프로그래밍언어

실습 13 (보충설명) – Simple Event Processing Simulation with Linked-List FIFO Queue

교수 김 영 탁

영남대학교 정보통신공학과

(Tel : +82-53-810-2497; Fax : +82-53-810-4742

http://antl.yu.ac.kr/; E-mail : ytkim@yu.ac.kr)

Outline

- **Simulation of Event Generations and Handlings with Linked List FIFO Queue**
  - 3 event generators
  - 2 event handlers
  - 2 linked-list FIFO queues (high priority, low priority)
- **Event Queue based on Doubly Linked List (DLL_EvQ)**
- **main()**
- **실행 결과**
- **Oral Test**

자기참조 구조체 (Self-referential Structure)

◆ 자기참조 구조체 (self-referential structure)

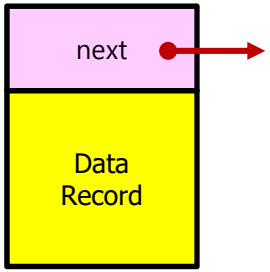
- 구조체에 포함된 항목으로 자신과 동일한 구조체를 가리키는 포인터를 포함하는 구조체
- 다양한 형태의 자료구조를 만들 수 있게 함

◆ 자기참조 구조체의 예

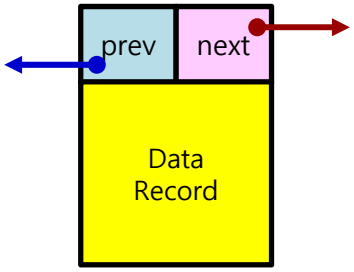
- Linked List Node (연결형 리스트의 리스트 노드)
- Binary Search Tree Node (이진 탐색 트리의 트리 노드)

연결형 리스트의 구조 (1)

- **List Node = data field + link field (next, prev)**
- **List Node with Data**

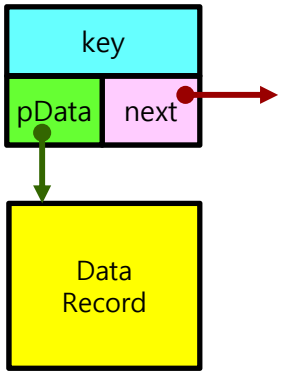


(a) List Node with Data for Singly Linked List

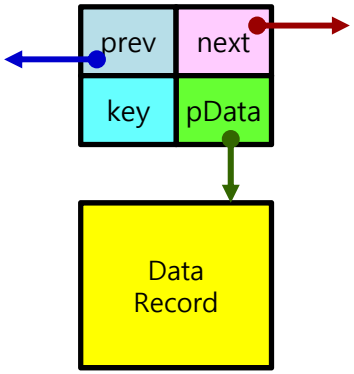


(b) Doubly Linked List Node with Data
for Doubly Linked List

- **List Node with Data Pointer**



(a) List Node with Data Pointer
for Singly Linked List



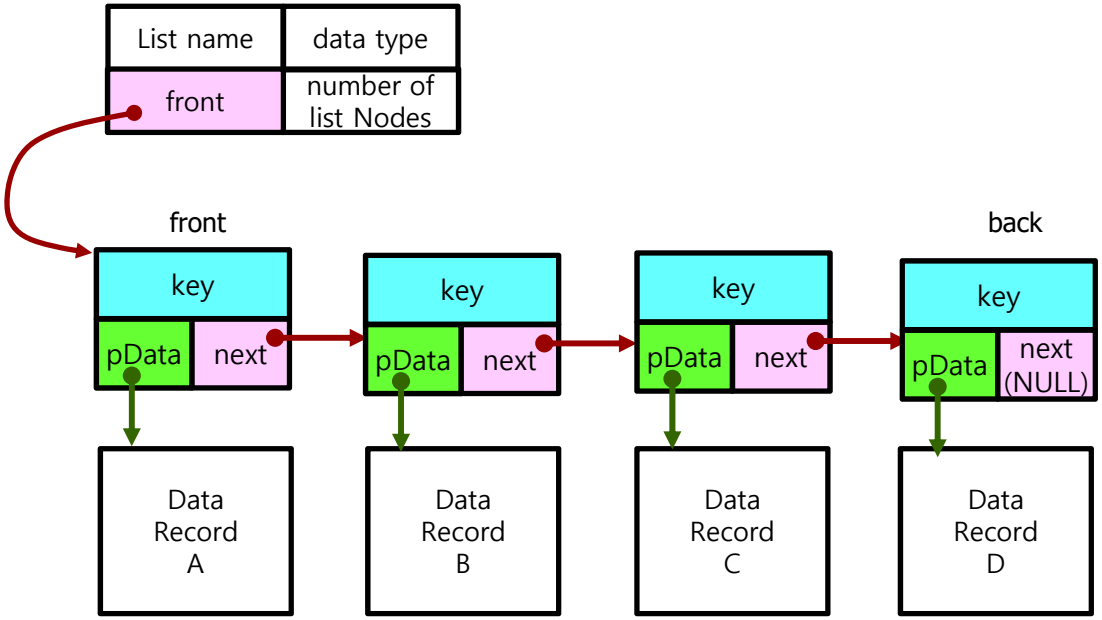
(b) List Node with Data Pointer
for Doubly Lniked List

