FROM RESEARCH TO INDUSTRY

cea tech

A Case Study on Formal Verification of the Anaxagoros Paging System with Frama-C

Allan Blanchard Nikolai Kosmatov
Matthieu Lemerre Frédéric Loulergue

list

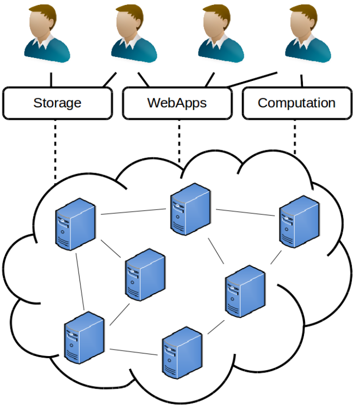
FMICS 2015 — June 22, 2015

LIFO DGA digiteo INSTITUT CARNOT CEA LIST

CONTENTS

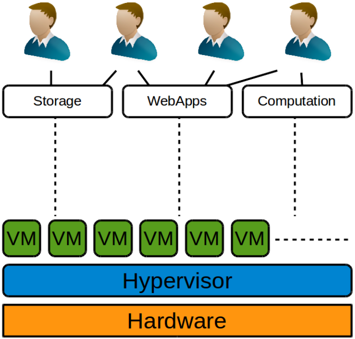
Anaxagoros Microkernel

- Clouds mutualize physical resources between users
  - Safety and security are crucial



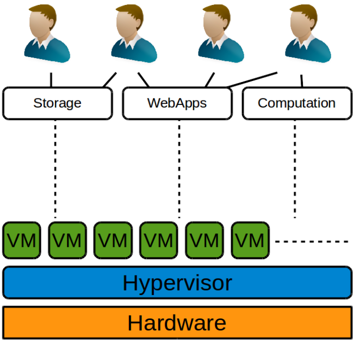
Anaxagoros Microkernel

- Clouds mutualize physical resources between users
  - Safety and security are crucial



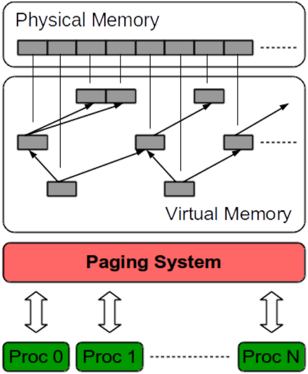
Anaxagoros Microkernel

- Clouds mutualize physical resources between users
  - Safety and security are crucial
- Anaxagoros
  - Secure microkernel hypervisor
  - Developped at CEA LIST by Matthieu Lemerre
  - Designed for resource isolation and protection
- Virtual memory system is a key module to ensure isolation



Virtual Memory Subsystem

- Organizes program address spaces
  - Creates a hierarchy of pages
  - Allows sharing when needed
- Controls accesses and modifications to the pages
  - Only owners can access their pages
  - Types of the pages limit possible actions
- Counts mappings, references, to each page

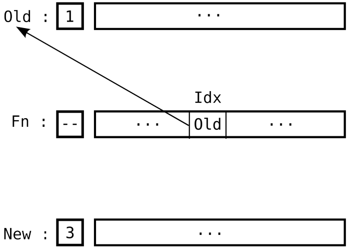


Verified function

```
#define NOF 2048
#define MAX  256

uint mappings[NOF];

int set_entry(uint fn, uint idx, uint new){
  uint c_n = mappings[new];
  if(c_n >= MAX) return 1;
  if(!CAS(&mappings[new], c_n, c_n+1))
    return 1;

  page_t p = get_frame(fn);
  uint old = atomic_exchange(&p[idx], new);

  if(!old) return 0;

  fetch_and_sub(&mappings[old], 1);
  return 0;
}
```

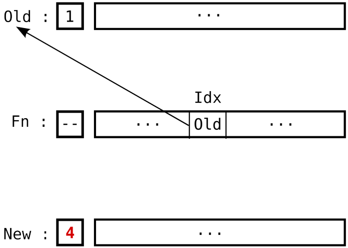
Verified function

```
#define NOF 2048
#define MAX  256

uint mappings[NOF];

int set_entry(uint fn, uint idx, uint new){
  uint c_n = mappings[new];
  if(c_n >= MAX) return 1;
  if(!CAS(&mappings[new], c_n, c_n+1))
    return 1;

  page_t p = get_frame(fn);
  uint old = atomic_exchange(&p[idx], new);

  if(!old) return 0;

  fetch_and_sub(&mappings[old], 1);
  return 0;
}
```



