

HW 4 Report

히프 정렬을 이용한 선점형 프로세스 스케줄러

자료구조 및 알고리즘 이해 (19-2, C070-1) / 이정원 교수님

201120696 최제현

문제분석 (요구사항)

- Heap을 이용하여 선점형 Process Scheduler를 구현한다.
 - Priority에 따른 우선순위 부여 (낮을수록 우선순위가 높음, minHeap)
- 사용한 자료구조/알고리즘에 대한 이해
 - Heap, Queue 자료구조
 - Heap 정렬 알고리즘
- 전체 소스코드 분석 및 이해
- 실행 결과 캡처 및 결과 분석

사용한 자료구조/알고리즘의 설명

1. Queue 자료구조

특징

- Queue라는 이름처럼 순차적으로 먼저 들어온 데이터가 먼저 나가는 자료구조이다.

- 선입선출(FIFO: First-In First-Out)

- ex. 매표소의 대기열

- 배열 방식 구현

1. 선형큐: 배열을 선형으로 사용하여 큐를 구현

- 삽입을 계속하기 위해서 요소들을 이동시켜야함
 - 문제점이 많아 사용되지 않음

2. 원형큐: 배열을 원형으로 사용하여 큐를 구현

- 큐의 전단과 후단을 관리하기 위한 2개의 변수 필요
 - front: 첫번째 요소 하나 앞의 인덱스 (삭제) / rear: 마지막 요소의 인덱스 (삽입)
 - 공백상태: front == rear
 - 포화상태: front % M == (rear+1) % M
 - 공백상태와 포화상태를 구별하기 위하여 하나의 공간은 항상 비워둔다.

- 연결리스트 방식 구현

1. 연결된 큐: 연결리스트로 구현된 큐

- 큐의 전단과 후단을 관리하기 위한 2개의 포인터 변수 필요 (front: 삭제 / rear: 삽입)

소스코드의 Queue 구현 방식 설명

소스코드상 큐는 배열 방식의 원형큐로 ProcessQueueType 구조체와 연산 함수들로 구현되어있다.

```
5  typedef struct {
6      ProcessElement *queue[MAX_ELEMENT+1];
7
8      int front;
9      int rear;
10 }ProcessQueueType;
```

queue
: MAX_ELEMENT(2,000) + 1 크기의 배열
(포화상태를 구별하기 위해 +1)

front: 첫번째 요소 하나 앞의 인덱스
rear: 마지막 요소의 인덱스

구현되어있는 Queue의 ADT(Abstract Data Type)은 다음과 같다.

- 객체: n개의 ProcessElement형으로 구성된 요소들의 순서있는 모임
- 연산:
 - createQueue() ::= 큐를 생성한다.
 - initQueue(q) ::= 큐를 초기화한다.
 - isEmptyQueue(q) ::= 큐가 비어있는지 검사한다.
 - isFullQueue(q) ::= 큐가 가득차있는지 검사한다.
 - getQueueSize(q) ::= 큐의 요소의 개수를 반환한다.
 - enqueueProcess(q, e) ::= 큐의 뒤에 요소를 추가한다.
 - dequeueProcess(q) ::= 큐의 앞에 있는 요소를 삭제하며 반환한다.
 - printQueue(s) ::= 큐를 프린트한다.

2. Heap 자료구조

특징

- 노드들이 저장하고 있는 Key들이 다음과 같은 식을 만족하는 완전이진트리
 - 최대 히프 (max heap): key (부모노드) \geq key (자식노드)
 - 최소 히프 (min heap): key (부모노드) \leq key (자식노드)
- 우선순위 큐 구현에 사용될 수 있다.
- n개의 노드를 가지고 있는 Heap의 높이는 완전이진트리이므로 $O(\log n)$
- Heap은 다음과 같은 이유로 일반적으로 배열을 이용하여 구현한다.
 - 완전이진트리이므로 각 노드에 번호를 붙일 수 있고, 부모노드와 자식 노드를 찾기가 쉽다.
 - 왼쪽 자식의 인덱스 = (부모의 인덱스) * 2
 - 오른쪽 자식의 인덱스 = (부모의 인덱스) * 2 + 1
 - 부모의 인덱스 = (자식의 인덱스) / 2

• Heap의 삽입 과정 (높이가 $O(\log n)$ 이므로 worst case $O(\log n)$)

- 새로운 요소가 들어오면, 일단 새로운 노드를 Heap의 마지막 노드에 삽입
- 삽입 후에 새로운 노드를 부모 노드들과 교환해서 Heap 성질을 만족시킨다.

• Heap의 삭제 과정 (높이가 $O(\log n)$ 이므로 worst case $O(\log n)$)

- 루트노드를 삭제한다.
- 마지막노드를 루트노드로 이동한다.
- 루트에서부터 단말노드까지의 경로에 있는 노드들을 교환하여 Heap 성질을 만족시킨다.

소스코드의 Heap 구현 방식 설명

소스코드상 Heap은 ProcessHeapType 구조체와 연산 함수들로 구현되어있다.

```
5     typedef struct {  
6         ProcessElement *heap[MAX_ELEMENT+1];  
7         int size;  
8     }ProcessHeapType;
```

heap
: MAX_ELEMENT(2,000) + 1 크기의 배열
(루트노드를 1부터 시작하기 때문에 +1)

size: 현재 heap의 요소의 개수
(마지막 노드의 인덱스 번호)

구현되어있는 Heap의 ADT(Abstract Data Type)은 다음과 같다.