연결형 리스트의 구조 (2)

◆ List

- linked list of Nodes
- list abstract data type

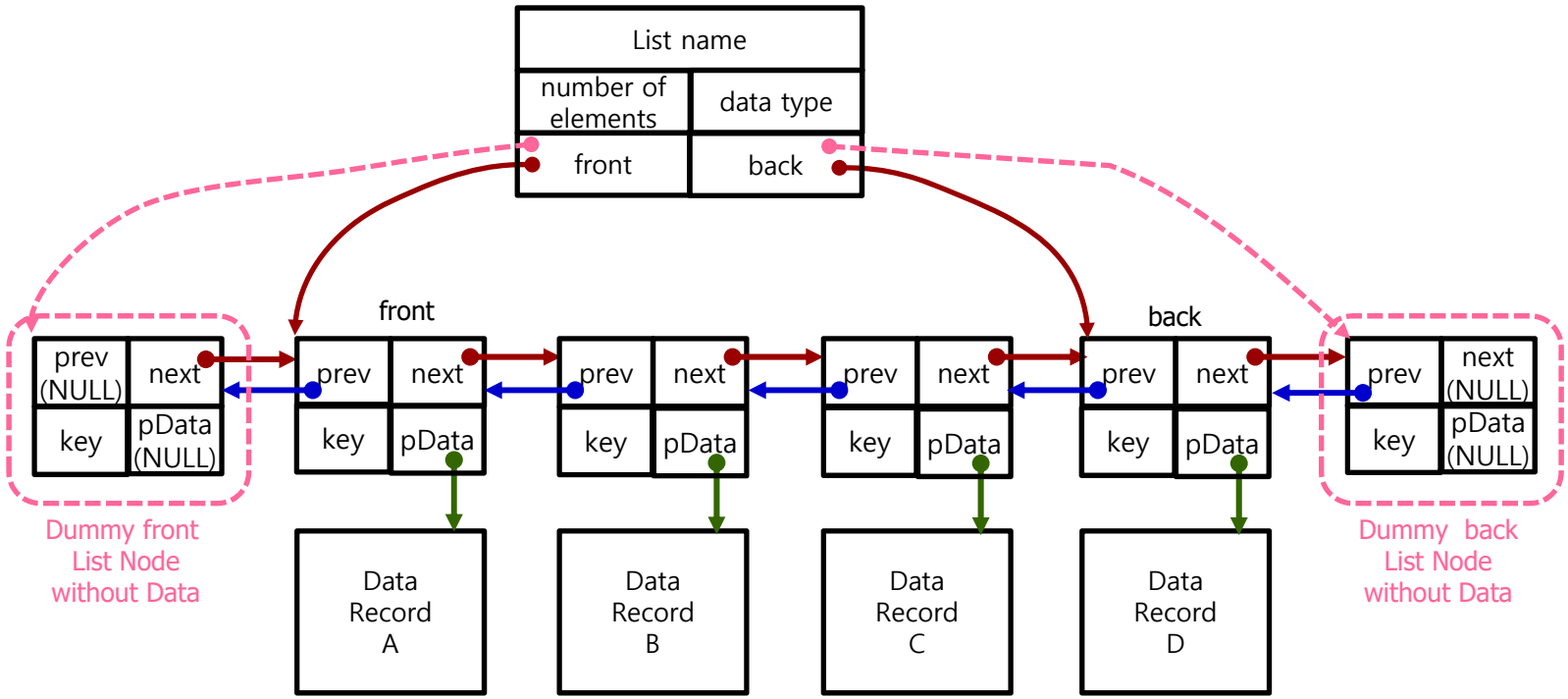
◆ Single Linked List (SLL)



