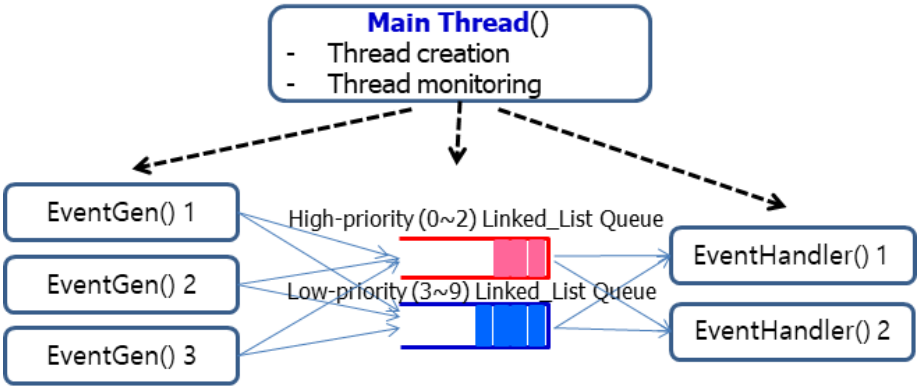
2021-1 프로그래밍언어 실습 13

13.1 Event Processing Simulation을 위한 Multi-thread 구현

(1) Functional Block Diagram



(2) 구조체 Event 및 관련함수

```
/* Event.h */

#ifndef EVENT_H
#define EVENT_H

#include <stdio.h>
#include "ConsoleDisplay.h"
#include "SimParams.h"

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

typedef struct
{
    int event_no;
    int event_gen_addr;
    int event_handler_addr;
    int event_pri;   // event_priority
    LARGE_INTEGER t_gen;
    LARGE_INTEGER t_proc;
    double t_elapsed; // elapsed time for event processing
    EventStatus eventStatus;
} Event;

void printEvent(Event* pEvt);
void printEvent_withTime(Event* pEv);
void calc_elapsed_time(Event *pEv, LARGE_INTEGER freq);
#endif
```

(3) Simulation Parameters

```
/* SimParam.h Simulation Parameters */

#ifndef SIMULATION_PARAMETERS_H
#define SIMULATION_PARAMETERS_H

#define NUM_EVENT_GENERATORS 3
#define NUM_EVENTS_PER_GEN 20
#define NUM_EVENT_HANDLERS 2
#define TOTAL_NUM_EVENTS (NUM_EVENTS_PER_GEN * NUM_EVENT_GENERATORS)
```

```
#define PLUS_INF INT_MAX
#define MAX_ROUND 1000

#define NUM_PRIORITY 10
#define PRIORITY_THRESHOLD 3 // 0 ~ 2: High Priority, 3 ~ 9: low priority
#define EVENT_PER_LINE 5

#endif
```

13.2 Doubly Linked List (DLL) - based Event Queue

(1) Doubly Linked ListNode (DLLN), DLLN_Ev, DLL_EvQ

```
/* DLL_EvQ.h   */
#ifndef LinkedList_QUEUE_H
#define LinkedList_QUEUE_H

#include <Windows.h>
#include <stdio.h>
#include <thread>
#include <mutex>
#include "Event.h"

using namespace std;

// doubly linked list node (DLLN)
typedef struct DLLN
{
    struct DLLN *next;
    struct DLLN *prev;
    Event *pEv;
} DLLN_Ev;

typedef struct
{
    mutex cs_EvQ;
    int priority;
    DLLN_Ev *front;
    DLLN_Ev *back;
    int num_event;
} DLL_EvQ;

void initDLL_EvQ(DLL_EvQ *DLL_EvQ, int priority);
Event *enDLL_EvQ(DLL_EvQ *DLL_EvQ, Event *pEv);
Event *deDLL_EvQ(DLL_EvQ *DLL_EvQ);
void printDLL_EvQ(DLL_EvQ *DLL_EvQ);
#endif
```

13.3 Threads: Event Generator, Event Forwarder

(1) Thread Parameter, Thread Status Monitor

```
/* Thread.h */
#ifndef THREAD_H
#define THREAD_H
#include <Windows.h>
#include <thread>
#include <mutex>
#include <process.h>
#include "Event.h"
#include "SimParams.h"
#include "DLL_EvQ.h"

using namespace std;
```

