**Lab 9**

|  |
| --- |
| 그림입니다. 원본 그림의 이름: YU_UI_RGB-10.png 원본 그림의 크기: 가로 2256pixel, 세로 3047pixel 프로그램 이름 : Adobe ImageReady |

|  |  |
| --- | --- |
| 과목명 | 객체지향프로그래밍과자료구조 |
| 교수님 | 김영탁 교수님 |
| 이 름 | 김주환 |
| 학 번 | 21812158 |
| 일 자 | 2021.11.05.금 |

|  |
| --- |
| /\* main.cpp \*/  /\* Description  \* 멀티 스레드 구현  \* 우선순위 Heap Queue 구현  \* Programmed by J. H. Kim  \* Last updated : 2021-11-05 \*/  #include <stdio.h>  #include <stdlib.h>  #include <Windows.h>  #include <thread>  #include <mutex>  #include "Multi\_Thread.h"  #include "HeapPrioQ\_CS.h"  #include "Event.h"  #include "ConsoleDisplay.h"  #include "SimParams.h"  #include <time.h>  #include <conio.h>  using namespace std;  void main() {  // 시간 관련 변수  LARGE\_INTEGER QP\_freq, t\_before, t\_after;  LONGLONG t\_diff;  double elapsed\_time, min\_elapsed\_time, max\_elapsed\_time;  double avg\_elapsed\_time, total\_elapsed\_time;  HeapPrioQ\_CS<int, Event> heapPriQ\_Event(30, string("HeapPriorityQueue\_Event"));  Event\* pEvent, \* pEv\_min\_elasped\_time, \* pEv\_max\_elapsed\_time;  int myAddr = 0;  int pkt\_event\_handler\_addr, eventPriority;  ThreadParam\_Event thrdParam\_EventGen, thrdParam\_EventHndlr;  HANDLE hThrd\_EventGenerator, hThrd\_EventHandler;  mutex cs\_main; // 콘솔 출력에 대한 임계구역  mutex cs\_thrd\_mon; // 공유자원 관리에 대한 임계구역    ThreadStatusMonitor thrdMon;  HANDLE consHndlr;  THREAD\_FLAG eventThreadFlag = RUN;  int count, numEventGenerated, numEventProcessed;  int num\_events\_in\_PrioQ;  Event eventProcessed[TOTAL\_NUM\_EVENTS];  consHndlr = initConsoleHandler();  QueryPerformanceFrequency(&QP\_freq); // 주파수  srand(time(NULL));  thrdMon.pFlagThreadTerminate = &eventThreadFlag;  thrdMon.totalEventGenerated = 0;  thrdMon.totalEventProcessed = 0;  for (int ev = 0; ev < TOTAL\_NUM\_EVENTS; ev++) {  thrdMon.eventProcessed[ev].setEventNo(-1); // mark as not-processed  thrdMon.eventProcessed[ev].setEventPri(-1);  }  /\* Create and Activate Thread\_EventHandler \*/  thrdMon.numEventProcessed = 0;  thrdParam\_EventHndlr.role = EVENT\_HANDLER;  thrdParam\_EventHndlr.myAddr = 1; // link address  thrdParam\_EventHndlr.pCS\_main = &cs\_main;  thrdParam\_EventHndlr.pCS\_thrd\_mon = &cs\_thrd\_mon;  thrdParam\_EventHndlr.pPriQ\_Event = &heapPriQ\_Event;  thrdParam\_EventHndlr.maxRound = MAX\_ROUND;  thrdParam\_EventHndlr.QP\_freq = QP\_freq;  thrdParam\_EventHndlr.pThrdMon = &thrdMon;  thread thrd\_EvProc(EventProc, &thrdParam\_EventHndlr); // Event 처리 Thread 실행  cs\_main.lock();  printf("Thread\_EventGen is created and activated ...\n");  cs\_main.unlock();  /\* Create and Activate Thread\_EventGen \*/  thrdMon.numEventGenerated = 0;  thrdParam\_EventGen.role = EVENT\_GENERATOR;  thrdParam\_EventGen.myAddr = 0; // my Address  thrdParam\_EventGen.pCS\_main = &cs\_main;  thrdParam\_EventGen.pCS\_thrd\_mon = &cs\_thrd\_mon;  thrdParam\_EventGen.pPriQ\_Event = &heapPriQ\_Event;  thrdParam\_EventGen.targetEventGen = NUM\_EVENTS\_PER\_GEN;  thrdParam\_EventGen.maxRound = MAX\_ROUND;  thrdParam\_EventGen.QP\_freq = QP\_freq;  thrdParam\_EventGen.pThrdMon = &thrdMon;  thread thrd\_EvGen(EventGen, &thrdParam\_EventGen); // Event 생성 Thread 실행  cs\_main.lock();  printf("Thread\_EventGen is created and activated ...\n");  cs\_main.unlock();  /\* periodic monitoring in main() \*/  for (int round = 0; round < MAX\_ROUND; round++) {  cs\_main.lock();  cls(consHndlr);  gotoxy(consHndlr, 0, 0);  printf("Thread monitoring by main() ::\n");  printf(" round(%2d): current total\_event\_gen (%2d), total\_event\_proc(%2d)\n",  round, thrdMon.totalEventGenerated, thrdMon.totalEventProcessed);  printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  numEventGenerated = thrdMon.numEventGenerated;  printf("Events generated (current total = %2d)\n ", numEventGenerated);  count = 0;  for (int ev = 0; ev < numEventGenerated; ev++) {  pEvent = &thrdMon.eventGenerated[ev];  if (pEvent != NULL) {  cout << \*pEvent << " ";  if (((ev + 1) % EVENTS\_PER\_LINE) == 0)  printf("\n ");  }  } //end for  printf("\n");  printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  num\_events\_in\_PrioQ = heapPriQ\_Event.size();  printf("Events currently in Priority\_Queue (%d) : \n", num\_events\_in\_PrioQ);  heapPriQ\_Event.fprint(cout);  printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  numEventProcessed = thrdMon.totalEventProcessed;  printf("Events processed (current total = %d): \n ", numEventProcessed);  count = 0;  total\_elapsed\_time = 0.0;  for (int ev = 0; ev < numEventProcessed; ev++) {  pEvent = &thrdMon.eventProcessed[ev];  if (pEvent != NULL) {  pEvent->printEvent\_proc();  if (((ev + 1) % EVENTS\_PER\_LINE) == 0)  printf("\n ");  }  if (ev == 0) {  min\_elapsed\_time = max\_elapsed\_time = total\_elapsed\_time =  pEvent->getEventElapsedTime(); // in milli-second  pEv\_min\_elasped\_time = pEv\_max\_elapsed\_time = pEvent;  }  else {  if (min\_elapsed\_time > pEvent->getEventElapsedTime()) {  min\_elapsed\_time = pEvent->getEventElapsedTime(); // in milli-second  pEv\_min\_elasped\_time = pEvent;  }  if (max\_elapsed\_time < pEvent->getEventElapsedTime()) {  max\_elapsed\_time = pEvent->getEventElapsedTime(); // in milli-second  pEv\_max\_elapsed\_time = pEvent;  }  total\_elapsed\_time += pEvent->getEventElapsedTime();  }  } //end for showing eventProcessed  printf("\n");  if (numEventProcessed > 0) {  printf("numEventProcessed = %d\n", numEventProcessed);  printf("min\_elapsed\_time = %8.2lf[ms]; ", min\_elapsed\_time);  cout << \*pEv\_min\_elasped\_time << endl;  printf("max\_elapsed\_time = %8.2lf[ms]; ", max\_elapsed\_time);  cout << \*pEv\_max\_elapsed\_time << endl;  avg\_elapsed\_time = total\_elapsed\_time / numEventProcessed;  printf("avg\_elapsed\_time = %8.2lf[ms]; \n", avg\_elapsed\_time);  }  if (numEventProcessed >= TOTAL\_NUM\_EVENTS) {  eventThreadFlag = TERMINATE; // set 1 to terminate threads  cs\_main.unlock();  break;  } //end if  cs\_main.unlock();  Sleep(100);  } //end for (int round = 0; ...)  thrd\_EvProc.join();  thrd\_EvGen.join();  printf("Hit any key to terminate : ");  \_getch();  } |
| /\* Event.h \*/  #ifndef E\_H  #define E\_H  #include <iostream>  #include <string>  #include <fstream>  #include <Windows.h> // for LARGE\_INTEGER used in QueryPerformanceCounter()  #include <iomanip>  using namespace std;  enum EventStatus { GENERATED, ENQUEUED, PROCESSED, UNDEFINED };  #define MAX\_EVENT\_PRIORITY 100  #define NUM\_EVENT\_GENERATORS 10  class Event {  friend ostream& operator<<(ostream& fout, const Event& e);  public:  Event(); // default constructor  Event(int event\_id, int event\_pri, int srcAddr); //constructor  void printEvent\_proc();  void setEventHandlerAddr(int evtHndlerAddr) { event\_handler\_addr = evtHndlerAddr; }  void setEventGenAddr(int genAddr) { event\_gen\_addr = genAddr; }  void setEventNo(int evtNo) { event\_no = evtNo; }  void setEventPri(int pri) { event\_pri = pri; }  void setEventStatus(EventStatus evtStatus) { eventStatus = evtStatus; }  void setEventGenTime(LARGE\_INTEGER t\_gen) { t\_event\_gen = t\_gen; }  void setEventProcTime(LARGE\_INTEGER t\_proc) { t\_event\_proc = t\_proc; }  LARGE\_INTEGER getEventGenTime() { return t\_event\_gen; }  LARGE\_INTEGER getEventProcTime() { return t\_event\_proc; }    void setEventElaspsedTime(double t\_elapsed\_ms) { t\_elapsed\_time\_ms = t\_elapsed\_ms; }  double getEventElapsedTime() { return t\_elapsed\_time\_ms; }    int getEventPri() { return event\_pri; }  int getEventNo() { return event\_no; }  // 우선순위에 따라 비교  bool operator>(Event& e) { return (event\_pri > e.event\_pri); }  bool operator<(Event& e) { return (event\_pri < e.event\_pri); }  private:  int event\_no;  int event\_gen\_addr;  int event\_handler\_addr;  int event\_pri; // event\_priority  // 시간  LARGE\_INTEGER t\_event\_gen;  LARGE\_INTEGER t\_event\_proc;  double t\_elapsed\_time\_ms;    EventStatus eventStatus;  };  Event\* genRandEvent(int evt\_no);  #endif // !E\_H |
| /\* Event.cpp \*/  #include "Event.h"  Event::Event() {  event\_no = 0;  event\_gen\_addr = 0;  event\_handler\_addr = -1; // event handler is not defined at this moment  event\_pri = 0; // event\_priority  eventStatus = GENERATED;  }  Event::Event(int evt\_no, int evt\_pri, int evtGenAddr) {  event\_no = evt\_no;  event\_gen\_addr = evtGenAddr;  event\_handler\_addr = -1; // event handler is not defined at this moment  event\_pri = evt\_pri; // event\_priority  eventStatus = GENERATED;  }  Event\* genRandEvent(int evt\_no) {  Event\* pEv;  int evt\_prio;  int evt\_generator\_id;  evt\_prio = rand() % MAX\_EVENT\_PRIORITY;  evt\_generator\_id = rand() % NUM\_EVENT\_GENERATORS;  pEv = new Event(evt\_no, evt\_prio, evt\_generator\_id);  return pEv;  }  void Event::printEvent\_proc() {  cout << "Ev(no:" << setw(2) << event\_no << ", pri:" << setw(2) << event\_pri;  cout.precision(2);  cout.setf(ios::fixed);  cout << ", t\_elapsed:" << setw(8) << t\_elapsed\_time\_ms << ") ";  }  ostream& operator<<(ostream& fout, const Event& evt) {  fout << "Ev(no:" << setw(3) << evt.event\_no;  fout << ", pri:" << setw(3) << evt.event\_pri << ")";  return fout;  } |
| /\* T\_Entry.h \*/  #ifndef T\_E\_H  #define T\_E\_H  #include <ostream>  #include <iomanip>  using namespace std;  template<typename K, typename V>  class T\_Entry {  friend ostream& operator<<(ostream& fout, T\_Entry<K, V>& entry) {  fout << "[Key: " << setw(2) << entry.getKey() << /\*"." << entry.getValue() <<\*/ "]";  return fout;  }  public:  T\_Entry(K key, V value) { \_key = key; \_value = value; }  T\_Entry() { \_key = 999; }  ~T\_Entry() {}  void setKey(const K& key) { \_key = key; }  void setValue(const V& value) { \_value = value; }  K getKey() const { return \_key; }  V getValue() const { return \_value; }  // Key 값 비교  bool operator>(const T\_Entry& right) { return (\_key > right.getKey()); }  bool operator>=(const T\_Entry& right) { return (\_key >= right.getKey()); }  bool operator<(const T\_Entry& right) { return (\_key < right.getKey()); }  bool operator<=(const T\_Entry& right) { return (\_key <= right.getKey()); }  bool operator==(const T\_Entry& right) { return ((\_key == right.getKey()) && (\_value == right.getValue())); }  T\_Entry& operator=(T\_Entry& right);  void fprint(ostream& fout);  private:  K \_key;  V \_value;  };  template<typename K, typename V>  T\_Entry<K, V>& T\_Entry<K, V>::operator=(T\_Entry<K, V>& right) {  \_key = right.getKey();  \_value = right.getValue();  return \*this;  }  template<typename K, typename V>  void T\_Entry<K, V>::fprint(ostream& fout) {  fout << "[Key:" << setw(2) << this->getKey() << ", " << \*(this->getValue()) << "]";  }  #endif // !T\_E\_H |
| /\* TA\_Entry.h \*/  #ifndef TA\_E\_H  #define TA\_E\_H  #include "T\_Entry.h"  enum SortingDirection { INCREASING, DECREASING };  template<typename K, typename V>  class TA\_Entry {  public:  TA\_Entry(int n, string nm);  ~TA\_Entry();  int size() { return num\_elements; }  bool empty() { return num\_elements == 0; }  string getName() { return name; }  void reserve(int new\_capacity); // 용량 재설정  void insert(int i, T\_Entry<K, V> element); // 삽입  void remove(int i); // 삭제  T\_Entry<K, V>& at(int i); // i번째 인덱스의 element 반환  void set(int i, T\_Entry<K, V> element); // i번째 인덱스를 element로 설정  T\_Entry<K, V> getMin(int begin, int end);  T\_Entry<K, V> getMax(int begin, int end);  // 탐색 및 정렬  void shuffle();  int sequential\_search(T\_Entry<K, V> search\_key);  int binary\_search(T\_Entry<K, V> search\_key);  void selection\_sort(SortingDirection sd);  void quick\_sort(SortingDirection sd);  // 출력  void fprint(ofstream& fout, int elements\_per\_line);  void fprintSample(ofstream& fout, int elements\_per\_line, int num\_sample\_lines);  bool isValidIndex(int i);  T\_Entry<K, V>& operator[](int index) { return t\_array[index]; }  protected:  T\_Entry<K, V>\* t\_array; // element 배열  int num\_elements; // 현재 개수  int capacity; // 배열의 크기(용량), 실제 사용에서는 크기를 한칸 늘려서 사용한다.  string name;  };  template<typename K, typename V>  TA\_Entry<K, V>::TA\_Entry(int n, string nm) {  t\_array = new T\_Entry<K, V>[n];  if (t\_array == NULL) {  cout << "Error in creation of dynamic array of size (" << n << ") !!" << endl;  exit;  }  capacity = n;  num\_elements = 0;  name = nm;  }  template<typename K, typename V>  TA\_Entry<K, V>::~TA\_Entry() {  if (t\_array != NULL)  delete[] t\_array;  }  template<typename K, typename V>  void TA\_Entry<K, V>::reserve(int new\_capacity) {  if (capacity >= new\_capacity)  return; // already big enough  T\_Entry<K, V>\* t\_newGA = new T\_Entry<K, V>[new\_capacity];  if (t\_newGA == NULL) {  cout << "Error in creation of dynamic array of size (" << new\_capacity << ") !!" << endl;  exit;  }  cout << this->getName() << " expands capacity to " << setw(3)  << new\_capacity << endl;  for (int i = 0; i < num\_elements; i++)  t\_newGA[i] = t\_array[i];  delete[] t\_array;  t\_array = t\_newGA;  capacity = new\_capacity;  }  template<typename K, typename V>  void TA\_Entry<K, V>::insert(int i, T\_Entry<K, V> element) {  if (isValidIndex(i)) {  for (int j = num\_elements - 1; j >= i; j--)  t\_array[j + 1] = t\_array[j]; //shift up elements in one position  t\_array[i] = element;  num\_elements++;  }  }  template<typename K, typename V>  void TA\_Entry<K, V>::remove(int i) {  if (isValidIndex(i)) {  for (int j = i + 1; j < num\_elements; j++)  t\_array[j - 1] = t\_array[j]; //shift down elements in one position  num\_elements--;  }  if (num\_elements < (capacity / 2)) {  int new\_capacity = capacity / 2;  TA\_Entry<K, V>\* t\_newGA = new TA\_Entry<K, V>[new\_capacity];  if (t\_newGA == NULL) {  return; // new memory allocation failed.  // Just return without modification.  }  cout << this->getName()  << " reduces capacity to " << setw(3)  << new\_capacity << endl;  for (int i = 0; i < num\_elements; i++)  t\_newGA[i] = t\_array[i];  delete[] t\_array;  t\_array = t\_newGA;  capacity = new\_capacity;  }  }  template<typename K, typename V>  T\_Entry<K, V>& TA\_Entry<K, V>::at(int i) {  if (isValidIndex(i)) {  return t\_array[i];  }  }  template<typename K, typename V>  void TA\_Entry<K, V>::set(int i, T\_Entry<K, V> element) {  if (isValidIndex(i)) {  t\_array[i] = element;  }  }  template<typename K, typename V>  T\_Entry<K, V> TA\_Entry<K, V>::getMin(int begin, int end) {  V minValue;  int index\_min;  minValue = t\_array[begin];  index\_min = begin;  for (int i = begin + 1; i <= end; i++) {  if (t\_array[i] < minValue) { // T must provide operator<() overloading !!  minValue = t\_array[i];  index\_min = i;  }  }  return t\_array[index\_min];  }  template<typename K, typename V>  T\_Entry<K, V> TA\_Entry<K, V>::getMax(int begin, int end) {  V maxValue;  int index\_max;  maxValue = t\_array[begin];  index\_max = begin;  for (int i = begin + 1; i <= end; i++) {  if (t\_array[i] > maxValue) { // T must provide operator>() overloading !!  maxValue = t\_array[i];  index\_max = i;  }  }  return t\_array[index\_max];  }  template<typename K, typename V>  void TA\_Entry<K, V>::shuffle() {  srand(time(0));  int index1, index2;  int rand\_1, rand\_2;  V temp;  for (int i = 0; i < num\_elements; i++) {  rand\_1 = rand();  rand\_2 = rand();  index1 = ((rand\_1 << 15) | rand\_2) % num\_elements;  rand\_1 = rand();  rand\_2 = rand();  index2 = ((rand\_1 << 15) | rand\_2) % num\_elements;  temp = t\_array[index1];  t\_array[index1] = t\_array[index2];  t\_array[index2] = temp;  }  }  template<typename K, typename V>  int TA\_Entry<K, V>::sequential\_search(T\_Entry<K, V> search\_key) {  int index;  K key;  if (search\_key == "ST\_ID") { // student\_ID  for (int index = 0; index < num\_elements; index++) {  t\_array[index].getKey(search\_key, &key);  if (key == search\_key)  return index;  }  }  return -1;  }  template<typename K, typename V>  int TA\_Entry<K, V>::binary\_search(T\_Entry<K, V> search\_key) {  K key;  int low, mid, high; int loop = 1;  low = 0; high = num\_elements - 1;  while (low <= high) {  cout << setw(2) << loop << "-th loop: current search range [" << setw(3) << low << ", " << setw(3) << high << "]" << endl;  mid = (low + high) / 2;  t\_array[mid].getKey(search\_key, &key);  if (key == search\_key)  return mid;  else if (key > search\_key)  high = mid - 1;  else  low = mid + 1; loop++;  }  }  template<typename K, typename V>  void TA\_Entry<K, V>::selection\_sort(SortingDirection sd) {  int index\_min, index\_max; // index of the element with minimum value  T\_Entry<K, V> tempElement;  K minKey, maxKey, key;  for (int i = 0; i < num\_elements - 1; i++) {  if (sd == INCREASING) { // sorting in increasing (non\_decreasing) order  index\_min = i;  key = t\_array[i].getKey(); // 키 반환  minKey = (K)key;  for (int j = i + 1; j < num\_elements; j++) {  key = t\_array[j].getKey();  if ((K)key < minKey) {  index\_min = j;  minKey = (K)key;  }  }  if (index\_min != i) { // if a smaller element is found, then swap  tempElement = t\_array[index\_min];  t\_array[index\_min] = t\_array[i];  t\_array[i] = tempElement;  }  }  else { // sorting in decreasing (non\_increasing) order  index\_max = i;  key = t\_array[i].getKey();  maxKey = (K)key;  for (int j = i + 1; j < num\_elements; j++) {  key = t\_array[j].getKey();  if ((K)key > maxKey) {  index\_max = j;  maxKey = (K)key;  }  }  if (index\_max != i) { // if a smaller element is found, then swap  tempElement = t\_array[index\_max];  t\_array[index\_max] = t\_array[i];  t\_array[i] = tempElement;  }  }  } // end for  }  template<typename T>  void \_quick\_sort(T\* array, int size, int left, int right,  SortingDirection sd = INCREASING) {  int pI, newPI; // pivot index  if (left >= right) {  return;  }  else { //select a pI (pivotIndex) in the range left ≤ pI ≤ right  pI = (left + right) / 2;  }  newPI = \_partition(array, size, left, right, pI, sd);  if (left < (newPI - 1)) {  \_quick\_sort(array, size, left, newPI - 1, sd);  // recursively sort elements on the left of pivotNewIndex  }  if ((newPI + 1) < right) {  \_quick\_sort(array, size, newPI + 1, right, sd);  // recursively sort elements on the right of pivotNewIndex  }  }  template<typename K, typename V>  void TA\_Entry<K, V>::quick\_sort(SortingDirection sd) {  \_quick\_sort(t\_array, size, 0, num\_elements - 1, sd);  }  template<typename K, typename V>  void TA\_Entry<K, V>::fprint(ofstream& fout, int elements\_per\_line) {  int count = 0;  while (count < num\_elements) {  for (int i = 0; i < elements\_per\_line; i++) {  fout << t\_array[count] << " ";  count++;  if (count % elements\_per\_line == 0)  fout << endl;  }  }  cout << endl;  }  template<typename K, typename V>  void TA\_Entry<K, V>::fprintSample(ofstream& fout, int elements\_per\_line, int num\_sample\_lines) {  string T\_type;  int last\_block\_start;  int count = 0;  T\_type = typeid(T\_Entry<K, V>).name();  for (int i = 0; i < num\_sample\_lines; i++) {  for (int j = 0; j < elements\_per\_line; j++) {  if (count >= num\_elements) {  fout << endl;  return;  }  if ((T\_type == string("int")) || (T\_type == string("double")) ||  (T\_type == string("class std::basic\_string<char,struct std::char\_traits<char>,class std::allocator<char> > ")))  fout << setw(10) << t\_array[count];  else  fout << t\_array[count] << " ";  count++;  } fout << endl;  }  if (count < (num\_elements - elements\_per\_line \* num\_sample\_lines))  count = num\_elements - elements\_per\_line \* num\_sample\_lines;  fout << " . . . . . " << endl;  for (int i = 0; i < num\_sample\_lines; i++) {  for (int j = 0; j < elements\_per\_line; j++) {  if (count >= num\_elements) {  fout << endl;  return;  }  if ((T\_type == string("int")) || (T\_type == string("double")) ||  (T\_type == string("class std::basic\_string<char, struct std::char\_traits<char>, class std::allocator<char> > ")))  fout << setw(10) << t\_array[count];  else  fout << t\_array[count] << " ";  count++;  } fout << endl;  } fout << endl;  }  template<typename K, typename V>  bool TA\_Entry<K, V>::isValidIndex(int i) {  if ((i < 0) || (i > num\_elements))  return false;  else  return true;  }  #endif // !TA\_E\_H |
| /\* CompleteBinaryTree.h \*/  #ifndef COMPLETE\_BINARY\_TREE\_H  #define COMPLETE\_BINARY\_TREE\_H  #include "TA\_Entry.h"  #include "T\_Entry.h"  #define CBT\_ROOT 1  template<typename K, typename V>  class CompleteBinaryTree : public TA\_Entry<K, V> {  public:  CompleteBinaryTree(int capa, string nm);  int add\_at\_end(T\_Entry<K, V>& elem); // CBT의 last node에 element 삽입  T\_Entry<K, V>& getEndElement() { return this->t\_array[end]; } // last node 반환  T\_Entry<K, V>& getRootElement() { return this->t\_array[CBT\_ROOT]; } // root node 반환  int getEndIndex() { return end; }  void removeCBTEnd(); // last node element 제거  void printCBT(ostream& fout);  void printCBT\_byLevel(ostream& fout);  protected:  void \_printCBT\_byLevel(ostream& fout, int p, int level);  int parentIndex(int index) { return index / 2; } // parent node index  int leftChildIndex(int index) { return index \* 2; } // left child node index  int rightChildIndex(int index) { return (index \* 2 + 1); } // right child node index  bool hasLeftChild(int index) { return ((index \* 2) <= end); } // left child node에 element가 있는지 확인  bool hasRightChild(int index) { return ((index \* 2 + 1) <= end); } // right child node에 element가 있는지 확인  int end; // last node index  };  template<typename K, typename V>  CompleteBinaryTree<K, V>::CompleteBinaryTree(int capa, string nm)  :TA\_Entry<K, V>(capa + 1, nm) {  end = 0; // reset to empty  }  template<typename K, typename V>  void CompleteBinaryTree<K, V>::printCBT(ostream& fout) {  int cnt = 1;  if (end <= 0) {  fout << this->getName() << " is empty now !!" << endl;  return;  }  int count = 0;  for (int i = 1; i <= end; i++) {  fout << setw(3) << this->t\_array[i];  if ((cnt++ % 5) == 0)  fout << endl;  //if ((((count + 1) % 10) == 0) && (i != end))  //fout << endl;  count++;  }  }  template<typename K, typename V>  void CompleteBinaryTree<K, V>::\_printCBT\_byLevel(ostream& fout, int index, int level) {  int index\_child;  if (hasRightChild(index)) {  index\_child = rightChildIndex(index);  this->\_printCBT\_byLevel(fout, index\_child, level + 1);  }  for (int i = 0; i < level; i++)  fout << " ";  this->t\_array[index].fprint(fout);  fout << endl;  if (hasLeftChild(index)) {  index\_child = leftChildIndex(index);  this->\_printCBT\_byLevel(fout, index\_child, level + 1);  }  }  template<typename K, typename V>  void CompleteBinaryTree<K, V>::printCBT\_byLevel(ostream& fout) {  if (end <= 0) {  fout << "CBT is EMPTY now !!" << endl;  return;  }  this->\_printCBT\_byLevel(fout, CBT\_ROOT, 0);  }  template<typename K, typename V>  int CompleteBinaryTree<K, V>::add\_at\_end(T\_Entry<K, V>& elem) {  if (end >= this->capacity) {  cout << this->getName() << " is FULL now !!" << endl;  return -1;  }  end++;  this->t\_array[end] = elem;  return end;  }  template<typename K, typename V>  void CompleteBinaryTree<K, V>::removeCBTEnd() {  end--;  this->num\_elements--;  }  #endif |
| /\* HeapPrioQ.h \*/  #ifndef HEAP\_PRIO\_QUEUE\_H  #define HEAP\_PRIO\_QUEUE\_H  #include <mutex>  #include "CompleteBinaryTree.h"  using namespace std;  template<typename K, typename V>  class HeapPrioQ\_CS : public CompleteBinaryTree<K, V> {  public:  HeapPrioQ\_CS(int capa, string nm);  ~HeapPrioQ\_CS();  bool isEmpty() { return (this->end <= 0); }  bool isFull() { return (this->end >= this->heapPriQ\_capa); }  T\_Entry<K, V>\* insert(T\_Entry<K, V>& elem); // 전달받은 element를 삽입, add\_at\_end()  T\_Entry<K, V>\* removeHeapMin(); // 우선순위가 가장 높은 element 추출 후 반환  T\_Entry<K, V>\* getHeapMin(); // 우선순위가 가장 높은 element 반환  void fprint(ostream& fout);  int size() { return this->end; }  private:  int heapPriQ\_capa; // Heap priority Queue의 capacity  mutex cs\_priQ; // priority queue의 운용에 사용되는 mutex  };  template<typename K, typename V>  HeapPrioQ\_CS<K, V>::HeapPrioQ\_CS(int capa, string nm)  :CompleteBinaryTree<K, V>(capa, nm) {  heapPriQ\_capa = capa;  }  template<typename K, typename V>  HeapPrioQ\_CS<K, V>::~HeapPrioQ\_CS()  { }  template<typename K, typename V>  T\_Entry<K, V>\* HeapPrioQ\_CS<K, V>::insert(T\_Entry<K, V>& elem) {  int index, parent\_index;  T\_Entry<K, V> temp;  if (isFull()) {  cout << "HeapPrioQ is Full !!" << endl;  return NULL;  }  cs\_priQ.lock();  index = this->add\_at\_end(elem); // last node에 element insert    /\* up-heap bubbling \*/  while (index != CBT\_ROOT) {  parent\_index = this->parentIndex(index); // parent index 저장  if (this->t\_array[index].getKey() >= this->t\_array[parent\_index].getKey())  break;  else { // child node의 우선순위가 parent node의 우선순위보다 높은 경우  temp = this->t\_array[index];  this->t\_array[index] = this->t\_array[parent\_index];  this->t\_array[parent\_index] = temp;  index = parent\_index;  }  }  cs\_priQ.unlock();  T\_Entry<K, V>\* pRoot = &(this->t\_array[CBT\_ROOT]);  return pRoot;  }  template<typename K, typename V>  T\_Entry<K, V>\* HeapPrioQ\_CS<K, V>::removeHeapMin() {  int index\_p, index\_c, index\_rc;  T\_Entry<K, V>\* pMinElem;  T\_Entry<K, V> temp, t\_p, t\_c;  int HPQ\_size = size();  if (HPQ\_size <= 0) { // element가 없다.  return NULL;  }  cs\_priQ.lock();  pMinElem = (T\_Entry<K, V>\*) new T\_Entry<K, V>;  \*pMinElem = this->getRootElement(); // root node element  if (HPQ\_size == 1) {  this->removeCBTEnd();  }  else {  index\_p = CBT\_ROOT;  this->t\_array[CBT\_ROOT] = this->t\_array[this->end]; // last node element를 root node element로 이동  this->end--;    /\* down-heap bubbling \*/  while (this->hasLeftChild(index\_p)) { // left child 존재  index\_c = this->leftChildIndex(index\_p);  index\_rc = this->rightChildIndex(index\_p);  // right child가 있고 right child의 우선순위가 left child의 우선순위보다 높은 경우 아래의 조건문을 실행  if (this->hasRightChild(index\_p) && (this->t\_array[index\_c] > this->t\_array[index\_rc]))  index\_c = index\_rc;  t\_p = this->t\_array[index\_p];  t\_c = this->t\_array[index\_c];  if (t\_p > t\_c) { // child의 우선순위가 더 높다.  //swap(index\_u, index\_c);  temp = this->t\_array[index\_p];  this->t\_array[index\_p] = this->t\_array[index\_c];  this->t\_array[index\_c] = temp;  index\_p = index\_c;  }  else  break;  } // end while  }  cs\_priQ.unlock();  return pMinElem;  }  template<typename K, typename V>  T\_Entry<K, V>\* HeapPrioQ\_CS<K, V>::getHeapMin() {  T\_Entry<K, V>\* pMinElem;  if (this->end <= 0) {  return NULL;  }  pMinElem = (T\_Entry<K, V>\*) new T\_Entry<K, V>;  \*pMinElem = this->getRootElement();  return pMinElem;  }  template<typename K, typename V>  void HeapPrioQ\_CS<K, V>::fprint(ostream& fout) {  if (size() <= 0) {  fout << "HeapPriorityQueue is Empty !!" << endl;  return;  }  else  CompleteBinaryTree<K, V>::printCBT(fout);  }  #endif |
| /\* SimParam.h Simulation Parameters \*/  #ifndef SIMULATION\_PARAMETERS\_H  #define SIMULATION\_PARAMETERS\_H  #define NUM\_EVENT\_GENERATORS 1  #define NUM\_EVENTS\_PER\_GEN 50  #define NUM\_EVENT\_HANDLERS 1  #define TOTAL\_NUM\_EVENTS (NUM\_EVENTS\_PER\_GEN \* NUM\_EVENT\_GENERATORS)  #define PRI\_QUEUE\_CAPACITY 50  #define PLUS\_INF INT\_MAX  #define MAX\_ROUND 1000  #define EVENTS\_PER\_LINE 5  #endif // !SIMULATION\_PATAMETERS\_H |
| /\* Multi\_thread.h \*/  #ifndef MULTI\_THREAD\_H  #define MULTI\_THREAD\_H  #include <iostream>  #include <fstream>  #include <Windows.h>  #include <thread>  #include <process.h>  #include <string>  #include "HeapPrioQ\_CS.h"  #include "Event.h"  #include "SimParams.h"  using namespace std;  enum ROLE {  EVENT\_GENERATOR,  EVENT\_HANDLER  };  enum THREAD\_FLAG {  INITIALIZE,  RUN, TERMINATE  };  // 모니터링 관련 구조체  typedef struct ThreadStatusMonitor {  int numEventGenerated; // 생성 이벤트 수  int numEventProcessed; // 처리 이벤트 수  int totalEventGenerated; // 총 생성 이벤트 수  int totalEventProcessed; // 총 처리 이벤트 수  // used for monitoring only  Event eventGenerated[TOTAL\_NUM\_EVENTS]; // 생성된 이벤트  Event eventProcessed[TOTAL\_NUM\_EVENTS]; // 처리된 이벤트  THREAD\_FLAG\* pFlagThreadTerminate; // Thread 실행 여부 결정  } ThreadStatusMonitor;  // Thread에 전달되는 파라메터 관련 구조체  typedef struct ThreadParam {  mutex\* pCS\_main; // 콘솔 출력 mutex  mutex\* pCS\_thrd\_mon; // 공유자원 관리에 대한 mutex  HeapPrioQ\_CS<int, Event>\* pPriQ\_Event; // Heap Priority Queue의 포인터  FILE\* fout;  ROLE role;  int myAddr; // Thread 주소  int maxRound;  int targetEventGen; // 이벤트를 얼마나 생성할지 결정  LARGE\_INTEGER QP\_freq; // used in measurements  ThreadStatusMonitor\* pThrdMon; // 모니터링  } ThreadParam\_Event;  void EventGen(ThreadParam\_Event\* pParam);  void EventProc(ThreadParam\_Event\* pParam);  #endif |
| /\* ConsoleDisplay.h \*/  #ifndef CONSOLE\_DISPLAY\_H  #define CONSOLE\_DISPLAY\_H  #include <Windows.h>  HANDLE initConsoleHandler();  void cls(HANDLE hConsole);  void closeConsoleHandler(HANDLE hndlr);  int gotoxy(HANDLE consoleHandler, int x,  int y);  #endif |
| /\* ConsoleDisplay.cpp \*/  #include <stdio.h>  #include "ConsoleDisplay.h"  HANDLE consoleHandler;  HANDLE initConsoleHandler() {  HANDLE stdCnslHndlr;  stdCnslHndlr = GetStdHandle(STD\_OUTPUT\_HANDLE);  consoleHandler = stdCnslHndlr;  return consoleHandler;  }  void closeConsoleHandler(HANDLE hndlr) {  CloseHandle(hndlr);  }  int gotoxy(HANDLE consHndlr, int x, int y) {  if (consHndlr == INVALID\_HANDLE\_VALUE)  return 0;  COORD coords = { static\_cast<short>(x), static\_cast<short>(y) };  SetConsoleCursorPosition(consHndlr, coords);  }  void cls(HANDLE hConsole) {  CONSOLE\_SCREEN\_BUFFER\_INFO csbi;  SMALL\_RECT scrollRect;  COORD scrollTarget;  CHAR\_INFO fill;  // Get the number of character cells in the current buffer.  if (!GetConsoleScreenBufferInfo(hConsole, &csbi)) {  return;  }  // Scroll the rectangle of the entire buffer.  scrollRect.Left = 0;  scrollRect.Top = 0;  scrollRect.Right = csbi.dwSize.X;  scrollRect.Bottom = csbi.dwSize.Y;  // Scroll it upwards off the top of the buffer with a magnitude of the entire height.  scrollTarget.X = 0;  scrollTarget.Y = (SHORT)(0 - csbi.dwSize.Y);  // Fill with empty spaces with the buffer's default text attribute.  fill.Char.UnicodeChar = TEXT(' ');  fill.Attributes = csbi.wAttributes;  // Do the scroll  ScrollConsoleScreenBuffer(hConsole, &scrollRect, NULL, scrollTarget, &fill);  // Move the cursor to the top left corner too.  csbi.dwCursorPosition.X = 0;  csbi.dwCursorPosition.Y = 0;  SetConsoleCursorPosition(hConsole, csbi.dwCursorPosition);  } |
| /\* Thread\_EventGenerator.cpp \*/  #include <Windows.h>  #include "Multi\_Thread.h"  #include "HeapPrioQ\_CS.h"  #include "Event.h"  #include "SimParams.h"  using namespace std;  using std::this\_thread::sleep\_for;  void EventGen(ThreadParam\_Event\* pParam) {  // 구조체 변수 가져옴  HeapPrioQ\_CS<int, Event>\* pPriQ\_Event = pParam->pPriQ\_Event;  int myRole = pParam->role;  int myAddr = pParam->myAddr;  int maxRound = pParam->maxRound;  int targetEventGen = pParam->targetEventGen;  LARGE\_INTEGER QP\_freq = pParam->QP\_freq;  ThreadStatusMonitor\* pThrdMon = pParam->pThrdMon;  T\_Entry<int, Event>\* pEntry, entry\_event;  Event event, \* pEvent;  int event\_no = 0;  int event\_priority = 0;  int event\_gen\_count = 0;  int event\_handler\_addr;  LARGE\_INTEGER t\_gen; // 이벤트 생성 시간  for (int round = 0; round < maxRound; round++) {  if (event\_gen\_count >= targetEventGen) {  if (\*pThrdMon->pFlagThreadTerminate == TERMINATE)  break; // Thread 종료  else {  sleep\_for(std::chrono::milliseconds(500));  continue;  }  }    // Event 초기화  event\_no = event\_gen\_count + NUM\_EVENTS\_PER\_GEN \* myAddr;  event\_priority = targetEventGen - event\_gen\_count - 1;  event.setEventNo(event\_no);  event.setEventPri(event\_priority);  event.setEventGenAddr(myAddr);  event.setEventHandlerAddr(-1); // event handler is not defined yet !!  QueryPerformanceCounter(&t\_gen); // 시간 측정  event.setEventGenTime(t\_gen);  event.setEventStatus(GENERATED);  // T\_Entry 초기화  entry\_event.setKey(event.getEventPri());  entry\_event.setValue(event);  // 삽입을 실패한 경우  while (pPriQ\_Event->insert(entry\_event) == NULL) {  pParam->pCS\_main->lock(); // 콘솔 출력 관련 mutex  cout << "PriQ\_Event is Full, waiting ..." << endl;  pParam->pCS\_main->unlock();  sleep\_for(std::chrono::milliseconds(100));  }  pParam->pCS\_main->lock(); // 콘솔 출력 관련 mutex  cout << "Successfully inserted into PriQ\_Event " << endl;  pParam->pCS\_main->unlock();  pParam->pCS\_thrd\_mon->lock(); // 공유자원 관련 mutex  pThrdMon->eventGenerated[pThrdMon->totalEventGenerated] = event;  pThrdMon->numEventGenerated++;  pThrdMon->totalEventGenerated++;  pParam->pCS\_thrd\_mon->unlock();  event\_gen\_count++;  //Sleep(100 + rand() % 300);  sleep\_for(std::chrono::milliseconds(10));  }  } |
| /\* Thread\_EventHandler.cpp \*/  #include <Windows.h>  #include "Multi\_Thread.h"  #include "HeapPrioQ\_CS.h"  #include "Event.h"  #include <mutex>  using namespace std;  using std::this\_thread::sleep\_for;  void EventProc(ThreadParam\_Event\* pParam) {  // 구조체 변수  HeapPrioQ\_CS<int, Event>\* pPriQ\_Event = pParam->pPriQ\_Event;  int myRole = pParam->role;  int myAddr = pParam->myAddr;  int maxRound = pParam->maxRound;  int targetEventGen = pParam->targetEventGen;  LARGE\_INTEGER QP\_freq = pParam->QP\_freq;  ThreadStatusMonitor\* pThrdMon = pParam->pThrdMon;  T\_Entry<int, Event>\* pEntry;  Event event, \* pEvent, \* pEventProc;  int event\_no = 0;  int eventPriority = 0;  int event\_gen\_count = 0;  int num\_pkt\_processed = 0;  LARGE\_INTEGER t\_gen, t\_proc; // event 처리 시간  LONGLONG t\_diff;  double elapsed\_time;  for (int round = 0; round < maxRound; round++) {  if (\*pThrdMon->pFlagThreadTerminate == TERMINATE)  break; // Thread 종료  // Queue에 이벤트가 있는 경우  if (!pPriQ\_Event->isEmpty()) {  pEntry = pPriQ\_Event->removeHeapMin();  event = pEntry->getValue();  pParam->pCS\_thrd\_mon->lock(); // 공유자원 관련 mutex  event.setEventHandlerAddr(myAddr);  QueryPerformanceCounter(&t\_proc); // 처리 시간 측정  event.setEventProcTime(t\_proc);  t\_gen = event.getEventGenTime(); // 생성 시간 불러오기  t\_diff = t\_proc.QuadPart - t\_gen.QuadPart; // 생성 시간 - 처리 시간  elapsed\_time = ((double)t\_diff / QP\_freq.QuadPart); // in second  event.setEventElaspsedTime(elapsed\_time \* 1000); // in milli-second  pThrdMon->eventProcessed[pThrdMon->totalEventProcessed] = event;  pThrdMon->numEventProcessed++;  pThrdMon->totalEventProcessed++;  pParam->pCS\_thrd\_mon->unlock();  } // end if  sleep\_for(std::chrono::milliseconds(100 + rand() % 100));  } // end for  } |
|  |

**2. 2021-2 객체지향형 프로그래밍과 자료구조 실습 Oral Test**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 학번 | 21812158 | 성명 | 김주환 | 점수 |  |

|  |
| --- |
| **(1) 스레드를 생성하면서 인수 (parameter)를 전달하는 방법과 스레드를 소멸시키는 방법에 대하여 예를 들어 설명하라.**  스레드 파라메터 구조체 포인터(pParam)를 통하여, 스레드 생성 및 실행에 관련된 정보를 main() 함수로부터 전달받으며, 파라메터 구조체는 필요에 따라 정의    thread의 join() 함수를 사용하여 생성된 스레드가 스스로 함수 실행을 종료할 때까지 대기    기본족으로 자동 종료하지만, return이나 ExitThread 함수를 통해 종료코드를 반환하면서 종료시킬 수 있다.  TerminateThread는 종료코드를 반환하지 않고 바로 종료시킨다. |
| **(2) 이벤트를 생성하는 스레드와 이벤트를 처리하는 스레드가 분리되어 있는 멀티스레드 구조에서 이벤트의 생성으로부터 처리까지 경과된 시간을 측정하기 위하여 구현되어야 하는 기능을 상세하게 설명하라.**  Event의 생성에서 처리까지 경과된 시간을 측정하기 위해서는 각 스레드에서 특정 파라메터를 Event에 전달하여야 한다. 이에 대해 알아보겠다.  Event class는 아래의 시간 정보들을 가진다.    생성 스레드에서 이벤트 생성 시간을 저장한다.    처리 스레드에서 이벤트 처리 시간을 저장하고, 이전에 저장했던 생성시간과의 차이를 이용하여 생성으로부터 처리까지 걸린 시간을 구합니다. |
| **(3) 멀티스레드 구조의 병렬처리 시스템에서 공유자원 (예: 큐)을 다수의 스레드가 공유하는 경우 임계 구역 설정이 필요한 이유에 대하여 설명하고, 임계 구역을 생성, 설정 및 해제하는 방법에 대하여 예를 들어 설명하라.**  다중 스레드 사용을 지원하는 운영체제는 프로그램 실행 중에 스레드 또는 프로세스 간에 교체가 일어날 수 있게 하여, 다수의 스레드/프로세스가 병렬로 처리될 수 있도록 관리하여야 한다. Context switching이 일어나면, 현재 실행 중이던 스레드/프로세스의 중간 상태가 임시 저장되고, 다른 스레드/프로세스가 실행된다. 프로그램 실행 중에 특정 구역은 실행이 종료될 때까지 스레드/프로세서 교체가 일어나지 않도록 관리하여야 하는 경우가 있다.  은행 입금 및 출금 스레드 예시를 통해 critical section으로 보호하여야 하는 구역을 알아보겠다.    위의 경우는 입금과 출금 중간에 context switching이 발생한 경우다.  공유자원을 스레드 A에서 사용하는 중에 스레드 B에서 사용했기 때문에 문제가 발생했다.  이런 문제를 해결하기 위해서 critical section을 설정해주겠다.    mutex를 통해 critical section을 지정해서 각 스레드가 공유자원을 사용하는 중에 context switching이 일어나지 않도록 구현했다.  아래의 함수를 통해 세마포를 선언하고, critical section설정을 시작하고, 종료한다. |
| **(4) 멀티스레드 구조의 병렬 처리 시스템의 실행 상황을 모니터링 하는 방법에 대하여 설명하라.**    스레드의 진행 상황을 파악하기 위한 ThreadStatusMonitor구조체를 정의한다.  main() 함수에서 ThreadStatusMonitor 구조체 변수를 생성하고, 그 구조체 변수의 주소를 스레드 생성의 인수로 전달한다.  각 스레드의 진행상황(예 : 이벤트 생성, 이벤트 처리 등)을 ThreadStatusMonitor 구조체 변수에 기록한다.  main() 함수에서 주기적으로 threadStatusMonitor 구조체 변수의 내용을 파악 및 분석한다.  전체 스레드의 진행 상황을 한 눈에 쉽게 볼 수 있도록 현황판 형식으로 콘솔에 출력한다.  main문 내부의 반복문에서 각 라운드마다 주기적으로 상태를 출력합니다. 이 때 매 반복마다 ConsoleDisplay파일의 cls()로 실행창을 초기화하고 gotoxy()로 (0, 0)에서부터 다시 출력합니다. |