**CS3103 2023/2024 Sem B**

**Assignment 2**

**Instructions**

1. **Due: April xxxth (Sunday), 12:59 PM**. (10 marks deducted if late for <=12 hours, 20 marks deducted if late for 12-24 hours, and so on).
2. Please write your answer directly on this document and submit it to Canvas as *doc*, *docx* or *pdf* files. Answers are allowed in text only. Any form of image/snapshot is not allowed.
3. Before you start, please take the time to review the course policy on academic integrity at: <https://www.cityu.edu.hk/ah/academic_honesty.htm>.

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

SID: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Question 1 (20 marks)**: Consider the following resource allocation graph:

A diagram of a network

Description automatically generated

Question 1.1 (5 marks) Does the above allocation graph contain a deadlock? Explain your answer using no more than two sentences.

Answer: No. A possible execution sequence is P2, P1, P3.

Question 1.2 (5 marks) Assume now that P2 also demands resource R1. Does this allocation graph contain a deadlock? Explain your answer using no more than two sentences.

Answer: Yes. If P2 demands R1, none of P1, P2, or P3 will get all resource they need to be processed. Therefore, there is a deadlock.

Question 1.3 (5 marks) Assume the allocation graph in problem b), and, in addition, assume that R2 has now three instances. Does this allocation graph contain a deadlock? Explain your answer using no more than two sentences.

Answer: No. The only possible execution sequence is P1, P2, P3.

Question 1.4 (5 marks) Add to the original allocation graph an additional process P4 that demands an instance of R1. Does the allocation graph contain a deadlock? Explain your answer using no more than two sentences.

Answer: No. A possible execution sequence is P2, P1, P3, P4.

**Question 2 (20 marks)** Consider the following snapshot of a system.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Allocation | | | | Max | | | |
|  | A | B | C | D | A | B | C | D |
| P0 | 2 | 1 | 3 | 2 | 5 | 3 | 5 | 4 |
| P1 | 1 | 2 | 2 | 1 | 3 | 5 | 7 | 3 |
| P2 | 4 | 3 | 4 | 4 | 5 | 5 | 5 | 5 |
| P3 | 1 | 4 | 1 | 2 | 3 | 7 | 2 | 4 |
| P4 | 4 | 3 | 2 | 4 | 7 | 5 | 3 | 6 |

|  |  |  |  |
| --- | --- | --- | --- |
| Available | | | |
| A | B | C | D |
| 2 | 2 | 2 | 1 |

Question 2.1 (15 marks): Suppose we use Banker’s algorithm to decide whether the system is currently in the safe sate, please answer the values of the Need matrix.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Need | | | |
|  | A | B | C | D |
| P0 | 3 | 2 | 2 | 2 |
| P1 | 2 | 3 | 5 | 2 |
| P2 | 1 | 2 | 1 | 1 |
| P3 | 2 | 3 | 1 | 2 |
| P4 | 3 | 2 | 1 | 2 |

Question 2.2 (5 marks): Is the system in a safe state? If so, please list a safe sequence. If not, please explain why.

Answer:

Yes, one possible safe sequence is <P2,P0,P1,P3,P4>

**Question 3 (10 marks)** Given the following reference string of size 12:

5, 3, 2, 1, 5, 4, 6, 3, 7, 9, 5, 1

Run (a) the FIFO replacement algorithm, (b) the LRU page replacement algorithm on the above reference string with a 4-frame physical memory. Please compute the number of page faults and page fault ratio (number of page hits divided by the number of references) by FIFO and LRU respectively on that string, respectively.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **5** | **3** | **2** | **1** | **5** | **4** | **6** | **3** | **7** | **9** | **5** | **1** |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Answer to (a) FIFO** | | | | | | | | | | | | |
| Number of page fault: 11; Page fault ratio: 11/12 = 0.92 | | | | | | | | | | | | |
| **5** | **3** | **2** | **1** | **5** | **4** | **6** | **3** | **7** | **9** | **5** | **1** |
| 5 | 5 | 5 | 5 | 5 | 4 | 4 | 4 | 4 | 9 | 9 | 9 |
|  | 3 | 3 | 3 | 3 | 3 | 6 | 6 | 6 | 6 | 5 | 5 |
|  |  | 2 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 1 |
|  |  |  | 1 | 1 | 1 | 1 | 1 | 7 | 7 | 7 | 7 |
| **Answer to (b) LRU** | | | | | | | | | | | | |
| Number of page fault: 11; Page fault ratio: 11/12 = 0.92 | | | | | | | | | | | | |
| **5** | **3** | **2** | **1** | **5** | **4** | **6** | **3** | **7** | **9** | **5** | **1** |
| 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 7 | 7 | 7 | 7 |
|  | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 9 | 9 | 9 |
|  |  | 2 | 2 | 2 | 2 | 6 | 6 | 6 | 6 | 5 | 5 |
|  |  |  | 1 | 1 | 1 | 1 | 3 | 3 | 3 | 3 | 1 |

**Question 4 (20 marks)** In a machine using multi-level paging policy, the virtual address is subdivided into 4 segments as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| 10-bit | 8-bit | 6-bit | 8-bit |

We use a 3-level page table, such that the first 10-bit are for the first level and so on. The memory is indexed in bytes. 2 bytes per entry, no matter what level.

Question 4.1: How many bytes are in one page of such a system (i.e., page size)? Why?

Answer

The page field is 8-bit wide, then the page size is 256 bytes.

Question 4.2: How many entries are needed for the page tables of different levels in a process that has 256KB of memory, which starts at address 0? Why?

Answer:

The field to indexing 3rd level page tables are 6-bits, so each 3rd level page table points to 64 pages.

To index the program's address space consisting of 1024 pages, we need 1024 entries of 3rd level page table and 16 third-level page tables.

The field to indexing 2nd level page tables are 8-bits, so each 2nd level page table points to 256 3rd page tables.

To index 3rd level page tables, we need 16 entries of 3rd level page table and 1 2nd level page table.

The field to indexing 1st level page tables are 10-bits, so each 1st level page table points to 1024 2nd level page tables.

To index 2nd level page tables, we need 1 entry of 2nd level page table and 1 1st level page table.

**Question 5 (15 marks)** Please explore how to use this simulator by yourself and answer the following question using the simulator.

A page replacement algorithm simulator **paging-policy.py** is available at **/public/cs3103/Assignment2** on the gateway server, which allows you to play around with different page-replacement policies. The default policy is FIFO, though others are available, including LRU.

For example, let's examine how LRU performs with a series of page references with page cache size 3 (i.e., the number of frames is 3):

0 1 2 0 1 3 0 3 1 2 1

To do so, run the simulator as follows

$ ./paging-policy.py --addresses=0,1,2,0,1,3,0,3,1,2,1 --policy=LRU --cachesize=3 -c

And what you would see is:

ARG addresses 0,1,2,0,1,3,0,3,1,2,1

ARG addressfile

ARG numaddrs 10

ARG policy LRU

ARG clockbits 2

ARG cachesize 3

ARG maxpage 10

ARG seed 0

ARG notrace False

Solving...

Access: 0 MISS LRU -> [0] <- MRU Replaced:- [Hits:0 Misses:1]

Access: 1 MISS LRU -> [0, 1] <- MRU Replaced:- [Hits:0 Misses:2]

Access: 2 MISS LRU -> [0, 1, 2] <- MRU Replaced:- [Hits:0 Misses:3]

Access: 0 HIT LRU -> [1, 2, 0] <- MRU Replaced:- [Hits:1 Misses:3]

Access: 1 HIT LRU -> [2, 0, 1] <- MRU Replaced:- [Hits:2 Misses:3]

Access: 3 MISS LRU -> [0, 1, 3] <- MRU Replaced:2 [Hits:2 Misses:4]

Access: 0 HIT LRU -> [1, 3, 0] <- MRU Replaced:- [Hits:3 Misses:4]

Access: 3 HIT LRU -> [1, 0, 3] <- MRU Replaced:- [Hits:4 Misses:4]

Access: 1 HIT LRU -> [0, 3, 1] <- MRU Replaced:- [Hits:5 Misses:4]

Access: 2 MISS LRU -> [3, 1, 2] <- MRU Replaced:0 [Hits:5 Misses:5]

Access: 1 HIT LRU -> [3, 2, 1] <- MRU Replaced:- [Hits:6 Misses:5]

FINALSTATS hits 6 misses 5 hitrate 54.55

The complete set of possible arguments for paging-policy is listed on the following page, and includes a number of options for varying the policy, how addresses are specified/generated, and other important parameters such as the size of the cache.