이중 연결형 리스트 (Doubly Linked List)의 구조

◆ Structure of Doubly Linked List (DLL)

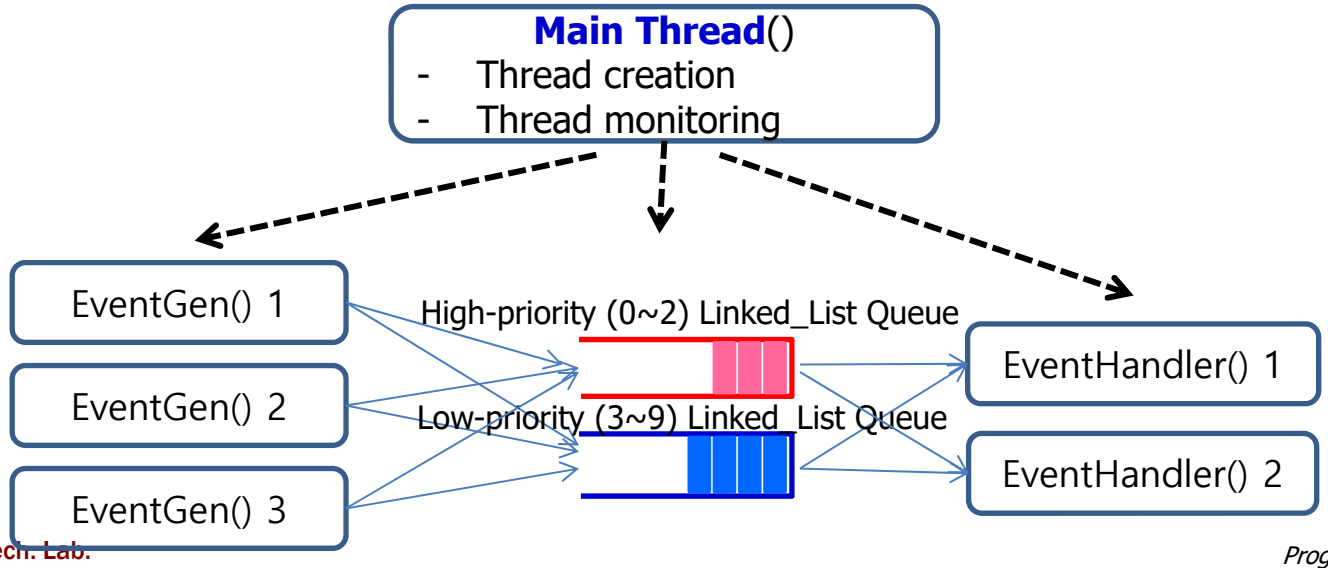
- **front** pointer**:** the pointer that points the front (head, first) Node
- **back** pointer: the pointer that points the last (end, tail, last) Node
- number of elements: current number of elements in DLL



Multi-thread와 Linked List Queue의 응용 예제

◆ Simple Simulation of Event Generator, Linked List Queues, Event Handler

- Two kinds of Threads
  - Event Generator
  - Event Handler
- Two shared Doubly Linked List Queue
  - EventQ_HighPriority
  - EventQ_LowPriority



이벤트 (Event) 이란 ?

◆ 이벤트 (Event)

- 다양하게 발생되는 사건들을 모델링
- 이벤트의 중요 항목
  - 생성자 주소: 이벤트가 발생된 노드의 주소
  - 처리자 주소: 이벤트를 처리한 노드의 주소
  - 이벤트번호 (event id): 동일한 송신-수신 단말장치간에 전달되는 이벤트들을 구분하기 위한 번호.
  - 우선순위 (priority 또는 precedence): 이벤트의 종류에 따라 우선적으로 처리하여야 하는 필요성을 나타내는 정보

