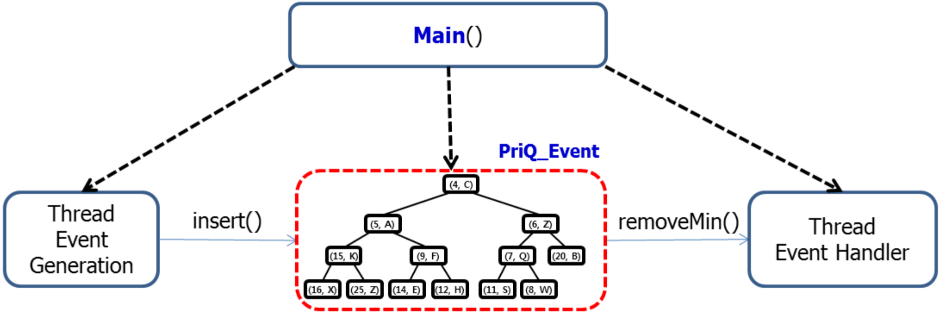
2021-1 프로그래밍언어 실습 12

12.1 우선순위 기반의 이벤트(사건) 처리 시뮬레이션을 위한 Multi-thread

(1) Functional Block Diagram



(2) Event

```
/* Event.h */

#ifndef EVENT_H
#define EVENT_H

#include <stdio.h>
#include <Windows.h>

#define NUM_PRIORITY 100
#define EVENT_PER_LINE 5

enum EventStatus { GENERATED, ENQUEUED, PROCESSED, UNDEFINED };
extern const char *strEventStatus[];

typedef struct
{
	int ev_no;
	int ev_generator;
	int ev_handler;
	int ev_pri;	// ev_priority
	LARGE_INTEGER ev_t_gen; // for performance monitoring
	LARGE_INTEGER ev_t_handle; // for performance monitoring
	double elap_time; // for performance monitoring
	EventStatus eventStatus;
} Event;

void printEvent(Event* pEv);
void fprintEvent(FILE *fout, Event* pEv);
Event *genEvent(Event *pEv, int event_Gen_ID, int ev_no, int ev_pri);
void calc_elapsed_time(Event* pEv, LARGE_INTEGER freq);
void printEvent_withTime(Event* pEv);

#endif
```

12.2 Priority Queue

(1) Priority Queue

```
/* PriorityQueue_Event.h */

#ifndef PRIORITY_QUEUE_H
#define PRIORITY_QUEUE_H
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Event.h"

#define POS_ROOT 1
#define MAX_NAME_LEN 80
#define TRUE 1
#define FALSE 0


typedef struct CBTN_Event
{
	int priority;
	Event event;
} CBTN_Event;

typedef struct PriorityQueue
{
	char PriQ_name[MAX_NAME_LEN];
	int priQ_capacity;
	int priQ_size;
	int pos_last;
	CBTN_Event *pCBT_Event;
	mutex cs_PriQ;
} PriQ_Event;

PriQ_Event *initPriQ_Event(PriQ_Event *pPriQ_Event, const char *name, int capacity);
Event *enPriQ_Event(PriQ_Event *pPriQ_Event, Event ev);
Event *dePriQ_Event(PriQ_Event *pPriQ_Event);
void printPriQ_Event(PriQ_Event *pPriQ_Event);
void fprintPriQ_Event(FILE *fout, PriQ_Event *pPriQ_Event);
void deletePriQ_Event(PriQ_Event *pPriQ_Event);
#endif
```

12.3 시뮬레이션 구성

(1) SimParam.h

```
/* 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 1
#define PLUS_INF INT_MAX
#define MAX_ROUND 1000

#endif
```

12.4 Event 생성 및 처리 Thread

(1) Thread.h

```
#include "SimParams.h"

using namespace std;
```

```
enum ROLE {EVENT_GENERATOR, EVENT_HANDLER};
enum THREAD_FLAG {INITIALIZE, RUN, TERMINATE};

#define THREAD_RETURN_CODE 7
typedef struct
{
    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
{
    mutex *pMTX_main;
    mutex *pMTX_thrd_mon;
    PriQ_Event *pPriQ_Event;
    ROLE role;
    int myAddr;
    int maxRound;
    int targetEventGen;
    ThreadStatusMonitor *pThrdMon;
} ThreadParam_Event;

void Thread_EventHandler(ThreadParam_Event *pParam);
void Thread_EventGenerator(ThreadParam_Event *pParam);
#endif
```

