

HW 2 Report

Round Robin Scheduling

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

201120696 최제현

문제분석 (요구사항)

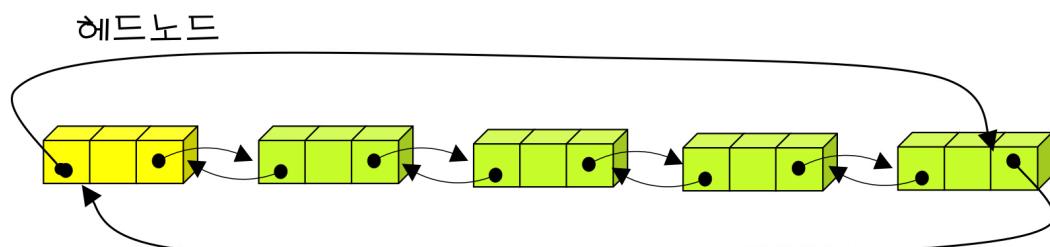
- 이중 원형 연결 리스트 (Doubly Circular Linked List) 자료구조의 이해 및 구현
- CPU Scheduling 기법, Round Robin Scheduling 기본 알고리즘의 대한 이해
- 구현된 소스코드 분석 및 이해
- 실행 결과 캡처 및 분석

사용한 자료구조/알고리즘의 설명

1. 이중 원형 연결 리스트 자료구조

특징

- 노드를 양방향으로 연결 해놓고, 양 끝단 노드도 서로 연결한 연결 리스트
- 장점으로, 단순 연결 리스트 보다 특정 노드를 빠르게 찾을 수 있다. (특히 마지막 노드)
- 단점으로, 단순 연결 리스트 보다 메모리 공간을 많이 차지하고 코드가 복잡하다.



구현한 자료구조의 설명

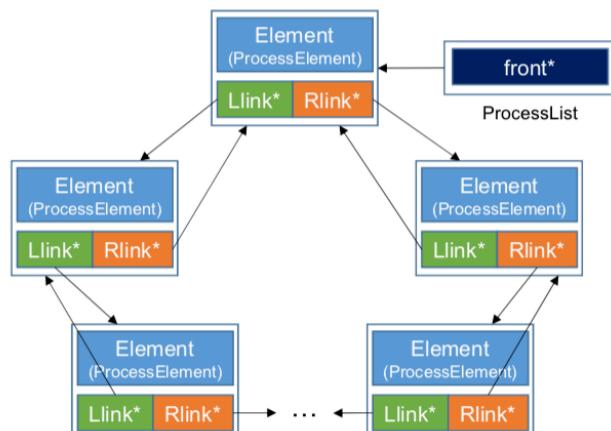
실제적인 자료의 형태는 아래의 구조체 및 이미지와 같은 형태의 이중 원형 연결 리스트 자료구조를 구성하고 있으며, front 포인터를 통해 첫번째 노드의 주소값을 갖고 있다.

```
//자료구조 선언
typedef struct ProcessElement {
    char *name;

    int executionTime;
    int remainingTime;
    int arrivalTime;
}ProcessElement;

typedef struct ProcessNode {
    ProcessElement element;
    struct ProcessNode *llink;
    struct ProcessNode *rlink;
}ProcessNode;

typedef struct ProcessList {
    struct ProcessNode *front;
}ProcessList;
```