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

typedef struct
{
	int eventsGen[NUM_EVENT_GENERATORS]; // generator ID
	int eventsProc[NUM_EVENT_HANDLERS]; // processor ID
	int totalEventGen;
	int totalEventProc;
	int numEventProcs_priH;
	int numEventProcs_priL;
	THREAD_FLAG *pFlagThreadTerminate;
	Event eventGenerated[TOTAL_NUM_EVENTS];
	Event eventProcessed[TOTAL_NUM_EVENTS];
} ThreadStatMon;

typedef struct
{
	FILE *fout;
	mutex *pCS_main;
	mutex *pCS_thrd_mon;
	DLL_EvQ *EvQ_PriH;
	DLL_EvQ *EvQ_PriL;
	ROLE role;
	int myAddr;
	int maxRound;
	int targetEventGen;
	LARGE_INTEGER PC_freq;
		// frequency of performance counter that is used to measure elapsed time
	ThreadStatMon *pThrdMon;
} ThreadParam_Ev;

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

(2) Thread_EventGenerator

Thread_EventGenerator()은 주기적으로 이벤트을 생성하여 DLL_EvQ에 enqueue시킨다. 이벤트의 no는 발생 순서에 따라 순차적으로 부여되며, 이벤트 발생지 주소는 사전에 스레드에 지정된 주소를 사용한다. 이벤트 우선 순위는 0 ~ 9사이의 값을 임의로 설정하며, 0~2의 우선순위는 higher priority로, 3~9의 우선순위는 lower priority로 구분하여 각각 다른 DLL_EvQ를 사용하도록 한다. 각 이벤트의 처리 시간을 분석하기 위하여, 각 이벤트의 생성 시점을 QueryPerformanceCounter() 함수를 사용하여 측정한 후, 이를 이벤트의 t_gen에 기록한다.

생성된 이벤트는 그 이벤트의 우선 순위 값에 따라 high priority event queue 또는 low priority event queue에 enqueue시킨다.

Thread_EventGenerator()은 이벤트를 생성하여 enqueue시킨 후, Thread Status Monitor 정보를 update하며, Event eventGenerated[] 배열에 추가한다. 이벤트 생성 후에는 10 ~ 100 ms 동안 sleep한다.

(3) Thread_EventHandler()

Thread_EventHandler()는 high priority event queue와 low priority eventt queue를 순차적으로 확인하여 이벤트를 처리한다. 먼저 high priority event queue에 있는 모든 이벤트를 처리한 후, low priority event queue의 이벤트를 하나 처리하고, 다시 high priority event queue의 패킷을 모두 처리하여 항상 우선 순위가 높은 이벤트가 존재하는 경우 이를 먼저 처리할 수 있게 한다.

이벤트가 처리된 시간을 QueryPerformanceCounter() 함수를 사용하여 측정한 후, 이를 그 이벤트의 t_proc에 기록한다.

하나의 이벤트을 처리한 후에는 100 ~ 500ms을 sleep하도록 한다.

13.4 Thread Monitoring

(1) Thread status monitoring

각 스레드의 처리 상태를 주기적으로 확인하기 위하여 구조체 ThreadStatusMonitor를 사용하며, 현재까지 생성된 총 이벤트 수, 현재까지 처리 완료된 총 이벤트 수 (우선 순위별로 구분할 것), 처리 완료된 이벤트의 배열 등이 포함된다.

(2) 주기적인 모니터링 결과 출력

구조체 ThreadStatusMonitor의 내용은 main() 함수에 의하여 주기적으로 출력되도록 한다. 스레드 모니터링 결과의 출력에서는 상황판 출력과 같이 화면의 지정된 위치에 출력이 발생하도록 할 것. (즉, scrolling 기능 없이, 각 round 별로 화면을 지운 후, 새로운 결과가 출력 될 수 있게 할 것.)

13.5 이벤트 처리 성능 측정 및 통계 분석

(1) 이벤트 처리 성능 측정

이벤트가 생성된 후 두 개의 FIFO queue를 거쳐 event 처리 스레드에 의하여 처리될 때까지 걸린 경과시간을 측정하기 위하여 QueryPerformacneCounter() 함수를 사용하며, 이벤트의 t_gen, t_procd에 처리 시점을 기록하고, 경과시간을 계산하여 t_elapsed에 기록한다.