(2) Thread Event Generator

```
/* Thread_EventGenenerator.cpp */

#include <Windows.h>
#include <time.h>
#include "Thread.h"
#include "PriQ_Event.h"
#include "Event.h"

void Thread_EventGenerator(ThreadParam_Event* pParam)
{
    PriQ_Event *pPriQ_Event = pParam->pPriQ_Event;
    int myRole = pParam->role;
    int myAddr = pParam->myAddr;
    int maxRound = pParam->maxRound;
    int event_gen_count = 0;
    ThreadStatusMonitor *pThrdMon = pParam->pThrdMon;
    pPriQ_Event = pParam->pPriQ_Event;
    int targetEventGen = pParam->targetEventGen;
    Event* pEv;

    srand(time(NULL));
    for (int round = 0; round < maxRound; round++)
    {
        if (event_gen_count >= targetEventGen)
        {
            if (*pThrdMon->pFlagThreadTerminate == TERMINATE)
```

```
                break;
            else {
                Sleep(500);
                continue;
            }
        }
        pEv = (Event *)calloc(1, sizeof(Event));
        pEv->ev_generator = myAddr;
        pEv->ev_handler = -1; // event handler is not defined yet !!
        pEv->ev_no = event_gen_count + NUM_EVENTS_PER_GEN*myAddr;
        //pEv->ev_pri = eventPriority = rand() % NUM_PRIORITY;
        pEv->ev_pri = targetEventGen - event_gen_count -1;
        QueryPerformanceCounter(&pEv->ev_t_gen);
        pThrdMon->eventGenerated[pThrdMon->totalEventGenerated] = *pEv;
        while (enPriQ_Event(pPriQ_Event, *pEv) == NULL)
        {
            Sleep(500);
        }
        free(pEv);
        pParam->pMTX_thrd_mon->lock();
        pThrdMon->numEventGenerated++;
        pThrdMon->totalEventGenerated++;
        pParam->pMTX_thrd_mon->unlock();
        event_gen_count++;
        //Sleep(100 + rand() % 300);
        Sleep(10);
    }
}
```

(3) Event Handler

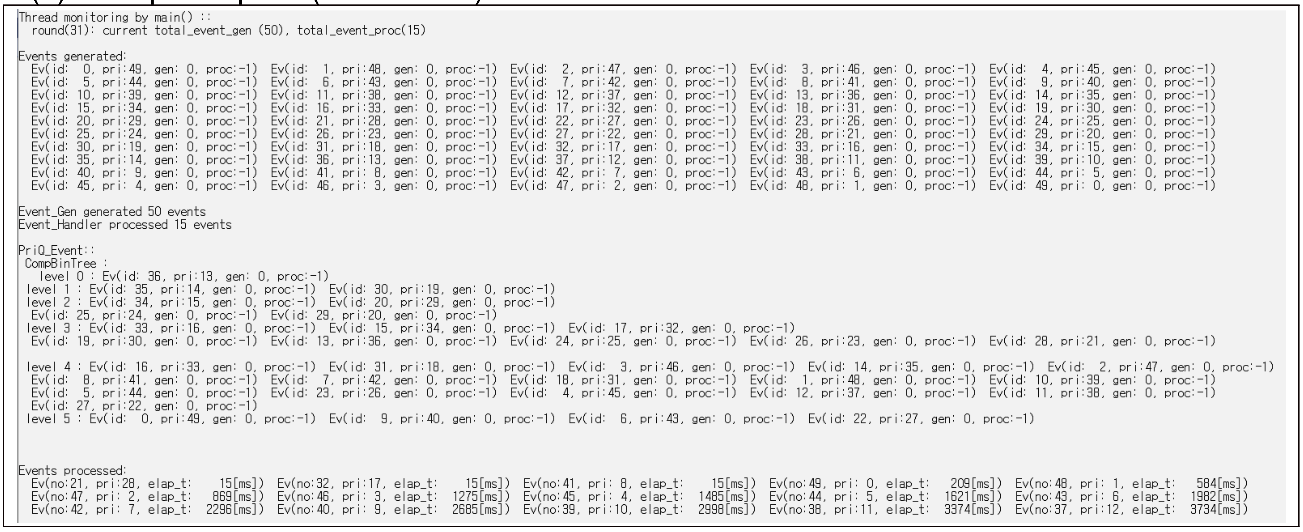
```
/* Thread_EventHandler.cpp */

#include <Windows.h>
#include <time.h>
#include "Thread.h"
#include "PriQ_Event.h"
#include "Event.h"

void Thread_EventHandler(ThreadParam_Event* pParam)
{
    Event *pEv, *pEvProc, ev;
    int myRole = pParam->role;
    int myAddr = pParam->myAddr;
    PriQ_Event* pPriQ_Event = pParam->pPriQ_Event;
    ThreadStatusMonitor* pThrdMon = pParam->pThrdMon;
    int maxRound = pParam->maxRound;
    int targetEventGen = pParam->targetEventGen;

    srand(time(NULL));
    for (int round = 0; round < maxRound; round++)
    {
        if (*pThrdMon->pFlagThreadTerminate == TERMINATE)
                break;
        if ((pEv = dePriQ_Event(pPriQ_Event)) != NULL)
        {
            //printf("Thread_EventProc::deLL_EventQ_from_HighPri_LL_EventQ : ");
            //printEvent(pEv);
            //printf("\n");
            pParam->pMTX_thrd_mon->lock();
            QueryPerformanceCounter(&pEv->ev_t_handle);
            pEv->ev_handler = myAddr;
            pThrdMon->eventProcessed[pThrdMon->totalEventProcessed] = *pEv;
            pThrdMon->numEventProcessed++;
            pThrdMon->totalEventProcessed++;
            pParam->pMTX_thrd_mon->unlock();
            free(pEv);
        }
        Sleep(100 + rand() % 300);
    }

}
```

