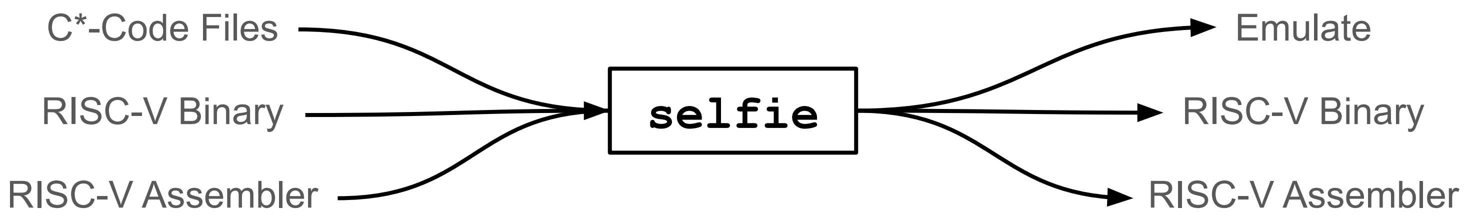
Automated testing of atomic instructions (lr/sc) implementations in selfie

Luis Thiele

Table of Contents

Revisiting `selfie`

The selfie System

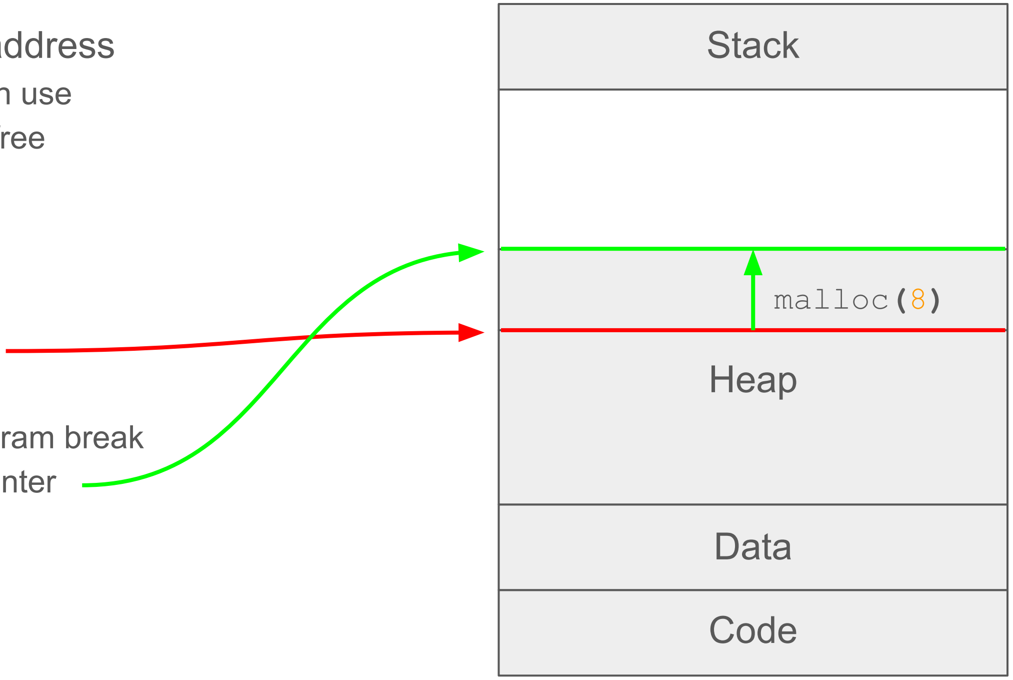


github.com/cksystemsteaching/selfie

Bump-Pointer Allocator

- Variable stores last allocated address
  - Addresses below value are in use
  - Addresses above value are free

- On `malloc(8)` Call:
  - Load value of bump-pointer
  - Add `8` to value
  - Syscall `BRK` to increase program break
  - Store new value of bump-pointer
  - Return new value



Processes vs. Threads

Processes:

