OPERATING SYSTEMS PROJECT #3

CS 4328.004

ANDY HERR, jah534
AUSTIN FAULKNER, a_f408

1. A Brief Overview of the Design & Implementation of the Page Replacement Algorithms Code

The program is set up to be inheritance based in general. At the top is a `Simulator.h` specification file and its corresponding implementation file `Simulator.cpp`. In `Simulator.h` several variables are declared: `refStringLen = 100`, `freeFrames`, `usedFrames`, `numFaults`, and `static const std::array<std::size_t, refStringLen> refString;`, the latter four with *protected* access specification.

In the `Simulator.cpp` implementation file, we have the following `refString`

```
const std::array<std::size_t, Simulator::refStringLen> Simulator::refString =
{
   {
      43,43,48,30,12,49,48,21,42,43,38,12,47,25,34,24,41,39,45,6,
      13,49,13,25,28,48,27,7,30,46,22,2,31,41,32,41,23,2,8,6,46,
      17,7,23,8,45,33,28,4,31,14,31,24,27,25,39,5,3,10,13,21,35,
      29,21,38,47,31,24,12,41,37,19,34,15,46,49,49,4,22,45,16,23,
      40,16,9,31,2,5,0,4,4,39,12,23,11,22,43,21,15,33
   }
};
```

along with the following "parent" functions:

`Simulator::Simulator(std::size_t numFrames) : freeFrames(numFrames) {}`

// Initializes numFrames by constructor,

`void Simulator::noReplace(std::size_t refStringIndex) { // Empty function. }`

// Do nothing if there isn't a page fault

`void Simulator::outputNumFaults(const std::string fileName) const`

// This simply outputs `numFaults` to a specified "filename" which is comma separated.

And finally there is the main function which allows all derived classes to run their simulations:

See Next Page for the `runSim()` Function

Date: November 29, 2021.

```
void Simulator::runSim()
{
   std::size_t curRef = 0;

   //Iterates through the reference string.
   //If the current element is not in `usedFrames`, runs replacement
   //algorithm, which will vary between subclasses.
   //Otherwise, nothing really happens.
   for (std::size_t refStringIndex = 0;
      refStringIndex < refString.size();
      refStringIndex++)
   {
      curRef = refString[refStringIndex];

      //Read: If current reference is in usedFrames.
      //Have to do it this way since contains() is a C++20 thing,
      //And the school's compilers only go up to C++11.
      if (usedFrames.find(curRef) != usedFrames.end())
      {
         noReplace(refStringIndex);
      }
      else
      {
         numFaults++;
         replacementAlgo(refStringIndex);
         usedFrames.insert(curRef);
      }
   }
}
```

It's worth mentioning that there are two virtual functions declared in `Simulator.h`. There are:

```
virtual void noReplace(std::size_t refStringIndex);

virtual void replacementAlgo(std::size_t refStringIndex) = 0;
```

These allow for the declarations to be *overridden* in the derived class specification files, and then defined in the derived class implementation files.

The files inheriting the `Simulator.h` and `Simulator.cpp` semantics are files corresponding to behavior pertaining to the page replacement algorithms we are interested in. Namely, these files are `FIFO.h` and `FIFO.cpp`; `LRU.h` and `LRU.cpp`; and `OPT.h` and `OPT.cpp`. FIFO indicates the "First In, First Out" page replacement algorithm. LRU indicates the "Least Recently Used" page replacement algorithm. And OPT indicates the "Optimal Page Replacement" algorithm.