12.4 main() function

(1) Example of main() function

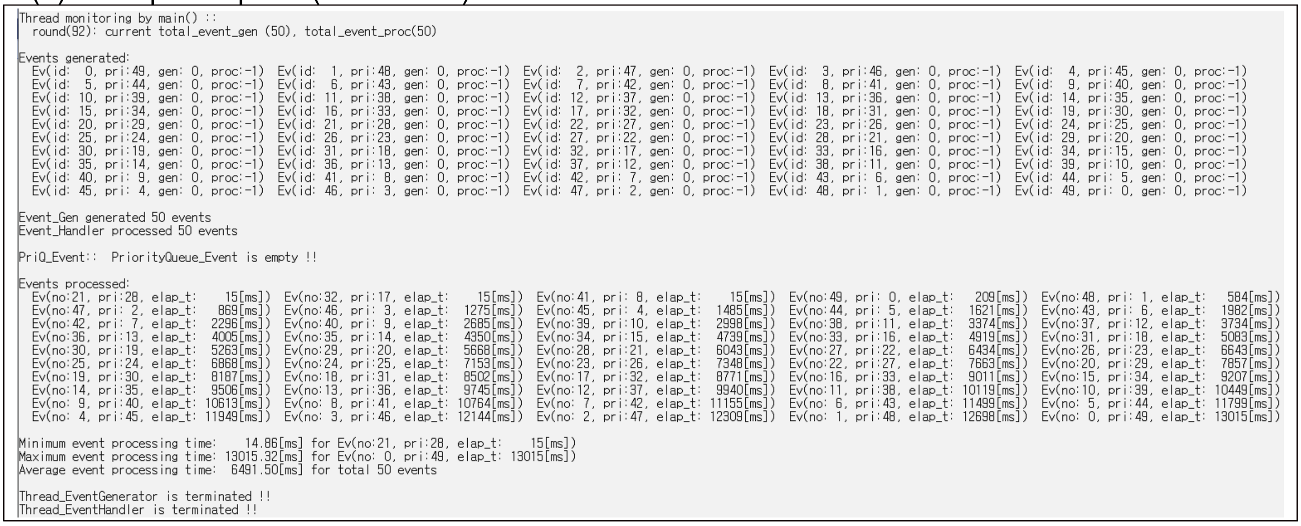
```
/* main_EventGen_CirQ_EventHandler.cpp */

#include <stdio.h>
#include <stdlib.h>
#include <Windows.h>
#include <mutex>
#include "Thread.h"
#include "PriQ_Event.h"
#include "Event.h"
#include "ConsoleDisplay.h"

using namespace std;

void main()
{
    PriQ_Event priQ_Event;
    Event *pEv;
    int myAddr = 0;
    int ev_handler, eventPriority;

    initPriQ_Event(&priQ_Event, "PriQ_Event", 1);

    ThreadParam_Event thrdParam_EventGen, thrdParam_EventHndlr;
    HANDLE hThrd_EventGenerator, hThrd_EventHandler;
    mutex cs_main; // console display
    mutex cs_thrd_mon; // thread monitoring
    ThreadStatusMonitor thrdMon;
    HANDLE consHndlr;
    THREAD_FLAG eventThreadFlag = RUN;
    int count, numEventGenerated, numEventProcessed;
    LARGE_INTEGER freq;

    consHndlr = initConsoleHandler();

    thrdMon.pFlagThreadTerminate = &eventThreadFlag;
    thrdMon.totalEventGenerated = 0;
    thrdMon.totalEventProcessed = 0;
    for (int ev = 0; ev < TOTAL_NUM_EVENTS; ev++)
    {
        thrdMon.eventProcessed[ev].ev_no = -1; // mark as not-processed
        thrdMon.eventProcessed[ev].ev_pri = -1;
    }
    QueryPerformanceFrequency(&freq);
    /* Create and Activate Thread_EventHandler */

    thrdMon.numEventProcessed = 0;
    thrdParam_EventHndlr.role = EVENT_HANDLER;
    thrdParam_EventHndlr.myAddr = 1; // link address
    thrdParam_EventHndlr.pMTX_main = &cs_main;
    thrdParam_EventHndlr.pMTX_thrd_mon = &cs_thrd_mon;
    thrdParam_EventHndlr.pPriQ_Event = &priQ_Event;
    thrdParam_EventHndlr.maxRound = MAX_ROUND;
    thrdParam_EventHndlr.pThrdMon = &thrdMon;

    thread thrd_ev_handler(Thread_EventHandler, &thrdParam_EventHndlr);
    cs_main.lock();
    printf("Thread_EventHandler 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.pMTX_main = &cs_main;
    thrdParam_EventGen.pMTX_thrd_mon = &cs_thrd_mon;
    thrdParam_EventGen.pPriQ_Event = &priQ_Event;
    thrdParam_EventGen.targetEventGen = NUM_EVENTS_PER_GEN;
    thrdParam_EventGen.maxRound = MAX_ROUND;
    thrdParam_EventGen.pThrdMon = &thrdMon;

    thread thrd_ev_generator (Thread_EventGenerator, &thrdParam_EventGen);
    cs_main.lock();
    printf("Thread_EventGen is created and activated ...\n");
    cs_main.unlock();

    for (int round = 0; round < MAX_ROUND; round++)
    {
        //cs_main.lock();
        system("cls");
        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");
        printf("Events generated: \n    ");
        count = 0;