- 객체: n개의 ProcessElement형의 우선 순위를 가진 요소들의 모임
- 연산:
 - createHeap() := heap을 생성한다.
 - initHeap(h) := heap을 초기화한다.
 - isEmptyHeap(h) := heap이 비어있는지 검사한다.
 - isFullHeap(h) := heap이 가득차있는지 검사한다.
 - insertMinHeap(h, e) := heap에 요소를 minHeap 기준으로 추가한다.
 - deleteMinHeap(h) := heap의 최소 요소를 삭제 및 반환 후, minHeap 기준으로 트리 정리를 한다.
 - heapSort(q) := 큐를 minHeap 기준으로 heap 정렬한다.

3. Heap 정렬 알고리즘

특징

- Heap 자료구조를 이용하여 정렬한 것
- 최소힙(MinHeap), 최대힙(MaxHeap)에 따라 각각 오름차순, 내림차순으로 정렬된다.
- 하나의 요소를 Heap에 삽입하거나 삭제할 때 시간이 $O(\log n)$ 만큼 소요되고
요소의 개수가 n 개이므로 전체적으로 $O(n \log n)$ 시간이 걸린다.
- 전체 정렬이 아닌, 가장 큰 값 몇 개만 필요할 때 유용하다.
가장 큰 값 k 개, 전체 요소 개수 n 일 때, 삽입 n 번, 삭제 k 번 $\Rightarrow (n + k)\log n$
- 정렬 과정
 1. 정렬해야 할 n 개의 요소들을 모두 heap에 삽입
 2. 삽입 된 heap의 n 개의 요소들을 삭제하며 그 순서대로 저장

소스코드의 Heap 구현 방식 설명

입력받은 queue를 정렬하는 용도로 사용되며, 정렬시 heap을 생성하고 정렬 후 메모리 해제한다.

```
64 void heapSort(ProcessQueueType *q) {  
65     ProcessHeapType *h = createHeap();  
66     initHeap(h);  
67  
68     while (!isEmptyQueue(q)) insertMinHeap(h, dequeueProcess(q));  
69     while (!isEmptyHeap(h)) enqueueProcess(q, deleteMinHeap(h));  
70  
71     free(h);  
72 }
```

