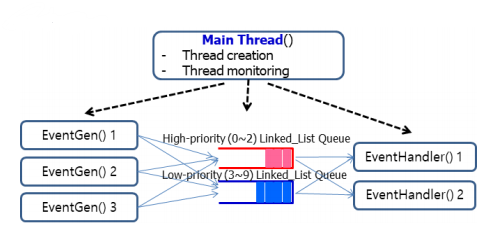
**Lab 13**

|  |
| --- |
| 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 | 2021.06.04.금 |

**13.1 Event Processing Simulation을 위한 Multi-thread 구현**

(1) Functional Block Diagram



(2) 구조체 Event 및 관련 함수

|  |
| --- |
| /\* Event.h \*/  #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[];  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, gen:%2d. proc:%2d) ", pEvent->event\_no, pEvent->event\_pri, pEvent->event\_gen\_addr, pEvent->event\_handler\_addr);  }  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;  } |

(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 List Node (DLLN), DLLN\_Ev, DLL\_EvQ

|  |
| --- |
| /\* DLL\_EvQ.h \*/  #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;  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(); // 누가 사용할지 모르기 때문에 락??  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) { // 전달 받은 이벤트가 NULL이면 NULL 반환  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) { // 동적할당에 실패하면 NULL 반환  printf("Error in enDLL\_EvQ :: memory allocation for new DLLN failed !!\n");  printf("Press any key to continue ...\n");  getc(stdin);  return NULL;  }  pLN\_Ev->pEv = pEv; // 새로운 이벤트  pEvQ->cs\_EvQ.lock();  if (pEvQ->num\_event == 0) { // 큐가 비어있는 경우  pEvQ->front = pEvQ->back = pLN\_Ev; // front와 back은 동일  pLN\_Ev->prev = pLN\_Ev->next = NULL; // 큐 내부에 하나의 이벤트만 존재하기 때문에 양 옆은 NULL  pEvQ->num\_event = 1; // 큐 내부에 하나의 이벤트만 존재  }  else { // 큐가 비어있지 않은 경우  pLN\_Ev->prev = pEvQ->back; // 큐의 끝에 연결  pEvQ->back->next = pLN\_Ev; // 큐의 끝에 연결  pEvQ->back = pLN\_Ev; // 큐의 끝에 연결  pLN\_Ev->next = NULL; // 큐의 back임을 명시  pEvQ->num\_event++; // 큐 내부 수 1만큼 증가  }  pEvQ->cs\_EvQ.unlock();  return pLN\_Ev->pEv;  }  /\* DLL\_EvQ.cpp (2) \*/  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; // 큐가 비어있으면 NULL 반환  }  else {  pLN\_Ev\_OldFront = pEvQ->front; // 추출할 이벤트  pEv = pEvQ->front->pEv; // 추출할 이벤트 저장  pEvQ->front = pEvQ->front->next; // front 재조정  if (pEvQ->front != NULL) // 새로운 front가 NULL이 아닌 경우  pEvQ->front->prev = NULL; // 큐의 front임을 명시  pEvQ->num\_event--; // 큐 내부 수 1만큼 감소  free(pLN\_Ev\_OldFront); // release memory for the current front DLLN  pEvQ->cs\_EvQ.unlock();  return pEv;  }  }  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;  }  } |

**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];  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;  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 ~ 3) Thread\_EventGenerator() & Thread\_EventHandler()

|  |
| --- |
| /\* Thread.cpp (1) \*/  #include <Windows.h>  #include <time.h>  #include <thread>  #include <mutex>  #include "Thread.h"  #include "DLL\_EvQ.h"  #include "Event.h"  #include "SimParams.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();  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.cpp (2) \*/  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();  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  } |

**13.4 Thread Monitoring**

(1) Thread status monitoring

텍스트이(가) 표시된 사진

자동 생성된 설명

(2) 추가적인 모니터링 결과 출력(지정된 위치에서 출력)

|  |
| --- |
| /\* 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);  } |

**13.5 이벤트 처리 성능 측정 및 통계 분석**

(1) 이벤트 처리 성능 측정

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

(2) 이벤트 처리 성능의 통계 분석

텍스트이(가) 표시된 사진

자동 생성된 설명

**13.6 main() function**

(1) main() function

|  |
| --- |
| /\*  \* File Name : main\_EventGen\_DLL\_EvQ\_EventProc.cpp  \* Purpose and basic functionality  \* : Doubly Linked List 형태의 FIFO Queue 구현  \* Writer : 김주환(21812158)  \* Date : 2021. 06. 04.  \*/  #define \_CRT\_SECURE\_NO\_WARNINGS  #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];  /\* main\_EventGen\_DLL\_EvQ\_EventProc.cpp (2) \*/  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();  }  /\* main\_EventGen\_DLL\_EvQ\_EventProc.cpp (3) \*/  /\* 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("\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");  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 .....)  /\* main\_EventGen\_DLL\_EvQ\_EventProc.cpp (4) \*/  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");  } |

(3) Example output (초기 상태)

텍스트, 검은색이(가) 표시된 사진

자동 생성된 설명

(4) Example output (최종 완료 상태)