        numEventGenerated = thrdMon.totalEventGenerated;
        for (int i = 0; i < numEventGenerated; i++)
        {
            pEv = &thrdMon.eventGenerated[i];
            if (pEv != NULL)
            {
                printEvent(pEv);
                if (((i + 1) % EVENT_PER_LINE) == 0)
                    printf("\n    ");
            }
        }
        printf("\n");
        printf("Event_Gen generated %2d events\n", thrdMon.numEventGenerated);
        printf("Event_Handler processed %2d events\n", thrdMon.numEventProcessed);
        printf("\n");
        printf("PriQ_Event::");    printPriQ_Event(&priQ_Event);
        printf("\n");
        printf("Events processed: \n    ");
        count = 0;
        numEventProcessed = thrdMon.totalEventProcessed;
        for (int i = 0; i < numEventProcessed; i++)
        {
            pEv = &thrdMon.eventProcessed[i];
            if (pEv != NULL)
            {
                calc_elapsed_time(pEv, freq);
                printEvent_withTime(pEv);
                if (((i + 1) % EVENT_PER_LINE) == 0)
                    printf("\n    ");
            }
        }
        printf("\n");

        if (numEventProcessed >= TOTAL_NUM_EVENTS)
        {
            eventThreadFlag = TERMINATE; // set 1 to terminate threads
            break;
        }

        //cs_main.unlock();
        Sleep(100);
    }

    /* Analyze the event processing times */
    double min, max, avg, sum;
    int min_ev, max_ev;
    min = max = sum = thrdMon.eventProcessed[0].elap_time;
    min_ev = max_ev = 0;
    for (int i = 1; i < TOTAL_NUM_EVENTS; i++)
    {
        sum += thrdMon.eventProcessed[i].elap_time;
        if (min > thrdMon.eventProcessed[i].elap_time)
        {
            min = thrdMon.eventProcessed[i].elap_time;
            min_ev = i;
        }
        if (max < thrdMon.eventProcessed[i].elap_time)
        {
            max = thrdMon.eventProcessed[i].elap_time;
            max_ev = i;
        }
    }
    avg = sum / TOTAL_NUM_EVENTS;
    printf("Minimum event processing time: %8.2lf[ms] for ", min * 1000);
    printEvent_withTime(&thrdMon.eventProcessed[min_ev]); printf("\n");
    printf("Maximum event processing time: %8.2lf[ms] for ", max * 1000);
    printEvent_withTime(&thrdMon.eventProcessed[max_ev]); printf("\n");
    printf("Average event processing time: %8.2lf[ms] for total %d events\n", avg * 1000,
           TOTAL_NUM_EVENTS);
    printf("\n");

    thrd_ev_generator.join();
    printf("Thread_EventGenerator is terminated !!\n");
    thrd_ev_handler.join();
    printf("Thread_EventHandler is terminated !!\n");
}
```

