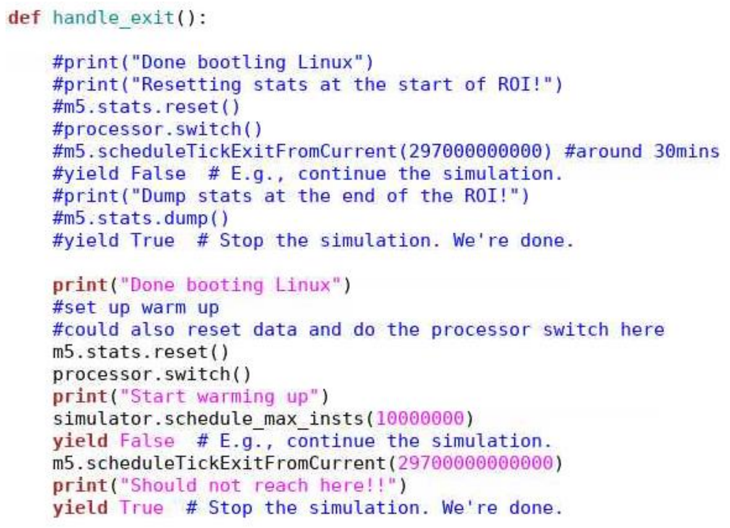
An adaptive reference-count-based Cache replacement policy

For prototyping purposes, we simply hijack the source files of one of the existing replacement policies implemented in Gem5 and replace it with our replacement policy. Thus, we don’t need to change the scons build scripts and so on to add our replacement policy officially.

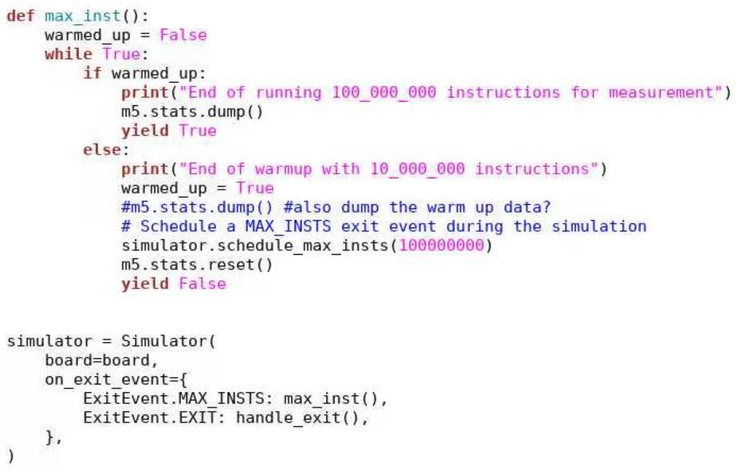
Here are the files we modified:

1. **X86-spec-cpu2017-benchmarks.py** under /gem5/configs/example/gem5_library

   We modify the event exist handling function so that the simulation could pause based on the number of executed instructions. A simple warm up flag is added to tell whether the simulator state is in warming up or measuring state. When simulation stops at the warming up state, we reset the statics and proceed to the measuring state. When simulation stops at the measuring state, we collect all the statics and exit the simulation.



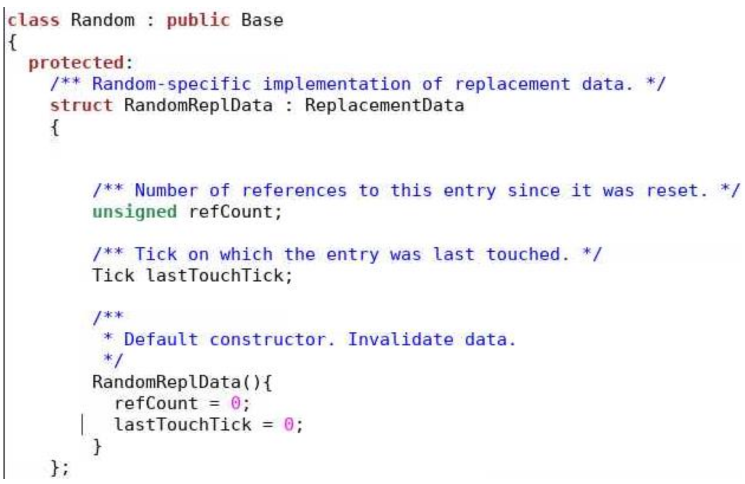
```
def handle_exit():

    #print("Done bootling Linux")
    #print("Resetting stats at the start of ROI!")
    #m5.stats.reset()
    #processor.switch()
    #m5.scheduleTickExitFromCurrent(297000000000) #around 30mins
    #yield False  # E.g., continue the simulation.
    #print("Dump stats at the end of the ROI!")
    #m5.stats.dump()
    #yield True  # Stop the simulation. We're done.

    print("Done booting Linux")
    #set up warm up
    #could also reset data and do the processor switch here
    m5.stats.reset()
    processor.switch()
    print("Start warming up")
    simulator.schedule_max_insts(10000000)
    yield False  # E.g., continue the simulation.
    m5.scheduleTickExitFromCurrent(29700000000000)
    print("Should not reach here!!")
    yield True  # Stop the simulation. We're done.
```



