Virtual Memory Lab

This lab project addresses the implementation of page-replacement algorithms in a demand-paging system.

Each process in a demand-paging system has a page table that contains a list of entries. For each logical page of the process, there is an entry in the table that indicates if the page is in memory. If the page is in memory, the memory frame number that page is resident in is indicated. Also, for each page, the time at which the page has arrived in memory, the time at which it has been last referenced, and the number of times the page has been referenced since the page arrived in memory are maintained.

The page table data structure is a simple array of page-table entries (PTEs). Each PTE contains five fields as defined below:

```
struct PTE {
    int is_valid;
    int frame_number;
    int arrival_timestamp;
    int last_access_timestamp;
    int reference_count;
}
```

Each process in the system has a page table that is simply an array of PTEs. Each process also has a pool of frames that is allocated. The frame pool for a process is represented as an array of integers, where each Integer represents the frame number of a frame that is free and is available for use by the process. Note that in order to get the frame number in the pool, you first need to access the integer in the array.

This lab project aims to exercise the various policies for page replacement. In particular, we study the following three page-replacement policies:

1. First-In-First-Out (FIFO)
2. Least-Recently-Used (LRU)
3. Least-Frequently-Used (LFU)

In order to implement the above policies, we need to develop corresponding functions that process page accesses for a process. That is, for each process, given its page table, a logical page number being referenced and the free frame pool of the process, the functions should determine the memory frame number for the logical page. Also, the functions should modify the page table and the free frame pool appropriately. The details of the functions with respect to the different policies are described below. You need to develop code for these functions that

implement the specifications. Place the code in a file called **virtual.c**. You should include the oslabs.h file.

process_page_access_fifo

This function implements the logic to process a page access in a system that uses the First-In First-Out (FIFO) policy for page replacement. Specifically, it takes four inputs:

1. process page table
2. logical page number
3. process frame pool
4. current timestamp.

The function determines the memory frame number for the logical page and returns this number.

First the function checks if the page being referenced is already in memory (i.e., the page-table entry has the valid bit true). If so, it returns the frame number, after modifying the last_access_timestamp and the reference_count fields of the page-table entry.

If the page being referenced is not in memory, the function checks if there are any free frames (i.e., the process frame pool is not empty). If so, a frame is removed from the process frame pool and the frame number is inserted into the page-table entry corresponding to the logical page number. In addition, the other fields of the page-table entry are set appropriately. The function returns the frame number associated with the page-table entry.

If the page being referenced is not in memory and there are no free frames for the process, a page needs to be replaced. The function selects among all the pages of the process that are currently in memory (i.e., they have valid bits as true) the page that has the smallest arrival_timestamp. It marks that page_table entry as invalid, along with setting the frame_number, arrival_timestamp, last_access_timestamp and reference_count to -1. It then sets the frame_number of the page-table entry of the newly-referenced page to the newly freed frame. It also sets the arrival_timestamp, the last_access_timestamp and the reference_count fields of the page-table entry appropriately. Finally, the function returns this frame number.

The signature of the method is as follows:

```
int process_page_access_fifo(struct PTE page_table[TABLEMAX],int *table_cnt, int
page_number, int frame_pool[POOLMAX],int *frame_cnt, int current_timestamp);
```

A sample execution input and output:

input/output	parameter	value
input	page_table	[IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:true,FN:10,ATS:3,LATS:3,RC:1]

		[IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:true,FN:20,ATS:2,LATS:4,RC:2] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:true,FN:30,ATS:1,LATS:1,RC:1]
input	table_cnt	8
input	page_number	0
input	frame_pool	EMPTY
input	frame_cnt	0
input	current_timestamp	12
output	page_table	[IV:true,FN:30,ATS:12,LATS:12,RC:1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:true,FN:10,ATS:3,LATS:3,RC:1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:true,FN:20,ATS:2,LATS:4,RC:2] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1]
output	table_cnt	8
output	frame_pool	EMPTY
output	frame_cnt	0
output	int	30

Please refer to Section 3.4 of the Modern Operating Systems book for a detailed discussion of the First-In First-Out algorithm.

count_page_faults_fifo

This function simulates the processing of a sequence of page references in a system that uses the First-In-First-Out (FIFO) policy for page replacement. Specifically, it takes three inputs:

1. process page table
2. sequence of logical page numbers
3. process frame pool.