heap 생성 및 초기화
큐의 모든 요소를 MinHeap에 삽입
MinHeap의 모든 요소를 다시 큐에 삽입
heap 메모리 해제

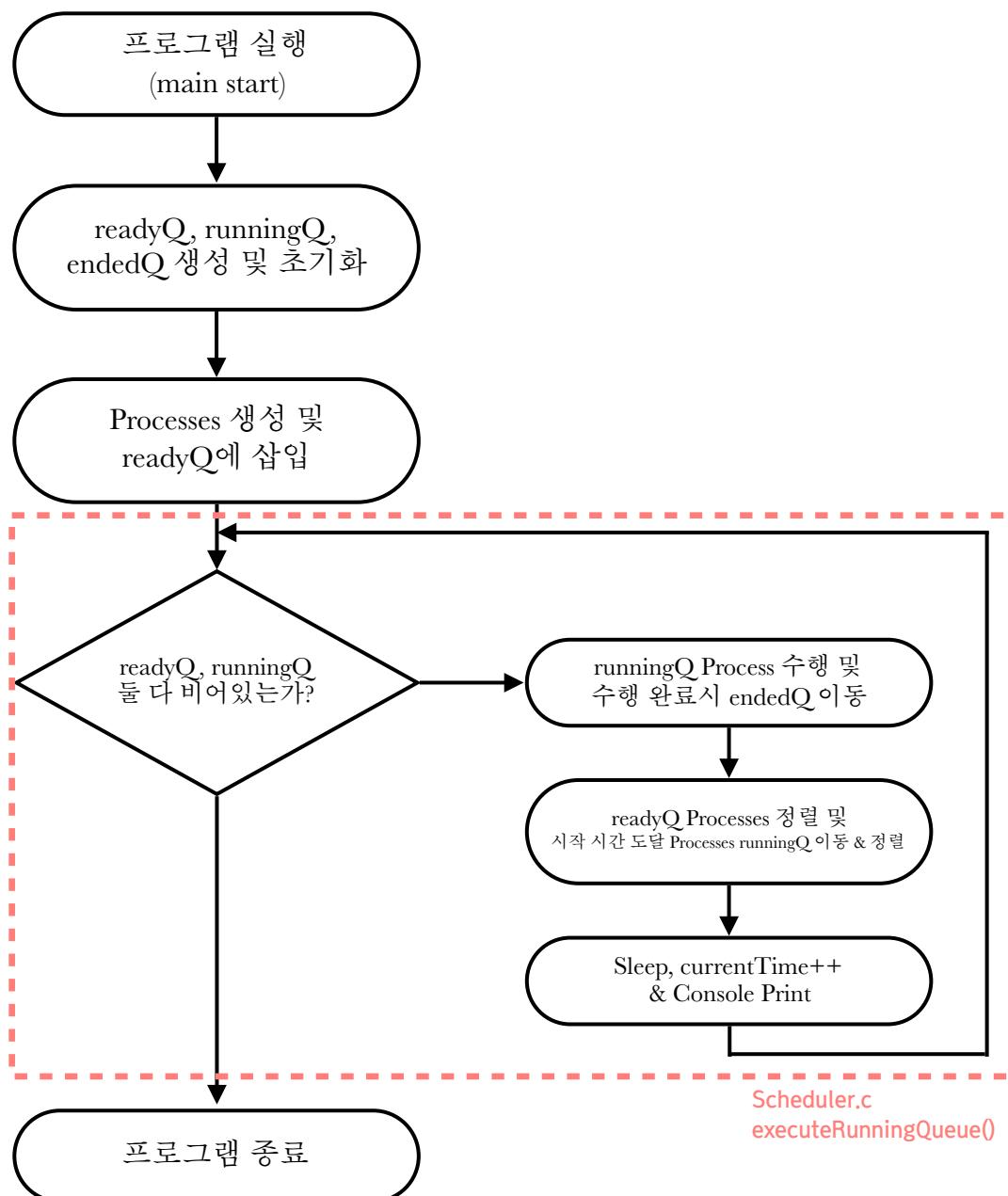
4. Heap 정렬을 이용한 선점형 프로세스 스케줄러 알고리즘

구현한 알고리즘 설명

스케줄러 알고리즘을 되도록 핵심적인 로직위주로 플로우차트를 그렸다.

빨간 네모박스 부분이 Scheduler.c 내의 executeRunningQueue 함수 내에서 수행하는 부분이다.

한 사이클마다 현재 runningQ에서 가장 낮은 값의 priority를 갖는 프로세스 1sec 수행하고
readyQ, runningQ 정렬이 필요할 시 priority 기준으로 최소힙 정렬을 진행하고
readyQ → runningQ → endedQ 각각 이동 상황에 맞춰 Process를 이동시키는 등
선점형 프로세스 스케줄러 역할을 수행한다.



소스코드 분석

기존 제공된 코드

 HW_4_Main.c
 HW_4_Main.h
 HW_4_Process.c
 HW_4_Process.h
 HW_4_ProcessHeap.c
 HW_4_ProcessHeap.h
 HW_4_ProcessQueue.c
 HW_4_ProcessQueue.h
 HW_4_Scheduler.c
 HW_4_Scheduler.h

<Main>

프로그램 실행 엔트리 파일
readyQ, runningQ, endedQ를 생성, 초기화하고
프로세스 생성하고 readyQ에 삽입한다.
조건이 비어있는 while이 주어졌으며, 본인 작성 코드에서 추가 설명하겠다.

<Process>

프로세스 구조체 타입이 선언되어있으며, 프로세스 생성 함수가 있다.
ProcessElement구조체에 name, priority, arriveTime, processingTime, restTime 등 프로세스의 기본적인 정보를 담을 수 있다.

<ProcessHeap>

Heap 자료구조를 선언 및 구현한 파일

<ProcessQueue>

배열을 이용한 원형 Queue 자료구조를 선언 및 구현한 파일

<Scheduler>

프로세스 스케줄링을 담당하며 프린트 함수도 갖고 있다.

본인 작성 코드

[ProcessHeap.c] insertMinHeap 함수

```
21 void insertMinHeap(ProcessHeapType *h, ProcessElement* item) {  
22     + if (isFullHeap(h)) return;  
23     +  
24     + int idx;  
25     + ProcessElement *temp;  
26  
27     +     idx = ++h->size;  
28     +     h->heap[idx] = item;  
29     +  
30     +     while (idx != 1 && h->heap[idx]->priority < h->heap[idx / 2]->priority) {  
31     +         temp = h->heap[idx / 2];  
32     +         h->heap[idx / 2] = h->heap[idx];  
33     +         h->heap[idx] = temp;  
34     +  
35     +         idx /= 2;  
36     +     }  
37 }
```

#22

heap의 크기가 2000개의 process를 가질 수 있도록 충분히 크게 할당되어있지만
먼저 heap이 가득찼는지 확인하고, 가득찼다면 insert를 수행하지 않도록 예외처리를 하였다.

#24~25

재사용할 변수들을 생성하였다.

int idx: 입력받은 item의 노드 index를 나타냄

processElement *temp: 노드 교환시 사용

#27~28

<heap의 마지막 노드에 새로운 노드를 삽입하는 과정>

heap의 마지막 노드의 인덱스를 의미하는 size 변수를 증감연산자로 1만큼 증가시킨 후, idx에 할당을 하였다.
또한, 마지막 노드에 입력받은 item을 삽입하였다.

#30~36

<삽입된 새로운 노드를 부모 노드들과 교환하여 MinHeap 성질을 만족시키는 과정>

초기 idx 변수의 인덱스 값은 삽입된 process 노드를 가리킨다.

idx 변수의 인덱스 노드 기준으로 부모노드와 비교하며 (min heap)

더 낮은 값의 process priority를 가지면 부모노드와 교환하는 것을 반복한다.

배열을 이용해 heap이 구현되어있으므로 현재 노드는 idx, 부모 노드는 idx / 2로 쉽게 접근할 수 있다.

한 번 교환하고 나면 부모노드의 위치가 입력받은 item이 담긴 노드의 위치가 되므로 idx를 idx / 2로 변경한다.

min priority를 가지는 item을 입력받아 교환을 반복하다보면 루트 노드까지 도달하게 되는데

루트 노드에 도달한 경우에도 비교 & 교환과정이 종료된다.

[ProcessHeap.c] deleteMinHeap 함수

```
39     ProcessElement* deleteMinHeap(ProcessHeapType *h) {
40     +     if (isEmptyHeap(h)) return NULL;
41     +
42     +     ProcessElement *item, *temp;
43     +     int child;
44     +
45     +     item = h->heap[1];
46     +
47     +     h->heap[1] = h->heap[h->size--];
48     +     child = 2;
49     +
50     +     while (child <= h->size) {
51     +         if (child < h->size && h->heap[child]->priority > h->heap[child + 1]->priority) child++;
52     +         if (h->heap[child / 2]->priority < h->heap[child]->priority) break;
53     +
54     +         temp = h->heap[child / 2];
55     +         h->heap[child / 2] = h->heap[child];
56     +         h->heap[child] = temp;
57     +
58     +         child *= 2;
59     +     }
60     +
61     +     return item;
62 }
```

#40

먼저 heap이 비어있는지 확인하고, 비어있다면 delete를 수행하지 않도록 예외처리를 하였다.