이벤트 처리에 걸린 경과시간을 계산하기 위하여 Thread_EventHandler()에서 이벤트를 큐로부터 dequeue한 후 해당 이벤트에 대하여 calc_elapsed_time() 함수를 실행한다.

(2) 이벤트 처리 성능의 통계 분석

이벤트 처리 성능의 통계 분석을 하기 위하여 각 이벤트의 생성 및 처리가 모두 완료된 후, 모든 이벤트의 처리 시간 중 최소값, 최대값, 평균값을 찾고, 이를 출력한다.

13.6 main() function

(1) main() function

```
/* main_EventGen_DLL_EvQ_EventProc.cpp */
#include <stdio.h>
#include <stdlib.h>
#include <Windows.h>
#include <time.h>
#include <thread>
#include <mutex>
#include "Thread.h"
#include "DLL_EvQ.h"
#include "Event.h"
#include "ConsoleDisplay.h"

using namespace std;

void main()
{
	FILE *fout;
	DLL_EvQ dll_EvQ_PriH, dll_EvQ_PriL;
	Event *pEvent;
	int myAddr = 0;
	int event_handler_addr, eventPriority;
	LARGE_INTEGER pc_freq;

	fout = fopen("SimOutput.txt", "w");
	if (fout == NULL)
	{
		printf("Error in opening SimOutput.txt file in write mode !!\n");
		exit;
	}
	initDLL_EvQ(&dll_EvQ_PriH, 0);
	initDLL_EvQ(&dll_EvQ_PriL, 1);
	srand(time(NULL));
```

```
ThreadParam_Ev thrdParam_EventGen[NUM_EVENT_GENERATORS],
        thrdParam_EventHndlr[NUM_EVENT_HANDLERS];
thread thread_evHandlers[NUM_EVENT_HANDLERS];
thread thread_evGens[NUM_EVENT_GENERATORS];
mutex cs_main;
mutex cs_thrd_mon;
ThreadStatMon thrdMon;
HANDLE consHndlr;
THREAD_FLAG eventThreadFlag = RUN;
int count, totalEventGenerated, totalEventProcessed;
Event eventProcessed[TOTAL_NUM_EVENTS];

consHndlr = initConsoleHandler();
QueryPerformanceFrequency(&pc_freq);

thrdMon.pFlagThreadTerminate = &eventThreadFlag;
thrdMon.totalEventGen = 0;
thrdMon.totalEventProc = 0;
thrdMon.numEventProcs_priH = 0;
thrdMon.numEventProcs_priL = 0;
for (int ev = 0; ev < TOTAL_NUM_EVENTS; ev++)
{
    thrdMon.eventProcessed[ev].event_no = -1; // mark as not-processed
    thrdMon.eventProcessed[ev].event_pri = -1;
}

/* Create and Activate Thread_EventHandler */
for (int p = 0; p < NUM_EVENT_HANDLERS; p++)
{
    thrdMon.eventsProc[p] = 0;
    thrdParam_EventHndlr[p].fout = fout;
    thrdParam_EventHndlr[p].role = EVENT_HANDLER;
    thrdParam_EventHndlr[p].myAddr = p; // Event handler address
    thrdParam_EventHndlr[p].pCS_main = &cs_main;
    thrdParam_EventHndlr[p].pCS_thrd_mon = &cs_thrd_mon;
    thrdParam_EventHndlr[p].EvQ_PriH = &dll_EvQ_PriH;
    thrdParam_EventHndlr[p].EvQ_PriL = &dll_EvQ_PriL;
    thrdParam_EventHndlr[p].maxRound = MAX_ROUND;
    thrdParam_EventHndlr[p].pThrdMon = &thrdMon;
    thrdParam_EventHndlr[p].PC_freq = pc_freq;

    thread_evHandlers[p] = thread(Thread_EventHandler, &thrdParam_EventHndlr[p]);
    //cs_main.lock();
    printf("%d-th thread_EventHandler is created and activated (id: %d)\n", p,
            thread_evHandlers[p].get_id());
    //cs_main.unlock();
}

/* Create and Activate Thread_EventGenerators */
for (int g = 0; g < NUM_EVENT_GENERATORS; g++)
{
    thrdMon.eventsGen[g] = 0;
    thrdParam_EventGen[g].role = EVENT_GENERATOR;
    thrdParam_EventGen[g].myAddr = g; // my Address of event generator
    thrdParam_EventGen[g].pCS_main = &cs_main;
    thrdParam_EventGen[g].pCS_thrd_mon = &cs_thrd_mon;
    thrdParam_EventGen[g].EvQ_PriH = &dll_EvQ_PriH;
    thrdParam_EventGen[g].EvQ_PriL = &dll_EvQ_PriL;
    thrdParam_EventGen[g].targetEventGen = NUM_EVENTS_PER_GEN;
    thrdParam_EventGen[g].maxRound = MAX_ROUND;
    thrdParam_EventGen[g].pThrdMon = &thrdMon;
    thrdParam_EventGen[g].PC_freq = pc_freq;
```

```
        thread_evGens[g] = thread(Thread_EventGenerator, &thrdParam_EventGen[g]);
        //cs_main.lock();
        printf("%d-th thread_EventGen is created and activated (id: %d)\n", g,
                        thread_evGens[g].get_id());
        //cs_main.unlock();
    }

    /* Monitoring thread progress in rounds */
    for (int round = 0; round < MAX_ROUND; round++)
    {
        cs_main.lock();
        system("cls");
        gotoxy(consHndlr, 0, 0);
        printf("Thread monitoring by main() :: round(%2d): \n", round);
        cs_thrd_mon.lock();
        for (int i = 0; i < NUM_EVENT_GENERATORS; i++)
        {
            printf("    Event_Gen[%d] generated %2d events.\n", I,
                            thrdMon.eventsGen[i]);
        }