Event

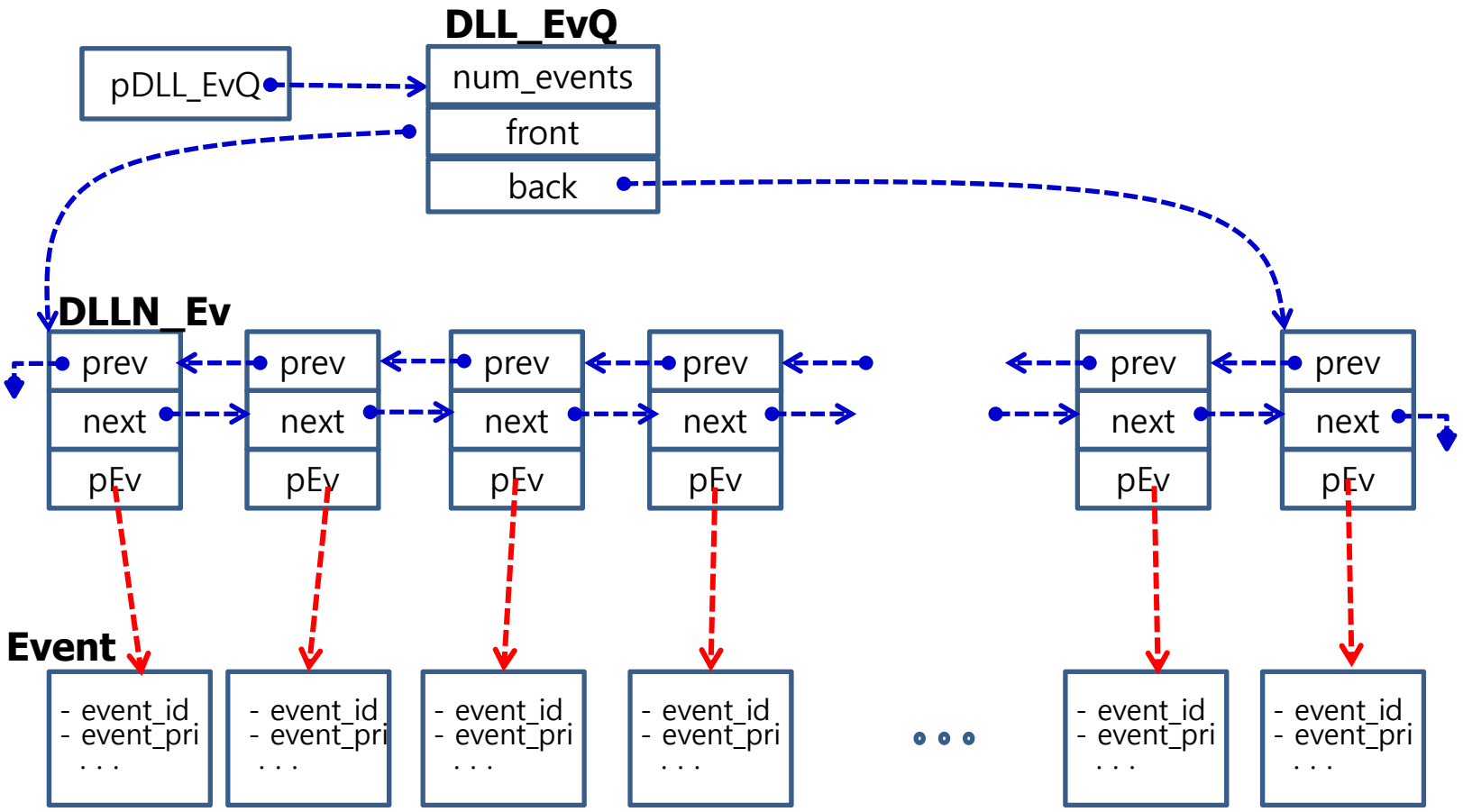
```
/* Event.h  (1) */
#ifndef EVENT_H
#define EVENT_H

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

#define NUM_PRIORITY 10
#define PRIORITY_THRESHOLD 3
#define EVENT_PER_LINE 5

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

```
/* Event.h  (2) */

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;
    EventStatus eventStatus;
} Event;