The function returns the estimated number of page faults for the reference string, with respect to the pool of frames allocated to the process. For each logical page number (in the sequence), the function simulates the processing of the page access in the FIFO system. It keeps track of the number of page faults that occur in the system as it simulates the processing of the entire sequence of logical page numbers.

When simulating the processing of a page access, the function first checks if the page being referenced is already in memory (i.e., the page-table entry has the valid bit true). If so, it modifies the last_access_timestamp and the reference_count fields of the page-table entry. It then moves on to the processing of the next entry in the sequence of logical page numbers.

In order to simulate timestamps, the function starts with a timestamp of 1 and increments it whenever the processing of a new page access is begun.

If the page being referenced is not in memory, the function checks if there are any free frames (i.e., the process frame pool is not empty). If so, a frame is removed from the process frame pool and the frame number is inserted into the page-table entry corresponding to the logical page number. In addition, the other fields of the page-table entry are set appropriately. The function counts this page access as a page fault.

If the page being referenced is not in memory and there are no free frames for the process, a page needs to be replaced. The function selects among all the pages of the process that are currently in memory (i.e., they have valid bits as true) the page that has the smallest arrival_timestamp. It marks that page-table entry as invalid, along with setting the arrival_timestamp, last_access_timestamp and reference_count to -1. It then sets the frame_number of the page-table entry of the newly-referenced page to the newly freed frame. It also sets the arrival_timestamp, the last_access_timestamp and the reference_count fields of the page-table entry appropriately. The function counts this page access as a page fault.

The function returns the total number of page faults encountered in the simulation.

The signature of the method is as follows:

int count_page_faults_fifo(struct PTE page_table[TABLEMAX],int table_cnt, int refrence_string[REFERENCEMAX],int reference_cnt,int frame_pool[POOLMAX],int frame_cnt);

A sample execution input and output:

input/output	parameter	value
input	page_table	[IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1]
input	table_cnt	8
input	reference_string	[0, 3, 2, 6, 3, 4, 5, 2, 4, 5, 6]
input	reference_cnt	11
input	frame_pool	[0, 1, 2]
input	frame_cnt	3
output	faults	8

Please refer to Section 3.4 of the Modern Operating Systems book for a detailed discussion of the First-In First-Out algorithm.

process_page_access_lru

This function implements the logic to process a page access in a system that uses the Least-Recently-Used (LRU) policy for page replacement. Specifically, it takes four inputs:

1. process page table
2. logical page number
3. process frame pool
4. current timestamp.

The function determines the memory frame number for the logical page and returns this number.

First the function checks if the page being referenced is already in memory (i.e., the page-table entry has the valid bit true). If so, it returns the frame number, after modifying the last_access_timestamp and the reference_count fields of the page-table entry.

If the page being referenced is not in memory, the function checks if there are any free frames (i.e., the process frame pool is not empty). If so, a frame is removed from the process frame pool and the frame number is inserted into the page-table entry corresponding to the logical page number. In addition, the other fields of the page-table entry are set appropriately. The function returns the frame number associated with the page-table entry.

If the page being referenced is not in memory and there are no free frames for the process, a page needs to be replaced. The function selects among all the pages of the process that are currently in memory (i.e., they have valid bits as true) the page that has the smallest last_access_timestamp. It marks that page-table entry as invalifd, along with setting the frame_number, arrival_timestamp, last_access_timestamp and reference_count to -1. It then sets the frame_number of the page-table entry of the newly-referenced page to the newly freed frame. It also sets the arrival_timestamp, the last_access_timestamp and the reference_count fields of the page-table entry appropriately. Finally, the function returns this frame number.

