```
//
//  main.cpp
//  SO - TF Memoria
//
//  Created by Victor Manuel Cruz Reyes on 11/27/17.
//  Copyright © 2017 Victor Manuel Cruz Reyes. All rights reserved.
//

#include <iostream>
#include <cmath>
#include <deque>
#include <fstream>
using namespace std;

////////////////////////// Variables Globales ////////////////////////
//Matriz como Memoria Principal
int memoriaPrincipal[128][2] = {0};
//Matriz de Memoria Secundaria
int memoriaSecundaria[256][2] = {0};
//Contador de Memoria (Basado en Marcos)
int marcosLibres = 128;
//Fila Prioridad
deque<int> filaPrioridad;
//Tipo de Algoritmo a usar
string algoritmo = " ";
//Contador de Tiempo
//Cargar Pagina, Swapp IN/OUT = +1 | Accesar/Modificar, Liberar = .1
double tiempoGlobal = 0.0;
//Contador de Swapps
int swappingGlobal = 0;
//Matriz Info Procesos
//Nombre | Tamaño | Tiempo de Entrada | Tiempo de Salida | Page Faults
double matrizProcesos[256][5] = {0};
int cantidadProcesos = 0;
////////////////////////// Variables Globales ////////////////////////


////////////////////////// Imprimir Estructuras ////////////////////////
//Imprimir Memoria Principal
void imprimirMemoriaPrincipal () {
    cout<<"Memoria Principal"<<endl;
    for (int c = 0; c<128; c++) {
        for (int x = 0; x<2; x++) {
            cout<<memoriaPrincipal[c][x]<<" ";
        } cout<<endl;
    } cout<<endl;
}

//Imprimir Memoria Secundaria
//Tamaño = 256
void imprimirMemoriaSecundaria () {
    cout<<"Memoria Secundaria"<<endl;
    for (int c = 0; c<256; c++) {
        for (int x = 0; x<2; x++) {
```

```
            cout<<memoriaSecundaria[c][x]<<" ";
        } cout<<endl;
    } cout<<endl;
}

//Imprimir Matriz Procesos
void imprimirMatrizProcesos () {
    cout<<"Matriz Procesos"<<endl;
    for (int c = 0; c<256; c++) {
        for (int x = 0; x<5; x++) {
            cout<<matrizProcesos[c][x]<<" ";
        } cout<<endl;
    } cout<<endl;
}

//Imprimir Fila
void imprimirFila () {
    cout<<"Fila"<<endl;
    for (int c = 0; c<filaPrioridad.size(); c++) {
        cout<<filaPrioridad[c]<<" ";
    } cout<<endl;
}
////////////////////////// Imprimir Estructuras //////////////////////////


////////////////////////// Reiniciar Estructuras ////////////////////////////////
//Borrar Memorias para nueva corrida
//Falta borrar fila
void borrarMemorias () {
    //Memoria Principal
    for (int c = 0; c<128; c++) {
        for (int x = 0; x<2; x++) {
            memoriaPrincipal[c][x]=0;
        }
    }
    //Memoria Secundaria
    for (int c = 0; c<256; c++) {
        for (int x = 0; x<2; x++) {
            memoriaSecundaria[c][x]=0;
        }
    }
    //Matriz Procesos
    for (int c = 0; c<256; c++) {
        for (int x = 0; x<5; x++) {
            matrizProcesos[c][x]=0;
        }
    }
    //Fila
    while (filaPrioridad.size()>0) {
        filaPrioridad.pop_front();
    }
    //Marcos Libres
    marcosLibres = 128;
    //Tiempo Global
```

```
    tiempoGlobal = 0.0;
    //Swapping Global
    swappingGlobal = 0;
    //Cantidad de Procesos
    cantidadProcesos = 0;
}
////////////////////////// Reiniciar Estructuras ////////////////////////


////////////////////////// Verificacion de Procesos ////////////////////////
//Se registra un nuevo Proceso en la Matriz de Procesos
void nuevoProceso(int nombreP, int numPagP) {
    matrizProcesos[cantidadProcesos][0]=nombreP;
    matrizProcesos[cantidadProcesos][1]=numPagP;
    matrizProcesos[cantidadProcesos][2]=tiempoGlobal;
    cantidadProcesos++;
}


//Funcion que Verifica el Nuevo Proceso a Ingresar
//Recibe Nombre y Tamaño en Numero de Paginas
//Regresa la verificacion
bool verificarProcesoNuevo(int nombreP, int tamPagP) {
    bool verificacion = true;
    //Verificar si habia sido cargado antes
    for (int c = 0; c<256; c++) {
        if (matrizProcesos[c][0]==nombreP) {
            cout<<"El proceso con ese nombre ya existe."<<endl;
            cout<<"Se rechaza."<<endl;
            verificacion = false;
            //Tamaño 256
