This project consists of 4 parts: implementing the 3 system calls tslsmval, getsmval, and setsmval, and also using these system calls in the dining philosophers problem.

System calls

1. Tslsmval

Since the values are to be either 0 or 1, we use the ds value as the lock.

First we get the message that contains the id of the integer to check from the input message.

(if the id is not between 0 and 15 inclusivereturn EINVAL)

Then we get the value of that integer in data store.

Final we check the value: using ds\_retreive\_u32

* if the value is 1, we don’t get the lock and return -1.
* If the value is 0, we get the lock, overwrite the value with 1 using ds\_publish\_u32 Then return OK
* If the value is not 0 or 1 we return EPERM

1. Getsmval

First get the message that contains the id of the integer to check from the input message.

(if the id is not between 0 and 15 inclusive return EINVAL)

Get the value of the integer using ds\_retreive\_u32

In the reply message put the value.

Return OK if successful

1. Setsmval

First get the id and the value to store in the specified ID from the input message.

(if the id is not between 0 and 15 inclusivereturn EINVAL)

Next overwrite the value of the given id in ds using ds\_publish\_u32

Return OK if successful

Dining Philosophers

Parent process(systest.c)

* Initializes integers 0-15 to 0 with setsmval (above)
* menu to test every function as well as run a dining philosophers simulation.
* When the dining philosophers simulation is ran:
  + Resets all integers 0-15 to 0
  + Forks 5 child processes indexed 1 through 5
  + Children go into a separate executable (diner) with their index as an argument
* Parent process waits for all children to complete.
* Back to menu

Child process (diner.c)

Functions

* *int getChopStick(int)*

calls tslsmval returns 0 if lock is retrieved, else -1

* *int unlockChopStick(int)*

releases the lock on specified int, using smval

* *int getWriteToFile()*

uses integer 0 as a writelock

calls tslsmval, returns 0 if lock is retrieved, else -1

* *int unlockWriteToFile()*

releases the lock on 0, using setsmval

* *int writeToFile(char\*)*Appends to the file dinersEating the input character string
* *void writingToFile(char\*)*

tries to get the writelock until it gets it (sleeping every attempt)

calls write to file on input character string

* *int grabFirstChopStick(int)*

continually tries to grab the specified int chopstick until it gets the lock

reports weather or not it got the lock with a timestamp

* *int grabSecondChopStick(int)*

attempts to get chopstick (specified int +1), if it fails it will release chopstick on the specified int (avoids deadlock)

reports when it gets the chopsticks or releases them with timestamp

* main program
  + each child process will first report that they are hungry with a timestamp
  + next they will continually try to grab first chopstick then grab second chopstick until successful, sleeping between each attempt.
  + Once they got both of their chopsticks, report eating with time stamp and sleep.
  + Release both chopsticks and exit reporting that the chopsticks are returned with timestamp