#42~43

재사용할 변수들을 생성하였다.

processElement *item: 삭제된 루트 노드를 보관하고 있다가, 반환하기 위해 사용

processElement *temp: 노드 교환시 사용

int child: 루트 노드가 삭제되면 루트로 올라간 마지막 노드의 왼쪽 자식노드 index를 의미, 오른쪽 자식노드는 child + 1로 표현

#45~48

<루트노드 삭제 및 마지막노드를 루트노드로 이동시키는 과정>

먼저 함수 종료시 반환하기 위해 item에 루트노드를 할당

루트노드에 마지막노드를 할당하고, 증감연산자로 heap의 size를 1만큼 감소시켰다.

이는 논리적으로 루트노드를 삭제하고 마지막노드를 루트노드로 옮긴 셈이 된다.

그리고, 루트 노드로 올라갔으므로 child index는 2이고, 이 값을 할당한다.

#50~59

<루트에서부터 단말노드까지의 노드들을 교환하여 MinHeap 성질을 만족시키는 과정>

#50, #58

기본적으로 반복문을 한 칸씩 내려가며 단말노드에 도달할 때까지 #51~#56의 로직을 반복하도록 하였다.

#51~56

먼저, 왼쪽 자식노드, 오른쪽 자식노드 중 더 작은 priority를 가지는 자식노드로 child index를 가리키도록 하였다.

더 작은 자식노드와 부모노드를 비교하였을 때, 부모노드의 priority가 더 작으면 반복문을 break하여 교환을 멈추도록 하였고 더 크면 부모노드와 자식노드를 교환하는 것을 반복하도록 하였다.

배열을 이용해 heap이 구현되어있으므로 현재 부모노드는 child / 2, 자식노드는 child or child+1로 쉽게 접근할 수 있다.

[ProcessHeap.c] heapSort 함수

```
64     void heapSort(ProcessQueueType *q) {  
65     +     ProcessHeapType *h = createHeap();  
66     +     initHeap(h);  
67     +  
68     +     while (!isEmptyQueue(q)) insertMinHeap(h, dequeueProcess(q));  
69     +     while (!isEmptyHeap(h)) enqueueProcess(q, deleteMinHeap(h));  
70  
71     +     free(h);  
72 } ↷
```

#65~66

heap 정렬에 사용될 heap h를 생성하고, 정상적으로 사용할 수 있도록 초기화하였다.

#68~69

입력받은 q가 Empty가 될 때까지 반복하여, process를 dequeueProcess하고 h에 insertMinHeap한다.
h가 Empty가 될 때까지 반복하여, process를 deleteMinHeap하고, 입력받은 q에 enqueueProcess한다.
minHeap에 전부 삽입하고, 순차적으로 삭제된 것을 큐에 넣는 형태로 정렬이 이뤄진다.

#71

heap 정렬이 마무리 되었으므로 동적할당된 heap h를 메모리 해제한다.

[Scheduler.c] manageReadyQueue 함수

```
2     void manageReadyQueue(int currentTime, ProcessQueueType *readyQ, ProcessQueueType *runningQ) {  
3     +     heapSort(readyQ);  
4  
5     +     ProcessElement* temp;  
6     +     for (int i = getQueueSize(readyQ); i > 0; i--) {  
7     +         temp = dequeueProcess(readyQ);  
8     +         if (temp->arriveTime == currentTime) {  
9     +             enqueueProcess(runningQ, temp);  
10    +             heapSort(runningQ);  
11    +         }  
12    +         else {  
13    +             enqueueProcess(readyQ, temp);  
14    +         }  
15    +     }  
16 }
```

#3

오버헤드가 존재할 수 있겠지만, 실제 OS처럼 readyQueue를 manage하는 타이밍에
새로운 process가 언제든 readyQ에 들어와있을 수 있다고 가정하고 readyQ를 매번 heapSort 하도록 하였다.

#5~15

priority 오름차순으로 정렬된 readyQ의 모든 프로세스를 순서대로 한번씩 dequeue하고
arriveTime과 currentTime을 비교하여 runningQ에 들어갈 타이밍인지 확인하였다.

arriveTime에 도달한 프로세스를 발견하면 runningQ에만 enqueue하여 readyQ에서는 옮겨지게 된다.

runningQ에 enqueue하는 삽입 타이밍에 runningQ를 heapSort 하여 기준으로 priority-based execute할 수 있도록 하였다.

arriveTime에 아직 도달하지 않았다면 다시 readyQ에 enqueue하게 된다.

readyQ의 size만큼 반복한 작업이므로 자연스럽게 priority 오름차순 정렬되어있다.