            //Por razones de optimizacion, sale rapido del Ciclo
            c=256;
        }
    }
    //Verificar el tamaño maximo = 128 Paginas / minimo = 1 Pagina
    if (tamPagP>128) {
        if (tamPagP<1) {
            cout<<"El proceso es menor al debido (1 Bytes)."<<endl;
            cout<<"Se rechaza."<<endl;
        }
        cout<<"El proceso es mayor al debido (2048 Bytes)."<<endl;
        cout<<"Se rechaza."<<endl;
        verificacion = false;
    }
    //Si es valido, se registra el nuevo proceso en la Matriz de Procesos
    if (verificacion) {
        nuevoProceso(nombreP, tamPagP);
    }
    return verificacion;
}

//Funcion que Verifica Procesos Existentes
//Recibe Nombre y Tamaño en Numero de Paginas
```

```
//Regresa la verificacion
bool verificarProceso(int nombreP, int tamPagP) {
    int direccion = 0;
    bool verificacion = false;
    //Busca el Proceso en la Matriz de Procesos
    for (int c = 0; c<256; c++) {
        if (matrizProcesos[c][0]==nombreP) {
            direccion = c;
            verificacion = true;
            //Por razones de optimizacion, sale rapido del Ciclo
            c = 256;
        }
    }
    //El proceso existe
    if (verificacion == true) {
        //Si el tamaño es mayor al Proceso
        if (matrizProcesos[direccion][1]<tamPagP) {
            cout<<"La direccion es mayor al tamaño del proceso."<<endl;
            cout<<"Se rechaza."<<endl;
            verificacion = false;
            return verificacion;
        }
        //Si el proceso ya habia terminado
        if (matrizProcesos[direccion][3]>0) {
            cout<<"El proceso ya habia sido liberado."<<endl;
            cout<<"Se rechaza."<<endl;
            verificacion = false;
            return verificacion;
        }
    } else {
        //El proceso no existe
        cout<<"El proceso no existe"<<endl;
        cout<<"Se rechaza."<<endl;
    }

    return verificacion;
}
////////////////////////// Verificacion de Procesos //////////////////////////


////////////////////////// Algoritmo LRU //////////////////////////
//Funcion que modifica la Fila en funcion al Algoritmo de Remplazo Least Recent Used
//Recibe el marco en Memoria Principal que se acceso
//Borra el marco de la fila y lo ingresa al final de esta
void filaLRU(int marco) {
    for (int c = 0; c<filaPrioridad.size(); c++) {
        if (filaPrioridad[c]==marco) {
            filaPrioridad.erase(filaPrioridad.begin()+c);
            filaPrioridad.push_back(c);
            //Por razones de optimizacion, sale rapido del Ciclo
            c=128;
        }
    }
}
```

```
////////////////////////// Algoritmo LRU //////////////////////////

////////////////////////// Estadisticas //////////////////////////
void estadisticasProcesos() {
    double promTurnAround = 0.0;
    int contador = 0;
    //Turnaround Time de Cada Proceso
    printf("TurnAround por Proceso:\n");
    for (int c = 0; c<256; c++) {
        if (matrizProcesos[c][3]>0) {
            promTurnAround += matrizProcesos[c][3];
            contador++;
            printf("Proceso %.0f = %.3f \n", matrizProcesos[c][0], matrizProcesos[c]
             [3]);
        }
    }
    //Turnaround promedio
    printf("Turnaround promedio = %.3f \n", promTurnAround/contador);
    //Page Faults por Proceso
    printf("Page Faults por Proceso: \n");
    for (int c = 0; c<256; c++) {
        if (matrizProcesos[c][4]>0) {
            printf("Proceso %.0f = %.0f \n", matrizProcesos[c][0], matrizProcesos[c]
             [4]);
        }
    }
    //Numero de Swaps In y Swaps Out
    printf("Numero total de Swaps = %d \n", swappingGlobal);
}
////////////////////////// Estadisticas //////////////////////////

////////////////////////// Liberar Proceso //////////////////////////
//Funcion responsable de liberar un Proceso
//Recibe el Nombre del Proceso
//Verifica que el Proceso exista y no haya salido
//Saca el proceso de: Fila, Memoria Principal y Memoria Secundaria