텍스트이(가) 표시된 사진

자동 생성된 설명

Oral Test

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **(1) 일반 함수와 스레드의 차이점을 다음과 같은 표를 만들어 비교하여 설명하라.**   |  |  |  | | --- | --- | --- | | 항목 | 일반 함수 | 스레드 | | 인수(parameter/argument)  전달 | call-by-value 방식  call-by-pointer 방식  call-by-reference 방식 | 구조체로 원하는 정보의 집합(Thread Parameter)을 만들어서 call-by-pointer 방식으로 전달 | | 함수 원형에서의  return type 지정 | 함수를 생성할 때 지정한 자료형에 따라 return type이 지정된다. | void로 지정 | | 함수/스레드의 실행 결과 및 exit code의 전달 | return : 함수 종료  exit : 프로세스 종료  함수 내부에서 계산된 결과를 return 하여 main() 문으로 반환한다. | return : 스레드 내부의 여러 함수들이 전부 반환되어야 종료  exit : 에러코드를 반환하고 프로세스 종료 | | 함수/스레드의 호출/생성 | 함수 이름(인수) | thread(함수 이름, 전달 구조체); | |

|  |
| --- |
| **(2) Doubly Linked List 기반의 FIFO Queue의 기본 동작 (enQueue(), deQueue(), isEmpty())이 어떻게 실행되는가에 대하여 상세하게 설명하라.**     * 구조 자기참조 구조체를 통한 연결형 리스트 해당 구조체는 이전 이벤트, 다음 이벤트, 이벤트 포인터를 멤버로 가집니다. * 동작 원리 위의 자료와 같이 이벤트 생성 시 우선순위를 High와 Low로 나누어 큐에 저장하고 High를 먼저 처리하고 나중에 Low를 처리한다. 자료를 추가하고 추출할 때 각각 back과 front에 연결 해주면 되기 때문에 다른 큐에 비해 효율적이다. * enQueue() * DLLN\_Ev의 생성 * 큐 내부 이벤트 수가 0인 경우 큐의 front, back에 생성한 이벤트를 지정 생성한 이벤트의 이전, 이후를 NULL로 지정 큐 내부의 이벤트 수를 1로 변경 * 그 외의 경우 생성한 이벤트의 이전 값을 큐의 back 값으로 지정 큐의 back 다음 값을 생성한 이벤트로 지정 큐의 back 값을 생성한 이벤트로 지정 생성한 이벤트가 back 값이므로 다음 값을 NULL로 지정 * deQueue() * 큐 내부 이벤트 수가 0인 경우 NULL 반환 -> 큐가 비어있음 * 그 외의 경우 기존 front의 위치 및 이벤트를 버퍼로 이동 front 위치를 다음 노드로 이동 변경된 front의 이벤트가 NULL이 아니면 변경된 front의 이전 이벤트를 NULL로 지정 큐 내부 이벤트 수를 1만큼 감소 추출한 이벤트 반환      * isEmpty() * 큐 내부 이벤트 수가 0인지 판단 큐가 비어 있는 경우와 비어 있지 않은 경우를 나누어 실행한다. (실행 예시는 상단에서 설명했습니다.) |

|  |
| --- |
| **(3) Doubly Linked List 기반의 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.**   * monitoring mutex 제거     모니터링 관련 mutex를 제거 후 실행하면 라운드 반복이 멈추지 않고 MAX\_ROUND까지 반복되는 문제가 발생한다. 이는 각 스레드가 공유자원을 변경하거나 사용하는 중에 개입해서 생각한 대로 모니터링 변수가 변경되지 않아서 발생한 문제다.   * monitoring mutex 추가     위에서 발생한 문제를 해결하기 위해서는 모니터링과 관련된 변수가 각 스레드 실행에서 변경되거나 관련된 동작을 실행할 때 mutex를 통해 lock()해주고 그 외의 경우에는 unlock()해주면 문제없이 실행된다. |

|  |
| --- |
| **(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.**   * 헤더 Windows.h, time.h * 시간 변수 LARGE\_INTEGER t\_gen; // 생성 시간  LARGE\_INTEGER t\_proc; // 처리 시간  double t\_elapsed; // 경과한 시간 위의 변수는 구조체 Event 내에 선언해야 한다. * 주파수 main()에서 LARGE\_INTEGER pc\_freq;로 주파수를 저장할 변수를 선언한다. QueryPerformanceFrequency(&pc\_freq);를 이용해 주파수를 저장 주파수를 저장한 주소를 스레드 파라메터에 저장한다. * 시간 저장(Event\* pEv;) 생성 스레드에서 QueryPerformanceCounter(&pEv->t\_gen);를 이용해 생성 시간을 저장 처리 스레드에서 QueryPerformanceCounter(&pEv->t\_proc);를 이용해 처리 시간을 저장 * 계산 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; } 저장한 특정 이벤트 처리 시간에서 생성 시간을 뺀 값에 주파수를 나눔 이 때 각 변수에는 “처리 시간.QuardPart”와 같은 형식을 사용 * 단위 변환 ms로 표현하기 위해서는 계산한 결과값에 1000을 곱함 |