(2) Example output 1 (initial status)



```
Thread monitoring by main() ::
  round(31): current total_event_gen (50), total_event_proc(15)

Events generated:
  Ev(id:  0, pri:49, gen: 0, proc:-1)  Ev(id:  1, pri:48, gen: 0, proc:-1)  Ev(id:  2, pri:47, gen: 0, proc:-1)  Ev(id:  3, pri:46, gen: 0, proc:-1)  Ev(id:  4, pri:45, gen: 0, proc:-1)
  Ev(id:  5, pri:44, gen: 0, proc:-1)  Ev(id:  6, pri:43, gen: 0, proc:-1)  Ev(id:  7, pri:42, gen: 0, proc:-1)  Ev(id:  8, pri:41, gen: 0, proc:-1)  Ev(id:  9, pri:40, gen: 0, proc:-1)
  Ev(id: 10, pri:39, gen: 0, proc:-1)  Ev(id: 11, pri:38, gen: 0, proc:-1)  Ev(id: 12, pri:37, gen: 0, proc:-1)  Ev(id: 13, pri:36, gen: 0, proc:-1)  Ev(id: 14, pri:35, gen: 0, proc:-1)
  Ev(id: 15, pri:34, gen: 0, proc:-1)  Ev(id: 16, pri:33, gen: 0, proc:-1)  Ev(id: 17, pri:32, gen: 0, proc:-1)  Ev(id: 18, pri:31, gen: 0, proc:-1)  Ev(id: 19, pri:30, gen: 0, proc:-1)
  Ev(id: 20, pri:29, gen: 0, proc:-1)  Ev(id: 21, pri:28, gen: 0, proc:-1)  Ev(id: 22, pri:27, gen: 0, proc:-1)  Ev(id: 23, pri:26, gen: 0, proc:-1)  Ev(id: 24, pri:25, gen: 0, proc:-1)
  Ev(id: 25, pri:24, gen: 0, proc:-1)  Ev(id: 26, pri:23, gen: 0, proc:-1)  Ev(id: 27, pri:22, gen: 0, proc:-1)  Ev(id: 28, pri:21, gen: 0, proc:-1)  Ev(id: 29, pri:20, gen: 0, proc:-1)
  Ev(id: 30, pri:19, gen: 0, proc:-1)  Ev(id: 31, pri:18, gen: 0, proc:-1)  Ev(id: 32, pri:17, gen: 0, proc:-1)  Ev(id: 33, pri:16, gen: 0, proc:-1)  Ev(id: 34, pri:15, gen: 0, proc:-1)
  Ev(id: 35, pri:14, gen: 0, proc:-1)  Ev(id: 36, pri:13, gen: 0, proc:-1)  Ev(id: 37, pri:12, gen: 0, proc:-1)  Ev(id: 38, pri:11, gen: 0, proc:-1)  Ev(id: 39, pri:10, gen: 0, proc:-1)
  Ev(id: 40, pri: 9, gen: 0, proc:-1)  Ev(id: 41, pri: 8, gen: 0, proc:-1)  Ev(id: 42, pri: 7, gen: 0, proc:-1)  Ev(id: 43, pri: 6, gen: 0, proc:-1)  Ev(id: 44, pri: 5, gen: 0, proc:-1)
  Ev(id: 45, pri: 4, gen: 0, proc:-1)  Ev(id: 46, pri: 3, gen: 0, proc:-1)  Ev(id: 47, pri: 2, gen: 0, proc:-1)  Ev(id: 48, pri: 1, gen: 0, proc:-1)  Ev(id: 49, pri: 0, gen: 0, proc:-1)

Event_Gen generated 50 events