        printf("Event_Generators have generated total %2d events\n",
                        thrdMon.totalEventGen);
        totalEventGenerated = thrdMon.totalEventProc;
        printf("\nTotal Generated Events (current total %d events)\n      ",
                        totalEventGenerated);
        for (int ev = 0; ev < totalEventGenerated; ev++)
        {
            pEvent = &thrdMon.eventGenerated[ev];
            if (pEvent != NULL)
            {
                printEvent(pEvent);
                if (((ev + 1) % EVENT_PER_LINE) == 0)
                    printf("\n      ");
            }
        }
        printf("\n");

        printf("\nEvent_Handlers have processed total %2d events ",
                        thrdMon.totalEventProc);
        printf("(event__PriH (%2d), event_PriL (%2d))\n", thrdMon.numEventProcs_priH,
                        thrdMon.numEventProcs_priL);
        for (int i = 0; i < NUM_EVENT_HANDLERS; i++)
        {
            printf("    Event_Proc[%d] processed %2d events.\n", I,
                        thrdMon.eventsProc[i]);
        }

        printf("\n");
        printf("DLL_EvQ_PriH (%3d events):\n      ", dll_EvQ_PriH.num_event);
                        printDLL_EvQ(&dll_EvQ_PriH);
        printf("\n");
        printf("DLL_EvQ_PriL (%3d events):\n      ", dll_EvQ_PriL.num_event);
                        printDLL_EvQ(&dll_EvQ_PriL);
        printf("\n");
        totalEventProcessed = thrdMon.totalEventProc;
        printf("\nTotal Processed Events (current total %d events):\n      ",
                        totalEventProcessed);
        count = 0;
        for (int ev = 0; ev < totalEventProcessed; ev++)
        {
            pEvent = &thrdMon.eventProcessed[ev];
            if (pEvent != NULL)
```

```
                {
                    printEvent(pEvent);
                    if (((ev + 1) % EVENT_PER_LINE) == 0)
                        printf("\n      ");
                }
            }
            printf("\n");
            cs_thrd_mon.unlock();
            if (totalEventProcessed >= TOTAL_NUM_EVENTS)
            {
                printf("!!! TotalEventProcessed (%d) is reached to target
                   TOTAL_NUM_EVENTS(%d)\n", totalEventProcessed, TOTAL_NUM_EVENTS);
                eventThreadFlag = TERMINATE; // set 1 to terminate threads
                cs_main.unlock();
                break;
            }
            cs_main.unlock();
            Sleep(100);
        } // end for (int round .....)

        for (int p = 0; p < NUM_EVENT_HANDLERS; p++)
        {
            thread_evHandlers[p].join();
        }
        printf("All threads of event handlers are terminated !!\n");

        for (int g = 0; g < NUM_EVENT_GENERATORS; g++)
        {
            thread_evGens[g].join();
        }
        printf("All threads of event generators are terminated !!\n");

        //calc_elapsed_time(thrdMon.eventProcessed, thrdMon.numPktProcs, freq);
        double min, max, avg, sum;
        int min_event, max_event;
        min = max = sum = thrdMon.eventProcessed[0].t_elapsed;
        min_event = max_event = 0;
        for (int i = 1; i < TOTAL_NUM_EVENTS; i++)
        {
            sum += thrdMon.eventProcessed[i].t_elapsed;
            if (min > thrdMon.eventProcessed[i].t_elapsed)
            {
                min = thrdMon.eventProcessed[i].t_elapsed;
                min_event = i;
            }
            if (max < thrdMon.eventProcessed[i].t_elapsed)
            {
                max = thrdMon.eventProcessed[i].t_elapsed;
                max_event = i;
            }
        }
        avg = sum / (double) TOTAL_NUM_EVENTS;
        printf("Minimum event processing time: %8.2lf[ms] for ", min * 1000);
        printEvent_withTime(&thrdMon.eventProcessed[min_event]); printf("\n");
        printf("Maximum event processing time: %8.2lf[ms] for ", max * 1000);
        printEvent_withTime(&thrdMon.eventProcessed[max_event]); printf("\n");
        printf("Average event processing time: %8.2lf[ms] for total %d events\n", avg * 1000,
           TOTAL_NUM_EVENTS);
        printf("\n");
}
```