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

#endif
```

```
/* Event.cpp */
#include <stdio.h>
#include <stdlib.h>
#include "Event.h"

const char *strEventStatus[] = { "GENERATED", "ENQUED", "PROCESSED", "UNDEFINED" };

void printEvent(Event* pEvent)
{
      printf("Ev(no:%3d, pri:%2d)  ", pEvent->event_no, pEvent->event_pri);
}
void printEvent_withTime(Event* pEv)
{
       printf("Ev(no:%3d, pri:%2d, %6.0lf[ms])  ", pEv->event_no, pEv->event_pri,
          pEv->t_elapsed * 1000);
}

void calc_elapsed_time(Event *pEv, LARGE_INTEGER freq)
{
       LONGLONG t_diff_LL;
       double t_elapsed;

       t_diff_LL = pEv->t_proc.QuadPart - pEv->t_gen.QuadPart;
       t_elapsed = (double) t_diff_LL / (double)freq.QuadPart;
       pEv->t_elapsed = t_elapsed;
}
```

Doubly Linked List Event Queue

◆ Doubly Linked List Event Queue



Linked List Queue for Event Handling

```
/* DLL_EvQ.h  (1) */
#ifndef DLL_EvQ_H
#define DLL_EvQ_H

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

using namespace std;

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

```
/* DLL_EvQ.h (2) */

typedef struct
{
    char name[50];
    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
```

```
/* DLL_EvQ.cpp (1) */
#include "DLL_EvQ.h"

void initDLL_EvQ(DLL_EvQ *pEvQ, int pri)
{
   pEvQ->cs_EvQ.lock();
   pEvQ->priority = pri;
   pEvQ->front = pEvQ->back = NULL;
   pEvQ->num_event = 0;
   pEvQ->cs_EvQ.unlock();
}

Event * enDLL_EvQ(DLL_EvQ *pEvQ, Event *pEv)
{
   DLLN_Ev *pLN_Ev;
   if (pEv == NULL)
   {
      printf("Error in enDLL_EvQ :: DLL_EvQ is NULL !!\n");
      printf("Press any key to continue ...\n");
      getc(stdin);
      return NULL;
   }
   pLN_Ev = (DLLN_Ev *)calloc(1, sizeof(DLLN_Ev));
   if (pLN_Ev == NULL)
   {
      printf("Error in enDLL_EvQ :: memory allocation for new DLLN failed !!\n");
      printf("Press any key to continue ...\n");
      getc(stdin);
      return NULL;
   }
```

```
/* DLL_EvQ.cpp (2) */

   pLN_Ev->pEv = pEv;
   pEvQ->cs_EvQ.lock();
   if (pEvQ->num_event == 0) // currently empty
   {
      pEvQ->front = pEvQ->back = pLN_Ev;
      pLN_Ev->prev = pLN_Ev->next = NULL;
      pEvQ->num_event = 1;
   }
   else
   {
      pLN_Ev->prev = pEvQ->back;
      pEvQ->back->next = pLN_Ev;
      pEvQ->back = pLN_Ev;
      pLN_Ev->next = NULL;
      pEvQ->num_event++;
   }
   pEvQ->cs_EvQ.unlock();
   return pLN_Ev->pEv;
}
```

```
/* DLL_EvQ.cpp (3) */

Event *deDLL_EvQ(DLL_EvQ *pEvQ)
{
   Event *pEv;
   DLLN_Ev *pLN_Ev_OldFront;

   pEvQ->cs_EvQ.lock();
   if (pEvQ->num_event <= 0)
   {
      pEvQ->cs_EvQ.unlock();
      return NULL;  // DLL_EvQ is Empty
   }
   else
   {
      pLN_Ev_OldFront = pEvQ->front;
      pEv = pEvQ->front->pEv;
      pEvQ->front = pEvQ->front->next;
      if (pEvQ->front != NULL)
         pEvQ->front->prev = NULL;
      pEvQ->num_event--;
      free(pLN_Ev_OldFront); // release memory for the current front DLLN
      pEvQ->cs_EvQ.unlock();
      return pEv;
   }
}
```

```
/* DLL_EvQ.cpp (4) */

void printDLL_EvQ(DLL_EvQ *pEvQ)
{
   int index = 0;
   int count;
   Event *pEv;
   DLLN_Ev *pLN_Ev;
   if (pEvQ == NULL)
   {
      printf("Error in printDLL_EvQ :: DLL_EvQ is NULL !!");
      printf("Press any key to continue ...\n");
      getc(stdin);
   }
   //printf("DLL_EvQ(num_event: %2d):\n   ", pEvQ->num_event);
   if (pEvQ->num_event <= 0)
      return;
   pLN_Ev = pEvQ->front;
   count = 0;
   while (pLN_Ev != NULL)
   {
      pEv = pLN_Ev->pEv;
      if (pEv == NULL)
         break;
      printEvent(pEv); printf(" ");
      count++;
      if ((count % 5) == 0)
         printf("\n   ");
      pLN_Ev = pLN_Ev->next;
   }
}
```