Event_Handler processed 15 events

PriQ_Event::
 CompBinTree :
   level 0 : Ev(id: 36, pri:13, gen: 0, proc:-1)
 level 1 : Ev(id: 35, pri:14, gen: 0, proc:-1)  Ev(id: 30, pri:19, gen: 0, proc:-1)
 level 2 : Ev(id: 34, pri:15, gen: 0, proc:-1)  Ev(id: 20, pri:29, gen: 0, proc:-1)
  Ev(id: 25, pri:24, gen: 0, proc:-1)  Ev(id: 29, pri:20, gen: 0, proc:-1)
 level 3 : Ev(id: 33, pri:16, gen: 0, proc:-1)  Ev(id: 15, pri:34, gen: 0, proc:-1)  Ev(id: 17, pri:32, gen: 0, proc:-1)
  Ev(id: 19, pri:30, gen: 0, proc:-1)  Ev(id: 13, pri:36, gen: 0, proc:-1)  Ev(id: 24, pri:25, gen: 0, proc:-1)  Ev(id: 26, pri:23, gen: 0, proc:-1)  Ev(id: 28, pri:21, gen: 0, proc:-1)

 level 4 : Ev(id: 16, pri:33, gen: 0, proc:-1)  Ev(id: 31, pri:18, gen: 0, proc:-1)  Ev(id:  3, pri:46, gen: 0, proc:-1)  Ev(id: 14, pri:35, gen: 0, proc:-1)  Ev(id:  2, pri:47, gen: 0, proc:-1)
  Ev(id:  8, pri:41, gen: 0, proc:-1)  Ev(id:  7, pri:42, gen: 0, proc:-1)  Ev(id: 18, pri:31, gen: 0, proc:-1)  Ev(id:  1, pri:48, gen: 0, proc:-1)  Ev(id: 10, pri:39, gen: 0, proc:-1)
  Ev(id:  5, pri:44, gen: 0, proc:-1)  Ev(id: 23, pri:26, gen: 0, proc:-1)  Ev(id:  4, pri:45, gen: 0, proc:-1)  Ev(id: 12, pri:37, gen: 0, proc:-1)  Ev(id: 11, pri:38, gen: 0, proc:-1)
  Ev(id: 27, pri:22, gen: 0, proc:-1)
 level 5 : Ev(id:  0, pri:49, gen: 0, proc:-1)  Ev(id:  9, pri:40, gen: 0, proc:-1)  Ev(id:  6, pri:43, gen: 0, proc:-1)  Ev(id: 22, pri:27, gen: 0, proc:-1)



Events processed:
  Ev(no:21, pri:28, elap_t:    15[ms])  Ev(no:32, pri:17, elap_t:    15[ms])  Ev(no:41, pri: 8, elap_t:    15[ms])  Ev(no:49, pri: 0, elap_t:   209[ms])  Ev(no:48, pri: 1, elap_t:   584[ms])
  Ev(no:47, pri: 2, elap_t:   869[ms])  Ev(no:46, pri: 3, elap_t:  1275[ms])  Ev(no:45, pri: 4, elap_t:  1485[ms])  Ev(no:44, pri: 5, elap_t:  1621[ms])  Ev(no:43, pri: 6, elap_t:  1982[ms])
  Ev(no:42, pri: 7, elap_t:  2296[ms])  Ev(no:40, pri: 9, elap_t:  2685[ms])  Ev(no:39, pri:10, elap_t:  2998[ms])  Ev(no:38, pri:11, elap_t:  3374[ms])  Ev(no:37, pri:12, elap_t:  3734[ms])
```

