<u>Completed and incomplete task:</u>

1. **Finished tasks :**
    - Implement modules by following steps in chapter 5 of OSP2
    - Collect and output system performance (total page faults) to the log
    - Implemented FIFO and LRU page replacement algorithms.
    - Test the FIFO and LRU and collect their performance data
    - Attached Readme file with Report.
    - Prepared three minutes short presentation.
    - Compared all algorithms implemented based on our experiment in your readme file/report and presentation.

2. **Incomplete task : none**

Summary of Replacement algorithms:

Page replacement algorithms decide which memory pages to page out (swap out, write to disk) when a page of memory needs to be allocated. Paging happens when a page fault occurs and a free page cannot be used to satisfy the allocation, either because there are none, or because the number of free pages is lower than some threshold.When the page that was selected for replacement and paged out is referenced again it has to be paged in (read in from disk), and this involves waiting for I/O completion. This determines the *quality* of the page replacement algorithm: the less time waiting for page-ins, the better the algorithm.

- **Summary of FIFO replacement algorithms:**

Replace the page that has been in memory for the longest time."

The idea is obvious from the name – the operating system keeps track of all the pages in memory in a queue **or** keep track of reference time when first created. When a page needs to be replaced, the page at the front of the queue (the oldest page) is selected and in case of time reference method maxmium difference between current and reference time is selected from all pages.

Advantage:

The first-in, first-out (FIFO) page replacement algorithm is a low-overhead algorithm that requires little bookkeeping on the part of the operating system. But the oldest page may be needed again soon some page may be important It will get old, but replacing it will cause an immediate Page Fault.

CODE:

we have keep track of reference time in PageTableEntry when page enters first time in memory. And whenever page fault occurs, we loop through all pages and replace the page that has maximum difference between current time and first time referred. Before that some cases has been taken cared. By this oldest page will be replaced.

case 1: if((newFrame.getPage() == null) && (!newFrame.isReserved()) && (newFrame.getLockCount() == 0))
case 2 : if((!newFrame.isDirty()) && (!newFrame.isReserved()) && (newFrame.getLockCount() == 0))
case 3 : if((!newFrame.isReserved()) && (newFrame.getLockCount() == 0)) case 4 : for(int i = 0; i < MMU.getFrameTableSize(); i++)

```
{
        newFrame = MMU.getFrame(i);
                PageTableEntry newPage = newFrame.getPage();
                if(Math.abs(HClock.get() - newPage.reftimer) > maxTillNow){
                        newFrameMax = newFrame;
                        maxTillNow= Math.abs(HClock.get() - newPage.reftimer);
                }
}
```

Here newFrame is frame will be returned in case1, case2 and case3 and newFrameMax in case4 which basically max difference between current time and refer time.

- **Summary of LRU replacement algorithms**

Keep track of when a page is used. Replace the page that has been used least recently.

Advantage:

Treat the RAM as a cache. In order to be an effective cache, it needs to keep the items most likely to be requested in memory.LRU keeps the things that were most recently used in memory.

CODE:

MMU update reference counter time every time when page is accessed unlike FIFO where it is updated only on creation time. And whenever page fault occurs, we loop through all pages and replace the page that has maximum difference between current time and first time referred. Before that some cases has been taken cared which has been described above in FIFO. By this least recently used are not be replaced. the one with older reference time will be selected.

Below is snapshot : here if LRU is true than we are updating page refernce time with current time in do_refer function.

```
static public PageTableEntry do_refer(int memoryAddress,

                                          int referenceType, ThreadCB thread)

  {

 …………………………….

if(LRU == true)

     page.reftimer = HClock.get();
return page;
}
```

Compare of Replacement algorithms:

A. **Experiment configuration Memory Management**

Run Number	Memory Reads	Memory Writes	Simulation Length
1	10	90	250000
2	20	80	250000
3	30	70	250000
4	40	60	250000
5	50	50	250000
6	60	40	250000
7	70	30	250000
8	80	20	250000
9	90	10	250000

B. **Table for performance data FIFO**

Run Number	CPU Util	Task Created	Finished Task	Unfinished Task	PageFault Count	Algo
1	68.46	12	8	4	857	FIFO
2	74.73	16	12	4	850	FIFO
3	65.98	15	11	4	773	FIFO
4	85.11	17	14	3	943	FIFO

5	77.31	15	12	3	961	FIFO
6	75.1	18	12	6	1136	FIFO
7	81.6	16	12	4	1144	FIFO
8	83.56	16	11	5	1229	FIFO
9	88.22	18	15	3	1205	FIFO

C. <u>Table for performance data LRU</u>

Run Number	CPU Util	Task Created	Finished Task	Unfinished Task	PageFault Count	Algo
1	68.4	12	8	4	805	LRU
2	74.73	16	12	4	874	LRU
3	80.67	14	9	5	850	LRU
4	71.93	14	8	6	925	LRU
5	74.95	14	11	3	895	LRU
6	95	18	12	6	1136	LRU
7	84.58	13	9	4	1058	LRU
8	87.86	14	10	4	1066	LRU
9	97.13	18	14	4	1043	LRU

- A page fault occurs when the referenced page by the CPU can not be mapped to any frame in the main memory (or TLB). We have used static count variable to count number of page fault in FIFO and LRU algorithm.

- As from above FIFO and LRU table, page fault count is less in LRU such as run number 1, 3, 4, 5, 6, 7, 8 and 9. This is mostly because according to <u>temporal locality of reference</u>, memory that has been accessed recently is more likely to be accessed again soon.

- In run number 2, FIFO shows less page fault than LRU. This may be due to frequently accessing the same page for longer running time or may be oldest page used more.

- We have also observed that as number of reads increase and write decreases , page fault count increases both in case LRU and FIFO. This may be due to increase in CPU utilization time in both the case.

Conclusion:

We found LRU performance is better in most cases.This may be due to temporal locality of reference is better in LRU than FIFO.In some cases, FIFO performance better because of more random access.

In addition, LRU has **more overhead** because you have to keep track of when the pages were used. And if size is larger, this may take longer to process. However the theory is that you will end up having more hits because you keep in memory the recently used pages.