[Scheduler.c] executeRunningQueue 함수

```
18     int executeRunningQueue(int *currentTime, ProcessQueueType *readyQ, ProcessQueueType *runningQ,
19                             ProcessQueueType *endedQ) {
20         +     if (!isEmptyQueue(runningQ)) {
21             +         ProcessElement* runningProcess = runningQ->queue[runningQ->front + 1 % MAX_ELEMENT];
22             // peek
23             +             if (--(runningProcess->restTime) == 0) {
24                 +                 enqueueProcess(endedQ, dequeueProcess(runningQ));
25
26             +                 printf("\n\n%s process will be terminated.\n", runningProcess->name);
27             +                 printf("End time of %s process : %d sec\n\n", runningProcess->name,
28                   *currentTime);
29             +                 Sleep(2000);
30         }
31
32         +         manageReadyQueue(*currentTime, readyQ, runningQ);
33
34         +         system("cls");
35         +         printf("current time : %d sec (starting at 0 sec)\n\n", *currentTime);
36         +         printf("[Current Situation]\n");
37         +         printAll(readyQ, runningQ, endedQ);
38
39         +         Sleep(2000);
40
41         +         if (isEmptyQueue(readyQ) && isEmptyQueue(runningQ)) {
42             +             system("cls");
43             +             printf("\n\nAll processes have finished running.\n");
44             +             printf("Last time of process execution : %d sec\n\n\n", *currentTime);
45             +             printf("[Final Result]\n");
46             +             printAll(readyQ, runningQ, endedQ);
47             +             Sleep(5000);
48         }
49
50         +         (*currentTime)++;
51
52         +         return 1;
53     }
```

⚠ 기본적으로 과제코드와 함께 주어진 ‘과제4.exe’ 실행파일과 동일하게 동작하도록 작성하였다.

⚠ 채점 기준에 따라, 메인 함수를 수정하지 않기 위해 executeRunningQueue 함수 주도로 프로그램이 동작하도록 작성하였다.

⚠ 어디까지나 가상 스케줄링 확인 프로그램이므로, 보기 편하게 하기 위해 실제로는 2,000ms씩 Sleep 하였다. (콘솔표기는 1sec 씩 실행)

⚠ 이 프로그램에선 runningQ는 manageReadyQueue에 의해서만 삽입되도록 설계되어있고 삽입 후 heapSort 해주므로, executeRunningQueue 함수 내에서는 항상 runningQ가 priority 오름차순으로 정렬되어있다.

#19~28

<runningQ의 highest priority process를 1 sec 만큼 실행>

#19

runningQ가 비어있다면 실행하지 않는다.

#20

ProcessQueueType의 연산에 peek이 따로 없어서, 직접 객체에 접근하여 runningQ의 맨 앞 process를 찾아내 runningProcess 포인터 변수에 할당하였다.

앞서 manageReadyQueue에서 했던 방식처럼 ProcessQueueType의 연산만으로 맨 앞의 process를 찾을 수도 있겠지만 여기서는 오버헤드가 너무 심하기에 직접 찾도록 하였다.
유지보수 및 가독성이 더 좋은 코드로 만드려면 peek 연산을 추가 구현하고 사용하는게 좋을 것이지만, 과제의 범위를 벗어나므로 따로 구현하지 않았다.

#21~26

runningProcess의 restTime을 증감연산자로 1sec 만큼 먼저 감소시키고, restTime이 0이 됐는지 확인한다.
restTime = 0이면 endedQ로 이동시키는 작업을 하고, 콘솔에 프린트하여 프로세스 종료 상황을 알려준다.

#30

현 currentTime에 맞게 ReadyQ를 manage한다.

runningQ로 옮길 것이 있으면 호출한 manageReadyQueue 함수 내에서 옮겨주고 heapSort도 해준다.

#32

콘솔에 현 스케줄링 상황을 프린트 해준다.

#37

process가 수행되는 중에 따로 timer tick ISR에서 스케줄링 처리되는 실제 OS 커널구조와 많이 다르지만,
콘솔 보기 편한 용도 & 프로세스 수행되는 것처럼 느껴지기 위한 딜레이 코드이다.

#39~47

readyQ와 runningQ가 모두 비어있으면

모든 프로세스가 종료되었음을 프린트하고, 0(false)을 반환하여 프로그램을 종료하도록 하였다.

#50~52

currentTime을 1sec씩 추가하고, 반복해서 executeRunningQueue가 실행되도록 1(true)을 반환하였다.

Sleep과 currentTime을 붙여놓는 것이 덜 어색해보이는데, ‘과제4.exe’ 파일과 동일하게 동작하도록 하기위해 떨어뜨려 놓았다.

[main.c] main 함수 일부분

```
@@ -19,7 +19,7 @@ int main(void) {
 19    19        enqueueProcess(readyQ, makeProcessElement("pro4", 2, 1, 2, 2));
 20    20        enqueueProcess(readyQ, makeProcessElement("pro5", 4, 1, 3, 3));
 21    21
 22    -    while () {
 22    +    while (executeRunningQueue(&currentTime, readyQ, runningQ, endedQ)) {
 23    23
 24    24    }
 25    25 }
```

#22

readyQ, runningQ 둘 중 하나라도 empty가 아닌 경우 1을 반환받아 반복해서 executeRunningQueue를 반복 호출하도록 하였고,
readyQ, runningQ 둘 다 empty인 경우 0을 반환받아 프로그램이 종료되도록 하였다.