- Independent Memory

System Calls:

- `fork()`
- `wait(uint64_t* wstatus)`
- `exit(uint64_t exitcode)`

Threads:

- Shared Memory (except Stack)

System Calls:

- `pthread_create()`
- `pthread_join(uint64_t* wstatus)`
- `pthread_exit(uint64_t exitcode)`

Load Reserved & Store Conditional (1)

- “Extended” Load & Store Instructions
- Load Reserved:
  - Load
  - Reservation on Address
- Store Conditional:
  - Condition: Reservation on Address
    - True: Store
    - False: No Store, Mark Unsuccessful

Load Reserved & Store Conditional (2)

Load:

- **`ld rd,imm(rs1)`**
  - **`rd = memory[rs1 + imm]`**

Store:

- **`sd rs2,imm(rs1)`**
  - **`memory[rs1 + imm] = rs2`**

Load Reserved:

- **`lr.d rd,(rs1)`**
  - **`rd = memory[rs1]`**

Store Conditional:

- **`sc.d rd,rs2,(rs1)`**
  - `SUCCESS:` **`memory[rs1] = rs2`**
  - `SUCCESS:` **`rd = 0`**
  - `FAILURE:` **`rd = 1`**

Load Reserved & Store Conditional (3)

"**LR-SC** Loop":

```
do {
    value = lr(address);
    // edit value here
} while (sc(address, value));
// sc returns 1 if unsuccessful
```

- Shared Memory Affected
  - Threads!
- Make Code Thread-Safe

- Threads:
  - Thread A
  - Thread B
- Execution:
  - `lr`
  - `sc`
  - `lr`
  - `sc`
- Interleaved Execution:
  - `lr`
  - `lr`
  - `sc`
  - `sc`
  - `lr`
  - `sc`

Treiber-Stack Assignment

Treiber-Stack

- Thread-Shared Stack
- Machine Instructions Only
    - No System Calls!
- Heap Memory
    - Uses `malloc`
    - Thus uses system call `BRK`
- Macros:
    - `void      init_stack()`
    - `void      push(uint64_t value)`
    - `uint64_t pop()`

Thread-Safe `malloc` (1)

- "Old" `malloc(8)` Call:
  - **`LD`** value of bump-pointer
  - Add `8` to value
  - Syscall `BRK` to increase program break
  - **`SD`** new value of bump-pointer
  - Return new value
- Issues:
  - Not thread-safe
  - Syscalls force context switches!

- "New" `malloc(8)` Call:
  - **`LR`** value of bump-pointer
  - Add `8` to value
  - Syscall `BRK` to increase program break
  - **`SC`** new value of bump-pointer
  - SUCCESS: Return new value
  - FAILURE: Jump back to **`LR`**
- Short version:
  - **`LD`** / **`LR`**
  - `BRK`
  - **`SD`** / **`SC`**

Thread-Safe `malloc` (2)

Code:

```
1.  pthread_create();
2.  malloc(8);
```

- Old `malloc`:
  - **LD**
  - BRK
  - **LD**
  - BRK
  - **SD**
  - **SD**
- New `malloc`:
  - **LR**
  - BRK
  - **LR**
  - BRK
  - **SC**
  - **LR**
  - BRK
  - **SC**
  - **LR**
  - BRK
  - **SC**
  - **LR**
  - BRK
  - ...

Old Assignment vs. New Assignment

Old Assignment:

- `treiber-stack`
  - Implement **`lr`** & **`sc`**
  - Implement treiber-stack

New Assignment(s):

- `threadsafe-malloc`
  - Implement **`lr`** & **`sc`**
  - Make `malloc` thread-safe
  - No context switches on `malloc`
- `treiber-stack`
  - Implement treiber-stack
  - Make treiber-stack thread-safe

Automated Tests (1)

No-Context-Switch `malloc` Test:

- Easy solution:
  - Make sure Thread A runs first
  - Thread A calls `malloc`
  - Thread A prints eg. “`Hello`”
  - Thread B prints eg. “`World`”
- Success:
  - `malloc` did not force a switch
  - “`Hello World`”
- Failure:
  - “`World Hello`”

```
pid = pthread_create();
if (pid == 0) {
  // child
  child = 1;
  malloc(8);
  write(1, “Hello   ”, 8);
} else {
  // parent
  while (child == 0)
    wait((uint64_t*) 0);
  write(1, “World   ”, 8);
}
```

Automated Tests (2)

`LR` & `SC` Semantics Test:

- Requirements:
  - `LR` coroutine (returns value)
  - `SC` coroutine (returns `1` on `FAILURE`)
- Solution:
  - Interleaved execution
  - 2nd `SC` must mark failure
  - 1st `SC` decides final value

```
uint64_t lr(uint64_t address);
uint64_t sc(uint64_t address,
            uint64_t value);
```

```
address = malloc(8);
*address = 7;
lr(address);
// force switch
lr(address);
if (sc(address, 42))
  return 7;
pthread_wait(status);
// switch
c = sc(address, 7);
pthread_exit(c);
// switch
return *status * *address;
```

Automated Tests (3)

Thread-Safe `malloc` Test:

- Force context switch between **`LR`** & **`SC`**
  - Context switch by timeout
- Idea:
  - Repeat useless loop
  - `malloc` just before switch by timeout
- Solution:
  - Thread A measures endless loop
  - Thread B stops endless loop
  - proceed as explained…
  - Child may force switch by `pthread_wait`
- Success:
  - Different addresses by `malloc`

```
zero = 0;
loop = 1;
while (zero < loop)
  counter = counter + 1;
// switch
loop = 0;
// force switch
i = 2;
while (i < counter);
  i = i + 1;
malloc(8);
// switch
malloc(8);
```

Automated Tests (4)

Thread-Safe Treiber-Stack Test:

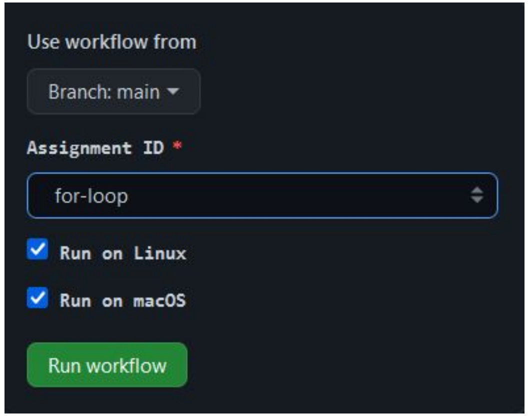
- “
  - “
- “
  - “
  - `push`/`pop` just before switch by timeout
- “
  - “
  - “
  - “
- “
  - `{push’d} = {pop’d}` (overwrites, detached head…)

```
zero = 0;
loop = 1;
while (zero < loop)
  counter = counter + 1;
// switch
loop = 0;
// force switch
i = 2;
while (i < counter);
  i = i + 1;
push(8);
// switch
push(8);
```

Changes to `selfie`

Changes to `selfie`

- Improve github actions
  - No more running out of quota
  - Private repo: Only run on `main` branch and only linux
  - Online dispatcher
- New assignments
  - `logical-and-or-not` (boolean)
  - `lazy-evaluation`
  - `(threadsafe-malloc, treiber-stack)`
- Restructure code
  - Array & Struct assignments a lot easier
  - Grammar also restructured


Use workflow from

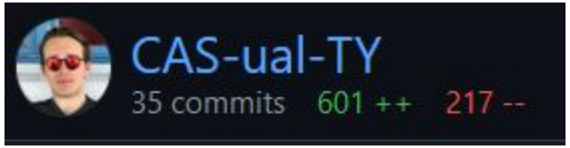
Branch: main

Assignment ID *

for-loop

✓ Run on Linux

✓ Run on macOS


CAS-ual-TY

35 commits 601 ++ 217 --

+706 −596

+266 −6