(3) Example output 2 (final status)



```
Thread monitoring by main() ::
  round(92): current total_event_gen (50), total_event_proc(50)

Events generated:
  Ev(id:  0, pri:49, gen: 0, proc:-1)  Ev(id:  1, pri:48, gen: 0, proc:-1)  Ev(id:  2, pri:47, gen: 0, proc:-1)  Ev(id:  3, pri:46, gen: 0, proc:-1)  Ev(id:  4, pri:45, gen: 0, proc:-1)
  Ev(id:  5, pri:44, gen: 0, proc:-1)  Ev(id:  6, pri:43, gen: 0, proc:-1)  Ev(id:  7, pri:42, gen: 0, proc:-1)  Ev(id:  8, pri:41, gen: 0, proc:-1)  Ev(id:  9, pri:40, gen: 0, proc:-1)
  Ev(id: 10, pri:39, gen: 0, proc:-1)  Ev(id: 11, pri:38, gen: 0, proc:-1)  Ev(id: 12, pri:37, gen: 0, proc:-1)  Ev(id: 13, pri:36, gen: 0, proc:-1)  Ev(id: 14, pri:35, gen: 0, proc:-1)
  Ev(id: 15, pri:34, gen: 0, proc:-1)  Ev(id: 16, pri:33, gen: 0, proc:-1)  Ev(id: 17, pri:32, gen: 0, proc:-1)  Ev(id: 18, pri:31, gen: 0, proc:-1)  Ev(id: 19, pri:30, gen: 0, proc:-1)
  Ev(id: 20, pri:29, gen: 0, proc:-1)  Ev(id: 21, pri:28, gen: 0, proc:-1)  Ev(id: 22, pri:27, gen: 0, proc:-1)  Ev(id: 23, pri:26, gen: 0, proc:-1)  Ev(id: 24, pri:25, gen: 0, proc:-1)
  Ev(id: 25, pri:24, gen: 0, proc:-1)  Ev(id: 26, pri:23, gen: 0, proc:-1)  Ev(id: 27, pri:22, gen: 0, proc:-1)  Ev(id: 28, pri:21, gen: 0, proc:-1)  Ev(id: 29, pri:20, gen: 0, proc:-1)
  Ev(id: 30, pri:19, gen: 0, proc:-1)  Ev(id: 31, pri:18, gen: 0, proc:-1)  Ev(id: 32, pri:17, gen: 0, proc:-1)  Ev(id: 33, pri:16, gen: 0, proc:-1)  Ev(id: 34, pri:15, gen: 0, proc:-1)
  Ev(id: 35, pri:14, gen: 0, proc:-1)  Ev(id: 36, pri:13, gen: 0, proc:-1)  Ev(id: 37, pri:12, gen: 0, proc:-1)  Ev(id: 38, pri:11, gen: 0, proc:-1)  Ev(id: 39, pri:10, gen: 0, proc:-1)
  Ev(id: 40, pri: 9, gen: 0, proc:-1)  Ev(id: 41, pri: 8, gen: 0, proc:-1)  Ev(id: 42, pri: 7, gen: 0, proc:-1)  Ev(id: 43, pri: 6, gen: 0, proc:-1)  Ev(id: 44, pri: 5, gen: 0, proc:-1)
  Ev(id: 45, pri: 4, gen: 0, proc:-1)  Ev(id: 46, pri: 3, gen: 0, proc:-1)  Ev(id: 47, pri: 2, gen: 0, proc:-1)  Ev(id: 48, pri: 1, gen: 0, proc:-1)  Ev(id: 49, pri: 0, gen: 0, proc:-1)

Event_Gen generated 50 events
Event_Handler processed 50 events

PriQ_Event::  PriorityQueue_Event is empty !!

Events processed:
  Ev(no:21, pri:28, elap_t:    15[ms])  Ev(no:32, pri:17, elap_t:    15[ms])  Ev(no:41, pri: 8, elap_t:    15[ms])  Ev(no:49, pri: 0, elap_t:   209[ms])  Ev(no:48, pri: 1, elap_t:   584[ms])
  Ev(no:47, pri: 2, elap_t:   869[ms])  Ev(no:46, pri: 3, elap_t:  1275[ms])  Ev(no:45, pri: 4, elap_t:  1485[ms])  Ev(no:44, pri: 5, elap_t:  1621[ms])  Ev(no:43, pri: 6, elap_t:  1982[ms])