The signature of the method is as follows:

```
int process_page_access_lru(struct PTE page_table[TABLEMAX],int *table_cnt, int
page_number, int frame_pool[POOLMAX],int *frame_cnt, int current_timestamp);
```

A sample execution input and output:

input/output	parameter	value
input	page_table	[IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:true,FN:10,ATS:3,LATS:3,RC:1]

		[IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:true,FN:20,ATS:2,LATS:4,RC:2] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:true,FN:30,ATS:1,LATS:1,RC:1]
input	table_cnt	8
input	page_number	0
input	frame_pool	EMPTY
input	frame_cnt	0
input	current_timestamp	12
output	page_table	[IV:true,FN:30,ATS:12,LATS:12,RC:1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:true,FN:10,ATS:3,LATS:3,RC:1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:true,FN:20,ATS:2,LATS:4,RC:2] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1]
output	table_cnt	8
output	int	30

Please refer to Section 3.4 from the Modern Operating Systems book for a detailed discussion of the Least-Recently-Used algorithm.

count_page_faults_lru

This function simulates the processing of a sequence of page references in a system that uses the Least-Recently-Used (LRU) policy for page replacement. Specifically, it takes three inputs:

1. process page table
2. sequence of logical page numbers
3. process frame pool.

The function returns the estimated number of page faults for the reference string, with respect to the pool of frames allocated to the process. For each logical page number (in the sequence), the function simulates the processing of the page access in the LRU system. It keeps track of the number of page faults that occur in the system as it simulates the processing of the entire sequence of logical page numbers.

When simulating the processing of a page access, the function first checks if the page being referenced is already in memory (i.e., the page-table entry has the valid bit true). If so, it modifies the last_access_timestamp and the reference_count fields of the page-table entry. It then moves on to the processing of the next entry in the sequence of logical page numbers.

In order to simulate timestamps, the function starts with a timestamp of 1 and increments it whenever the processing of a new page access is begun.

If the page being referenced is not in memory, the function checks if there are any free frames (i.e., the process frame pool is not empty). If so, a frame is removed from the process frame pool and the frame number is inserted into the page-table entry corresponding to the logical page number. In addition, the other fields of the page-table entry are set appropriately. The function counts this page access as a page fault.

If the page being referenced is not in memory and there are no free frames for the process, a page needs to be replaced. The function selects among all the pages of the process that are currently in memory (i.e., they have valid bits as true) the page that has the smallest last_access_timestamp. It marks that page-table entry as invalid, along with setting the arrival_timestamp, last_access_timestamp and reference_count to 0. It then sets the frame_number of the page-table entry of the newly-referenced page to the newly freed frame. It also sets the arrival_timestamp, the last_access_timestamp and the reference_count fields of the page-table entry appropriately. The function counts this page access as a page fault.

The function returns the total number of page faults encountered in the simulation.

The signature of the method is as follows:

int count_page_faults_lru(struct PTE page_table[TABLEMAX],int table_cnt, int refrence_string[REFERENCEMAX],int reference_cnt,int frame_pool[POOLMAX],int frame_cnt);

A sample execution input and output:

input/output	parameter	value
input	page_table	[IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1]
input	table_cnt	8
input	reference_string	[0, 3, 2, 6, 3, 4, 5, 2, 4, 6, 5]
input	reference_cnt	11
input	frame_pool	[0, 1, 2]
input	frame_cnt	3
output	faults	9

Please refer to Section 3.4 of the Modern Operating Systems book for a detailed discussion of the Least-Recently-Used algorithm.

process_page_access_lfu

This function implements the logic to process a page access in a system that uses the Least-Frequently-Used (LFU, also known as Not-Frequently-Used) policy for page replacement. Specifically, it takes four inputs: 1) process page table, 2) logical page number, 3) process frame pool, and 4) current timestamp.

The function determines the memory frame number for the logical page and returns this number.

First the function checks if the page being referenced is already in memory (i.e., the page-table entry has the valid bit true). If so, it returns the frame number, after modifying the last_access_timestamp and the reference_count fields of the page-table entry.

If the page being referenced is not in memory, the function checks if there are any free frames (i.e., the process frame pool is not empty). If so, a frame is removed from the process frame pool and the frame number is inserted into the page-table entry corresponding to the logical page number. In addition, the other fields of the page-table entry are set appropriately. The function returns the frame number associated with the page-table entry.