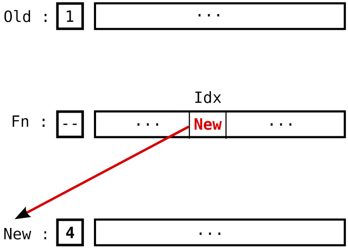
Verified function

```
#define NOF 2048
#define MAX  256

uint mappings[NOF];

int set_entry(uint fn, uint idx, uint new){
  uint c_n = mappings[new];
  if(c_n >= MAX) return 1;
  if(!CAS(&mappings[new], c_n, c_n+1))
    return 1;

  page_t p = get_frame(fn);
  uint old = atomic_exchange(&p[idx], new);

  if(!old) return 0;

  fetch_and_sub(&mappings[old], 1);
  return 0;
}
```



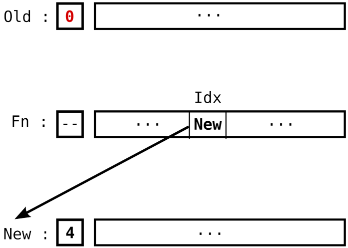
Verified function

```
#define NOF 2048
#define MAX  256

uint mappings[NOF];

int set_entry(uint fn, uint idx, uint new){
  uint c_n = mappings[new];
  if(c_n >= MAX) return 1;
  if(!CAS(&mappings[new], c_n, c_n+1))
    return 1;

  page_t p = get_frame(fn);
  uint old = atomic_exchange(&p[idx], new);

  if(!old) return 0;

  fetch_and_sub(&mappings[old], 1);
  return 0;
}
```



Verified function

```
#define NOF 2048
#define MAX  256

uint mappings[NOF];

int set_entry(uint fn, uint idx, uint new){
  uint c_n = mappings[new];
  if(c_n >= MAX) return 1;
  if(!CAS(&mappings[new], c_n, c_n+1))
    return 1;

  page_t p = get_frame(fn);
  uint old = atomic_exchange(&p[idx], new);

  if(!old) return 0;

  fetch_and_sub(&mappings[old], 1);
  return 0;
}
```



Verified memory invariant

- Maintain the count of mappings on pages
  - Each page descriptor contains a counter that must be equal to the number of mappings to the described page
  - Assuming Occ^v represents the number of occurrences of v in all pagetables, we want to prove :

$$\boxed{\forall e, validpage(e) \Rightarrow Occ^e = mappings[e] \leq MAX}$$

Formal Verification

- Concurrency issues
  - Pages might be modified by different processus simultaneously
  - It creates a gap between the actual number of mappings and the counter

New invariant :

$$\forall e, validpage(e) \Rightarrow Occ^e \leq mappings[e] \leq MAX$$

and more precisely,

$$\forall e, validpage(e) \Rightarrow \exists k.\ k \geq 0 \wedge Occ^e + k = mappings[e] \leq MAX$$

This k is actually the number of threads that have introduced a difference in the counter, difference of at most 1.

Frama-C and WP plugin

- Our verification is conducted with Frama-C :
  - A framework for analysis of C programs
  - Provides a specification language called ACSL
  - We use the WP plugin for deductive proof
- Frama-C and WP do not support concurrency
  - We simulate concurrent executions
  - We prove the invariant on the simulation

Simulation of the concurrency

- We model the execution context, we have for each thread :
  - global arrays representing the value of each local variable
  - a global array representing its position in the execution
- We simulate every atomic step with a function taking in parameter the thread we want to execute
- We create an infinite loop that randomly chooses a thread and makes it perform a step of execution according to its current position

Simulation of the concurrency

Original Code

```
#define NOF 2048
#define MAX  256

uint mappings[NOF];

int set_entry(uint fn, uint idx, uint new){
  uint c_n = mappings[new];
  if(c_n >= MAX) return 1;
  if(!CAS(&mappings[new], c_n, c_n+1))
    return 1;

  page_t p = get_frame(fn);
  uint old = atomic_exchange(&p[idx], new);

  if(!old) return 0;

  fetch_and_sub(&mappings[old], 1);
  return 0;
}
```