  Ev(no:42, pri: 7, elap_t:  2296[ms])  Ev(no:40, pri: 9, elap_t:  2685[ms])  Ev(no:39, pri:10, elap_t:  2998[ms])  Ev(no:38, pri:11, elap_t:  3374[ms])  Ev(no:37, pri:12, elap_t:  3734[ms])
  Ev(no:36, pri:13, elap_t:  4005[ms])  Ev(no:35, pri:14, elap_t:  4350[ms])  Ev(no:34, pri:15, elap_t:  4739[ms])  Ev(no:33, pri:16, elap_t:  4919[ms])  Ev(no:31, pri:18, elap_t:  5083[ms])
  Ev(no:30, pri:19, elap_t:  5263[ms])  Ev(no:29, pri:20, elap_t:  5668[ms])  Ev(no:28, pri:21, elap_t:  6043[ms])  Ev(no:27, pri:22, elap_t:  6434[ms])  Ev(no:26, pri:23, elap_t:  6643[ms])
  Ev(no:25, pri:24, elap_t:  6868[ms])  Ev(no:24, pri:25, elap_t:  7153[ms])  Ev(no:23, pri:26, elap_t:  7348[ms])  Ev(no:22, pri:27, elap_t:  7663[ms])  Ev(no:20, pri:29, elap_t:  7857[ms])
  Ev(no:19, pri:30, elap_t:  8187[ms])  Ev(no:18, pri:31, elap_t:  8502[ms])  Ev(no:17, pri:32, elap_t:  8771[ms])  Ev(no:16, pri:33, elap_t:  9011[ms])  Ev(no:15, pri:34, elap_t:  9207[ms])
  Ev(no:14, pri:35, elap_t:  9506[ms])  Ev(no:13, pri:36, elap_t:  9745[ms])  Ev(no:12, pri:37, elap_t:  9940[ms])  Ev(no:11, pri:38, elap_t: 10119[ms])  Ev(no:10, pri:39, elap_t: 10449[ms])
  Ev(no: 9, pri:40, elap_t: 10613[ms])  Ev(no: 8, pri:41, elap_t: 10764[ms])  Ev(no: 7, pri:42, elap_t: 11155[ms])  Ev(no: 6, pri:43, elap_t: 11499[ms])  Ev(no: 5, pri:44, elap_t: 11799[ms])
  Ev(no: 4, pri:45, elap_t: 11949[ms])  Ev(no: 3, pri:46, elap_t: 12144[ms])  Ev(no: 2, pri:47, elap_t: 12309[ms])  Ev(no: 1, pri:48, elap_t: 12698[ms])  Ev(no: 0, pri:49, elap_t: 13015[ms])

Minimum event processing time:    14.86[ms] for Ev(no:21, pri:28, elap_t:    15[ms])
Maximum event processing time: 13015.32[ms] for Ev(no: 0, pri:49, elap_t: 13015[ms])
Average event processing time:  6491.50[ms] for total 50 events

Thread_EventGenerator is terminated !!
Thread_EventHandler is terminated !!
```

<Oral Test>

Q 12.1 다중 스레드 구조의 프로그램에서 공유 자원의 사용에 대한 Critical section (임계구역) 설정이 필요한 이유에 대하여 예를 들어 구체적으로설명하고, 임계구역 설정을 하기 위하여 mutex를 설정하고 사용하는 방법에 대하여 예를 들어 구체적으로 설명하라.

Q 12.2 스레드를 생성하는 방법과 생성되는 스레드에 파라메터를 전달하는 방법에 대하여 예를 들어 구체적으로 설명하라.

Q 12.3 Multi-thread의 동작 상태를 monitoring하여, 주기적으로 상태를 출력하는 방법에 대하여 예를 들어 구체적으로 설명하라. 특히 관련 구조체, 구조체의 변수를 누가 언제 변경하고, 누가 언제 출력하게 되는가에 상세하게 설명하라.

Q 12.4 우선 순위를 고려한 Event처리를 위하여 사용되는 Priority Queue에서 우선 순위가 높은 event가 우선적으로 처리될 수 있는 내부 구조와 동작 원리에 대하여 예를 들어 구체적으로 설명하라.