//Se registra Turnaround-Time
void sacarProceso(int nombreP) {
    //Output (Input de proceso)
    printf("L %d \n", nombreP);
    //Si el proceso existe y no ha salido
    if (verificarProceso(nombreP, 0)) {
        //Borrar de la Fila
        int x = 0;
        int posicion = 0;
        while (x!=filaPrioridad.size()) {
            posicion = filaPrioridad[x];
            if (memoriaPrincipal[posicion][0]==nombreP) {
                filaPrioridad.erase(filaPrioridad.begin() + x);
            }
            else {
                x++;
            }
        }
```

```
        //Borrar de la Memora Principal
        printf("Memoria Principal:\n");
        for (int c = 0; c<128; c++) {
            if (memoriaPrincipal[c][0]==nombreP) {
                memoriaPrincipal[c][0]=0;
                memoriaPrincipal[c][1]=0;
                //Se liberan Marcos
                marcosLibres++;
                //Tiempo Global + 0.1 (Liberar Pagina)
                tiempoGlobal += 0.1;
                //Output Marco liberado
                printf("Marco %d liberado\n", c);
            }
        }
        //Borrar de la Memora Secundaria
        printf("Memoria Secundaria:\n");
        for (int c = 0; c<256; c++) {
            if (memoriaSecundaria[c][0]==nombreP) {
                memoriaSecundaria[c][0]=0;
                memoriaSecundaria[c][1]=0;
                //Tiempo Global + 0.1 (Liberar Pagina)
                tiempoGlobal += 0.1;
                //Output Marco liberado
                printf("Marco %d liberado\n", c);
            }
        }
        //Generar Turnaround Time
        for (int c = 0; c<256; c++) {
            if (nombreP==matrizProcesos[c][0]) {
                matrizProcesos[c][3]=tiempoGlobal;
                //Por razones de optimizacion, sale rapido del Ciclo
                c=256;
            }
        }
    }
}
////////////////////////// Liberar Proceso //////////////////////////

////////////////////////// Funciones Swapping //////////////////////////
//Funcion que registra e informa de un Page Fault
//Recibe el nombre del Proceso Implicado
void pageFault(int nombreP) {
    //Busca el proceso en la Matriz de Procesos
    for (int c = 0; c<256; c++) {
        if (nombreP==matrizProcesos[c][0]) {
            matrizProcesos[c][4]+=1;
            cout<<"Page Fault: Pagina no esta en Memoria Principal."<<endl;
            cout<<"Se procede a buscarla en Memoria Secundaria."<<endl;
            //Por razones de optimizacion, sale rapido del Ciclo
            c=256;
        }
    }
}
```

```
//Swapping In
//Funcion que Mueve Pagina a Memoria Secundaria de Primaria
//Libera posicion en Memoria Principal
//Recibe la Posicion en Memoria Principal del Proceso a Salir
void swappingIn(int posicionFuera) {
    //Verifica espacios libres en Memoria Secundaria
    bool hayMemoria = false;
    //Nombre y Pagina del Marco a Sacar
    int nombre = memoriaPrincipal[posicionFuera][0];
    int pagina = memoriaPrincipal[posicionFuera][1];
    //Output de la Pagina y Proceso a Salir
    printf("Sale pagina %d del Proceso %d \n", pagina, nombre);
    for (int c = 0; c<256; c++) {
        if (memoriaSecundaria[c][0]==0) {
            hayMemoria = true;
            //Output de Direccion en Memoria Secundaria del Proceso que Salio
            printf("Posicion en Memoria Secundaria: %d (Swapp In)\n", c);
            //Proceso entra en Memoria Secundaria
            memoriaSecundaria[c][0]=memoriaPrincipal[posicionFuera][0];
            memoriaSecundaria[c][1]=memoriaPrincipal[posicionFuera][1];
            //Tiempo Global + 1 (Cargar Pagina)
            tiempoGlobal += 1;
            //Swapping Global + 1 (Swapp In)
            swappingGlobal++;
            //Por razones de optimizacion, sale rapido del Ciclo
            c=256;
        }
    }
    if (hayMemoria == false) {
        cout<<"Error critico: No hay memoria Secundaria."<<endl;
    }
}

//¿2 o 3 segundos? Swapp out, Swapp in, ¿+cargar?
