Course: ECE 408 (Operating Systems)

Name: Josh Martin

HMW: 6

Date: 04/25/2019

Introduction

Methodology

```
#define FRAME_SIZE 256
#define TOTAL_NUMBER_OF_FRAMES 128
#define TLB_SIZE 16
#define PAGE_TABLE_SIZE 256
#define ADDRESS_BUFFER_SIZE  10

typedef struct {
    int frame_number;
    int reference_bit;
}Page_table_item;

typedef struct {
    int pageNumber;
    int frameNumber;
} Translation_Lookaside_Buffer;

typedef struct TLB_data {
int hits;//counts TLB hits
    int entries;//counts the number of entries in the TLB
} TLB_data;

typedef struct data_struct {
    FILE *address_file;
    FILE *backing_store;
    TLB_data TLB;
    int faults;// counts page faults

    Page_table_item pageTable[PAGE_TABLE_SIZE];
    Translation_Lookaside_Buffer TLB_table[TLB_SIZE];
    int frame_table[TOTAL_NUMBER_OF_FRAMES]; // TODO: Change name
    signed char physicalMemory[TOTAL_NUMBER_OF_FRAMES][FRAME_SIZE];
    int firstAvailableFrame; //tracks the first available frame
    int firstAvailablePageTableNumber; //tracks the first available page table entry
```

```
} memory;

memory * new_virtual_memory();

#endif //HW5_VIRTUAL_MEMORY_H
void FIFO_algorthim(memory *_this, int pageNumber, int frameNumber) {

    int i;  // if it's already in the TLB, break
    for(i = 0; i < _this->TLB.entries; i++){
        if(_this->TLB_table[i].pageNumber == pageNumber){
            break;
        }
    }

    if(i == _this->TLB.entries){
        if(_this->TLB.entries < TLB_SIZE){
            // insert the page and frame on the end
            _this->TLB_table[_this->TLB.entries].pageNumber = pageNumber;
            _this->TLB_table[_this->TLB.entries].frameNumber = frameNumber;
        }
        else{// otherwise shift everything
            for(i = 0; i < TLB_SIZE - 1; i++){
                _this->TLB_table[i].pageNumber = _this->TLB_table[i + 1].pageNumber;
                _this->TLB_table[i].frameNumber = _this->TLB_table[i +1].frameNumber;
            }
            _this->TLB_table[_this->TLB.entries-1].pageNumber = pageNumber;
            _this->TLB_table[_this->TLB.entries -1 ].frameNumber = frameNumber;
        }
    }
    else{ // index is not <==> to # of entries
        for(; i < _this->TLB.entries - 1; i++){
            _this->TLB_table[i].pageNumber = _this->TLB_table[i+1].pageNumber;
            _this->TLB_table[i].frameNumber = _this->TLB_table[i+1].frameNumber;
        }
    }
}

void setIntoTLB(memory *_this, int pageNumber, int frameNumber) {

    FIFO_algorthim(_this, pageNumber, frameNumber);

    if(_this->TLB.entries < TLB_SIZE){
        _this->TLB.entries++;
    }
}
```

```
void getStore(memory *_this, int pageNumber, int frame_number) {

    if (fseek(_this->backing_store, pageNumber * PAGE_TABLE_SIZE, SEEK_SET) != 0) {
        fprintf(stderr, "Error seeking in backing store\n");
    }

    // load the bits into the first available frame in the physical memory 2D array
    if (fread(_this->physicalMemory[frame_number], sizeof(signed char), PAGE_TABLE_SIZE, _th
        fprintf(stderr, "Error reading from backing store\n");
    }

    _this->TLB_table[_this->TLB.entries].frameNumber = frame_number;
    _this->TLB_table[_this->TLB.entries].pageNumber = pageNumber;

    _this->TLB.entries++;
    _this->TLB.entries %= TLB_SIZE;
}
int updateFramePointer(memory * _this, int frame_number){
    frame_number = _this->firstAvailableFrame++;
    _this->firstAvailableFrame %= TOTAL_NUMBER_OF_FRAMES;

    return frame_number;
}
int second_chance(memory * _this, int frame_number, int pageNumber){
    if(_this->frame_table[frame_number] != -1 ){

        int checkRefBit = _this->frame_table[frame_number];
        while (_this->pageTable[checkRefBit].reference_bit){ // while there is no one to pro
            _this->pageTable[checkRefBit].reference_bit = 0;

            frame_number = updateFramePointer(_this,frame_number);
            checkRefBit = _this->frame_table[checkRefBit];