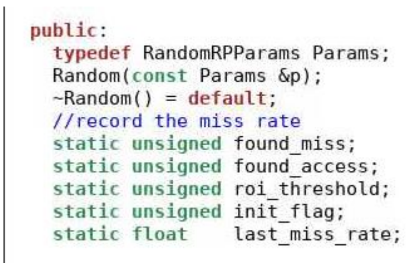
```
def max_inst():
    warmed_up = False
    while True:
        if warmed_up:
            print("End of running 100_000_000 instructions for measurement")
            m5.stats.dump()
            yield True
        else:
            print("End of warmup with 10_000_000 instructions")
            warmed_up = True
            #m5.stats.dump() #also dump the warm up data?
            # Schedule a MAX_INSTS exit event during the simulation
            simulator.schedule_max_insts(100000000)
            m5.stats.reset()
            yield False


simulator = Simulator(
    board=board,
    on_exit_event={
        ExitEvent.MAX_INSTS: max_inst(),
        ExitEvent.EXIT: handle_exit(),
    },
)
```

2. **Random_rp.hh** under /gem5/src/mem/cache/replacement_policies

We add variables to store the reference count and time stamp in the replacement data structure. We also add static variables to store the number of cache accesses, the number of cache misses, reference count threshold, and previous miss rate to calculate and compare miss rates. The interface of member functions is kept the same.



```
class Random : public Base
{
  protected:
    /** Random-specific implementation of replacement data. */
    struct RandomReplData : ReplacementData
    {


        /** Number of references to this entry since it was reset. */
        unsigned refCount;

        /** Tick on which the entry was last touched. */
        Tick lastTouchTick;

        /**
         * Default constructor. Invalidate data.
         */
        RandomReplData(){
          refCount = 0;
          lastTouchTick = 0;
        }
    };
```

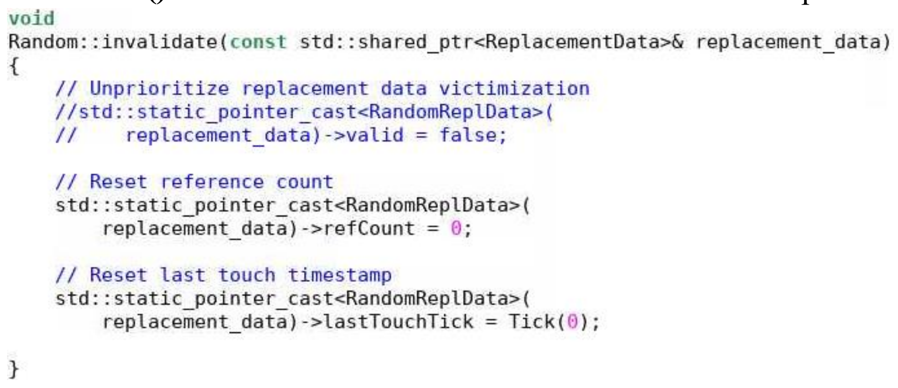


```
  public:
    typedef RandomRPParams Params;
    Random(const Params &p);
    ~Random() = default;
    //record the miss rate
    static unsigned found_miss;
    static unsigned found_access;
    static unsigned roi_threshold;
    static unsigned init_flag;
    static float    last_miss_rate;
```

