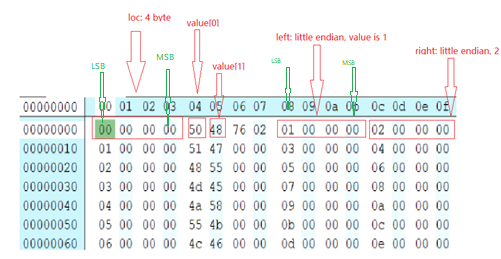
# Program 4

Yu yang 892449550



As the screenshot shows, C is using LSB. So the left child index of the first node: 01000000, is 00000001 in Big endian. Every node is padded into 16 bytes.

## Part1 C

#include <stdio.h>

#include <stdint.h>

#include <malloc.h>

#include <time.h>

#define TRUE 1

#define FALSE 0

typedef struct Node{

int32\_t loc;

char value[2];

int32\_t left;

int32\_t right;

} Node;

int main(int argn,char \*arg[]){

if(argn == 2){

printf("file name is %s\n",arg[1]);

}else{

perror("arg number doesn't match, exit");

exit(1);

}

// number of Node to be generated

int num = 10000;

int y = INT32\_MAX;

char \*fileName = arg[1];

FILE \*fp = fopen(fileName,"wb");

if(fp == NULL) {

perror("fopen");

exit(1);

}

Node \*head;

//!!note: Node size is padded to 16 byte

//INT32\_MAX = Ox7fff\_ffff, so the max number of Node

// should < INT32\_MAX, to prevent overflow

// if test case would override this number anyhow

// should comply with this rule

// because in the canvas page, there is no description about

// a argument of this number, this program does not perform

// overflow check

head = (Node\*)malloc(sizeof(struct Node) \* num);

if(head == NULL)

return -1;

//set all the nodes

for(int i = 0; i < num; i++){

(head+i)->loc = i;

(head+i)-> value[0] = 'A' + (rand() % 26);

(head+i)->value[1] = 'A' + (rand() % 26);

int32\_t left = (i+1)<<1;

int32\_t right= left + 1;

(head+i)->left = (left<=num)?(left-1):-1;

(head+i)->right= (right<=num)?(right-1):-1;

fwrite((head+i),sizeof(struct Node),1,fp);

}

if(fclose(fp)!=0){

perror("fclose");

exit(1);

}

free(head);

};

## Part2 Java

/\*\*

\*

\* writer preference implementation is based on wiki page:

\* https://en.wikipedia.org/wiki/Readers%E2%80%93writers\_problem

\*/

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.RandomAccessFile;

import java.nio.ByteOrder;

import java.nio.MappedByteBuffer;

import java.nio.channels.FileChannel;

import java.nio.charset.StandardCharsets;

import java.util.Date;

import java.util.Random;

import java.util.concurrent.Semaphore;

public class MMIO\_Reader {

//bTree file name

private static String FILE\_NAME;

//random generator ob

private Random rdG = new Random(new Date().getTime());

//randomAccessFile ob

private RandomAccessFile rf;

private MappedByteBuffer mBB;

private static long fileSize;

//reader will repeat this number

private static final int READER\_RUNNING\_ROUND= 10000;

//private static final int READER\_RUNNING\_ROUND= 1000;

private static final int SLEEP\_TIME = 20;

//number of reader/writer

private static final int READER\_WRITER\_NUMBER = 2;

//for random number gen object

private Semaphore semForRndGen = new Semaphore(1);

private int rdCount = 0, wrCount = 0;

//semaphore for writers preference, including

//reader mutex, writer mutex, reader entry mutex, resourse mutex;

private Semaphore rMutex = new Semaphore(1), wMutex = new Semaphore(1),

readTry = new Semaphore(1), resourseSemaphore = new Semaphore(1);

// Node data structure

private class Node{

int index;

int leftChildIndex;

int rightChildIndex;

char[] value;

public Node(int index, int leftChildIndex, int rightChildIndex, char[] value){

this.index = index;

this.leftChildIndex = leftChildIndex;

this.rightChildIndex = rightChildIndex;

this.value = value;

}

public Node(int index){

int basePointer = setFPLocation(index);

this.index= mBB.getInt() ;

this.value = new char[2];

value[0] = (char)(mBB.get()&0xff);

value[1] = (char)(mBB.get()&0xff);

//char in c is 1 byte, skip 2 padding bytes

mBB.position(basePointer + 8);

this.leftChildIndex = mBB.getInt();

this.rightChildIndex = mBB.getInt();

}

@Override

public String toString(){

return String.format("Node index %d, leftChildIndex %d, rightChildIndex %d, value[%c,%c]\n",this.index,this.leftChildIndex,this.rightChildIndex,this.value[0],this.value[1]);

}

}

public MMIO\_Reader(String fn){

this.FILE\_NAME = fn;

try {

//get file, map filem, set endian order

this.rf = new RandomAccessFile(FILE\_NAME,"rw");

this.fileSize = rf.length();

this.mBB = rf.getChannel().map(FileChannel.MapMode.READ\_WRITE,0,fileSize);

mBB.order(ByteOrder.LITTLE\_ENDIAN);

}catch(IOException ex){

ex.printStackTrace();

System.exit(1);

}

}

//get node by index

private Node getNode(int index){

int nodeTotalNumber = (int)getFileSize()>>4;

if(index >= nodeTotalNumber) {

System.err.println(String.format("invalid index, need to be smaller than %d, but %d provided", nodeTotalNumber, index));

System.exit(1);

}

return new Node(index);

}

//set file pointer to a node , return byte position

protected int setFPLocation(int nodeIndex){

mBB.position(nodeIndex<<4);

return nodeIndex<<4;

}

protected long getFileSize(){

return fileSize;

}

//generate two ascii character array

protected byte[] genRdValue(){

semForRndGen.acquireUninterruptibly(1);

byte[] result = new StringBuilder().append((char)(rdG.nextInt(26)+'A')).append((char)(rdG.nextInt(26)+'A')).toString().getBytes(StandardCharsets.US\_ASCII);

semForRndGen.release();

return result;

}

protected int getRandIndex(){

semForRndGen.acquireUninterruptibly(1);

int result = rdG.nextInt((int)(getFileSize()>>4));

semForRndGen.release();

return result;

}

private void go() {

int i = 0;

while(i++<READER\_WRITER\_NUMBER){

new Thread(new reader(),"reader" + i).start();

new Thread(new writer(),"writer" + i).start();

}

try {

this.rf.close();

}catch(IOException e){

e.printStackTrace();

}

}

private class reader implements Runnable{

private void printFromIndex(int index){

int nodeTotalNumber = (int)getFileSize()>>4;

int lIndex = 2\*(index + 1)-1;

//print if there is right child

//use writer preference

if(2\*(index+1)<nodeTotalNumber) {

readTry.acquireUninterruptibly(1);

rMutex.acquireUninterruptibly(1);

rdCount++;

if (rdCount == 1)

resourseSemaphore.acquireUninterruptibly(1);

rMutex.release();

readTry.release();

System.out.println("READER: " + getNode(2 \* (index + 1)).toString());

rMutex.acquireUninterruptibly(1);

rdCount--;

if(rdCount == 0)

resourseSemaphore.release();

rMutex.release();

}

//print all its left children

//use writer preference

while(lIndex<nodeTotalNumber){

readTry.acquireUninterruptibly(1);

rMutex.acquireUninterruptibly(1);

rdCount++;

if (rdCount == 1)

resourseSemaphore.acquireUninterruptibly(1);

rMutex.release();

readTry.release();

System.out.println("READER: " + getNode(lIndex).toString());

lIndex = 2\*(lIndex+1) -1;

rMutex.acquireUninterruptibly(1);

rdCount--;

if(rdCount == 0)

resourseSemaphore.release();

rMutex.release();

}

}

@Override

public void run(){

int i = READER\_RUNNING\_ROUND;

while(i-->0){

printFromIndex(getRandIndex());

try {

Thread.sleep(SLEEP\_TIME);

} catch(InterruptedException e){

System.err.println(e.toString());

System.exit(1);

}

}

}

}

private class writer implements Runnable{

private void modifyOneNodeVal(byte[] newVal, int nodeIndex){

mBB.position((nodeIndex<<4) + 4);

mBB.put(newVal);

System.out.println("WRITER: modified node#"+nodeIndex);

System.out.println(" " + getNode(nodeIndex).toString());

}

//overwrite a random node's value array

@Override

public void run(){

int i = READER\_RUNNING\_ROUND;

while(i-->0){

wMutex.acquireUninterruptibly(1);

wrCount++;

if(wrCount == 1)

readTry.acquireUninterruptibly(1);

wMutex.release();

resourseSemaphore.acquireUninterruptibly(1);

modifyOneNodeVal(genRdValue(),getRandIndex());

resourseSemaphore.release();

wMutex.acquireUninterruptibly(1);

wrCount--;

if(wrCount == 0)

readTry.release();

wMutex.release();

try {

Thread.sleep(SLEEP\_TIME);

} catch(InterruptedException e){

System.err.println(e.toString());

System.exit(1);

}

}

}

}

public static void main(String[] args){

//todo : cite reference wiki

//todo : take args file name

new MMIO\_Reader(args[0]).go();

}

}