To do so, run the simulator as follows:

$ ./paging-policy.py -h

Usage: paging-policy.py [options]

Options:

-h, --help show this help message and exit

-a ADDRESSES, --addresses=ADDRESSES

a set of comma-separated pages to access; -1 means randomly generate

-f ADDRESSFILE, --addressfile=ADDRESSFILE

a file with a bunch of addresses in it

-n NUMADDRS, --numaddrs=NUMADDRS

if -a (--addresses) is -1, this is the number of addrs to generate

-p POLICY, --policy=POLICY

replacement policy: FIFO, LRU, OPT, UNOPT, RAND, CLOCK

-b CLOCKBITS, --clockbits=CLOCKBITS

for CLOCK policy, how many clock bits to use

-C CACHESIZE, --cachesize=CACHESIZE

size of the page cache, in pages

-m MAXPAGE, --maxpage=MAXPAGE

if randomly generating page accesses, max page number

-s SEED, --seed=SEED random number seed

-N, --notrace do not print out a detailed trace

-c, --compute compute answers for me

As usual, "-c" is used to solve a particular problem, whereas without it, the accesses are just listed (and the program does not tell you whether or not a particular access is a hit or miss).

Other options include:

"-C/--cachesize" which changes the size of the page cache;

"-m/--maxpage" which is the largest page number that will be used if the simulator is generating references for you;

"-f/--addressfile" which lets you specify a file with addresses in them, in case you wish to get traces from a real application or otherwise use a long trace as input.

"-s/--seed" which specifies the seed for random addresses generating. The same seed leads to the same address reference streams.

"-n/--numaddrs " which specifies the number of addresses to generate.

For a cache of size 3, generate worst-case address reference streams (the number of addresses is 10 and the maximum of addresses is 10) for each of the following policies: FIFO and LRU. A worst-case reference stream for a specific policy, for example FIFO, causes the largest (among FIFO and LRU) number of misses when using this policy.

Give the address reference stream in the style of “./paging-policy.py --addresses=5,8,9,2,7,7,6,1,0,3 -p FIFO -C 3 -n 10 -c” or “./paging-policy.py --addresses=5,8,9,2,7,7,6,1,0,3 -p LRU -C 3 -n 10 -c”

For the worst-case reference streams, what is its hit rate and how much bigger of a cache is needed to improve performance dramatically(increase by more than 20%) and approach OPT?(If )

Answer:

E.g.

./paging-policy.py --addresses=0,1,2,3,4,0,1,2,3,4 -p FIFO -C 3 -n 10 -c

hit rate 0.0%.

To improve performance dramatically and approach OPT, a cache of size 5 or a larger number is needed.

./paging-policy.py --addresses=0,1,2,3,4,0,1,2,3,4 -p LRU -C 3 -n 10 -c

hit rate 0.0%

To improve performance dramatically and approach OPT, a cache of size 5 or a larger number is needed.

**Question 6 (15 marks)** A file system simulator **vsfs.py** is available at **/public/cs3103/Assignment2** on the gateway server. The simulator allows you to get close to how file system state changes when different operations take place by user and system behaviors. The possible operations are:

* mkdir() – creates a new directory
* create() – creates a new (empty) file
* open(), write(), close() – appends a block to a file
* link() – creates a hard link to a file
* unlink() – unlinks a file (Removing it if linkcnt == 0)

The simulator shows the state of the file system by printing the contents of four different data structures:

* inode bitmap – indicates which inodes are allocated
* inodes – table of inodes and their contents
* data bitmap – indicates which data blocks are allocated
* data – indicates contents of data blocks

The bitmaps is fairly straightforward to understand, with a 1 indicating that the corresponding inode or data block is allocated, and a 0 indicating said inode or data block is free.

