Question 1

The enclosed tarfile contains the following files: homework.c, make.sh, and test.py. The homework.c source file consists of two parts – an application based on the Python library, to use up lots of memory, and code to simulate page faults and page fault handling.

In particular, we install a signal handler to intercept signal 11, SIGSEVG or “segmentation violation”. (the name is historic, but means an illegal access to memory) Then we use the mprotect() system call to disable access (“PROT_NONE”) to most of the virtual address space of the process, skipping only the heap (used by malloc), the stack, and functions and variables defined in the homework.c file. When a page fault (signal 11) occurs, the fault function must restore permissions to the faulting page – since we are not told whether it is a read, write, or execute fault we enable all access – PROT_READ, PROT_WRITE, and PROT_EXEC.

Note that we cannot handle a fault in the fault handler, so it can only use:

- local variables
- global variables in homework.c, or from malloc in the setup function
- functions defined in homework.c. (this includes special versions of printf and mprotect, but not malloc)

If you are compiling on a non-CCIS system, you will need the Python development libraries and Python interpreter installed.

The first argument to homework.c is the string “q1” or “q2”. Upon startup, it installs the signal handler fault_q1 and runs setup_q1 (or equivalently 'q2') according to the argument passed, and then starts the python application which waits for connections. Type 'python test.py' in another window, which will request multiple files from the 'homework.c' application and then tell it to exit.

Note that a sample fault handler has been provided which just re-enables access to any faulting page – it will run if invoked as './homework sample'

Question 1 – FIFO replacement

Usage: ./homework q1 #   - simulate '#' pages of memory and FIFO page replacement

For this question you will write fault_q1 and setup_q1, which simulate FIFO page replacement. The memory size is given by the variable 'memsize_in_pages', and is passed on the command line as given above.

Excluding the pages skipped as described above (homework.c, stack, and heap), no more than 'memsize_in_pages' 4K pages of memory should be accessible (PROT_READ|PROT_WRITE|PROT_EXEC) at any time, and access to other pages should be disallowed. (PROT_NONE)

Count how many page faults occur when running with 1024, 512, and 256 pages of memory. For each memory size, report the values from 5 separate test runs, as well as the average.
**Question 2 – VMS Page Replacement**

Usage: `./homework q2 #npages #.#fraction` - run with '#npages' total memory, '#.#fraction' reserved

In Question 2 we simulate the VMS approximation to LRU. Given N pages of memory ('memsize_in_pages'), we keep a fraction of them (double 'reserve_fraction') in the **reserved pool**, and keep the remaining pages in the **active pool** where they are treated normally.

- Pages in the reserved pool are protected (PROT_NONE); when a fault occurs for one of them we (1) count a *soft fault*, (2) un-protect the page and move it into the active pool, and (3) move the oldest page in the active pool into the reserved pool and protect it.

- When a fault occurs for a page not in the reserved pool, we (1) count a *hard fault*, (2) un-protect the page and move it into the active pool, (2) move the oldest page in the active pool into the reserved pool, and (3) evict the oldest page in the reserved pool.

So at any time there should be 'memsize_in_pages * (1.0 - reserve_fraction)' pages which have access enabled, and another 'memsize_in_pages * reserve_fraction' pages which can be accessed without a hard fault.

Run 6 cases – 256, 512, and 1024 pages of memory, each with reserve fraction 0.1 and 0.2. Take 5 measurements for each case, and report the number of hard faults.

How does this compare with strict FIFO in Question 1?
Question 3 – Disk Performance

The XYZ-7600 disk drive has the following characteristics:

- 7200 RPM (5 ms/revolution)
- Block size of 4096 bytes
- 40 tracks divided into 4 zones:
  - 10 tracks of 54 blocks (LBA 0..539)
  - 10 tracks of 45 blocks (LBA 540..989)
  - 10 tracks of 36 blocks (LBA 990..1349)
  - 10 tracks of 27 blocks (LBA 1350..1619)
  for a total of 1620 blocks or \(6.6 \times 10^6\) bytes.
- A side-to-side seek time of 10ms, so that seeks from track \(i\) to track \(j\) take \(\frac{|i-j|}{4}\) ms.

For simplicity we assume that blocks are aligned identically on each track, as shown in the figure, rather than being offset from one track to the next.

A) How long would it take to read all the data on the drive if disk blocks had to be read sequentially? (i.e. the first block on each track had to be read before any other blocks on the track)

How fast could the contents of the disk be read if blocks could be read in any order?

Note that to read a disk block the head must be finished seeking and on the correct track in time for the first byte of the block, and has to remain there until the last byte is done.

B) Write a simulator which takes as input a list of block numbers (decimal, 0-1619, one per line) and calculates the minimum time that sequence of operations would take. You may use any language which is available on the CCIS Linux systems; please indicate in your submission what arguments and/or environment variables I will need to use in order to run it from the command line.

How long would this disk take for the following sequence? Please indicate the time at which each operation completes. (in ms from issuing the first request)

\[963\ 490\ 852\ 306\ 1133\ 918\ 906\ 207\ 746\ 821\ 604,\]
\[768\ 1511\ 1242\ 978\ 1119\ 559\ 60\ 1259\ 698\ 432\ 515\]
Question 4 – Paging simulation

At the beginning of the output from homework.c there is a table mapping address ranges to block numbers. Generate a fault trace with FIFO page replacement (question 1) with 512 entries. Translate the page address trace into a block number trace, and use your simulator to determine how long it would take to access that sequence of blocks.

Deliverables:

Turn in the following:

• homework.c with code for questions 1 and 2
• a document giving answers to the 4 questions. (including measured values, etc. - however, not long traces)
• the simulation code which you wrote
• for question 4, save the page fault trace, the page->block map, and the translated block trace. I should be able to run your simulator on the block trace you submit and get the same results as the answer you provide to question 4.

Please package these in a tar file and upload to Blackboard.