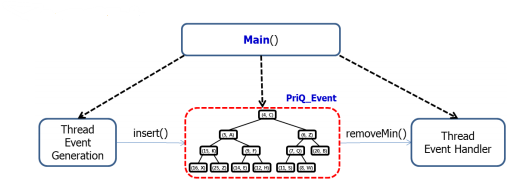
**Lab 12**

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

|  |  |
| --- | --- |
| 과목명 | 프로그래밍 언어 |
| 교수님 | 김영탁 교수님 |
| 이 름 | 김주환 |
| 학 번 | 21812158 |
| 일 자 | 2021.05.28.금 |

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

(1) Functional Block Diagram



/\* 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

(2) Event

/\* Event.cpp \*/

#include <stdio.h>

#include <stdlib.h>

#include <Windows.h>

#include "Event.h"

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

void printEvent(Event\* pEv) {

char str\_pri[6];

printf("Ev(id:%3d, pri:%2d, gen:%2d, proc:%2d) ", pEv->ev\_no, pEv->ev\_pri,

pEv->ev\_generator, pEv->ev\_handler);

}

Event\* genEvent(Event\* pEv, int event\_Gen\_ID, int ev\_no, int ev\_pri) {

pEv = (Event\*)calloc(1, sizeof(Event));

if (pEv == NULL)

return NULL;

pEv->ev\_generator = event\_Gen\_ID;

pEv->ev\_handler = -1; // event handler is not defined yet !!

pEv->ev\_no = ev\_no;

//pEv->ev\_pri = eventPriority = rand() % NUM\_PRIORITY;

pEv->ev\_pri = ev\_pri;

return pEv;

}

void calc\_elapsed\_time(Event\* pEv, LARGE\_INTEGER freq) {

LONGLONG t\_diff;

t\_diff = pEv->ev\_t\_handle.QuadPart - pEv->ev\_t\_gen.QuadPart;

pEv->elap\_time = (double)t\_diff / freq.QuadPart;

}

void printEvent\_withTime(Event\* pEv)

{

char str\_pri[6];

//printf("Ev(no:%3d, pri:%2d, src:%2d, proc:%2d) ", pEv->event\_no, pEv->event\_pri,

// pEv->event\_gen\_addr, pEv->event\_handler\_addr);

printf("Ev(no:%2d, pri:%2d, elap\_t:%6.0lf[ms]) ", pEv->ev\_no, pEv->ev\_pri, pEv->elap\_time \* 1000);

}

**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 <mutex>

#include "Event.h"

#define POS\_ROOT 1

#define MAX\_NAME\_LEN 80

#define TRUE 1

#define FALSE 0

using namespace std;

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

/\* PriorityQueue\_Event.cpp (1) \*/

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "PriorityQueue\_Event.h"

#include "Event.h"

bool hasLeftChild(int pos, PriQ\_Event\* pPriQ\_Event) {

if (pos \* 2 <= pPriQ\_Event->priQ\_size)

return TRUE;

else

return FALSE;

}

bool hasRightChild(int pos, PriQ\_Event\* pPriQ\_Event) {

if (pos \* 2 + 1 <= pPriQ\_Event->priQ\_size)

return TRUE;

else

return FALSE;

}

PriQ\_Event\* initPriQ\_Event(PriQ\_Event\* pPriQ\_Event, const char\* name, int capacity = 1) {

pPriQ\_Event->cs\_PriQ.lock();

strcpy(pPriQ\_Event->PriQ\_name, name);

pPriQ\_Event->priQ\_capacity = capacity;

pPriQ\_Event->pCBT\_Event = (CBTN\_Event\*)calloc((capacity + 1), sizeof(CBTN\_Event));

pPriQ\_Event->priQ\_size = 0;

pPriQ\_Event->pos\_last = 0;

pPriQ\_Event->cs\_PriQ.unlock();

return pPriQ\_Event;

}

void deletePriQ\_Event(PriQ\_Event\* pPriQ\_Event) {

pPriQ\_Event->cs\_PriQ.lock(); if (pPriQ\_Event->pCBT\_Event != NULL) free(pPriQ\_Event->pCBT\_Event); pPriQ\_Event->cs\_PriQ.unlock();