(2) Example output (초기 상태)

```
Thread monitoring by main() :: round(21):
  Event_Gen[0] generated 19 events.
  Event_Gen[1] generated 19 events.
  Event_Gen[2] generated 19 events.
Event_Generators have generated total 57 events

Total Generated Events (current total 10 events)
   Ev[  0, pri( 1), gen( 0), proc(-1)] Ev[ 20, pri( 1), gen( 1), proc(-1)] Ev[ 40, pri( 1), gen( 2), proc(-1)] Ev[  1, pri( 4), gen( 0), proc(-1)] Ev[ 41, pri( 4), gen( 2), proc(-1)]
   Ev[ 21, pri( 4), gen( 1), proc(-1)] Ev[ 22, pri( 9), gen( 1), proc(-1)] Ev[ 42, pri( 9), gen( 2), proc(-1)] Ev[  2, pri( 9), gen( 0), proc(-1)] Ev[ 43, pri( 8), gen( 2), proc(-1)]


Event_Handlers have processed total 10 events (event__PriH (10), event_PriL ( 0))
  Event_Proc[0] processed  5 events.
  Event_Proc[1] processed  5 events.

DLL_EvQ_PriH ( 15 events):
   Ev[ 30, pri( 1), gen( 1), proc(-1)]  Ev[ 10, pri( 1), gen( 0), proc(-1)]  Ev[ 50, pri( 1), gen( 2), proc(-1)]  Ev[ 51, pri( 2), gen( 2), proc(-1)]  Ev[ 31, pri( 2), gen( 1), proc(-1)]
   Ev[ 11, pri( 2), gen( 0), proc(-1)]  Ev[ 32, pri( 2), gen( 1), proc(-1)]  Ev[ 52, pri( 2), gen( 2), proc(-1)]  Ev[ 12, pri( 2), gen( 0), proc(-1)]  Ev[ 33, pri( 1), gen( 1), proc(-1)]
   Ev[ 53, pri( 1), gen( 2), proc(-1)]  Ev[ 13, pri( 1), gen( 0), proc(-1)]  Ev[ 36, pri( 1), gen( 1), proc(-1)]  Ev[ 56, pri( 1), gen( 2), proc(-1)]  Ev[ 16, pri( 1), gen( 0), proc(-1)]

DLL_EvQ_PriL ( 30 events):
   Ev[  1, pri( 4), gen( 0), proc(-1)]  Ev[ 41, pri( 4), gen( 2), proc(-1)]  Ev[ 21, pri( 4), gen( 1), proc(-1)]  Ev[ 22, pri( 9), gen( 1), proc(-1)]  Ev[ 42, pri( 9), gen( 2), proc(-1)]
   Ev[  2, pri( 9), gen( 0), proc(-1)]  Ev[ 43, pri( 8), gen( 2), proc(-1)]  Ev[ 23, pri( 8), gen( 1), proc(-1)]  Ev[  3, pri( 8), gen( 0), proc(-1)]  Ev[  5, pri( 5), gen( 0), proc(-1)]
   Ev[ 25, pri( 5), gen( 1), proc(-1)]  Ev[ 45, pri( 5), gen( 2), proc(-1)]  Ev[ 28, pri( 5), gen( 1), proc(-1)]  Ev[ 48, pri( 5), gen( 2), proc(-1)]  Ev[  8, pri( 5), gen( 0), proc(-1)]
   Ev[ 49, pri( 7), gen( 2), proc(-1)]  Ev[ 29, pri( 7), gen( 1), proc(-1)]  Ev[  9, pri( 7), gen( 0), proc(-1)]  Ev[ 54, pri( 8), gen( 2), proc(-1)]  Ev[ 34, pri( 8), gen( 1), proc(-1)]
   Ev[ 14, pri( 8), gen( 0), proc(-1)]  Ev[ 15, pri( 7), gen( 0), proc(-1)]  Ev[ 35, pri( 7), gen( 1), proc(-1)]  Ev[ 55, pri( 7), gen( 2), proc(-1)]  Ev[ 17, pri( 9), gen( 0), proc(-1)]
   Ev[ 57, pri( 9), gen( 2), proc(-1)]  Ev[ 37, pri( 9), gen( 1), proc(-1)]  Ev[ 58, pri( 7), gen( 2), proc(-1)]  Ev[ 38, pri( 7), gen( 1), proc(-1)]  Ev[ 18, pri( 7), gen( 0), proc(-1)]


Total Processed Events (current total 10 events):
   Ev[  0, pri( 1), gen( 0), proc( 0)] Ev[ 20, pri( 1), gen( 1), proc( 1)] Ev[ 40, pri( 1), gen( 2), proc( 1)] Ev[ 44, pri( 2), gen( 2), proc( 0)] Ev[ 24, pri( 2), gen( 1), proc( 1)]
   Ev[  4, pri( 2), gen( 0), proc( 0)] Ev[ 46, pri( 1), gen( 2), proc( 1)] Ev[ 26, pri( 1), gen( 1), proc( 0)] Ev[  6, pri( 1), gen( 0), proc( 1)] Ev[ 47, pri( 1), gen( 2), proc( 0)]
```