        }

        // Evict from the LRU
        for (int i = 0; i < TLB_SIZE ; ++i)
            if(_this->TLB_table[i].pageNumber == checkRefBit)
                _this->TLB_table[i].pageNumber = -1;

        _this->pageTable[checkRefBit].frame_number = -1;
        _this->pageTable[checkRefBit].reference_bit = 0;
    }

    _this->frame_table[frame_number] = pageNumber;
```

```
    getStore(_this, pageNumber, frame_number); // gets data from .bin
    _this->pageTable[pageNumber].frame_number = -1;
    _this->pageTable->reference_bit = 0;

    return frame_number;

}

int find_frame(memory * _this, int pageNumber){
    int i,
    frame_number = - 1;

    for(i = 0; i < TLB_SIZE; i++){
        if(_this->TLB_table[i].pageNumber == pageNumber){
            frame_number = _this->TLB_table[i].frameNumber;
            _this->TLB.hits++;

            _this->pageTable[i].reference_bit = 1;
        }
    }

    if(frame_number == -1){ // frame_number not found
        frame_number = _this->pageTable[pageNumber].frame_number;

        if(frame_number == -1){// the page is not found in those contents
            _this->faults++;

            frame_number = updateFramePointer(_this, frame_number);
            frame_number= second_chance(_this, frame_number, pageNumber);


        }
        else{
            _this->pageTable[pageNumber].reference_bit = 1;

            _this->TLB_table[_this->firstAvailablePageTableNumber].frameNumber = frame_numbe
            _this->TLB_table[_this->firstAvailablePageTableNumber].pageNumber = pageNumber;

            _this->firstAvailablePageTableNumber++;
            _this->firstAvailablePageTableNumber %= TLB_SIZE;

        }
    }
    return frame_number;
}
```

```
void getPage(
        memory * _this,
        int logical_address// reads in 32-bit numbers
        ){
    // Only concerned w/  RIGHTMOST 16-bit addresses of logical_address
    // logical_address is broken into 2, 8-bit segments:
    int pageNumber = ((logical_address & 0xFF00)>>8), // 8-bit page number
            offset = (logical_address & 0x00FF), // 8-bit page offset
            frame_number,
            physical_address;

     frame_number = find_frame(_this, pageNumber);

    setIntoTLB(_this, pageNumber, frame_number);
    physical_address = (frame_number << 8) | offset;
    printf("Virtual address: %d"
           " Physical address: %d Value: %d\n",
           logical_address,
           physical_address,
           _this->physicalMemory[frame_number][offset]
    );
}

FILE * openFile(char fileName[100], char read[3]){
    FILE * filPtr;
    if ((filPtr = fopen(fileName, read))  == NULL) {
        fprintf(stderr, "Error (%s): %s \n",fileName, strerror(errno));
        exit(EXIT_FAILURE);
    }
    return filPtr;
}

void print_stats(memory * _this, double total_addresses){

    // calculate and print out the stats
    printf("Number of translated addresses = %.0f\n", total_addresses);

    printf("Page Miss Rate: %.3f\n",_this->faults / total_addresses);
    printf("TLB Hit Rate: %.3f\n", _this->TLB.hits / total_addresses);

}

int main(int argc, char *argv[]) {

    double addresses_seen = 0.0;
```

```
    char address[ADDRESS_BUFFER_SIZE];
     memory * _this = new_virtual_memory();

    if (argc != 2) {
        fprintf(stderr,"Error: wrong number of Arguments passed");
        exit(EXIT_FAILURE);
    }

    _this->backing_store = openFile("BACKING_STORE.bin", "rb");
    _this->address_file = openFile(argv[1], "r");

    while (fgets(address, ADDRESS_BUFFER_SIZE, _this->address_file) != NULL ) {
        getPage(_this, atoi(address)); // get the physical address and value stored at that
        addresses_seen++;  // increment the number of translated addresses
    }

    print_stats(_this, addresses_seen);
    fclose(_this->address_file);
    fclose(_this->backing_store);

    return 0;
}
```

Results

Summary

Appendix