}

Event\* enPriQ\_Event(PriQ\_Event\* pPriQ\_Event, Event ev) {

int pos, pos\_parent; CBTN\_Event CBTN\_Ev\_tmp;

pPriQ\_Event->cs\_PriQ.lock(); // 인큐 중 스위치 되지 않도록 락

if (pPriQ\_Event->priQ\_size >= pPriQ\_Event->priQ\_capacity) {

// Priority Queue is full /\* Expand the capacity twice, and copy the entries \*/ CBTN\_Event \*newCBT\_Event;

int newCapacity;

newCapacity = 2 \* pPriQ\_Event->priQ\_capacity;

CBTN\_Event\* newCBT\_Event = (CBTN\_Event\*)calloc((newCapacity + 1), sizeof(CBTN\_Event));

if (newCBT\_Event == NULL) {

printf("Error in expanding CompleteBinaryTree for Priority Queue !!₩n");

exit(-1);

}

for (int pos = 1; pos <= pPriQ\_Event->priQ\_size; pos++)

newCBT\_Event[pos] = pPriQ\_Event->pCBT\_Event[pos];

free(pPriQ\_Event->pCBT\_Event);

pPriQ\_Event->pCBT\_Event = newCBT\_Event;

pPriQ\_Event->priQ\_capacity = newCapacity;

} //end - if

/\* insert at the last position \*/

pos = ++pPriQ\_Event->priQ\_size;

pPriQ\_Event->pCBT\_Event[pos].priority = ev.ev\_pri;

pPriQ\_Event->pCBT\_Event[pos].event = ev;

/\* up-heap bubbling \*/

while (pos != POS\_ROOT) {

pos\_parent = pos / 2;

if (pPriQ\_Event->pCBT\_Event[pos].priority >= pPriQ\_Event->pCBT\_Event[pos\_parent].priority) {

break;

// if the priority of the new packet is lower than its parent's priority, just stop up-heap bubbling

}

else {

CBTN\_Ev\_tmp = pPriQ\_Event->pCBT\_Event[pos\_parent];

pPriQ\_Event->pCBT\_Event[pos\_parent] = pPriQ\_Event->pCBT\_Event[pos];

pPriQ\_Event->pCBT\_Event[pos] = CBTN\_Ev\_tmp;

pos = pos\_parent;

}

} // end - while

pPriQ\_Event->cs\_PriQ.unlock();

return &(pPriQ\_Event->pCBT\_Event[pPriQ\_Event->pos\_last].event);

}

/\* PriorityQueue\_Event.cpp (2) \*/

Event\* dePriQ\_Event(PriQ\_Event\* pPriQ\_Event) {

Event\* pEv, ev;

CBTN\_Event CBTN\_Ev\_tmp;

int pos, pos\_root = 1, pos\_last, pos\_child;

if (pPriQ\_Event->priQ\_size <= 0)

return NULL; // Priority queue is empty

pPriQ\_Event->cs\_PriQ.lock(); // 동작 중 스위치 되지 않도록 락

pEv = (Event\*)calloc(1, sizeof(Event));

\*pEv = pPriQ\_Event->pCBT\_Event[1].event; // get the packet address of current top

pos\_last = pPriQ\_Event->priQ\_size;

pPriQ\_Event->priQ\_size--;

if (pPriQ\_Event->priQ\_size > 0) {

/\* put the last node into the top position \*/

pPriQ\_Event->pCBT\_Event[pos\_root] = pPriQ\_Event->pCBT\_Event[pos\_last];

/\* down heap bubbling \*/

pos = pos\_root;

while (hasLeftChild(pos, pPriQ\_Event)) {

pos\_child = pos \* 2;

if (hasRightChild(pos, pPriQ\_Event)) {

if (pPriQ\_Event->pCBT\_Event[pos\_child].priority >

pPriQ\_Event->pCBT\_Event[pos\_child + 1].priority)

pos\_child = pos \* 2 + 1; // if right child has higher priority, then select it

}

/\* if the Event in pos\_child has higher priority than Event in pos, swap them \*/

if (pPriQ\_Event->pCBT\_Event[pos\_child].priority >= pPriQ\_Event->pCBT\_Event[pos].priority)

break;

else {

CBTN\_Ev\_tmp = pPriQ\_Event->pCBT\_Event[pos];

pPriQ\_Event->pCBT\_Event[pos] = pPriQ\_Event->pCBT\_Event[pos\_child];

pPriQ\_Event->pCBT\_Event[pos\_child] = CBTN\_Ev\_tmp;