//Swapping Out
//Funcion encargada de sacar Proceso de Memoria Secundaria e ingresarlo a Memoria
 Principal
//Recibe Nompre y Marco del Proceso
void swappingOut(int nombreP, int marcoP, int direccionP) {
    bool enMemoriaSecundaria = false;
    int posicionFuera = 0;
    //Lo busca en Memoria Secundaria
    for (int c = 0; c<256; c++) {
        if (memoriaSecundaria[c][0]==nombreP) {
            if (memoriaSecundaria[c][1]==marcoP) {
                //Output (Direccion Memoria Secundaria)
                printf("-Pagina %d en Posicion %d de Memoria Secundaria \n", marcoP,
                 c);
                //Borrar de memoria secundaria
                memoriaSecundaria[c][0] = 0;
                memoriaSecundaria[c][1] = 0;
                enMemoriaSecundaria = true;
                //Tamaño 33
                //Por razones de optimizacion, sale rapido del Ciclo
```

```
                c=256;
            }
        }
    }
    //Mete en Memoria Principal si esta en Memoria Secundaria
    if (enMemoriaSecundaria) {
        //Si hay espacio libre
        if (marcosLibres>0) {
            for (int c=0; c<128; c++) {
                //Encuentra el Espacio Libre
                if (memoriaPrincipal[c][0] == 0) {
                    //Se ingresa a la Fila el # de posicion
                    filaPrioridad.push_back(c);
                    //Se ingresa el proceso a Memoria Principal
                    memoriaPrincipal[c][0] = nombreP;
                    memoriaPrincipal[c][1] = marcoP;
                    //Posicion
                    posicionFuera = c;
                    //Se actualizan los Marcos Disponibles
                    marcosLibres--;
                    //Por razones de optimizacion, sale rapido del Ciclo
                    c = 17;
                }
            }
        } else {
            //Saca proceso de acuerdo a prioridad
            posicionFuera = filaPrioridad.front();
            cout<<posicionFuera<<endl;
            filaPrioridad.pop_front();
            //Swapping
            swappingIn(posicionFuera);
            //Ingresa Pagina en Memoria Principal
            memoriaPrincipal[posicionFuera][0] = nombreP;
            memoriaPrincipal[posicionFuera][1] = marcoP;
            filaPrioridad.push_back(posicionFuera);
            //Tiempo Global + 1 (Cargar a Pagina/Swapping)
            tiempoGlobal+=1;
        }
        //Output Pagina a Ingresar y Numero de Marco en Memoria Principal
        printf("-Pagina %d en Marco %d \n", marcoP, posicionFuera);
        //Swapping Global + 1 (Swapp Out)
        swappingGlobal++;
        //Se accede en la nueva direccion
        //Output (Direcciones Reales y Virtuales)
        printf("Direccion Virtual: %d  Direccion Real: %d \n", direccionP,
         (posicionFuera*16)+(direccionP%16));
        //Tiempo Global + 0.1 (Acceder a Pagina)
        tiempoGlobal += 0.1;
    }
    else {
        cout<<"Error: La pagina existe, pero no se encontro en ninguna
         memoria."<<endl;
    }
}
```

```
/////////////////////////// Funciones Swapping ////////////////////////


/////////////////////////// Funcion Accesar ///////////////////////////
//ADPM = Direccion Virtual | Proceso | lee o se modifica
//Funcion Accesar Proceso (Leer o modificar)
//Verifica el Tamaño, Existencia y Terminacion del Proceso
//Si esta en Memoira principal lo accede sino, lo busca en Memoria Secundaria
//La lista de prioridades se modifica conforme al Algoritmo a usar.