The inodes each have three fields: the first field indicates the type of file (e.g., f for a regular file, d for a directory); the second indicates which data block belongs to a file (here, files can only be empty, which would have the address of the data block set to -1, or one block in size, which would have a non-negative address); the third shows the reference count for the file or directory. For example, the following inode is a regular file, which is empty (address field set to -1), and has just one link in the file system:

[f a:-1 r:1].

If the same file had a block allocated to it (say block 10), it would be shown as follows:

[f a:10 r:1].

If someone then created a hard link to this inode, it would then become:

[f a:10 r:2].

Finally, data blocks can either retain user data or directory data. If filled with directory data, each entry within the block is of the form (name, inumber), where "name" is the name of the file or directory, and "inumber" is the inode number of the file. Thus, an empty root directory looks like this, assuming the root inode is 0:

[(.,0) (..,0)].

If we add a single file "f" to the root directory, which has been allocated inode number 1, the root directory contents would then become:

[(.,0) (..,0) (f,1)].

If a data block contains user data, it is shown as just a single character within the block, e.g., "h". If it is empty and unallocated, just a pair of empty brackets ([]) are shown.

An entire file system is thus depicted as follows:

Chart, scatter chart

Description automatically generated

This file system has eight inodes and eight data blocks. The root directory contains three entries (other than "." and ".."), to "y", "z", and "f". By looking up inode 1, we can see that "y" is a regular file (type f), with a single data block allocated to it (address 1). In that data block 1 are the contents of the file "y": namely, "u". We can also see that "z" is an empty regular file (address field set to -1), and that "f" (inode number 3) is a directory, also empty. You can also see from the bitmaps that the first four inode bitmap entries are marked as allocated, as well as the first three data bitmap entries.

Following is some usage examples of the simulator vsfs.py:

$ ./vsfs.py -h //get usage information

$ ./vsfs.py -n 6 -s 16 //simulate 6 requests use seed 16

$ ./vsfs.py -n 6 -s 16 -c //simulate 6 requests use seed 16, and also shows the answer

$ ./vsfs.py -n 6 -s 16 -r //simulate 6 requests use seed 16, and use reverse mode (reverse mode will print the operations instead of the states to see if you can predict the state changes from the given operations)

Please answer the following question based this simulator:

Question 6.1 (2 marks): Given the two states of a file system before and after taking place a file system operation, which operation took place?

|  |
| --- |
| **Before** |
|  |
| **After** |
|  |

Answer:

link("/y", "/m") ; or link file “m” to file “y”

Question 6.2 (2 marks): Given the two states of a file system before and after writing new data to a file, which file is written? Which data block stores the new data?

|  |
| --- |
| **Before** |
|  |
| **After** |
|  |

Answer:

file “z” is modified, the new data is stored in data block 3.

Question 6.3 (2 marks): Given the two states of a file system before and after creating a new file, what is the name of the new file? What is the name of the new file’s parent directory?

|  |
| --- |
| **Before** |
|  |
| **After** |
|  |

Answer:

The new file name is “y” and the name of its parent directory is “f”.

Question 6.4 (3 marks): Now reduce the number of data blocks in the file system, to very low numbers (say two), and run the simulator for a hundred or so requests. What types of files end up in the file system in this highly constrained layout? What types of operations would fail?

Answer:

Empty files; write and mkdir.

Question 6.5 (6 marks): Now do the same, but with inodes. With very few inodes, what types of operations can succeed? Which will usually fail? What is the final state of the file system likely to be?

Answer:

With only two inodes, the simulator will only allocate one at most. The root directory already occupies this one, so all operations that need to allocate inodes will fail, such as create file or mkdir. And because the first operation must create a file or create a folder, the system will stop running at the first operation.

Note: with only two inodes, no operations can succeed.

* For “succeed” and “fail”, if answer “fail” and explain correctly
* For “succeed”
* For “fail”

The final state is only a blank root directory. You can also try other small inodes number. At this time, the system can create a small number of files or folders and perform other operations, but in the end, it usually stops running because it cannot continue to create them.