}

pos = pos\_child;

} // end while

} // end if

pPriQ\_Event->cs\_PriQ.unlock();

return pEv;

}

void printPriQ\_Event(PriQ\_Event\* pPriQ\_Event) {

int pos, count;

int eventPriority;

int level = 0, level\_count = 1;

Event\* pEv;

if (pPriQ\_Event->priQ\_size == 0) {

printf(" PriorityQueue\_Event is empty !!\n");

return;

}

pos = 1;

count = 1;

level = 0;

level\_count = 1; // level\_count = 2^^level

printf("\n CompBinTree :\n", level);

while (count <= pPriQ\_Event->priQ\_size) {

printf(" level%2d : ", level);

for (int i = 0; i < level\_count; i++) {

pEv = &(pPriQ\_Event->pCBT\_Event[pos].event);

eventPriority = pEv->ev\_pri;

printEvent(pEv);

pos++;

if ((count % EVENT\_PER\_LINE) == 0)

printf("\n ");

count++;

if (count > pPriQ\_Event->priQ\_size)

break;

}

printf("\n");

level++;

level\_count \*= 2;

} // end - while

printf("\n");

}

**12.3 시뮬레이션 구성**

/\* 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 10

#define PLUS\_INF INT\_MAX

#define MAX\_ROUND 1000

#endif

(1) SimParam.h

**12.4 Event 생성 및 처리 Thread**

/\* Thread.h \*/

#ifndef TH

#define TH

#include <mutex>

#include "priorityQueue\_Event.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]; // 생성된 순서대로 저장

Event eventProcessed[TOTAL\_NUM\_EVENTS]; // 처리된 순서대로 저장

THREAD\_FLAG\* pFlagThreadTerminate;

} ThreadStatusMonitor;

typedef struct

{

mutex\* pMTX\_main; // 메인과 동시에 처리할 경우 사용

mutex\* pMTX\_thrd\_mon; // 스레드 모니터링과 동시에 처리할 경우 사용

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

(1) Thread.h

(2) Thread Event Generator

/\* Thread\_Event Genenerator.cpp \*/

#include <Windows.h>

#include <time.h>

#include "Thread.h"

#include "PriorityQueue\_Event.h"

#include "Event.h"

void Thread\_EventGenerator(ThreadParam\_Event\* pParam) {

PriQ\_Event\* pCirQ\_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;

pCirQ\_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->numEventGenerated] = \*pEv;

/\*while (enPriQ\_Event(pCirQ\_Event, \*pEv) == NULL) {

Sleep(500); // 공간이 남아 있으면 인큐

}\*/

enPriQ\_Event(pCirQ\_Event, \*pEv);

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) Thread Event Handler

/\* Thread\_Event Handler.cpp \*/

#include <Windows.h>

#include <time.h>

#include "Thread.h"

#include "PriorityQueue\_Event.h"

#include "Event.h"

void Thread\_EventHandler(ThreadParam\_Event\* pParam) {

Event\* pEv, \* pEvProc;

int myRole = pParam->role;

int myAddr = pParam->myAddr;

PriQ\_Event\* pCirQ\_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(pCirQ\_Event)) != NULL) {

//printf("Thread\_EventProc::deLL\_EventQ\_from\_HighPri\_LL\_EventQ : ");

//printEvent(pEv);

//printf("\n");

QueryPerformanceCounter(&pEv->ev\_t\_handle);

pParam->pMTX\_thrd\_mon->lock();

pEv->ev\_handler = myAddr;

pThrdMon->eventProcessed[pThrdMon->totalEventProcessed] = \*pEv; // 동작 순서 확인

pThrdMon->numEventProcessed++;

pThrdMon->totalEventProcessed++;

pParam->pMTX\_thrd\_mon->unlock();

} // 요소가 있으면 디큐

Sleep(100 + rand() % 300);

}

}

**12.5 main() function**

/\* main\_EventGen\_CirQ\_EventHandler.cpp (1) \*/

/\*

\* 파일명 : main.cpp

\* 목적 및 기본 기능

\* Multi-Thread Priority Queue

\* 작성자 : 김주환(21812158)