(3) Example output (최종 완료 상태)

```
Thread monitoring by main() :: round(80):
  Event_Gen[0] generated 20 events.
  Event_Gen[1] generated 20 events.
  Event_Gen[2] generated 20 events.
Event_Generators have generated total 60 events

Total Generated Events (current total 60 events)
   Ev[  0, pri( 1), gen( 0), proc(-1)] Ev[ 20, pri( 1), gen( 1), proc(-1)] Ev[ 40, pri( 1), gen( 2), proc(-1)] Ev[ 41, pri( 4), gen( 2), proc(-1)] Ev[  1, pri( 4), gen( 0), proc(-1)]
   Ev[ 21, pri( 4), gen( 1), proc(-1)] Ev[ 22, pri( 9), gen( 1), proc(-1)] Ev[  2, pri( 9), gen( 0), proc(-1)] Ev[ 42, pri( 9), gen( 2), proc(-1)] Ev[  3, pri( 8), gen( 0), proc(-1)]
   Ev[ 43, pri( 8), gen( 2), proc(-1)] Ev[ 23, pri( 8), gen( 1), proc(-1)] Ev[ 44, pri( 2), gen( 2), proc(-1)] Ev[ 24, pri( 2), gen( 1), proc(-1)] Ev[  4, pri( 2), gen( 0), proc(-1)]
   Ev[ 25, pri( 5), gen( 1), proc(-1)] Ev[ 45, pri( 5), gen( 2), proc(-1)] Ev[  5, pri( 5), gen( 0), proc(-1)] Ev[  6, pri( 1), gen( 0), proc(-1)] Ev[ 26, pri( 1), gen( 1), proc(-1)]
   Ev[ 46, pri( 1), gen( 2), proc(-1)] Ev[  7, pri( 1), gen( 0), proc(-1)] Ev[ 27, pri( 1), gen( 1), proc(-1)] Ev[ 47, pri( 1), gen( 2), proc(-1)] Ev[  8, pri( 5), gen( 0), proc(-1)]
   Ev[ 48, pri( 5), gen( 2), proc(-1)] Ev[ 28, pri( 5), gen( 1), proc(-1)] Ev[  9, pri( 7), gen( 0), proc(-1)] Ev[ 29, pri( 7), gen( 1), proc(-1)] Ev[ 49, pri( 7), gen( 2), proc(-1)]
   Ev[ 50, pri( 1), gen( 2), proc(-1)] Ev[ 30, pri( 1), gen( 1), proc(-1)] Ev[ 10, pri( 1), gen( 0), proc(-1)] Ev[ 51, pri( 2), gen( 2), proc(-1)] Ev[ 31, pri( 2), gen( 1), proc(-1)]
   Ev[ 11, pri( 2), gen( 0), proc(-1)] Ev[ 52, pri( 2), gen( 2), proc(-1)] Ev[ 12, pri( 2), gen( 0), proc(-1)] Ev[ 32, pri( 2), gen( 1), proc(-1)] Ev[ 33, pri( 1), gen( 1), proc(-1)]
   Ev[ 53, pri( 1), gen( 2), proc(-1)] Ev[ 13, pri( 1), gen( 0), proc(-1)] Ev[ 54, pri( 8), gen( 2), proc(-1)] Ev[ 34, pri( 8), gen( 1), proc(-1)] Ev[ 14, pri( 8), gen( 0), proc(-1)]
