Project Name: **Virtual Memory Management**
Author: **Chengjun Yuan** <cy3yb@virginia.edu>
Time: **Nov.18.2015**

This project contains two code files and a makefile.

```
"p5cyuan.h"
"p5cyuan.c"
"Makefile"
```

In "**p5cyuan.h**", parameters of this virtual memory management are defined and set as below.

```
#define PAGE_SIZE 256                       // the capacity (number of bytes) of one page
#define PAGE_ENTRIES 16                     // the number of entries in page table
#define FRAME_SIZE 256                      // the capacity of one frame in physical memory
#define FRAME_ENTRIES 8                     // the number of frames
#define MEMORY_SIZE (FRAME_SIZE*FRAME_ENTRIES)   // total capacity of physical memory
#define TLB_ENTRIES 4                       // the number of entries in TLB
```

The data of TLB, page table and physical memory are defined by struct data with arrays. The "lastUseTime" is used to save the last used time of entries in TLB or frames in physical memory. The "isUse" denote whether one entry or frame is used or not. The "freeFrames" express the number of free frames in physical memory and is initiated as FRAME_ENTRIES.

```
typedef struct T_L_B{
        unsigned int pageNum[TLB_ENTRIES];
        unsigned int frameNum[TLB_ENTRIES];
        int lastUseTime[TLB_ENTRIES];
        bool inUse[TLB_ENTRIES];
}TLB;

typedef struct PAGE_TABLE{
        unsigned int frameNum[PAGE_ENTRIES];
        bool inUse[PAGE_ENTRIES];
}PageTable;

typedef struct PHYSICAL_MEMORY{
        char memory[FRAME_ENTRIES][FRAME_SIZE];
        unsigned int pageNum[FRAME_ENTRIES];
        int lastUseTime[FRAME_ENTRIES];
        bool inUse[FRAME_ENTRIES];
        int freeFrames;
}PhysicalMemory;
```

In addition, the definitions of all subroutines are presented in "**p5cyuan.h**" as shown below.

```
// initiate data structure of physical memory, page table and TLB.
void initiateMemory(PhysicalMemory *physicalMemory, PageTable *pageTable, TLB *tlb);

// extract page number and offset from virtual address.
unsigned int extractPageNum(unsigned int address);
unsigned int extractOffset(unsigned int address);

// search page number in TLB or page table. If not exist, return -1, else return frame number.
int searchTLB(TLB *tlb, unsigned int pageNum);
int searchPageTable(PageTable *pageTable, unsigned int pageNum);

// insert page number & frame number pair into TLB or page table.
void insertTLB(TLB *tlb, unsigned int pageNum, unsigned int frameNum);
void insertPageTable(PageTable *pageTable, unsigned int pageNum, unsigned int frameNum);

// update the last used time of one frame in physical memory.
void updatePhysicalMemoryLastUseTime(PhysicalMemory *physicalMemory, unsigned int frameNum);

// allocate free frame in physical memory, return frame number.
unsigned int allocateFrameNum(PhysicalMemory *physicalMemory, bool *replacement);

// load page data from BACKING_STORE.bin and save it in frame.
void loadPageDataFromBackingStore(PhysicalMemory *physicalMemory, unsigned int pageNum,
unsigned int frameNum);

// output final results on screen.
void outputStatisticResults(PhysicalMemory *physicalMemory, PageTable *pageTable, int
numTLBHits, int numPageFaults);
```

In "**p5cyuan.c**", the flow of the main function are introduced here. First, variables are defined.

```
FILE *fp,*fpWrite;   // file pointer for reading "addresses.txt" and writing "result.txt".
int numTLBHits=0,numPageFaults=0;  // variables for recording TLB hits and page faults.

// variables for virtual address, page number, frame number, offset, replaced page number and physical address.
unsigned int address, pageNum, frameNum, offset, replacedPageNum, physicalAddress;

bool replacement=false;   // denote whether a page in one frame is replaced.
PhysicalMemory physicalMemory;
TLB tlb;
PageTable pageTable;
```

Second, initiate the data structure or memory of the physical memory, page table, TLB. Their values are all set to zero and marked as free except the number of free frames in physical memory is initiated as FRAME_ENTRIES. The input file "**addresses.txt**" and output file "**result.txt**" are opened.

