernel architectures.
- Minimal hardware modifications.

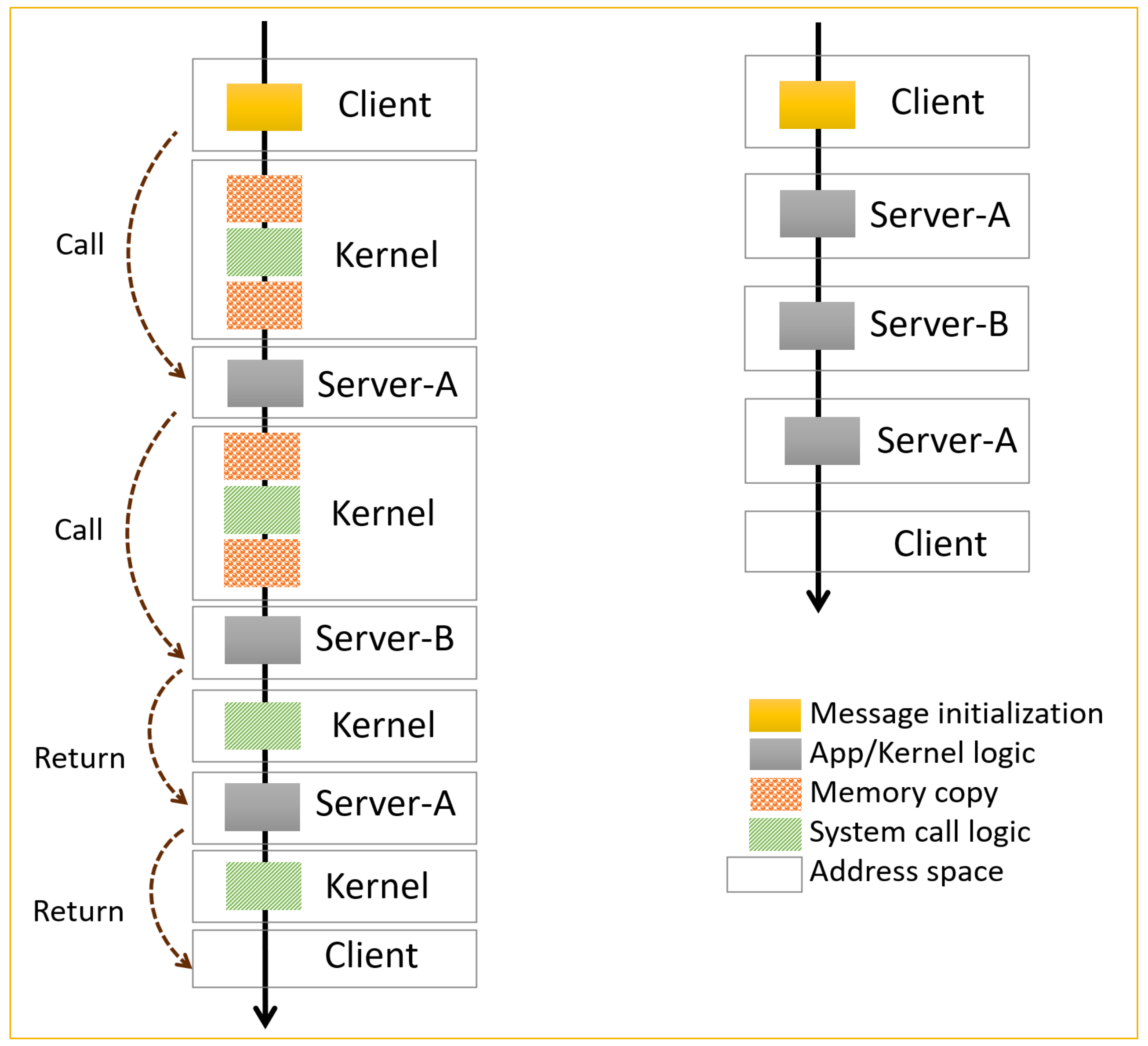

Fig.1. (a) Traditional system call, (b) Our design with zero copy message passing.

Design

We propose a separation of the checking logic into a control plane and a data plane, in which the former is done by software and the latter by hardware.

- **Control Plane**
  The OS kernel is responsible for controlling tasks, ensuring that the mapped region will never be overlapped by any mapping of the page table, and managing hardware extension. The OS kernel is responsible for ensuring that the mapped regions, implemented in the data plane, will never overlap, allowing for complete isolation.
- **Data Plane:**
  We use a new address-space mapping mechanism named relay segment which is a memory region with its virtual address ranges and physical address ranges. The address range information is recorded in Segment Register (Segment-reg), and an extension in MMU is responsible for translating the virtual addresses into physical addresses using Segment-reg. Segment-reg can be passed from a caller to a callee through syscall execution.