Doubly Linked List 구조의 FIFO Queue와 Multi-thread 응용 프로그램

Thread.h

```
/* Thread.h  (1)*/
#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];
    int eventsProc[NUM_EVENT_HANDLERS];
    int totalEventGen;
    int totalEventProc;
    int numEventProcs_priH;
    int numEventProcs_priL;
    THREAD_FLAG *pFlagThreadTerminate;
    Event eventGenerated[TOTAL_NUM_EVENTS];
    Event eventProcessed[TOTAL_NUM_EVENTS];
} ThreadStatMon;
```

```
/* Thread.h  (2)*/

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
```

Thread_EventGenerator

```
/* Thread_EventGen.cpp (1) */

#include <Windows.h>
#include <time.h>
#include <thread>
#include "Thread.h"
#include "DLL_EvQ.h"
#include "Event.h"
using namespace std;

void Thread_EventGenerator(ThreadParam_Ev* pThrdParam)
{

      Event *pEv;
   int event_no = 0;
   int event_pri = 0;
   int event_gen_count = 0;
   int myRole = pThrdParam->role;
   int myGenAddr = pThrdParam->myAddr;
   int targetEventGen = pThrdParam->targetEventGen;
   DLL_EvQ* pEvQ;
   DLL_EvQ* priH_EvQ = pThrdParam->EvQ_PriH;
   DLL_EvQ* priL_EvQ = pThrdParam->EvQ_PriL;
   ThreadStatMon* pThrdMon = pThrdParam->pThrdMon;
   int maxRound = pThrdParam->maxRound;
   pThrdParam->pCS_main->lock();
   printf("Thread_EventGenerator(%d): targetEventGen(%d)₩n", myGenAddr, targetEventGen);
   pThrdParam->pCS_main->unlock();
```

```
/* Thread_EventGen.cpp (2) */

    for (int round = 0; round < maxRound; round++)
    {
        if (event_gen_count < targetEventGen)
        {
            pEv = (Event *)calloc(1, sizeof(Event));
            pEv->event_gen_addr = myGenAddr;
            pEv->event_no = event_no = event_gen_count + (NUM_EVENTS_PER_GEN * myGenAddr);
            pEv->event_pri = event_pri = rand() % NUM_PRIORITY;
            pEv->event_handler_addr = -1;
            QueryPerformanceCounter(&pEv->t_gen);
            pEvQ = (event_pri < PRIORITY_THRESHOLD) ? priH_EvQ : priL_EvQ;
            while (enDLL_EvQ(pEvQ, pEv) == NULL)
            {
                Sleep(100);
            } // end while
            pThrdParam->pCS_thrd_mon->lock();
            pThrdMon->eventsGen[myGenAddr]++;
            pThrdMon->eventGenerated[pThrdMon->totalEventGen] = *pEv;
            pThrdMon->totalEventGen++;
            pThrdParam->pCS_thrd_mon->unlock();
            event_gen_count++;
        }
        else
        {
            if (*pThrdMon->pFlagThreadTerminate == TERMINATE)
                break;
        }  // end if
        Sleep(100 + rand() % 100);
    }  // end for round
}
```