   Ev[ 15, pri( 7), gen( 0), proc(-1)] Ev[ 35, pri( 7), gen( 1), proc(-1)] Ev[ 55, pri( 7), gen( 2), proc(-1)] Ev[ 16, pri( 1), gen( 0), proc(-1)] Ev[ 36, pri( 1), gen( 1), proc(-1)]
   Ev[ 56, pri( 1), gen( 2), proc(-1)] Ev[ 37, pri( 9), gen( 1), proc(-1)] Ev[ 17, pri( 9), gen( 0), proc(-1)] Ev[ 57, pri( 9), gen( 2), proc(-1)] Ev[ 58, pri( 7), gen( 2), proc(-1)]
   Ev[ 38, pri( 7), gen( 1), proc(-1)] Ev[ 18, pri( 7), gen( 0), proc(-1)] Ev[ 59, pri( 5), gen( 2), proc(-1)] Ev[ 39, pri( 5), gen( 1), proc(-1)] Ev[ 19, pri( 5), gen( 0), proc(-1)]


Event_Handlers have processed total 60 events (event__PriH (27), event_PriL (33))
  Event_Proc[0] processed 32 events.
  Event_Proc[1] processed 28 events.

DLL_EvQ_PriH (  0 events):

DLL_EvQ_PriL (  0 events):


Total Processed Events (current total 60 events):
   Ev[  0, pri( 1), gen( 0), proc( 0)] Ev[ 20, pri( 1), gen( 1), proc( 1)] Ev[ 40, pri( 1), gen( 2), proc( 1)] Ev[ 44, pri( 2), gen( 2), proc( 0)] Ev[  4, pri( 2), gen( 0), proc( 1)]
   Ev[ 24, pri( 2), gen( 1), proc( 0)] Ev[ 26, pri( 1), gen( 1), proc( 1)] Ev[  6, pri( 1), gen( 0), proc( 0)] Ev[ 46, pri( 1), gen( 2), proc( 0)] Ev[  7, pri( 1), gen( 0), proc( 1)]
   Ev[ 47, pri( 1), gen( 2), proc( 0)] Ev[ 27, pri( 1), gen( 1), proc( 1)] Ev[ 50, pri( 1), gen( 2), proc( 1)] Ev[ 30, pri( 1), gen( 1), proc( 0)] Ev[ 51, pri( 2), gen( 2), proc( 1)]
   Ev[ 10, pri( 1), gen( 0), proc( 0)] Ev[ 11, pri( 2), gen( 0), proc( 1)] Ev[ 31, pri( 2), gen( 1), proc( 0)] Ev[ 52, pri( 2), gen( 2), proc( 1)] Ev[ 12, pri( 2), gen( 0), proc( 0)]
   Ev[ 32, pri( 2), gen( 1), proc( 1)] Ev[ 33, pri( 1), gen( 1), proc( 0)] Ev[ 53, pri( 1), gen( 2), proc( 0)] Ev[ 13, pri( 1), gen( 0), proc( 1)] Ev[ 16, pri( 1), gen( 0), proc( 0)]
   Ev[ 36, pri( 1), gen( 1), proc( 1)] Ev[ 56, pri( 1), gen( 2), proc( 1)] Ev[ 41, pri( 4), gen( 2), proc( 0)] Ev[  1, pri( 4), gen( 0), proc( 0)] Ev[ 21, pri( 4), gen( 1), proc( 0)]
