/\*----------------------------------------------------------------------------

Meryl Mabin

CSE 410, section 001

proj04

---------------------------------------------------------------------------\*/

using namespace std;

#include <iostream>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <iostream>

#include <vector>

#include <sstream>

#include <stdio.h>

#include "reference.h"

struct page

{

int accessTime,

pageNum,

operation,

procID,

age;

bool useBit,

modBit;

};

int main( int Argc, char \*Arg[] )

{

string command,

num,

algorithm = "LRU";

int pageSize = 8, // args

numFrames = 32,

maxTLBsize = 0,

numProcesses = 1,

numRefs = 200,

integer = 0,

oldPageIndex,

currID,

numFaults = 0,

clockIndex = 0,

counter = 0,

clockHere = -1;

vector<page> TLB, // Always LRU

mainMem;

float refTime = 0;

struct ref theRef;

page currPage;

bool TLBflag = 0,

loop1Flag = 0,

loop2Flag = 0,

mem\_flag = 0;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Parse the command line

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for ( int i=1; i<Argc; i+=2 )

{

command = Arg[i];

// PAGE SIZE

if ( command == "-p" )

{

if ( i+1 < Argc )

{

num = Arg[i+1];

if ( isdigit(num[0]) )

{

pageSize = atoi( num.c\_str() );

}

else

{

cout << "Invalid page size.\n";

exit(1);

}

}

else

{

cout << "Missing a parameter.\n";

exit(1);

}

}

// FRAME NUMBER

else if ( command == "-f" )

{

if ( i+1 < Argc )

{

num = Arg[i+1];

if ( isdigit(num[0]) )

{

numFrames = atoi( num.c\_str() );

}

else

{

cout << "Invalid number of frames.\n";

exit(1);

}

}

else

{

cout << "Missing a parameter.\n";

exit(1);

}

}

// ALGORITHM

else if ( command == "-a" )

{

if ( i+1 < Argc )

{

num = Arg[i+1];

if (num == "LRU" || num == "FIFO" || num == "CLOCK")

{

algorithm = num;

}

else

{

cout << "Invalid algorithm.\n";

exit(1);

}

}

else

{

cout << "Missing a parameter.\n";

exit(1);

}

}

// TLB SIZE

else if ( command == "-t" )

{

if ( i+1 < Argc )

{

num = Arg[i+1];

if ( isdigit(num[0]) )

{

maxTLBsize = atoi( num.c\_str() );

}

else

{

cout << "Invalid size of TLB.\n";

exit(1);

}

}

else

{

cout << "Missing a parameter.\n";

exit(1);

}

}

// NUMBER OF PROCESSES

else if ( command == "-P" )

{

if ( i+1 < Argc )

{

num = Arg[i+1];

if ( isdigit(num[0]) )

{

numProcesses = atoi( num.c\_str() );

}

else

{

cout << "Invalid number of processes.\n";

exit(1);

}

}

else

{

cout << "Missing a parameter.\n";

exit(1);

}

}

// NUMBER OF REFERENCES

else if ( command == "-l" )

{

if ( i+1 < Argc )

{

num = Arg[i+1];

if ( isdigit(num[0]) )

{

numRefs = atoi( num.c\_str() );

}

else

{

cout << "Invalid number of references.\n";

exit(1);

}

}

else

{

cout << "Missing a parameter.\n";

exit(1);

}

}

// DATA

else if ( command == "-d" )

{

if ( i+1 < Argc )

{

num = Arg[i+1];

if ( isdigit(num[0]) )

{

integer = atoi( num.c\_str() );

}

else

{

cout << "Invalid -d integer.\n";

exit(1);

}

}

else

{

cout << "Invalid command.\n";

exit(1);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Paging stuff

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

refTime = 0;

for ( int i=0; i<numRefs; i++ )

{

TLBflag = 0;

mem\_flag = 0;

currPage.age = refTime;

currPage.useBit = 1;

// Ask for the reference

reference( &theRef, numProcesses, integer );

// Get page number

currPage.pageNum = theRef.address >> pageSize;

currPage.modBit = theRef.operation;

// If process ID changed from the last one, TLB should be cleared

if ( theRef.processid != currID ) TLB.clear();

currID = theRef.processid;

currPage.procID = theRef.processid;

// Check TLB for the page

for ( unsigned j=0; j<TLB.size(); j++ )

{

if ( currPage.pageNum == TLB[j].pageNum )

// Page is in TLB, so compute access time

{

refTime += .000000030;

currPage.age = refTime;

TLBflag = 1;

break;

}

}

if ( !TLBflag )

{

// If not in TLB, fault

if ( maxTLBsize != 0 )

{

fault(0);

}

// check main memory.

for ( unsigned j=0; j<mainMem.size(); j++ )

{

if ( currPage.pageNum == mainMem[j].pageNum )

{

// Found the page in the main memory. AWESOME.

mem\_flag = 1;