3. **Random_rp.cc** under /gem5/src/mem/cache/replacement_policies

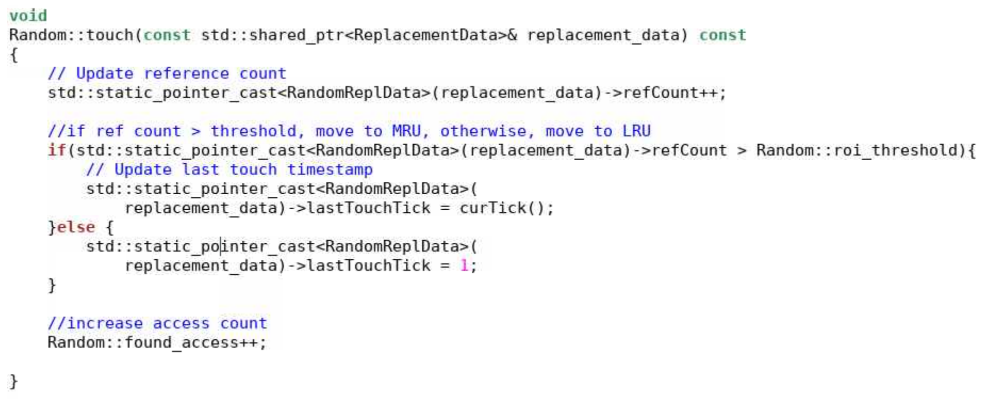
We implement the initialization of the added variables, collection of reference count, cache accesses, and cache misses, the calculation and comparison of miss rate, and the adaptive scheme to adjust the reference count threshold and select cache victim.

a. **Invalidate ()** function - reset the reference count and timestamp.



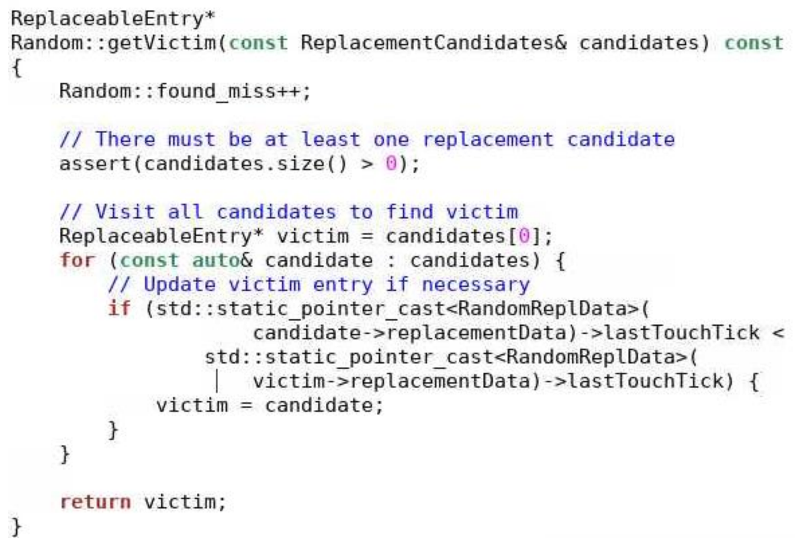
```
void
Random::invalidate(const std::shared_ptr<ReplacementData>& replacement_data)
{
    // Unprioritize replacement data victimization
    //std::static_pointer_cast<RandomReplData>(
    //    replacement_data)->valid = false;

    // Reset reference count
    std::static_pointer_cast<RandomReplData>(
        replacement_data)->refCount = 0;

    // Reset last touch timestamp
    std::static_pointer_cast<RandomReplData>(
        replacement_data)->lastTouchTick = Tick(0);

}
```

b. **Touch ()** function – update reference count, cache accesses number, time stamp on cache hit.



```
void
Random::touch(const std::shared_ptr<ReplacementData>& replacement_data) const
{
    // Update reference count
    std::static_pointer_cast<RandomReplData>(replacement_data)->refCount++;

    //if ref count > threshold, move to MRU, otherwise, move to LRU
    if(std::static_pointer_cast<RandomReplData>(replacement_data)->refCount > Random::roi_threshold){
        // Update last touch timestamp
        std::static_pointer_cast<RandomReplData>(
            replacement_data)->lastTouchTick = curTick();
    }else {
        std::static_pointer_cast<RandomReplData>(
            replacement_data)->lastTouchTick = 1;
    }

    //increase access count
    Random::found_access++;

}
```