   Ev[ 22, pri( 9), gen( 1), proc( 0)] Ev[  2, pri( 9), gen( 0), proc( 0)] Ev[ 42, pri( 9), gen( 2), proc( 1)] Ev[  3, pri( 8), gen( 0), proc( 0)] Ev[ 43, pri( 8), gen( 2), proc( 1)]
   Ev[ 23, pri( 8), gen( 1), proc( 0)] Ev[ 25, pri( 5), gen( 1), proc( 1)] Ev[ 45, pri( 5), gen( 2), proc( 0)] Ev[  5, pri( 5), gen( 0), proc( 0)] Ev[  8, pri( 5), gen( 0), proc( 1)]
   Ev[ 48, pri( 5), gen( 2), proc( 0)] Ev[ 28, pri( 5), gen( 1), proc( 1)] Ev[  9, pri( 7), gen( 0), proc( 0)] Ev[ 29, pri( 7), gen( 1), proc( 1)] Ev[ 49, pri( 7), gen( 2), proc( 0)]
   Ev[ 54, pri( 8), gen( 2), proc( 1)] Ev[ 34, pri( 8), gen( 1), proc( 1)] Ev[ 14, pri( 8), gen( 0), proc( 0)] Ev[ 15, pri( 7), gen( 0), proc( 1)] Ev[ 35, pri( 7), gen( 1), proc( 0)]
   Ev[ 55, pri( 7), gen( 2), proc( 1)] Ev[ 37, pri( 9), gen( 1), proc( 0)] Ev[ 17, pri( 9), gen( 0), proc( 0)] Ev[ 57, pri( 9), gen( 2), proc( 1)] Ev[ 58, pri( 7), gen( 2), proc( 0)]
   Ev[ 38, pri( 7), gen( 1), proc( 1)] Ev[ 18, pri( 7), gen( 0), proc( 0)] Ev[ 59, pri( 5), gen( 2), proc( 1)] Ev[ 39, pri( 5), gen( 1), proc( 0)] Ev[ 19, pri( 5), gen( 0), proc( 1)]

All threads of event handlers are terminated !!
All threads of event generators are terminated !!
Minimum event processing time:   143.47[ms] for Ev(no: 20, pri: 1,    143[ms])
Maximum event processing time:  8641.41[ms] for Ev(no: 23, pri: 8,   8641[ms])
Average event processing time:  5657.17[ms] for total 60 events
```

<Oral Test>

13.1 일반 함수와 스레드의 차이점을 다음과 같은 표를 만들어 비교하여 설명하라.

항목	일반 함수	스레드
인수(parameter/argument) 전달		
함수 원형에서의 return type 지정		
함수/스레드의 실행 결과의 전달		
함수/스레드의 호출/생성		

13.2 Doubly linked list 기반의 FIFO queue 의 기본 동작 (enQueue(), deQueue(), isEmpty())이 어떻게 실행되는가에 대하여 상세하게 설명하라.

13.3 Doubly linked list 기반의 FIFO queue 를 다중 프로세스 (또는 Multi-thread) 들이 공유하는 경우, critical section 을 설정하지 않았을 때 발생하는 문제를 예를 들어 설명하고, 이를 해결하는 방법에 대하여 상세하게 설명하라.

13.4 Multi-thread 기반의 프로그램 실행에서 하나의 사건 (event)가 발생되어 처리가 완료될 때까지 경과된 시간을 micro-second 단위로 정밀하게 측정하는 방법에 대하여 상세하게 설명하라.