If the page being referenced is not in memory and there are no free frames for the process, a page needs to be replaced. The function selects among all the pages of the process that are currently in memory (i.e., they have valid bits as true) the page that has the smallest reference_count. If multiple pages have the smallest reference_count, the one with the smallest arrival_timestamp among these is selected. After selecting the page for replacement, the function marks that page-table entry as invalid, and sets the frame_number, arrival_timestamp, last_access_timestamp and reference_count to -1. It then sets the frame_number of the page-table entry of the newly-referenced page to the newly freed frame. It also sets the arrival_timestamp, the last_access_timestamp and the reference_count fields of the page-table entry appropriately. Finally, the function returns this frame number.

The signature of the method is as follows:

```
int process_page_access_lfu(struct PTE page_table[TABLEMAX],int *table_cnt, int
page_number, int frame_pool[POOLMAX],int *frame_cnt, int current_timestamp);
```

A sample execution input and output:

input/output	parameter	value
input	page_table	[IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:true,FN:10,ATS:3,LATS:3,RC:1]

		[IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:true,FN:20,ATS:2,LATS:4,RC:2] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:true,FN:30,ATS:1,LATS:1,RC:1]
input	table_cnt	8
input	page_number	0
input	frame_pool	EMPTY
input	frame_cnt	0
input	current_timestamp	12
output	page_table	[IV:true,FN:30,ATS:12,LATS:12,RC:1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:true,FN:10,ATS:3,LATS:3,RC:1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:true,FN:20,ATS:2,LATS:4,RC:2] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:0,LATS:0,RC:0]
output	table_cnt	8
output	int	30

Please refer to Section 3.4 of the Modern Operating Systems book for a detailed discussion of the Not-Frequently-Used algorithm.

count_page_faults_lfu

This function simulates the processing of a sequence of page references in a system that uses the Least-Frequently-Used (LFU, also known as Not-Frequently-Used) policy for page replacement. Specifically, it takes three inputs:

1. process page table
2. sequence of logical page numbers
3. process frame pool.

The function returns the estimated number of page faults for the reference string, with respect to the pool of frames allocated to the process. For each logical page number (in the sequence), the function simulates the processing of the page access in the LFU system. It keeps track of the number of page faults that occur in the system as it simulates the processing of the entire sequence of logical page numbers.

When simulating the processing of a page access, the function first checks if the page being referenced is already in memory (i.e., the page-table entry has the valid bit true). If so, it modifies the last_access_timestamp and the reference_count fields of the page-table entry. It then moves on to the processing of the next entry in the sequence of logical page numbers.

In order to simulate timestamps, the function starts with a timestamp of 1 and increments it whenever the processing of a new page access is begun.

If the page being referenced is not in memory, the function checks if there are any free frames (i.e., the process frame pool is not empty). If so, a frame is removed from the process frame pool and the frame number is inserted into the page-table entry corresponding to the logical page number. In addition, the other fields of the page-table entry are set appropriately. The function counts this page access as a page fault.

If the page being referenced is not in memory and there are no free frames for the process, a page needs to be replaced. The function selects among all the pages of the process that are currently in memory (i.e., they have valid bits as true) the page that has the smallest reference_count. If multiple pages exist with the smallest reference_count, the one with the smallest arrival_timestamp is chosen. The function then marks that page-table entry as invalid, along with setting the arrival_timestamp, last_access_timestamp and reference_count to 0. It then sets the frame_number of the page-table entry of the newly-referenced page to the newly freed frame. It also sets the arrival_timestamp, the last_access_timestamp and the reference_count fields of the page-table entry appropriately. The function counts this page access as a page fault.

The function returns the total number of page faults encountered in the simulation.

The signature of the method is as follows:

int count_page_faults_lfu(struct PTE page_table[TABLEMAX],int table_cnt, int refrence_string[REFERENCEMAX],int reference_cnt,int frame_pool[POOLMAX],int frame_cnt);

A sample execution input and output:

input/output	parameter	value
input	page_table	[IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1] [IV:false,FN:-1,ATS:-1,LATS:-1,RC:-1]
input	table_cnt	8
input	reference_string	[0, 3, 2, 6, 3, 4, 5, 2, 6, 4, 5]
input	reference_cnt	11
input	frame_pool	[0, 1, 2]
input	frame_cnt	3
output	faults	10

Please refer to Section 3.4 of the Modern Operating Systems book for a detailed discussion of the Not-Frequently-Used algorithm.