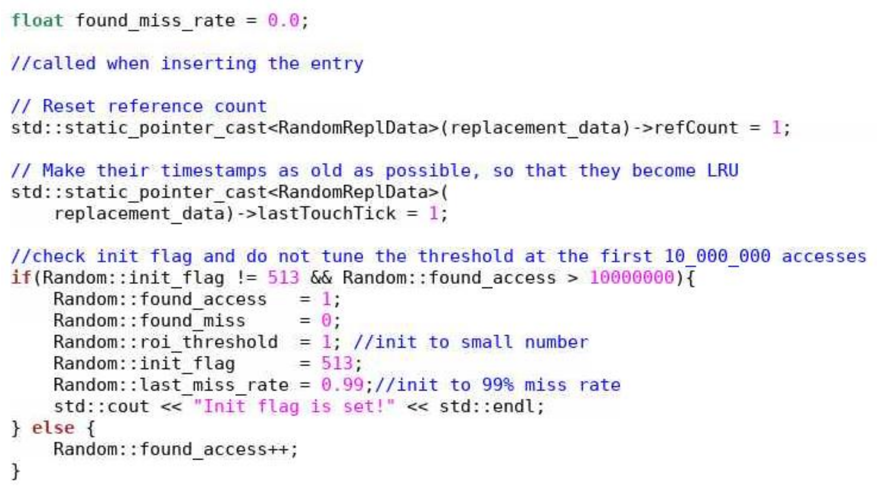
c. **getVictim ()** function – update cache miss number and select the LRU victim on cache miss.



```
ReplaceableEntry*
Random::getVictim(const ReplacementCandidates& candidates) const
{
    Random::found_miss++;

    // There must be at least one replacement candidate
    assert(candidates.size() > 0);

    // Visit all candidates to find victim
    ReplaceableEntry* victim = candidates[0];
    for (const auto& candidate : candidates) {
        // Update victim entry if necessary
        if (std::static_pointer_cast<RandomReplData>(
                    candidate->replacementData)->lastTouchTick <
                std::static_pointer_cast<RandomReplData>(
                |   victim->replacementData)->lastTouchTick) {
            victim = candidate;
        }
    }

    return victim;
}
```

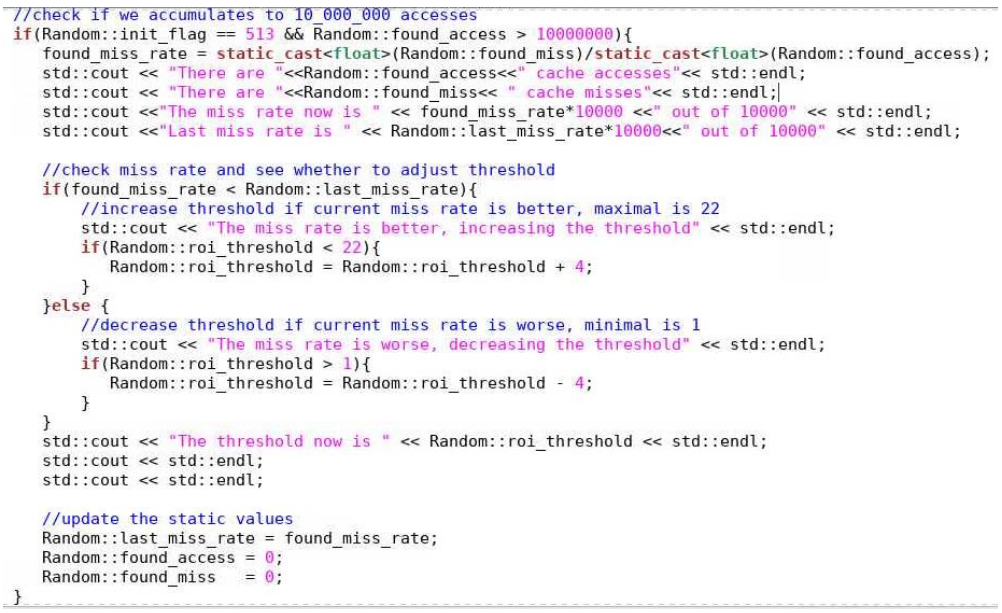
d. Reset () function**:**

The reset function is called when there is a new entry into the cache. We start the initialization and the tuning of the reference count threshold after the first 10 million cache accesses to warming up the cache. The previous miss rate is initialized to 0.99 to guarantee an increment of threshold at the first tuning.



```
float found_miss_rate = 0.0;

//called when inserting the entry

// Reset reference count
std::static_pointer_cast<RandomReplData>(replacement_data)->refCount = 1;

// Make their timestamps as old as possible, so that they become LRU
std::static_pointer_cast<RandomReplData>(
    replacement_data)->lastTouchTick = 1;

//check init flag and do not tune the threshold at the first 10_000_000 accesses
if(Random::init_flag != 513 && Random::found_access > 10000000){
    Random::found_access   = 1;
    Random::found_miss     = 0;
    Random::roi_threshold  = 1; //init to small number
    Random::init_flag      = 513;
    Random::last_miss_rate = 0.99;//init to 99% miss rate
    std::cout << "Init flag is set!" << std::endl;
} else {
    Random::found_access++;
}
```