출력된 결과화면 캡처 및 결과에 대한 분석

프로세스 스케줄러 테스트 1

$t = 0$ 일 때, 아래 표의 순서대로 ReadyQ에 Process 삽입

Name	Priority	Arrive_time	Process_time
Pro1	1	2	3
Pro2	2	3	2
Pro3	3	1	4
Pro4	2	1	2
Pro5	4	1	3

0sec

```
current time : 0 sec (starting at 0 sec)
[Current Situation]
=====
Ready Queue Information
Name Priority Arrive Time Processing Time Rest Time
pro1 1 2 3 3 3
pro2 2 3 2 2 2
pro4 2 1 2 2 2
pro3 3 1 4 4 4
pro5 4 1 3 3 3
=====
Running Queue Information
Queue is Empty
=====
Ended Queue Information
Queue is Empty
=====
```

1sec

```
current time : 1 sec (starting at 0 sec)
[Current Situation]
=====
Ready Queue Information
Name Priority Arrive Time Processing Time Rest Time
pro1 1 2 3 3 3
pro2 2 3 2 2 2
=====
Running Queue Information
Name Priority Arrive Time Processing Time Rest Time
pro4 2 1 2 2 2
pro3 3 1 4 4 4
pro5 4 1 3 3 3
=====
Ended Queue Information
Queue is Empty
=====
```

2sec

```
current time : 2 sec (starting at 0 sec)
[Current Situation]
=====
Ready Queue Information
Name Priority Arrive Time Processing Time Rest Time
pro2 2 3 2 2 2
=====
Running Queue Information
Name Priority Arrive Time Processing Time Rest Time
pro1 1 2 3 3 3
pro4 2 1 2 2 1
pro3 3 1 4 4 4
pro5 4 1 3 3 3
=====
Ended Queue Information
Queue is Empty
=====
```

3sec

```
current time : 3 sec (starting at 0 sec)
[Current Situation]
=====
Ready Queue Information
Queue is Empty
=====
Running Queue Information
Name Priority Arrive Time Processing Time Rest Time
pro1 1 2 3 3 2
pro4 2 1 2 2 1
pro2 2 3 2 2 2
pro3 3 1 4 4 4
pro5 4 1 3 3 3
=====
Ended Queue Information
Queue is Empty
=====
```

4sec

```
current time : 4 sec (starting at 0 sec)
[Current Situation]
=====
Ready Queue Information
Queue is Empty
=====
Running Queue Information
Name Priority Arrive Time Processing Time Rest Time
pro1 1 2 3 3 1
pro4 2 1 2 2 1
pro2 2 3 2 2 2
pro3 3 1 4 4 4
pro5 4 1 3 3 3
=====
Ended Queue Information
Queue is Empty
=====
```

pro1 process will be terminated.
End time of pro1 process : 5 sec

5sec

```
current time : 5 sec (starting at 0 sec)
[Current Situation]
=====
Ready Queue Information
Queue is Empty
=====
Running Queue Information
Name Priority Arrive Time Processing Time Rest Time
pro4 2 1 2 2 1
pro2 2 3 2 2 2
pro3 3 1 4 4 4
pro5 4 1 3 3 3
=====
Ended Queue Information
Name Priority Arrive Time Processing Time Rest Time
pro1 1 2 3 3 0
=====
pro4 process will be terminated.  
End time of pro4 process : 6 sec
```

6sec

current time : 6 sec (starting at 0 sec)

[Current Situation]

Ready Queue Information
Queue is Empty

Running Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro2	2	3	2	2
pro3	3	1	4	4
pro5	4	1	3	3

Ended Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro1	1	2	3	0
pro4	2	1	2	0

7sec

current time : 7 sec (starting at 0 sec)

[Current Situation]

Ready Queue Information
Queue is Empty

Running Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro2	2	3	2	1
pro3	3	1	4	4
pro5	4	1	3	3

Ended Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro1	1	2	3	0
pro4	2	1	2	0

pro2 process will be terminated.
End time of pro2 process : 8 sec

8sec

current time : 8 sec (starting at 0 sec)

[Current Situation]

Ready Queue Information
Queue is Empty

Running Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro3	3	1	4	4
pro5	4	1	3	3

Ended Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro1	1	2	3	0
pro4	2	1	2	0
pro2	2	3	2	0

9sec

current time : 9 sec (starting at 0 sec)

[Current Situation]

Ready Queue Information
Queue is Empty

Running Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro3	3	1	4	3
pro5	4	1	3	3

Ended Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro1	1	2	3	0
pro4	2	1	2	0
pro2	2	3	2	0

10sec

current time : 10 sec (starting at 0 sec)

[Current Situation]

Ready Queue Information
Queue is Empty

Running Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro3	3	1	4	2
pro5	4	1	3	3

Ended Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro1	1	2	3	0
pro4	2	1	2	0
pro2	2	3	2	0

11sec

current time : 11 sec (starting at 0 sec)

[Current Situation]

Ready Queue Information
Queue is Empty

Running Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro3	3	1	4	1
pro5	4	1	3	3

Ended Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro1	1	2	3	0
pro4	2	1	2	0
pro2	2	3	2	0

pro3 process will be terminated.
End time of pro3 process : 12 sec

12sec

current time : 12 sec (starting at 0 sec)

[Current Situation]

Ready Queue Information
Queue is Empty

Running Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro5	4	1	3	3

Ended Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro1	1	2	3	0
pro4	2	1	2	0
pro2	2	3	2	0
pro3	3	1	4	0

13sec

current time : 13 sec (starting at 0 sec)

[Current Situation]