\* 작성일 : 2021년 5월 28일

\*/

#include <stdio.h>

#include <stdlib.h>

#include <Windows.h>

#include <mutex>

#include "Thread.h"

#include "PriorityQueue\_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);

// Thread\_EventHandler 생성 및 활성화

thrdMon.numEventProcessed = 0;

thrdParam\_EventHndlr.role = EVENT\_HANDLER; // 역할

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");

/\* main\_EventGen\_CirQ\_EventHandler.cpp (2) \*/

cs\_main.unlock();

// Thread\_EventGen 생성 및 활성화

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"); // 라운드 진행마다 진행 현황 출력을 위해 출력창 초기화

gotoxy(consHndlr, 0, 0); // 커서 초기화

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;

/\* main\_EventGen\_CirQ\_EventHandler.cpp (3) \*/

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 2 to terminate threads

break;

}

//cs\_main.unlock();

Sleep(100); // 컨텍스트 스위치

}

/\* 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(); // 종료까지 기다림

printf("Thread\_EventGenerator is terminated !!\n");

thrd\_ev\_handler.join(); // 종료까지 기다림

printf("Thread\_EventHandler is terminated !!\n");

}

**12.6 ConsoleDisplay**

/\* 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) };

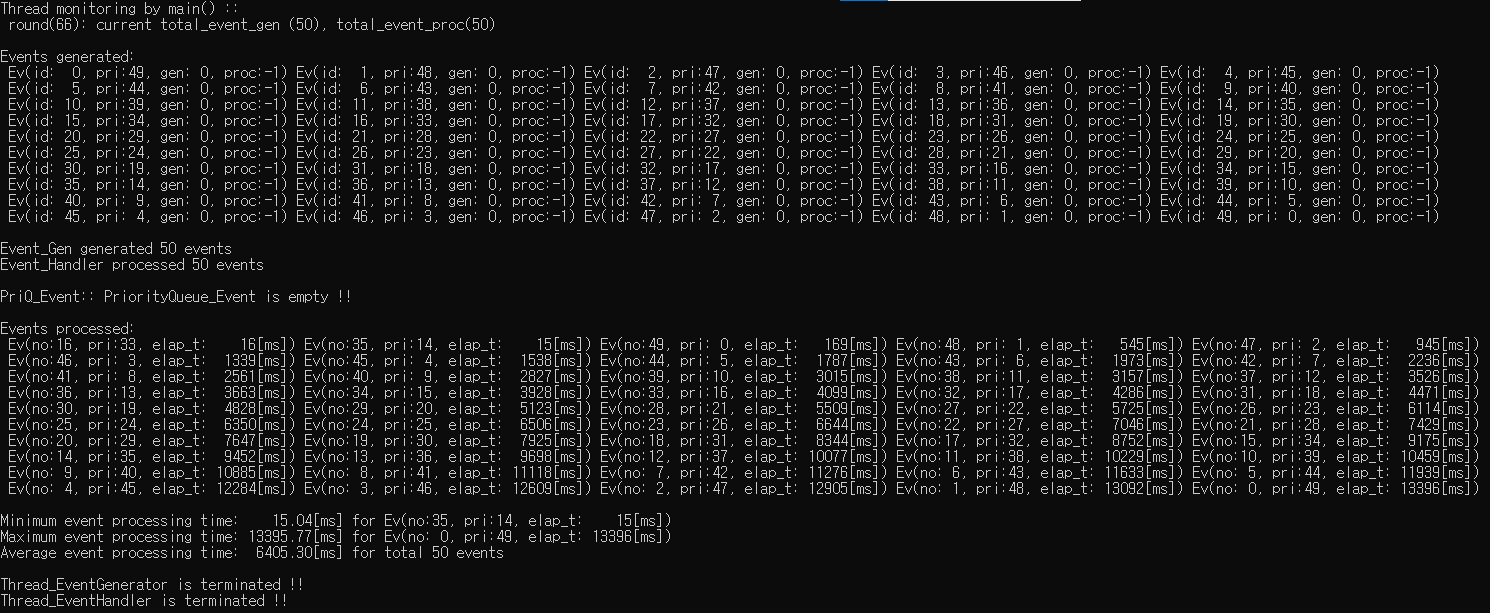
SetConsoleCursorPosition(consHndlr, coords);