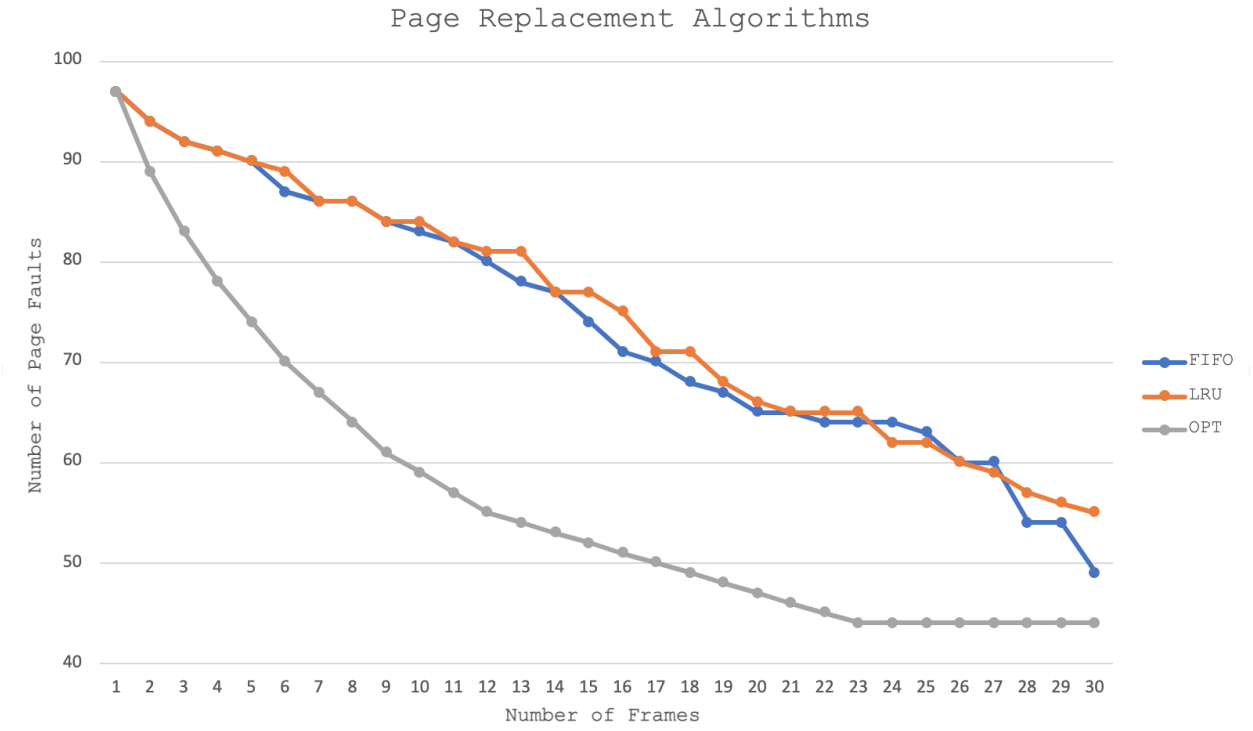
Each derived specification file (.h) has the crucial overridden declaration `void replacementAlgo(std::size_t refStringIndex) override;`. The definition of these are the heart of the program along with `runSim()`. For the sake of brevity we won't present the algorithms algorithms here. We note however that a `queue` is used for FIFO's algorithm, and a `list` is used for LRU. OPT is more involved. The OPT algorithm only considers the first reference to each frame, and ignores all subsequent references to it. To accomplish this, we copy usedFrames into another hash set that we can safely remove stuff from.

2. Instructions on How to Compile & Run on the CS Linux Servers

In the `program3_a_f408.zip` or `program3_jah534.zip` files, there is a `makefile`. When you issue the command `make run` on TX State Linux Server both the program `main.cpp` and the bash script `runSims.sh` will be executed. The output we are concerned with are the files `FIFO.csv` (the FIFO results), `LRU.csv` (the Least Recently Used results), `OPT.csv` (the Optimal Page Replacement results).

In order to clean up the directory in which the above `*.csv`, the `program3` executable, and any `*.o` files are held, simply issue the command `make clean`. The directory will still contain `main.cpp`[1], the `makefile`, the bash script `runSims.sh`, the README files, the directory `inc` containing the `*.h` specification files, the directory `src` containing the `*.cpp` implementation files, and the empy `obj` directory. Leaving these files intact allows for more simulations.

3. Results of the Experiment & and Their Interpretation



Observations: Each of the above functions are monotonic nonincreasing. That is, either the function is decreasing for FIFO, LRU, and OPT, respectively, or it is constant. What's strange about this behavior is **for the reference string given on page .1** is Belady's anomaly **does *not* present**. This means that at no point is there an increase in page faults for any increase in physical-frames.

It's worth further noting that FIFO and LRU compete to be the better page replacement algorithm, even though FIFO in general is not always very consistent in performing well (See Silberschatz, *et al.*, "Operating System Concepts," 10th ed. pg. 405 (print version)). It's apparent from the graph that OPT way out

[1] Which is in directory `src`; see below.

performs the other two algorithms. It roughly follows an inverse square law, or power law, which in general makes for best performance.