Ready Queue Information
Queue is Empty

Running Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro5	4	1	3	2

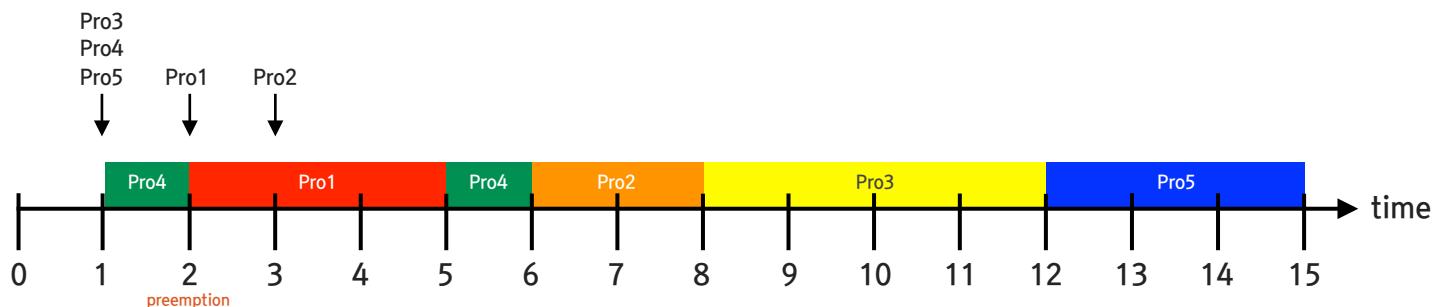
Ended Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro1	1	2	3	0
pro4	2	1	2	0
pro2	2	3	2	0
pro3	3	1	4	0

14sec	15sec
<pre>current time : 14 sec (starting at 0 sec) [Current Situation] ===== Ready Queue Information Queue is Empty ===== Running Queue Information Name Priority Arrive Time Processing Time Rest Time pro5 4 1 3 1 ===== Ended Queue Information Name Priority Arrive Time Processing Time Rest Time pro1 1 2 3 0 pro4 2 1 2 0 pro2 2 3 2 0 pro3 3 1 4 0 ===== pro5 process will be terminated. End time of pro5 process : 15 sec</pre>	<pre>All processes have finished running. Last time of process execution : 15 sec [Final Result] ===== Ready Queue Information Queue is Empty ===== Running Queue Information Queue is Empty ===== Ended Queue Information Name Priority Arrive Time Processing Time Rest Time pro1 1 2 3 0 pro4 2 1 2 0 pro2 2 3 2 0 pro3 3 1 4 0 pro5 4 1 3 0 =====</pre>

0sec에서의 ReadyQ와 2, 3sec에서 RunningQ를 보면
 프로그램 요구 사항인 히프 정렬을 이용해
 프로세스를 priority 기준 최소히프 정렬을 올바르게 진행한 것을 확인할 수 있다.

또한, 아래 예상 그림과 동일하게 선점형 시분할 프로세스 스케줄러 동작이 진행된 것을 확인할 수 있다.
 RunningQ에 현재 Running 중인 Process 보다 Priority가 높은 Process가 들어와 선점이 발생한 것은 2sec에서 확인할 수 있다.



프로세스 스케줄러 테스트 2

$t = 0$ 일 때, 아래 표의 순서대로 ReadyQ에 Process 삽입 (Pro1과 Pro4 Arrive_time을 변경)

Name	Priority	Arrive_time	Process_time
Pro1	1	9	3
Pro2	2	3	2
Pro3	3	1	4
Pro4	2	8	2
Pro5	4	1	3

0sec

```
current time : 0 sec (starting at 0 sec)
[Current Situation]
=====
Ready Queue Information
Name Priority   Arrive Time Processing Time   Rest Time
pro1    1          9            3                 3
pro2    2          3            2                 2
pro4    2          8            2                 2
pro3    3          1            4                 4
pro5    4          1            3                 3
=====
Running Queue Information
Queue is Empty
=====
Ended Queue Information
Queue is Empty
```

1sec

```
current time : 1 sec (starting at 0 sec)
[Current Situation]
=====
Ready Queue Information
Name Priority   Arrive Time Processing Time   Rest Time
pro1    1          9            3                 3
pro2    2          3            2                 2
pro4    2          8            2                 2
=====
Running Queue Information
Name Priority   Arrive Time Processing Time   Rest Time
pro3    3          1            4                 4
pro5    4          1            3                 3
=====
Ended Queue Information
Queue is Empty
```

2sec

```
current time : 2 sec (starting at 0 sec)
[Current Situation]
=====
Ready Queue Information
Name Priority   Arrive Time Processing Time   Rest Time
pro1    1          9            3                 3
pro2    2          3            2                 2
pro4    2          8            2                 2
=====
Running Queue Information
Name Priority   Arrive Time Processing Time   Rest Time
pro3    3          1            4                 3
pro5    4          1            3                 3
=====
Ended Queue Information
Queue is Empty
```

3sec

```
current time : 3 sec (starting at 0 sec)
[Current Situation]
=====
Ready Queue Information
Name Priority   Arrive Time Processing Time   Rest Time
pro1    1          9            3                 3
pro4    2          8            2                 2
=====
Running Queue Information
Name Priority   Arrive Time Processing Time   Rest Time
pro2    2          3            2                 2
pro3    3          1            4                 2
pro5    4          1            3                 3
=====
Ended Queue Information
Queue is Empty
```

4sec

```
current time : 4 sec (starting at 0 sec)
[Current Situation]
=====
Ready Queue Information
Name Priority   Arrive Time Processing Time   Rest Time
pro1    1          9            3                 3
pro4    2          8            2                 2
=====
Running Queue Information
Name Priority   Arrive Time Processing Time   Rest Time
pro2    2          3            2                 1
pro3    3          1            4                 2
pro5    4          1            3                 3
=====
Ended Queue Information
Queue is Empty
```

pro2 process will be terminated.
End time of pro2 process : 5 sec

5sec

```
current time : 5 sec (starting at 0 sec)
[Current Situation]
=====
Ready Queue Information
Name Priority   Arrive Time Processing Time   Rest Time
pro1    1          9            3                 3
pro4    2          8            2                 2
=====
Running Queue Information
Name Priority   Arrive Time Processing Time   Rest Time
pro3    3          1            4                 2
pro5    4          1            3                 3
=====
Ended Queue Information
Name Priority   Arrive Time Processing Time   Rest Time
pro2    2          3            2                 0
```

6sec

current time : 6 sec (starting at 0 sec)

[Current Situation]

Ready Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro1	1	9	3	3
pro4	2	8	2	2

Running Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro3	3	1	4	1
pro5	4	1	3	3

Ended Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro2	2	3	2	0

pro3 process will be terminated.

