I. Introduction
   - A. Brief Intro- System call trace recording and replaying
   - B. Project Motivation
     1. Accurately record and replay workloads using Linux system calls
     2. Uses:
        - a) Benchmarking operating system features/alterations
        - b) Analyzing program security and other properties

II. Workflow
   - A. Strace -> library -> Dataseries file -> replayer
     1. Minimize changes to strace
     2. No changes to DS library
     3. Glue code (C to C++) in helper “strace2ds” library
   - B. High level overview (diagram).

III. Recording
   - A. Why our approach?
     1. Problems with old approach (csv)
   - B. Approach:
     1. Minimize changes to strace code
     2. Every change wrapped in #ifdef ENABLE_DATASERIES
        - a) Enabled with ./configure --enable-dataseries
     3. Run-time option “-X <ds-filename>” needed to turn functionality at run time.
     4. When writing to DS file, normal strace output (e.g., pretty-printing) is off
     5. Strace2ds library also uses autoconf
   - C. Where/how we get information in strace
     1. Struct tcb
        - a) Most relevant information changed for each system call
     2. Common fields/v_args
        - a) Void * arrays
        - b) Common fields:
          - (1) Time called, time returned, retval, errno, pid
        - c) V_args
          - (1) Virtual arguments: to copy mem from ptrace’d process to strace’s addr space
          - (2) Used to pass pathnames, read/write buffers, stat structures, etc. for system calls that require them
     3. Our utility functions in util.c
        - a) ds_get_*
        - b) Get pathnames/different types of buffers for different system calls
        - c) Call strace’s umoven function, which copies data from the address space of process being traced to that of strace

    4. Switch block in syscall.c
        a) tcp->s_ent->sen
            (1) Strace's own identifying number for system calls
            (2) I.e. SEN_open
        b) Mostly in trace_syscall_exiting()
            (1) Called after actual system call is executed
            (2) Fields in struct tcb contain accurate values for that system call at that point
        c) Non-terminating system calls in trace_syscall_entering()
            (1) _exit(), execve()
            (2) These calls either never reach trace_syscall_exiting() or the values passed as arguments are inaccessible from trace_syscall_exiting()

D. What happens in the library
    1. Wrapper code (strace2ds.h, strace2ds.cpp)
    2. Fields table
    3. XML files (created from generate-xml.sh <tablefile>)
    4. DataSeriesOutputModule
        a) One DSOM object per trace
            (1) When initialized: creates configuration table with extent names (syscall names) and their relevant fields
        b) Walk through what happens for each system call
            (1) ds_write_record(...)
            (2) Creates map of field names to field values (string to void *)
            (3) Common fields stored in map (if present)
            (4) Check if syscall name matches a supported call
            (5) Make[syscall name]ArgsMap function
                (a) Stores system call's specific fields/arguments in map
            (6) Iterate through field names and write to DataSeries file
        c) Any specific system calls we want to point out?

E. How to run (as an option of strace)
    1. STRACE2DS=~/strace2ds ./strace -X foo.ds <executable>

IV. Replaying

A. Approach
    1. Base SystemCallTraceReplayModule class
    2. Individual system call module classes
    3. Priority Queue

B. Workflow

1. Initialize a module for each supported system call
2. Replay in order of unique_id number
3. processRow(): defined in each derived class
    a) Gets argument values from the ExtentSeries
    b) Actually replays system call
4. completeProcessing():
    a) after_sys_call()
        (1) Compares retval, errno
        (2) Prints system call fields if desired
    b) Adjusts series location
        (1) (not actually a pointer, but ++operator is overloaded in DataSeries so that we can move to the next row similarly)

C. Replaying Options
1. Default
    a) Prints a message when the first of one system call (i.e., close) is played, and when the last of that system call is played.
2. Verbose
    a) Prints each system calls common fields/arguments
3. Verify
    a) Verifies traced and replayed read/write, stat, getdents buffers contain the same data
4. Warn
    a) Prints a warning message if recorded and replayed retval/errno aren't the same
5. Abort
    a) Aborts replayer if recorded and replayed retval/errno aren't the same
6. Write pattern data
    a) Rand() or dev/urandom
    b) Repeated pattern
    c) 0s

D. Replaying Design Decisions
1. File descriptor map
    a) Maps recorded fd to replayed fd
    b) Map certain standard values prior to replay
        (1) STDIN, STDOUT, STDERR, AT_FDCWD
2. Using integer encoding of mode/flag values
    a) Instead of recovering these values from the boolean flag/mode fields specified in the SNIA doc (faster)
3. Rows per call (readv, writev, execve)

a) Most system calls require one record and take up one row in an extent

(1) Default value of rows_per_call is 1

b) Some require more than one

c) Rows_per_call will be set accordingly in processRow

4. System calls that don't make sense to replay (but we record them)

a) Ex: exit, execve

E. How to RUN syscall-replayer

1. ./system-call-replayer <foo.ds>

V. System calls supported

A. list

VI. Programs/utilities we have traced and replayed successfully

A. Cp, mv, rm, ls, ...

VII. Changes to SNIA doc

A. All changes redlined in DOC file

B. Added some system calls

1. Rename, getdents, openat, unlinkat

2. Added/removed/ certain fields

3. Made certain fields (non-)nullable

4. Fixed typos and inconsistencies

VIII. Other issues not yet covered

A. Testing

1. Short test programs for individual system calls

B. Stats we need (e.g., timing replayed syscalls)

C. More syscalls we know we need to capture (mmap group, exec*, clone, ioctl)

IX. To do next:

A. Strace will complain if it traces a syscall we're not capturing

B. Replay bigger and bigger apps

1. Ultimate goal: replay server apps (mysql, apache, etc.)