CS6210 Project 2 Report

Group Members: Bryan Hickerson, Daniel Hunnicutt

Implementation Summary

Service:

The service takes in requests from a client with parameters (x, p) to a function which does (2^x)modp. The service should then reply with the return value via a response. The data structure which manages the requests/responses is a ring buffer in shared memory.

1) Ring Buffer

The ring buffer is stored in shared memory. The basic idea is that the ring buffer is a fixed length data structure which has linked start and end nodes. A request is inserted into the ring buffer at the end position. Since the ring buffer is looped, the end node will eventually reach old data and overwrite it, but this is an intended behavior of a ring buffer. The ring buffer then chews requests in a simple round robin algorithm.

2) Request/Response Handling

```
struct request {
  int x;
  int p;
  pthread_cond_t response_ready;
};
```

A request is triggered by a conditional mutex variable in shared memory. The client executes a make_request() call which is part of our services library available to the client. Make_request() will write the service parameters, trigger the service that a new request has arrived via conditional variable, and return the request ID to the client.

Once the service has processed the requests, the response is generated and stored back in the ring buffer. The service then notifies the client that a response has been filled, and the client retrieves the response using another library call available to the clients.

Client:

The sample client is relatively simple. The idea is that the client makes many threads which generate service requests and synchronously receive responses.

1) Threads

The threads are pthreads created by using the pthread.h library. We create 128 threads a stagger them by 10useconds so that the requests are staggered by 10us.

2) Synchronous Calls

We make the thread requests synchronous by blocking on the thread using `pthread_mutex_lock` to block all other threads from executing while one thread has a lock. The thread releases the lock when the request is received.

3) Request/Response

The request and response handling is done via helper functions in our server library that the client has access to. These include make_request(), read_response() to aid the client and not giving them direct access to any data structures or shared memory that could cause problems in the service.

Results: