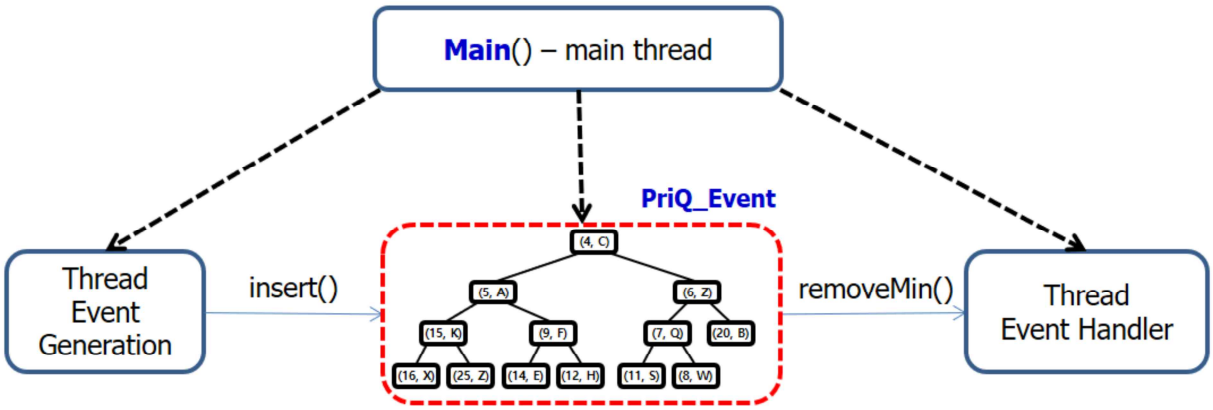
O-O Programming & Data Structure Lab. 9

9.1 Simulation of Priority-based Event Processing with Multi-thread and Heap Priority Queue



9.2 class Event

```
/* Event.h */
#include <iostream>
#include <string>
#include <fstream>
#include <iomanip>
#include <Windows.h>
#include <thread>
#include <process.h>
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; }
	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
	LARGE_INTEGER t_event_gen;
	LARGE_INTEGER t_event_proc;
	double t_elapsed_time_ms;
	EventStatus eventStatus;
};
```

```
Event* genRandEvent(int evt_no);
```

9.3 class T_Entry<K, V>

```
template<typename K, typename V>
class T_Entry
{
	friend ostream& operator<<(ostream& fout, T_Entry<K, V>& entry)
	{
		fout << "[" << entry.getKey() << ", " << *(entry.getValue()) << "]";
		return fout;
	}
public:
	T_Entry(K key, V value) { _key = key; _value = value; }
	T_Entry() { _key = 999; } // default constructor
	~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; }
	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;
}
```

9.4 class TA_Entry<K, V>

```
enum SortingDirection { INCREASING, DECREASING };

template<typename K, typename V>
class TA_Entry
{
public:
	TA_Entry(int n, string nm); // constructor
	~TA_Entry(); // destructor
	int size() { return num_elements; }
	bool empty() { return num_elements == 0; }
	string getName() { return name; }
	void reserve(int new_capacity);
	void insert(int i, T_Entry<K, V> element);
	void remove(int i);
	T_Entry<K, V>& at(int i);
	void set(int i, T_Entry<K, V>& element);
	T_Entry<K, V> getMin(int begin, int end);
	T_Entry<K, V> getMax(int begin, int end);
	void shuffle();
	int sequential_search(T_Entry<K, V> search_key); // search and return the index; -1 if not found
	int binary_search(T_Entry<K, V> search_key); // search and return the index; -1 if not found
	void selection_sort(SortingDirection sd);
	void quick_sort(SortingDirection sd);
	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_GA[index]; }
protected:
	T_Entry<K, V> *t_GA;
	int num_elements;
	int capacity;
	string name;
};
```

9.5 class CompleteBinaryTree<K, V>

```
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);
	T_Entry<K, V>& getEndElement() { return t_GA[end]; }
	T_Entry<K, V>& getRootElement() { return t_GA[CBT_ROOT]; }
	int getEndIndex() { return end; }
	void removeCBTEnd();
	void fprintCBT(ofstream &fout);
	void fprintCBT_byLevel(ofstream &fout);
protected:
	void _printCBT_byLevel(ofstream &fout, int p, int level);
	int parentIndex(int index) { return index / 2; }
	int leftChildIndex(int index) { return index * 2; }
	int rightChildIndex(int index) { return (index * 2 + 1); }
	bool hasLeftChild(int index) { return ((index * 2) <= end); }
	bool hasRightChild(int index) { return ((index * 2 + 1) <= end); }
	int end;
};
```

9.6 class HeapPrioQ<K, V>

```
template<typename K, typename V>
class HeapPrioQueue : public CompleteBinaryTree<K, V>
{
public:
	HeapPrioQueue(int capa, string nm);
	~HeapPrioQueue();
	bool isEmpty() { return size() == 0; }
	bool isFull() { return size() == capacity; }
	int insert(T_Entry<K, V>& elem);
	T_Entry<K, V>* removeHeapMin();
	T_Entry<K, V>* getHeapMin();
	void fprint(ofstream &fout);
	int size() {return end; }
private:
	mutex cs_priQ; // critical section for Heap Priority Queue
};
```

9.7 Simulation Parameters

```
/* 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 EVENT_PER_LINE 5

#endif
```

9.8 Multi_thread.h

```
/* Multi_thread.h */
#ifndef MULTI_THREAD_H
#define MULTI_THREAD_H
#include <iostream>
#include <fstream>
#include <Windows.h>
#include <thread>
#include <mutex>
```

```
#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 };

typedef struct ThreadStatusMonitor
{
	int numEventGenerated;
	int numEventProcessed;
	int totalEventGenerated;
	int totalEventProcessed;
	Event eventGenerated[TOTAL_NUM_EVENTS]; // used for monitoring only
	Event eventProcessed[TOTAL_NUM_EVENTS]; // used for monitoring only
	THREAD_FLAG *pFlagThreadTerminate;
}ThreadStatusMonitor;

typedef struct ThreadParam
{
	mutex *pCS_main;
	mutex *pCS_thrd_mon;
	HeapPrioQ_CS<int, Event> *pPriQ_Event;
	FILE *fout;
	ROLE role;
	int myAddr;
	int maxRound;
	int targetEventGen;
	LARGE_INTEGER QP_freq;
	ThreadStatusMonitor *pThrdMon;
}ThreadParam_Event;
#endif
```

9.10 Console Display for Thread Monitoring

```
/* 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
```

9.11 Thread_EventGen()

```
/* Thread_EventGenenerator.cpp */

. . . // 필요한 preprocessor 설정

void EventGen(ThreadParam_Event* pParam)
{
	ThreadParam_Event *pThrdParam;
	. . . . // 필요한 지역변수 선언

	pThrdParam = (ThreadParam_Event *)pParam;
	. . . . // 인수 전달


	for (int round = 0; round < maxRound; round++)
	{
```

```
            if (event_gen_count >= targetEventGen)
            {
                if (*pThrdMon->pFlagThreadTerminate == TERMINATE)
                    break;
                else {
                    Sleep(500);
                    continue;
                }
            }
            . . . . // 이벤트 생성 및 생성 시간 기록
            while (pPriQ_Event->insert(entry_event) == NULL)
                Sleep(50);
            . . . . // 스레드 모니터링 관련 정보 처리
            event_gen_count++;
            sleep_for(std::chrono::milliseconds(10));
        }
}
```

9.12 Thread_EventProc()

```
/* Thread_EventHandler.cpp */

. . . . // 필요한 preprocessor 설정

void EventProc(ThreadParam_Event* pParam)
{
    . . . .  // 필요한 지역변수 선언

    pThrdParam = (ThreadParam_Event *)pParam;
    . . . .  // 전달된 인수를 지역변수로 복사

    for (int round = 0; round < maxRound; round++)
    {
        if (*pThrdMon->pFlagThreadTerminate == TERMINATE)
            break;
        if (!pPriQ_Event->isEmpty())
        {
            . . . . //이벤트를 우선순위큐로부터 추출, 이벤트 처리시간 기록
            . . . . //스레드 모니터링 관련 정보 정리
        } // end if
        sleep_for(std::chrono::milliseconds(100 + rand() % 100));
    } // end for
}
```

9.13 main() function

The main() function should contain following procedure to manage student array.

```
/* main_EventGen_PriQ_EventHandler.cpp */

. . . . // 필요한 preprocessing 설정

void main()
{
    . . . . // 필요한 지역변수 선언
    HeapPrioQ_CS<int, Event> heapPriQ_Event(30, string("HeapPriorityQueue_Event"));
    Event *pEvent, *pEv_min_elasped_time, *pEv_max_elapsed_time;

    Event eventProcessed[TOTAL_NUM_EVENTS];

    . . . . // 초기화
    consHndlr = initConsoleHandler();
    QueryPerformanceFrequency(&QP_freq);
    srand(time(NULL));
```

```
    . . . . // 스레드 모니터링 준비

/* Create and Activate Thread_EventHandler */
. . . . // 이벤트 처리 스레드 생성을 위한 준비
thread thrd_EvProc(EventProc, &thrdParam_EventHndlr);
cs_main.lock();
printf("Thread_EventGen is created and activated ...\n");
cs_main.unlock();

/* Create and Activate Thread_EventGen */
. . . . // 이벤트 생성 스레드의 생성을 위한 준비
thread thrd_EvGen(EventGen, &thrdParam_EventGen);
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);
    . . . . // 현재까지 생성된 이벤트 들의 출력

    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) % EVENT_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();
        fout.close();

        printf("Hit any key to terminate : ");
        _getch();
}
```

9.14 Example of Output (첫 시작 부분)

```
Thread monitoring by main() ::
  round(13): current total_event_gen (41), total_event_proc(11)

*****************************
Events generated (current total = 41)
  Ev(no:  0, pri: 49) Ev(no:  1, pri: 48) Ev(no:  2, pri: 47) Ev(no:  3, pri: 46) Ev(no:  4, pri: 45)
  Ev(no:  5, pri: 44) Ev(no:  6, pri: 43) Ev(no:  7, pri: 42) Ev(no:  8, pri: 41) Ev(no:  9, pri: 40)
  Ev(no: 10, pri: 39) Ev(no: 11, pri: 38) Ev(no: 12, pri: 37) Ev(no: 13, pri: 36) Ev(no: 14, pri: 35)
  Ev(no: 15, pri: 34) Ev(no: 16, pri: 33) Ev(no: 17, pri: 32) Ev(no: 18, pri: 31) Ev(no: 19, pri: 30)
  Ev(no: 20, pri: 29) Ev(no: 21, pri: 28) Ev(no: 22, pri: 27) Ev(no: 23, pri: 26) Ev(no: 24, pri: 25)
  Ev(no: 25, pri: 24) Ev(no: 26, pri: 23) Ev(no: 27, pri: 22) Ev(no: 28, pri: 21) Ev(no: 29, pri: 20)
  Ev(no: 30, pri: 19) Ev(no: 31, pri: 18) Ev(no: 32, pri: 17) Ev(no: 33, pri: 16) Ev(no: 34, pri: 15)
  Ev(no: 35, pri: 14) Ev(no: 36, pri: 13) Ev(no: 37, pri: 12) Ev(no: 38, pri: 11) Ev(no: 39, pri: 10)
  Ev(no: 40, pri:  9)

*****************************
Events currently in Priority_Queue (31) :
  [Key: 8] [Key:26] [Key:17] [Key:32] [Key:27]
  [Key:23] [Key:18] [Key:39] [Key:33] [Key:31]
  [Key:28] [Key:35] [Key:24] [Key:22] [Key:19]
  [Key:49] [Key:43] [Key:46] [Key:34] [Key:47]
  [Key:40] [Key:42] [Key:29] [Key:48] [Key:38]
  [Key:44] [Key:25] [Key:45] [Key:36] [Key:37]
  [Key:20]

*****************************
Events processed (current total = 11):
  Ev(no: 8, pri:41, t_elapsed:   14.93) Ev(no:19, pri:30, t_elapsed:   14.87) Ev(no:28, pri:21, t_elapsed:    0.24) Ev(no:33, pri:16, t_elapsed:   14.96) Ev(no:34, pri:15, t_elapsed:  179.45)
  Ev(no:35, pri:14, t_elapsed:  298.95) Ev(no:36, pri:13, t_elapsed:  255.40) Ev(no:37, pri:12, t_elapsed:  285.24) Ev(no:38, pri:11, t_elapsed:  164.57) Ev(no:39, pri:10, t_elapsed:  314.14)
  Ev(no:40, pri: 9, t_elapsed:  165.06)
numEventProcessed = 11
min_elapsed_time =    0.24[ms]; Ev(no: 28, pri: 21)
max_elapsed_time =  314.14[ms]; Ev(no: 39, pri: 10)
avg_elapsed_time =  155.26[ms];
Successfully inserted into PriQ_Event
```

9.15 Example of Output (종료 단계)

```
Thread monitoring by main() ::
  round(59): current total_event_gen (50), total_event_proc(50)

*****************************
Events generated (current total = 50)
  Ev(no:  0, pri: 49) Ev(no:  1, pri: 48) Ev(no:  2, pri: 47) Ev(no:  3, pri: 46) Ev(no:  4, pri: 45)
  Ev(no:  5, pri: 44) Ev(no:  6, pri: 43) Ev(no:  7, pri: 42) Ev(no:  8, pri: 41) Ev(no:  9, pri: 40)
  Ev(no: 10, pri: 39) Ev(no: 11, pri: 38) Ev(no: 12, pri: 37) Ev(no: 13, pri: 36) Ev(no: 14, pri: 35)
  Ev(no: 15, pri: 34) Ev(no: 16, pri: 33) Ev(no: 17, pri: 32) Ev(no: 18, pri: 31) Ev(no: 19, pri: 30)
  Ev(no: 20, pri: 29) Ev(no: 21, pri: 28) Ev(no: 22, pri: 27) Ev(no: 23, pri: 26) Ev(no: 24, pri: 25)
  Ev(no: 25, pri: 24) Ev(no: 26, pri: 23) Ev(no: 27, pri: 22) Ev(no: 28, pri: 21) Ev(no: 29, pri: 20)
  Ev(no: 30, pri: 19) Ev(no: 31, pri: 18) Ev(no: 32, pri: 17) Ev(no: 33, pri: 16) Ev(no: 34, pri: 15)
  Ev(no: 35, pri: 14) Ev(no: 36, pri: 13) Ev(no: 37, pri: 12) Ev(no: 38, pri: 11) Ev(no: 39, pri: 10)
  Ev(no: 40, pri:  9) Ev(no: 41, pri:  8) Ev(no: 42, pri:  7) Ev(no: 43, pri:  6) Ev(no: 44, pri:  5)
  Ev(no: 45, pri:  4) Ev(no: 46, pri:  3) Ev(no: 47, pri:  2) Ev(no: 48, pri:  1) Ev(no: 49, pri:  0)


*****************************
Events currently in Priority_Queue (0) :
  HeapPriorityQueue is Empty !!


*****************************
Events processed (current total = 50):
  Ev(no: 9, pri:40, t_elapsed:    0.02) Ev(no:19, pri:30, t_elapsed:   15.56) Ev(no:27, pri:22, t_elapsed:   15.56) Ev(no:33, pri:16, t_elapsed:   15.30) Ev(no:34, pri:15, t_elapsed:  181.14)
  Ev(no:35, pri:14, t_elapsed:  195.51) Ev(no:36, pri:13, t_elapsed:  256.52) Ev(no:37, pri:12, t_elapsed:  300.83) Ev(no:38, pri:11, t_elapsed:  242.72) Ev(no:39, pri:10, t_elapsed:  287.08)
  Ev(no:40, pri: 9, t_elapsed:  166.21) Ev(no:41, pri: 8, t_elapsed:  197.23) Ev(no:42, pri: 7, t_elapsed:  256.54) Ev(no:43, pri: 6, t_elapsed:  257.69) Ev(no:44, pri: 5, t_elapsed:  152.14)
  Ev(no:45, pri: 4, t_elapsed:  333.34) Ev(no:46, pri: 3, t_elapsed:  272.30) Ev(no:47, pri: 2, t_elapsed:  287.76) Ev(no:48, pri: 1, t_elapsed:  181.66) Ev(no:49, pri: 0, t_elapsed:  151.08)
  Ev(no:32, pri:17, t_elapsed: 2797.46) Ev(no:31, pri:18, t_elapsed: 2919.16) Ev(no:30, pri:19, t_elapsed: 3039.69) Ev(no:29, pri:20, t_elapsed: 3237.35) Ev(no:28, pri:21, t_elapsed: 3448.09)
  Ev(no:26, pri:23, t_elapsed: 3676.02) Ev(no:25, pri:24, t_elapsed: 3827.05) Ev(no:24, pri:25, t_elapsed: 3962.10) Ev(no:23, pri:26, t_elapsed: 4096.92) Ev(no:22, pri:27, t_elapsed: 4323.02)
  Ev(no:21, pri:28, t_elapsed: 4488.90) Ev(no:20, pri:29, t_elapsed: 4639.90) Ev(no:18, pri:31, t_elapsed: 4851.13) Ev(no:17, pri:32, t_elapsed: 5017.19) Ev(no:16, pri:33, t_elapsed: 5212.96)
  Ev(no:15, pri:34, t_elapsed: 5348.90) Ev(no:14, pri:35, t_elapsed: 5531.42) Ev(no:13, pri:36, t_elapsed: 5758.35) Ev(no:12, pri:37, t_elapsed: 5909.41) Ev(no:11, pri:38, t_elapsed: 6121.19)
  Ev(no:10, pri:39, t_elapsed: 6241.40) Ev(no: 8, pri:41, t_elapsed: 6392.39) Ev(no: 7, pri:42, t_elapsed: 6544.42) Ev(no: 6, pri:43, t_elapsed: 6694.82) Ev(no: 5, pri:44, t_elapsed: 6889.63)
  Ev(no: 4, pri:45, t_elapsed: 7071.83) Ev(no: 3, pri:46, t_elapsed: 7237.39) Ev(no: 2, pri:47, t_elapsed: 7373.02) Ev(no: 1, pri:48, t_elapsed: 7552.56) Ev(no: 0, pri:49, t_elapsed: 7758.17)

numEventProcessed = 50
min_elapsed_time =     0.02[ms]; Ev(no:  9, pri: 40)
max_elapsed_time =  7750.17[ms]; Ev(no:  0, pri: 49)
avg_elapsed_time =  3234.56[ms];
```

9.16 Oral Test

(1) 스레드를 생성하면서 인수 (parameter)를 전달하는 방법과 스레드를 소멸시키는 방법에 대하여 예를 들어 설명하라.

(2) 이벤트를 생성하는 스레드와 이벤트를 처리하는 스레드가 분리되어 있는 멀티스레드 구조에서 이벤트의 생성으로부터 처리까지 경과된 시간을 측정하기 위하여 구현되어야 하는 기능을 상세하게 설명하라.

(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 해제 하는 방법에 대하여 예를 들어 설명하라.

(4) 멀티스레드 구조의 병렬 처리 시스템의 실행 상황을 모니터링 하는 방법에 대하여 설명하라.