Simulating Code

```
#define THD 16

uint pct[THD];

uint fn [THD];
uint idx[THD];
uint new[THD];
uint c_n[THD];
uint old[THD];

//@ghost uint ref[THD]

...
```

Simulation of the concurrency

Original Code

```
#define NOF 2048
#define MAX  256

uint mappings[NOF];

int set_entry(uint fn, uint idx, uint new){
  uint c_n = mappings[new];
  if(c_n >= MAX) return 1;
  if(!CAS(&mappings[new], c_n, c_n+1))
    return 1;

  page_t p = get_frame(fn);
  uint old = atomic_exchange(&p[idx], new);

  if(!old) return 0;

  fetch_and_sub(&mappings[old], 1);
  return 0;
}
```

Simulating Code

```
...

void gen_args(uint th){
  fn[th]  = random_page();
  idx[th] = random_idx();
  new[th] = random_page();
  pct[th] = 1;
}

...
```

Simulation of the concurrency

Original Code

```
#define NOF 2048
#define MAX  256

uint mappings[NOF];

int set_entry(uint fn, uint idx, uint new){
  uint c_n = mappings[new];
  if(c_n >= MAX) return 1;
  if(!CAS(&mappings[new], c_n, c_n+1))
    return 1;

  page_t p = get_frame(fn);
  uint old = atomic_exchange(&p[idx], new);

  if(!old) return 0;

  fetch_and_sub(&mappings[old], 1);
  return 0;
}
```

Simulating Code

```
...

void read_map_new(uint th){
  c_n[th] = mappings[new[th]];
  pct[th] = 2;
}

...
```

Simulation of the concurrency

Original Code

```
#define NOF 2048
#define MAX  256

uint mappings[NOF];

int set_entry(uint fn, uint idx, uint new){
  uint c_n = mappings[new];
  if(c_n >= MAX) return 1;
  if(!CAS(&mappings[new], c_n, c_n+1))
    return 1;

  page_t p = get_frame(fn);
  uint old = atomic_exchange(&p[idx], new);

  if(!old) return 0;

  fetch_and_sub(&mappings[old], 1);
  return 0;
}
```

Simulating Code

```
...

void test_map_new(uint th){
  pct[th] = (c_n[th] < MAX)? 3 : 0;
}

...
```

Simulation of the concurrency

Original Code

```
#define NOF 2048
#define MAX  256

uint mappings[NOF];

int set_entry(uint fn, uint idx, uint new){
  uint c_n = mappings[new];
  if(c_n >= MAX) return 1;
  if(!CAS(&mappings[new], c_n, c_n+1))
    return 1;

  page_t p = get_frame(fn);
  uint old = atomic_exchange(&p[idx], new);

  if(!old) return 0;

  fetch_and_sub(&mappings[old], 1);
  return 0;
}
```

Simulating Code

```
...

void cas_map_new(uint th){
  if(mappings[new[th]] == c_n[th]){
    mappings[new[th]] = c_n[th]+1;
    //@ghost ref[th] = new[th];
    pct[th] = 4;
  }
  else pct[th] = 0;
}

...
```

Simulation of the concurrency

Original Code

```
#define NOF 2048
#define MAX  256

uint mappings[NOF];

int set_entry(uint fn, uint idx, uint new){
  uint c_n = mappings[new];
  if(c_n >= MAX) return 1;
  if(!CAS(&mappings[new], c_n, c_n+1))
    return 1;

  page_t p = get_frame(fn);
  uint old = atomic_exchange(&p[idx], new);

  if(!old) return 0;

  fetch_and_sub(&mappings[old], 1);
  return 0;
}
```

Simulating Code

```
...

void exch_entry(uint th){
  page_t p = get_frame(fn[th]);
  old[th] = p[idx[th]];
  p[idx[th]] = new[th];
  //@ghost ref[th] = old[th];

  pct[th] = 5;
}

...
```