// Add time (one for accessing in RAM, one for TLB)

refTime += .000000030 + .000000120;

currPage.age = refTime;

// If the TLB is not full, move page to TLB

if ( TLB.size() < maxTLBsize )

{

TLB.push\_back( currPage );

}

else if ( TLB.size() > 0 )

{

// If TLB is full, do LRU replacement

oldPageIndex = 0;

for ( unsigned k=1; k<TLB.size(); k++ )

{

if ( TLB[k].age < TLB[oldPageIndex].age )

{

oldPageIndex = k;

}

} // end for

// Replace the oldest page in TLB with the current one

TLB[oldPageIndex] = currPage;

} // end else

break;

} // end if

} // end for

if ( !mem\_flag)

{

// Didn't find the page in main memory, so we had to find it on disk

fault(1);

numFaults ++;

refTime += 0.035 + .000000030 + .000000120;

// Move the page to the TLB and to main memory

// If the TLB is not full, move page to TLB

if ( TLB.size() < maxTLBsize ) TLB.push\_back( currPage );

else if ( TLB.size() > 0 )

{

// If TLB is full, do LRU replacement

oldPageIndex = 0;

for ( unsigned k=0; k<TLB.size(); k++ )

{

if ( TLB[k].age < TLB[oldPageIndex].age )

{

oldPageIndex = k;

}

} // end for

// Replace the oldest page in TLB with the current one

TLB[oldPageIndex] = currPage;

} // end else

// If main memory is not full, move page to main memory

if ( (mainMem.size() < numFrames) && (algorithm != "CLOCK") )

{

mainMem.push\_back( currPage );

}

else if ( (mainMem.size() > 0) && (algorithm != "CLOCK") )

{

// If main memory is full, use the specified replacement algorithm

if ( algorithm == "LRU" )

{

// Least recently used

oldPageIndex = 0;

for ( unsigned k=1; k<mainMem.size(); k++ )

{

if ( mainMem[k].age < mainMem[oldPageIndex].age )

{

oldPageIndex = k;

}

} // end for

// Replace the oldest page in main memory with the current one

mainMem[oldPageIndex] = currPage;

} // end if LRU

else if ( algorithm == "FIFO" )

{

// First-in first-out

// Erase the first element, and add new one to the end

mainMem.erase( mainMem.begin() );

mainMem.push\_back( currPage );

} // end if FIFO

}

else if ( algorithm == "CLOCK" )

{

clockHere = -1;

// check to see if the page is in memory

for ( unsigned l=0; l<mainMem.size(); l++ )

{

if ( mainMem[l].pageNum == currPage.pageNum )

{

mainMem[l].useBit = 1;

clockHere = l;

break;

}

}

// if the page isn't in mem and mem isn't full, put it in mem

// and set its use bit to 1

if ( (clockHere == -1) && mainMem.size() < numFrames )

{

currPage.useBit = 1;

mainMem.push\_back( currPage );

// advance the clock pointer

if ( clockIndex == numFrames - 1 )

{

clockIndex = 0;

}

else

{

clockIndex += 1;

}

}

// else if the page isn't in mem and mem is full, loop through

// mem, looking for a page that has its use bit set to 0 and

// changing 1's to 0's

else if ( (clockHere == -1) && mainMem.size() == numFrames )

{

while (true)

{

counter = 0;

while ( counter < mainMem.size() )

{

// Look for a page with use and mod bits = 0

if ( mainMem[clockIndex].useBit == 0 && mainMem[clockIndex].modBit == 0 )

{

mainMem[clockIndex] = currPage;

loop1Flag = 1;

break;

}

if ( clockIndex == numFrames - 1 )

{

clockIndex = 0;

}

else

{

clockIndex += 1;

}

counter++;

}

if ( !loop1Flag )

{

counter = 0;

while ( counter < mainMem.size() )

{

if ( mainMem[clockIndex].useBit == 0 && mainMem[clockIndex].modBit == 1 )

{

mainMem[clockIndex] = currPage;

loop2Flag = 1;

break;

}

mainMem[clockIndex].useBit = 0;

// advance clock pointer

if ( clockIndex == numFrames - 1 )

{

clockIndex = 0;

}

else

{

clockIndex += 1;

}

counter ++;

}

}

if (loop1Flag || loop2Flag) break;

}

}

} // end if CLOCK

}

}

if ( i%50 == 0 )

{

printf("\n%6s %6s %6s %6s\n", "proc#", "page#", "mod", "use");

for ( unsigned x=0; x<mainMem.size(); x++ )

{

printf("%6d %6d %6d %6d\n", mainMem[x].procID, mainMem[x].pageNum, mainMem[x].modBit, mainMem[x].useBit );

}

}

// Printing final information

if ( i == numRefs - 1 )

{

cout << "Total Time: " << refTime << endl;

cout << "Total Page Faults: " << numFaults << endl;

cout << "Hit Rate: " << (1 - (float(numFaults) / float(numRefs))) << endl;

}

} // end for

}