Raúl E. Negrón Otero

801-13-4680

CCOM4017

Dr. José Ortiz

Page Replacement Algorithms

Description of the program

These Python scripts simulate three page replacement algorithms. They are:

`optimal.py` - The theoretically optimal page replacement algorithm. It is impossible to implement in practice since you can't provide a list of future page requests in any normal user scenario. But, given a sequence of future page requests, this algorithm will replace pages which will be used again later than any other page in memory. Therefore, it avoids the most page faults and is used as a theoretical ceiling.

`second.py` - The second-chance page replacement algorithm is a modified version of the FIFO page replacement algorithm. The difference is that in second-chance, each page is assigned a Reference bit. By going through the list in FIFO order whenever a page fault occurs, if the current page has a Reference bit equal to 1, then the page is given a "second chance" and moved to the tail of the list, with its R bit cleared. If the page is in memory and gets a page hit, then its R bit is switched on as well. Therefore, for a list of pages with all their R bits equal to 1, the second-chance algorithm deteriorates into a FIFO algorithm with an initial clearing pass.

`wsclock.py` - A page replacement algorithm which tries to unite the ideas introduced by the *clock* algorithm and the concepts used in a *working set* model. It is implemented by keeping an internal logical clock and a circularly linked list. We also need a Reference bit and a Modified bit for each page frame. Additionally, a new numerical parameter named *Tau* is introduced. Every time a page fault occurs, the page pointed to by the clock hand (an element of the list) is inspected. If its R bit equals 1, the page has been referenced very recently and should be left alone. Before leaving it be, the algorithm clears its R bit.

If, however, the page has a Reference bit equal to 0, then its age (current clock time - page's stored clock time) is inspected. If it turns out that the page's age is greater than *Tau*, we say that the page is *not* in the working set and is therefore a candidate for removal. To make sure the page can be removed safely, its M bit is inspected. If the Modified bit equals 1, then the page is said to be "dirty"; a disk write is scheduled for the page and its M bit is cleared. The clock hand then moves on to see if it can find an old, "clean" page.

But if the page has an M bit equal to 0, then it is safe to remove and its space in memory is claimed. If the clock hand moves all the way around and ends up where it began without fixing the page fault, then there is at least one clean page now. The algorithm simply moves to claim the first clean page it finds.

How to use

Both `optimal.py` and `second.py` can be run as follows:

```
$ <script.py> <Maximum memory space> <access sequence file>
```

For example:

```
$ optimal.py 10 input.txt
```

`wsclock.py` requieres and additional argument, `Tau`, which is placed after the `<Maximum memory space>` argument.

For example:

```
$ wsclock.py 10 5 input.txt
```

Where `5` correspondonds to the parameter `Tau`.

For every script, the output is

```
$ Page Faults:<value>
```

where `<value>` is the number of page faults that occured using the specified page replacement algorithm with the specified memory space and the page access sequence provided.

Who helped

Nobody :(