The virtual memory address, physical memory address and the value stored in it are output in "**result.txt**" as the form like "*Virtual address: 16916 Physical address: 20 Value: 0*".

```
initiateMemory(&physicalMemory,&pageTable,&tlb);
fp=fopen("addresses.txt","r");
fpWrite=fopen("result.txt","w");
```

Third, while loop for reading virtual address from "**addresses.txt**", and extract the page number and offset. The page number is firstly searched in TLB, if not exist, then search the page table. If still not exist, then allocate a new frame and read the page from "BACKING_STORE.bin".

```
while((!feof(fp))&&numAddresses<1000){
        fscanf(fp,"%u\n",&address);
        pageNum=extractPageNum(address);
        offset=extractOffset(address);
        if(DEBUG)printf("read virtual address %u pageNum %u offset %u\n",address,pageNum,offset);
        frameNum=searchTLB(&tlb,pageNum);
        if(frameNum==-1){
                frameNum=searchPageTable(&pageTable,pageNum);
                if(frameNum==-1){
                        numPageFaults++;
        printf("virtual address %u contained in page %u causes a page fault \n",address,pageNum);
                        frameNum=allocateFrameNum(&physicalMemory,&replacement);
                        printf("page %u is loaded into frame %u \n",pageNum,frameNum);
                        if(replacement){
                                replacedPageNum=physicalMemory.pageNum[frameNum];
                                pageTable.inUse[replacedPageNum]=false;
                        }
                        physicalMemory.pageNum[frameNum]=pageNum;
                        insertPageTable(&pageTable,pageNum,frameNum);
                    loadPageDataFromBackingStore(&physicalMemory,pageNum,frameNum);
                }
                updatePhysicalMemoryLastUseTime(&physicalMemory,frameNum);
                insertTLB(&tlb,pageNum,frameNum);
        }else{
                updatePhysicalMemoryLastUseTime(&physicalMemory,frameNum);
                numTLBHits++;
        }
        fprintf(fpWrite,"Virtual address: %u Physical address: %u Value: %d
\n",address,((frameNum<<8)|offset),physicalMemory.memory[frameNum][offset]);
        numAddresses++;
}
fclose(fpWrite);
fclose(fp);
outputStatisticResults(&physicalMemory,&pageTable,numTLBHits,numPageFaults);
```

The LRU replacement algorithm is implemented in subroutine “insertTLB” when there are no free entries in TLB, and in subroutine “allocateFrameNum” when there are no free frames in physical memory. The codes will traverse the variables “lastUseTime” for each TLB entries or physical frames to figure out the longest used one and then replace it. The codes are shown below.

```
// LRU replacement algorithm for physical memory.
*replacement=true;
for(i=0;i<FRAME_ENTRIES;i++){
        if(numAddresses-physicalMemory->lastUseTime[i]>longestUseTime){
                longestUseTime=numAddresses-physicalMemory->lastUseTime[i];
                longestUseFrame=i;
        }
}
physicalMemory->lastUseTime[longestUseFrame]=numAddresses;
physicalMemory->inUse[longestUseFrame]=true;
return longestUseFrame;

// LRU replacement algorithm for TLB.
for(i=0;i<TLB_ENTRIES;i++){
        if(numAddresses-tlb->lastUseTime[i]>longestUseTime){
                longestUseTime=numAddresses-tlb->lastUseTime[i];
                longestUseTLB=i;
        }
}
tlb->pageNum[longestUseTLB]=pageNum;
tlb->frameNum[longestUseTLB]=frameNum;
tlb->lastUseTime[longestUseTLB]=numAddresses;
printf("frame %u containing page %u is stored in entry %d of the TLB \n",frameNum,pageNum,longestUseTLB);
```