Luis K. Pena
PID: 6300130
COP4610
Summer 2023

Programming Assignment 4

```
/*
 * This program implements a code in C that determines how many page faults
 * will occur in FIFO and LRU replacement policies, as well as tell us the
 * final state of memory when the processes finish.
 *
 * @author Luis Pena (PID 6300130)
 * @version 7/12/2023
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

int mem_size = 0;
int* memory;
int* page_refs;



//Get page references from a file and store them in an array
int get_refs(char* file_name, int mem_size)
{
   int num_refs = 0, page_ref, length;
   FILE *file;
```

```
//Open the file for reading
file = fopen(file_name, "r");

if (file == NULL)
{
   printf("Failed to open the file.\n");
   exit(1);
}

//Move file pointer to the end of the file
fseek(file, 0, SEEK_END);

//Get the position of the file pointer, which is the length of the file
length = ftell(file);

//Move file pointer back to the beginning of the file
fseek(file, 0, SEEK_SET);

//Allocate memory for page references array
page_refs = (int*) malloc((length) * sizeof(int));

//Initialize page references array to -1
for (int i = 0; i < length; ++i)
{
   page_refs[i] = -1;
}

//Read page references from the file
while (fscanf(file, "%d", &page_ref) == 1)
```

```
    {
        page_refs[num_refs++] = page_ref;
    }

    //Close the file
    fclose(file);

    return num_refs;
}

//FIFO page fault counter function
void fifo(char* file_name, int mem_size)
{

        //Instatiate and intialize global variables
        int num_refs = get_refs(file_name, mem_size);
        int page_faults = 0, oldest = 0, page_ref;
        bool found = false;

        //Allocate and intialize  memory
        memory = (int*)malloc(mem_size * sizeof(int));
        memset(memory, -1, mem_size * sizeof(int));

        //Check if page memory is in reference
        for(int i = 0; i < num_refs; i++)
        {
                page_ref = page_refs[i];
                found = false;
```

```
        //Set found to true if page reference found
        for(int j = 0; j < mem_size; j++)
        {
                if(memory[j] == page_ref)
                {
                        found = true;
                        break;
                }
        }

        //Page fault exists
        if(!found)
        {
                memory[oldest] = page_ref;
                oldest = (oldest + 1) % mem_size;
                page_faults += 1;
        }

}

//Print final page fault count
printf("%s: %d %s\n", "FIFO", page_faults, "page faults.");

//Print memory state label
printf("%s: ", "Final Memory State");

//Print numbers for occupied memory spaces
for(int k = 0; k < mem_size; k++)
{
```

```
                if(memory[k] != -1)
                {
                        printf("%d ", memory[k]);
                }
        }

        //Print newline
        printf("\n");

        //Free dynamically allocated memory
        free(memory);
}

//LRU page fault counter function
void lru(char* file_name, int mem_size)
{
        //Instantiate and initialize global variables
        int num_refs = get_refs(file_name, mem_size);
        int page_faults = 0, min_timestamp, min_timestamp_i, page_ref;
        bool found = false;

        //Allocate and initialize memory
        memory = (int*)malloc(mem_size * sizeof(int));
        memset(memory, -1, mem_size * sizeof(int));

        //Allocate memory for timestamps array
        int* timestamps = (int*)malloc(mem_size * sizeof(int));
        memset(timestamps, -1, mem_size * sizeof(int));
```

```
//Check if page memory is in reference
for(int i = 0; i < num_refs; i++)
{
	page_ref = page_refs[i];
	found = false;
	min_timestamp_i = 0;

	//Set found to true if page found and update timestamps
	for(int j = 0; j < mem_size; j++)
	{
		if(memory[j] == page_ref)
		{
			found = true;
			timestamps[j] = i;
			break;
		}
	}

	//Page fault exists
	if(!found)
	{
		min_timestamp = i;
		min_timestamp_i = 0;

		//Find page with minimum timestamp
		for(int k = 1; k < mem_size; k++)
		{
			if(timestamps[k] < min_timestamp)
			{
```

```
                                min_timestamp = timestamps[k];
                                min_timestamp_i = k;
                        }
                }

                //Replace page with minimum timestamp
                memory[min_timestamp_i] = page_ref;
                timestamps[min_timestamp_i] = i;
                page_faults += 1;
        }
}

//Print final page fault count
printf("%s: %d %s\n", "LRU", page_faults, "page faults.");

//Print memory state label
printf("%s: ", "Final Memory State");

for(int l = 0; l < mem_size; l++)
{
        if(memory[l] != -1)
        {
                printf("%d ", memory[l]);
        }
}

//Print newline
printf("\n");
```

```
        //Free dynamically allocated memory
        free(timestamps);
        free(memory);
}

//Main function
int main(int argc, char* argv[])
{
   if (argc != 3)
   {
      printf("Usage: %s pagereffile memorysize\n", argv[0]);
      return 1;
   }

   //Get file name and memory size from command line arguments
   char* file_name = argv[1];
   mem_size = atoi(argv[2]);

   // Check if memory size is within the valid range
   if (mem_size < 1 || mem_size > 10)
   {
      printf("Error: memory size must be between 1 and 10\n");
      return 1;
   }

   //Call page replacement functions
   fifo(file_name, mem_size);
   lru(file_name, mem_size);
```

```
    //Free dynamically allocated memory for page references
    free(page_refs);

    return 0;
}
```