}

텍스트이(가) 표시된 사진

자동 생성된 설명**12.6 Result**

(1) Initial status

(2) Final status

Oral Test

|  |
| --- |
| **(1) 다중 스레드 구조의 프로그램에서 공유 자원의 사용에 대한 Critical section(임계구역) 설정이 필요한 이유에 대하여 예를 들어 구체적으로 설명하고, 임계구역 설정을 하기 위하여 mutex를 설정하고 사용하는 방법에 대하여 예를 들어 구체적으로 설명하라.**     * Critical section 설정이 필요한 이유 * 은행  입금과 출금이 각각 스레드로 동작하고 계좌 잔고를 공유 자원으로 사용하는 방식으로 은행이 운영되고 있다고 가정합니다. 이 때 임계구역을 설정하지 않는다면 동시에 잔고의 값을 불러옵니다. 만약 1) 출금(80)이 먼저 끝나고 2) 입금(70)이 나중에 이루어진다면(이전 잔고 : 100) 1) 에서는 100 – 80 = 20이 되겠지만 2) 에서는 100 + 70 = 170이 될 것입니다. 원래대로라면 100 – 80 + 70 = 90이 되어야 합니다.  이러한 문제를 사전에 예방하고자 임계구역의 설정이 필요합니다. * mutex를 설정하고 사용하는 방법 * 헤더 #include <mutex> using namespace std; * 번수 선언 mutex pMTX; * 임계구역 설정 |

|  |
| --- |
| **(2) 스레드를 생성하는 방법과 생성되는 스레드에 파라메터를 전달하는 방법에 대하여 예를 들어 구체적으로 설명하라.**     * 스레드 생성 * 헤더 파일 #include <thread>를 통해 헤더 파일을 지정한다. * 스레드 함수의 구현 프로그램에 포함되는 함수 중, 병렬로 실행되어야 하는 함수를 스레드로 지정합니다. 구조체 포인터를 통해 스레드 생성 및 실행에 관련된 정보를 main() 함수로부터 전달받으며, 해당 구조체는 필요에 따라 정의할 수 있습니다. * 생성 thread 사용할\_스레드\_이름 ( 스레드 함수, 구조체 주소 ); * 스레드에 파라메터 전달 필요에 따라 파라메터 항목들을 포함하는 구조체를 정의합니다. 기본적으로 mutex에 관한 정보, 공유자원에 대한 정보, 파일 입출력에 관련된 정보를 포함합니다. |

|  |
| --- |
| **(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 대해 상세하게 설명하라.**       * 실행창 초기화 main문의 반복문 내에서 주기적으로 상태를 출력합니다. 이 때 반복문은 매 실행마다 system(“cls”);를 사용하여 실행창을 지웁니다. 이 함수의 헤더파일은 Windows.h입니다. * 구조체, 구조체 변수의 변경 구조체는 생성된 각 스레드 내부에서 동작의 순서를 확인하기 위해 생성하거나 처리한 이벤트를 배열에 순서대로 저장합니다. 구조체의 변수는 생성된 각 스레드 내부에서 각 라운드가 끝날 때 mutex를 사용하여 다른 스레드 중간에 개입하지 못하도록 만들고 변경합니다. |

|  |
| --- |
| **(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.**   * 내부 구조 우선순위 큐는 동적 확장성 배열을 사용한 완전 이진 트리 구조입니다. 공간이 가득 차면 배열의 크기를 2배만큼 확장합니다. * 동작 원리 * up-heap bubbling 자식 위치는 부모 위치의 2배입니다. 이때 자식의 우선순위가 부모 보다 높으면 스왑하고 위의 동작을 반복합니다. 반대의 경우에는 반복을 탈출합니다. * down-heap bubbling last node를 root로 변경하고 배열의 크기를 한 칸 줄입니다. 이때 현재 위치는 root입니다. 현 위치에서 자식이 어느 쪽에 있는 지, 둘 다 있다면 어느 쪽의 우선순위가 높은 지 판단합니다. 선택된 자식의 우선순위가 부모의 것보다 높다면 스왑합니다. 자식이 없는 경우까지 반복합니다. |