Simulation of the concurrency

Original Code

```
#define NOF 2048
#define MAX  256

uint mappings[NOF];

int set_entry(uint fn, uint idx, uint new){
  uint c_n = mappings[new];
  if(c_n >= MAX) return 1;
  if(!CAS(&mappings[new], c_n, c_n+1))
    return 1;

  page_t p = get_frame(fn);
  uint old = atomic_exchange(&p[idx], new);

  if(!old) return 0;

  fetch_and_sub(&mappings[old], 1);
  return 0;
}
```

Simulating Code

```
...

void test_old(uint th){
  pct[th] = (old[th])? 6 : 0;
}

...
```

Simulation of the concurrency

Original Code

```
#define NOF 2048
#define MAX  256

uint mappings[NOF];

int set_entry(uint fn, uint idx, uint new){
  uint c_n = mappings[new];
  if(c_n >= MAX) return 1;
  if(!CAS(&mappings[new], c_n, c_n+1))
    return 1;

  page_t p = get_frame(fn);
  uint old = atomic_exchange(&p[idx], new);

  if(!old) return 0;

  fetch_and_sub(&mappings[old], 1);
  return 0;
}
```

Simulating Code

```
...

void fas_map_old(uint th){
  mappings[old[th]]--;
  //@ghost ref[th] = 0;
  pct[th] = 0;
}

...
```

Simulation of the concurrency

Original Code

```
#define NOF 2048
#define MAX  256

uint mappings[NOF];

int set_entry(uint fn, uint idx, uint new){
  uint c_n = mappings[new];
  if(c_n >= MAX) return 1;
  if(!CAS(&mappings[new], c_n, c_n+1))
    return 1;

  page_t p = get_frame(fn);
  uint old = atomic_exchange(&p[idx], new);

  if(!old) return 0;

  fetch_and_sub(&mappings[old], 1);
  return 0;
}
```

Simulating Code

```
...

void interleaving(){
  while(true){
    uint th = choose_a_thread();

    switch(pct[th]){
    case 0 : gen_args(th);     break;
    case 1 : read_map_new(th); break;
    case 2 : test_map_new(th); break;
    case 3 : cas_map_new(th);  break;
    case 4 : exch_entry(th);   break;
    case 5 : test_old(th);     break;
    case 6 : fas_map_old(th);  break;
    }
  }
}
```

Parts of the module verified

- For low-level functions, we conducted a “classic” verification
  - Specification with ACSL
  - Automatic proof with WP and SMT Solver : CVC4/Z3
- For the concurrent function used to change pagetables :
  - First specification and proof for sequential version
  - Weakening of the invariant for concurrency
  - Creation and specification of the simulation and proof

Some interactive proofs

- Occurrence counting in arrays relies on :
  - Axiomatization of a simple recursive counting method
  - Lemmas that define properties about this function
- These lemmas could not be proved automatically
  - the proof is done in Coq by extracting them from WP

Lessons Learned, Limitations and Benefits

- Ability to treat concurrent programs
  - With a tool that originally does not handle parallelism
  - Proof done mostly automatically
  - Verification of properties in isolation
- Scalability
  - By-hand simulation is tedious and error prone
  - Could perfectly be automized
  - Need for specification mean for concurrent behaviors

Results

Our approach is valid as long as :

- This function is the only function allowed to modify pagetables
  - Actually, one another function is allowed to modify them,
  - It could be added to the analysis
- The program respects an interleaving semantics
  - In our case, it is true,
  - In the general case, the simulation would not be correct

Conclusion

We performed the deductive verification of a concurrent program in Frama-C that originally do not deal with it

- This method is quite simple
- Automatic proof saves a lot of time

We still need some improvement :

- Simulation could be automatically generated
- The specification language could include concurrency material
- We could perform the verification without simulation

Conclusion

We performed the deductive verification of a concurrent program in Frama-C that originally do not deal with it

- This method is quite simple
- Automatic proof saves a lot of time

We still need some improvement :

- Simulation could be automatically generated
- The specification language could include concurrency material
- We could perform the verification without simulation

Thank you for your attention !

Thank you for your attention

Direction de la Recherche Technologique
Département d'Ingénierie des Logiciels et des Systèmes
Laboratoire de Sûreté des Logiciels

Commissariat à l'énergie atomique et aux énergies alternatives
Institut Carnot CEA LIST
Centre de Saclay — 91191 Gif-sur-Yvette Cedex
Etablissement public à caractère industriel et commercial — RCS Paris B 775 685 019