//Recibe nombre del Proceso, Direccion y Algoritmo
//Regresa Output informativo
void tocaProceso(int nombreP, int direccionP, int accionP, string algoritmo) {
    //Output (Input de proceso)
    printf("A %d %d %d\n",direccionP, nombreP, accionP);
    //Direccion Real a Marco
    int numeroMarco = direccionP/16;
    //Verifica al proceso
    if (verificarProceso(nombreP, numeroMarco)) {
        int direccionAux = 0;
        bool estaMemoriaPrincipal = false;
        //La pagina del Proceso se busca en Memoria Principal
        for (int c = 0; c<128; c++) {
            if (memoriaPrincipal[c][0]==nombreP) {
                if (memoriaPrincipal[c][1]==numeroMarco) {
                    estaMemoriaPrincipal = true;
                    direccionAux = c;
                    //Por razones de optimizacion, sale rapido del Ciclo
                    c=128;
                }
            }
        }
        //Si esta en Memoria Principal
        if (estaMemoriaPrincipal) {
            //Tiempo Global + 0.1 (Acceder a Pagina)
            tiempoGlobal += 0.1;
            //Si se usa el Algoritmo LRU se modifica la fila
            if (algoritmo == "LRU") {
                filaLRU(direccionAux);
            }
            //Output (Direcciones Reales y Virtuales
            printf("Direccion Virtual: %d  Direccion Real: %d \n", direccionP,
             direccionAux*16+direccionP%16);
        }
        else {
            //No esta en Memoria Principal
            //Se genera un Page Fault
            pageFault(nombreP);
            //Lo busca en Memoria Secundaria
            swappingOut(nombreP, numeroMarco, direccionP);
        }
    }
}
/////////////////////////// Funcion Accesar ///////////////////////////
```

```
////////////////////////// Funcion Cargar ////////////////////////
//Algoritmo de Remplazo
//Para FIFO y LRU
//Recibe Nombre, Tamaño en Paginas, Auxiliar Numero de Pagina (Por referencia)
//Ingresa pagina liberando un marco en Memoria Principal (Swapping In)
void buscaMeteOcupadas(int nombreP, int tamPagP, int &auxNumPagP) {
    //Donde estaba el proceso a Salir
    int posicionFuera;
    while (auxNumPagP < tamPagP) {
        //Saca de Pagina de acuerdo a la Prioridad
        posicionFuera = filaPrioridad.front();
        filaPrioridad.pop_front();
        //Funcion Swapp In
        swappingIn(posicionFuera);
        //Ingresa pagina en Memoria Principal
        memoriaPrincipal[posicionFuera][0] = nombreP;
        memoriaPrincipal[posicionFuera][1] = auxNumPagP;
        filaPrioridad.push_back(posicionFuera);
        //Output Pagina a Ingresar y Numero de Marco en Memoria Principal
        printf("-Pagina %d en Marco %d \n", auxNumPagP, posicionFuera);
        //Tiempo Global + 1 (Cargar Pagina)
        tiempoGlobal += 1;
        //Se actualiza el auxiliar Numero de Pagina
        auxNumPagP++;
    }
    //Se notifica si se ingreso o no todo el proceso
    if (tamPagP == auxNumPagP) {
        printf("Se ingresaron todas las Paginas en Memoria Principal\n");
    } else {
        printf("Error: No se ingresaron las paginas desde %d hasta %d", auxNumPagP,
         tamPagP);
    }
}

//Tamaño 128
//Se puede optimizar con una fila
//Busca espacios libre e ingresa Paginas
//Recibe Nombre, Tamaño en Paginas, Auxiliar Numero de Pagina (Por referencia)
//Regresa Notificacion
void buscaMeteLibres(int nombreP, int tamPagP,int &auxNumPagP) {
    //Recorre Toda la Memoria
    for (int c=0; c<128; c++) {
        //Encuentra el Espacio Libre
        if (memoriaPrincipal[c][0] == 0) {
            //Si aun existen Marcos y Faltan Paginas por ingresar
            if (marcosLibres>0&&tamPagP!=auxNumPagP) {
                //Se ingresa a la Fila el # de posicion
                filaPrioridad.push_back(c);
                //Se ingresa el proceso a Memoria Principal
                memoriaPrincipal[c][0] = nombreP;
                memoriaPrincipal[c][1] = auxNumPagP;
                //Output Pagina a Ingresar y Numero de Marco en Memoria Principal
                printf("-Pagina %d en Marco %d \n", auxNumPagP, c);