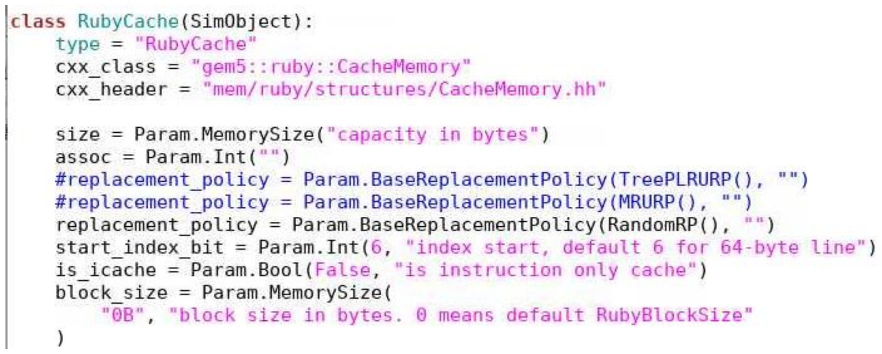
After the initialization, on every 10 million cache accesses, we calculate the current local cache miss rate and compare it with the previous one. If the miss rate improved, we continue increasing the threshold; otherwise, the threshold is decreased. The threshold is bounded to stay in a reasonable range.



```
//check if we accumulates to 10_000_000 accesses
if(Random::init_flag == 513 && Random::found_access > 10000000){
   found_miss_rate = static_cast<float>(Random::found_miss)/static_cast<float>(Random::found_access);
   std::cout << "There are "<<Random::found_access<<" cache accesses"<< std::endl;
   std::cout << "There are "<<Random::found_miss<< " cache misses"<< std::endl;
   std::cout <<"The miss rate now is " << found_miss_rate*10000 <<" out of 10000" << std::endl;
   std::cout <<"Last miss rate is " << Random::last_miss_rate*10000<<" out of 10000" << std::endl;

   //check miss rate and see whether to adjust threshold
   if(found_miss_rate < Random::last_miss_rate){
       //increase threshold if current miss rate is better, maximal is 22
       std::cout << "The miss rate is better, increasing the threshold" << std::endl;
       if(Random::roi_threshold < 22){
          Random::roi_threshold = Random::roi_threshold + 4;
       }
   }else {
       //decrease threshold if current miss rate is worse, minimal is 1
       std::cout << "The miss rate is worse, decreasing the threshold" << std::endl;
       if(Random::roi_threshold > 1){
          Random::roi_threshold = Random::roi_threshold - 4;
       }
   }
   std::cout << "The threshold now is " << Random::roi_threshold << std::endl;
   std::cout << std::endl;
   std::cout << std::endl;

   //update the static values
   Random::last_miss_rate = found_miss_rate;
   Random::found_access = 0;
   Random::found_miss   = 0;
}
```

4. **RubyCache.py** under /gem5/src/mem/ruby/structures

We modify what replacement policy would be used in the Ruby Cache hierarchy, where both L1 and L2 caches use the same replacement policy based on the current implementation. As we are hijacking the source files of the random replacement policy, we change the Ruby Cache to use random.



```
class RubyCache(SimObject):
    type = "RubyCache"
    cxx_class = "gem5::ruby::CacheMemory"
    cxx_header = "mem/ruby/structures/CacheMemory.hh"

    size = Param.MemorySize("capacity in bytes")
    assoc = Param.Int("")
    #replacement_policy = Param.BaseReplacementPolicy(TreePLRURP(), "")
    #replacement_policy = Param.BaseReplacementPolicy(MRURP(), "")
    replacement_policy = Param.BaseReplacementPolicy(RandomRP(), "")
    start_index_bit = Param.Int(6, "index start, default 6 for 64-byte line")
    is_icache = Param.Bool(False, "is instruction only cache")
    block_size = Param.MemorySize(
        "0B", "block size in bytes. 0 means default RubyBlockSize"
    )
```