End time of pro3 process : 7 sec

7sec

current time : 7 sec (starting at 0 sec)

[Current Situation]

Ready Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro1	1	9	3	3
pro4	2	8	2	2

Running Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro5	4	1	3	3

Ended Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro2	2	3	2	0
pro3	3	1	4	0

8sec

current time : 8 sec (starting at 0 sec)

[Current Situation]

Ready Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro1	1	9	3	3

Running Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro4	2	8	2	2
pro5	4	1	3	2

Ended Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro2	2	3	2	0
pro3	3	1	4	0

9sec

current time : 9 sec (starting at 0 sec)

[Current Situation]

Ready Queue Information

Queue is Empty

Running Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro1	1	9	3	3
pro4	2	8	2	2
pro5	4	1	3	2

Ended Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro2	2	3	2	0
pro3	3	1	4	0

10sec

current time : 10 sec (starting at 0 sec)

[Current Situation]

Ready Queue Information

Queue is Empty

Running Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro1	1	9	3	2
pro4	2	8	2	1
pro5	4	1	3	2

Ended Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro2	2	3	2	0
pro3	3	1	4	0

11sec

current time : 11 sec (starting at 0 sec)

[Current Situation]

Ready Queue Information

Queue is Empty

Running Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro1	1	9	3	1
pro4	2	8	2	1
pro5	4	1	3	2

Ended Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro2	2	3	2	0
pro3	3	1	4	0

pro1 process will be terminated.

End time of pro1 process : 12 sec

12sec

current time : 12 sec (starting at 0 sec)

[Current Situation]

Ready Queue Information

Queue is Empty

Running Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro4	2	8	2	1
pro5	4	1	3	2

Ended Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro2	2	3	2	0
pro3	3	1	4	0
pro1	1	9	3	0

pro4 process will be terminated.
End time of pro4 process : 13 sec

13sec

current time : 13 sec (starting at 0 sec)

[Current Situation]

Ready Queue Information

Queue is Empty

Running Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro5	4	1	3	2

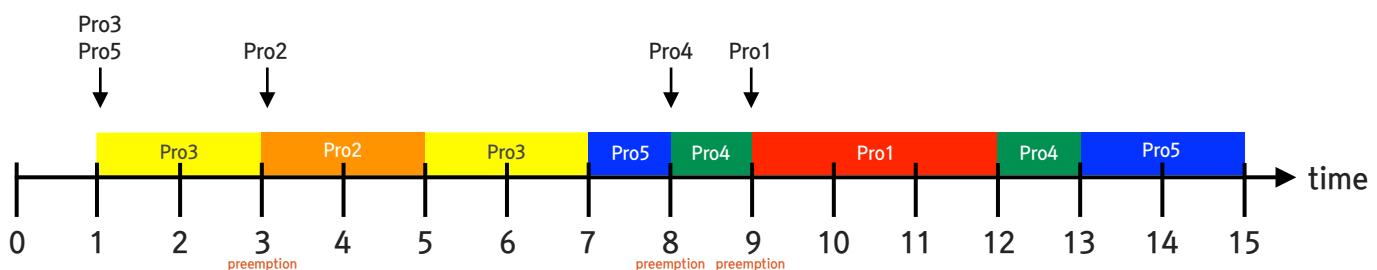
Ended Queue Information

Name	Priority	Arrive Time	Processing Time	Rest Time
pro2	2	3	2	0
pro3	3	1	4	0
pro1	1	9	3	0
pro4	2	8	2	0

14sec	15sec
<pre>current time : 14 sec (starting at 0 sec) [Current Situation] ===== Ready Queue Information Queue is Empty ===== Running Queue Information Name Priority Arrive Time Processing Time Rest Time pro5 4 1 3 1 ===== Ended Queue Information Name Priority Arrive Time Processing Time Rest Time pro2 2 3 2 0 pro3 3 1 4 0 pro1 1 9 3 0 pro4 2 8 2 0 ===== pro5 process will be terminated. End time of pro5 process : 15 sec</pre>	<pre>All processes have finished running. Last time of process execution : 15 sec [Final Result] ===== Ready Queue Information Queue is Empty ===== Running Queue Information Queue is Empty ===== Ended Queue Information Name Priority Arrive Time Processing Time Rest Time pro2 2 3 2 0 pro3 3 1 4 0 pro1 1 9 3 0 pro4 2 8 2 0 pro5 4 1 3 0 =====</pre>

테스트1과 동일하게 정상적으로 정렬되고, 스케줄링되는 것을 확인할 수 있다.

또한, 아래 예상 그림과 동일하게 선점이 3번 발생한 것을 확인 할 수 있다.



참고 자료

- 자료구조및알고리즘이해, 과제4 PDF, 소스코드
- 자료구조및알고리즘이해, 실습4 PDF, 소스코드
- 자료구조및알고리즘이해, 강의노트 6장 Queue
- 자료구조및알고리즘이해, 강의노트 8장 Heap