```

```
                //Tiempo Global + 1 (Cargar Pagina)
                tiempoGlobal += 1;
                //Se actualizan los Marcos Disponibles
                marcosLibres--;
                //Se actualiza el auxiliar Numero de Pagina
                auxNumPagP++;
            }
            else {
                //Tamaño 128
                //Por razones de optimizacion, sale rapido del Ciclo
                c = 128;
            }
        }
    }
    //Se notifica si se ingreso o no todo el proceso
    if (tamPagP == auxNumPagP) {
        printf("Se ingresaron todas las Paginas en Memoria Principal\n");
    } else {
        printf("No quedan espacios libres en Memoria Principal\n");
        printf("Se procede a Tecnica de Remplazo\n");
    }
}

//FUNCION INICIAL METER PROCESO
//Encargada de verificar proceso
//Si es valido, se procede a ingresarlo
//Recibe Tamaño y Nombre
void meterProceso(int tamañoP, int nombreP) {
    //Output (Input de proceso)
    printf("P %d %d \n",tamañoP, nombreP);
    //Tamaño del proceso en Paginas
    int tamPagP = ceil(tamañoP/16.0);
    //Auxiliar Numero de Pagina
    int auxNumPagP = 0;
    //Verifica el Proceso contra errores. (Si se habia cargado antes y tamaño
     maximo)
    if (verificarProcesoNuevo(nombreP, tamPagP)) {
        //Si existen Marcos Libres, se usan.
        if (marcosLibres>0) {
            buscaMeteLibres(nombreP, tamPagP, auxNumPagP);
        }
        //Si no existen Marcos Libres, se usa tecnica de Remplazo
        if (auxNumPagP<tamPagP) {
            buscaMeteOcupadas(nombreP, tamPagP, auxNumPagP);
        }
    }
}
////////////////////////// Funcion Cargar ////////////////////////


int main(int argc, const char * argv[]) {
    //Variables auxiliares del Leer Archivos
    int tamañoP;
    int nombreP;
```

```
int direccionP;
int accionP;
char input;
string comentario;
ifstream Texto;

//Se abre el archivo
//NOTA: El programa no verifica el archivo, por lo que cualquier error generado
 es debido al texto
//Un ejemplo de ello es P XX XX
//En donde al ingersar XX a un int produce error
//Se inica el programa pensando que el Texto esta bien formateado
Texto.open("operaciones.txt");

//Son dos corridas identicas, primero FIFO, despues LRU
for (int c = 0; c<2; c++) {
    if (c == 0) {
        cout<<"////////// FIFO //////////"<<endl;
        algoritmo = "FIFO";
    }
    if ( c== 1) {
        cout<<"////////// LRU //////////"<<endl;
        algoritmo = "LRU";
    }
    //Mientras no se llegue al fin del archivo, se lee
    //Se utiliza un switch para dirijir el input
    while (!Texto.eof()) {
        Texto>>input;
        switch (input) {
            case 'P':
                Texto>>tamañoP>>nombreP;
                meterProceso(tamañoP, nombreP);
                cout<<endl;
                break;
            case 'A':
                Texto>>direccionP>>nombreP>>accionP;
                tocaProceso(nombreP, direccionP, accionP, algoritmo);
                cout<<endl;
                break;
            case 'L':
                Texto>>nombreP;
                sacarProceso(nombreP);
                cout<<endl;
                break;
            case 'C':
                getline(Texto, comentario);
                cout<<"C"<<comentario<<endl<<endl;
                break;
            case 'F':
                getline(Texto, comentario);
                cout<<input<<endl;
                estadisticasProcesos();
                cout<<endl;
                break;
```

```
                case 'E':
                    getline(Texto, comentario);
                    cout<<input<<endl;
                    cout<<"Adios, ¡Nos vemos el Proximo semestre!"<<endl;
                    cout<<endl;
                    break;
                default:
                    break;
            }
        }

        //Regresa el apuntador
        Texto.clear();
        Texto.seekg(0, ios::beg);

        //Reinica Estructuras
        borrarMemorias();
    }

    Texto.close();
    return 0;
}
```