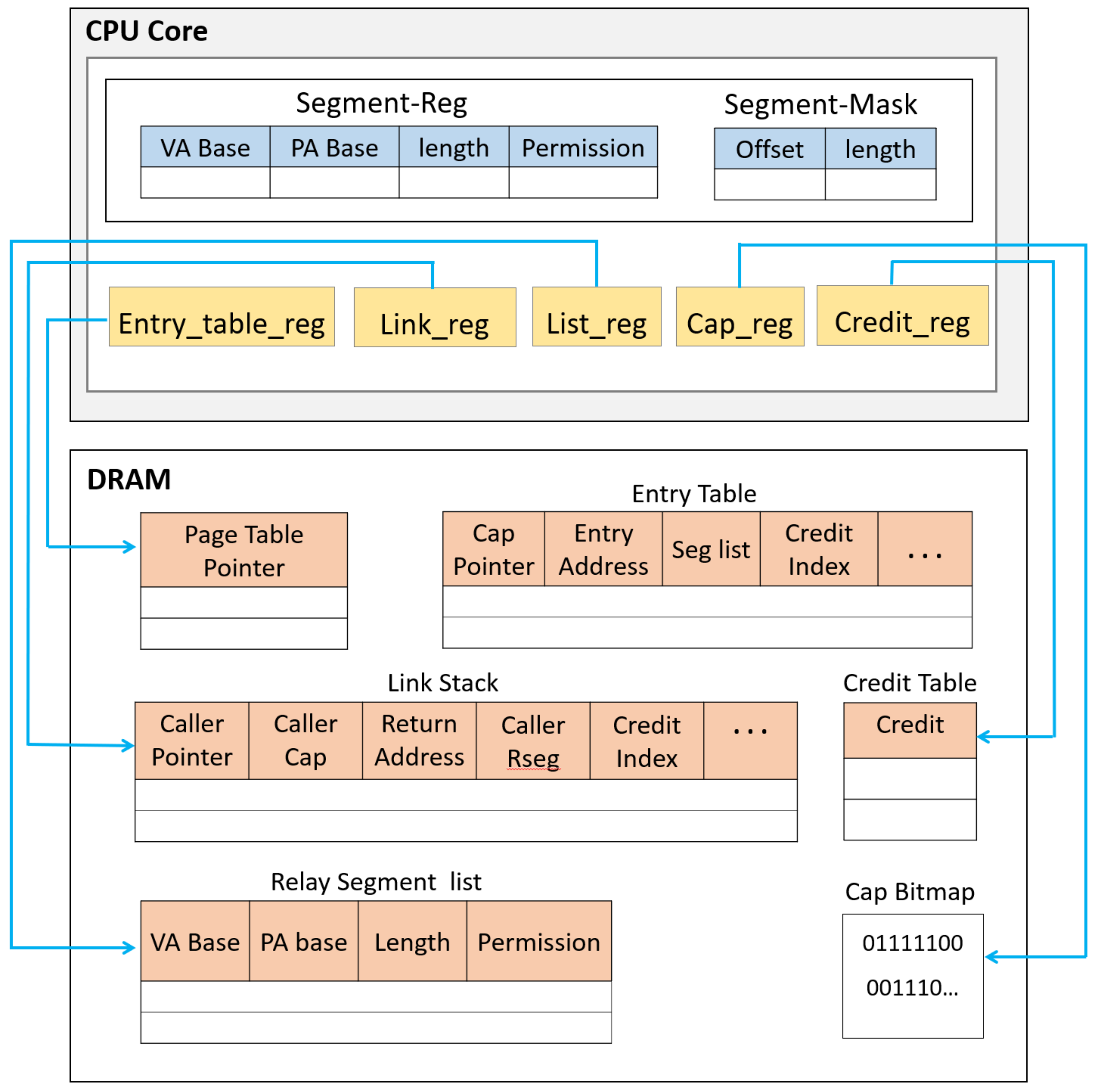

Fig.2. Syscall engine.

Implementation

Software implementation

Software components include kernel support to manage entry, call-cap, relay-segment, and exception handling. Syscall provides register_entry and unregister_entry which allocate and free a relay segment, grant_service and revoke_service grant and revoke the syscall capabilities to other threads.

Hardware implementation

- The syscall engine checks the caller's capability by reading the bit at #reg in the cap bitmap.
- Processor loads the target entry from entry-table (a global table indicated by entry-table-reg and checks the valid bit of the entry.
- Processor prepares a linkage record to record the caller's information and pushes it to the link stack.
- Processor updates states according to the fetched entry, including page table pointer, program counters, and others.

Evaluation

The OS implementation is small as it only affects system call-related modules in kernel. It will not affect other modules such as file system, memory and address space management, etc.

Project is still in design stage.

University of Idaho
www.uidaho.edu/csds
csds@uidaho.edu
(208) 885-4114