Thread_EventHandler

```
/* Thread_EventHandler.cpp (1) */

#include <Windows.h>
#include <time.h>
#include <thread>
#include <mutex>
#include "Thread.h"
#include "DLL_EvQ.h"
#include "Event.h"

using namespace std;
void Thread_EventHandler(ThreadParam_Ev* pThrdParam)
{

    int myRole = pThrdParam->role;
    int myProcAddr = pThrdParam->myAddr;
    Event* pEv;
    DLL_EvQ* pEvQ;
    DLL_EvQ *priH_EvQ = pThrdParam->EvQ_PriH;
    DLL_EvQ *priL_EvQ = pThrdParam->EvQ_PriL;
    ThreadStatMon* pThrdMon = pThrdParam->pThrdMon;
    int maxRound = pThrdParam->maxRound;
    Event* evProc = pThrdParam->pThrdMon->eventProcessed;
    int targetEventGen = pThrdParam->targetEventGen;
    LARGE_INTEGER PC_freq = pThrdParam->PC_freq; // frequence of performance counter
    pThrdParam->pCS_main->lock();
    printf("Thread_EventHandler(%d): targetEventGen(%d)₩n", myProcAddr, targetEventGen);
    pThrdParam->pCS_main->unlock();
```

```
/* Thread_EventHandler.cpp (2) */

    for (int round = 0; round < maxRound; round++)
    {
        if (*pThrdMon->pFlagThreadTerminate == TERMINATE)
            break;
        while ((pEv = deDLL_EvQ(priH_EvQ)) != NULL)
        {
            pThrdParam->pCS_thrd_mon->lock();
            pEv->event_handler_addr = myProcAddr;
            QueryPerformanceCounter(&pEv->t_proc);
            calc_elapsed_time(pEv, PC_freq);
            pThrdMon->eventProcessed[pThrdMon->totalEventProc] = *pEv;
            pThrdMon->eventsProc[myProcAddr]++;
            pThrdMon->totalEventProc++;
            pThrdMon->numEventProcs_priH++;
            free(pEv);  // free the memory space for a Packet
            pThrdParam->pCS_thrd_mon->unlock();
            Sleep(300 + rand() % 500);
        } // end while
        if ((pEv = deDLL_EvQ(priL_EvQ)) != NULL)
        {
            pThrdParam->pCS_thrd_mon->lock();
            pEv->event_handler_addr = myProcAddr;
            QueryPerformanceCounter(&pEv->t_proc);
            calc_elapsed_time(pEv, PC_freq);
            pThrdMon->eventProcessed[pThrdMon->totalEventProc] = *pEv;
            pThrdMon->eventsProc[myProcAddr]++;
            pThrdMon->totalEventProc++;
            pThrdMon->numEventProcs_priL++;
            free(pEv);
            pThrdParam->pCS_thrd_mon->unlock();
        } // end if
        _sleep(100 + rand() % 100);
    }  // end while
}
```

Console Display

```
/* ConsoleDisplay.h */
#ifndef CONSOLE_DISPLAY_H
#define CONSOLE_DISPLAY_H
#include <Windows.h>

HANDLE initConsoleHandler();
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);
}
```

SimParams.h

```
/* 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
```

main()

```
/* main_EventGen_DLL_EvQ_EventProc.cpp (1) */
#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;
    }
```

```
/* main_EventGen_DLL_EvQ_EventProc.cpp (2) */


    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;
    }
```

```
/* main_EventGen_DLL_EvQ_EventProc.cpp (3) */

    /* 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();
    }
```

```
/* main_EventGen_DLL_EvQ_EventProc.cpp (4) */

    /* 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();
    }
```

```
/* main_EventGen_DLL_EvQ_EventProc.cpp (5) */

    /* 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");
```

```
/* main_EventGen_DLL_EvQ_EventProc.cpp (6) */


        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("\nDLL_EvQ_PriH (%3d events):\n   ", dll_EvQ_PriH.num_event);
        printDLL_EvQ(&dll_EvQ_PriH);
        printf("\nDLL_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");
```

```
/* main_EventGen_DLL_EvQ_EventProc.cpp (7) */

        cs_thrd_mon.unlock();
        if (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");
```

```
/* main_EventGen_DLL_EvQ_EventProc.cpp (8) */

    //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");
}
```

◆ Thread Monitoring 결과 (중간 단계)

```
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)]
```

◆ Thread Monitoring 결과 (최종 단계)

```
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 다음 항목별로 비교하여 설명하라.
표를 만들어 비교할 것.

항목	일반 함수	스레드
인수(parameter/ argument) 전달		
함수 원형에서의 return type 지정		
함수/스레드의 실행 결과 및 exit code 의 전달		
함수/스레드의 호출/생성		

13.2 Doubly linked list기반의 FIFO queue의 기본 동작 (enQueue(), deQueue(), isEmpty())이 어떻게 실행되는가에 대하여 상세하게 설명하라.

13.3 Doubly linked list기반의 FIFO queue를 다중 프로세스 (또는 Multi-thread) 들이 공유하는 경우, critical section을 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-second단위로 정밀하게 측정하는 방법에 대하여 상세하게 설명하라.