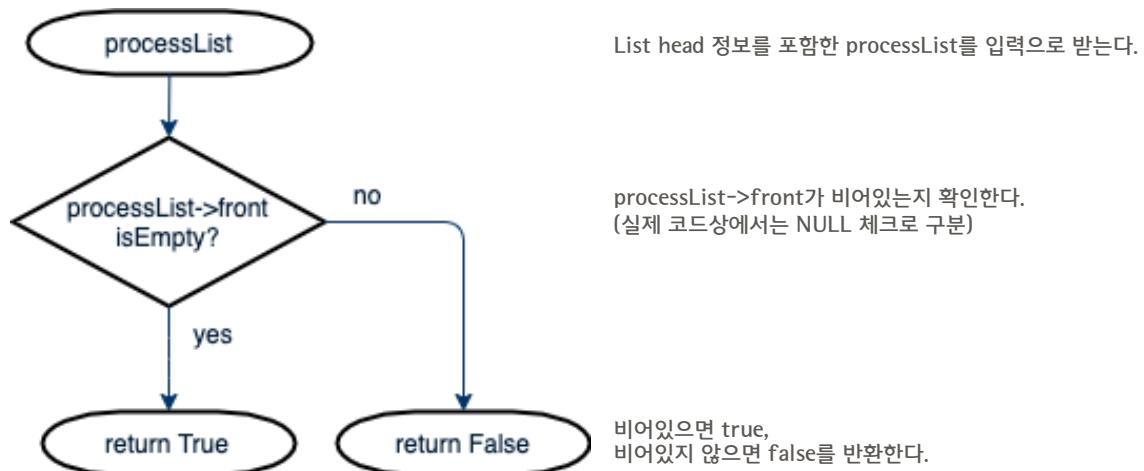


이중 원형 연결 리스트의 ADT(Abstract Data Type)은 다음과 같다.

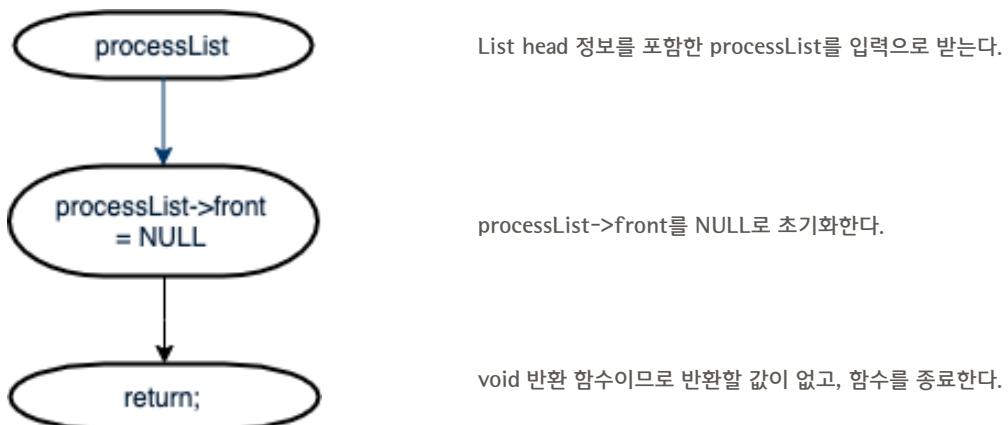
- 객체: n개의 ProcessElement형으로 구성된 모임
- 연산:
 - int isEmptyList(list) ::= 리스트가 비어있는지 확인한다.
 - void initList(list) ::= head 포인터를 Null로 만들어 리스트를 초기화한다.
 - void insertProcess(list, name, arrivalTime, executionTime) ::= 마지막 노드에 프로세스 ProcessElement 노드를 생성하고 삽입한다.
 - void removeProcess(list) ::= 현재 head 포인터가 가리키는 노드를 제거한다.
 - void printProcessList(list) ::= 프로세스 리스트를 프린트한다.
 - void runningTask(list, runningTime) ::= 리스트 내의 프로세스를 실행한다.

기본적으로 프로세스를 처리하기 위한 리스트 자료구조이기에, 프로세스 처리 관련 메서드들도 존재하는 것을 볼 수 있다. 프로세스 처리와 관련된 부분들을 제외하고 이중 원형 연결 리스트 자체의 내용에 중점을 두며 4개의 연산 함수들을 자세하게 설명하겠다. (isEmptyList, initList, insertProcess, removeProecess)

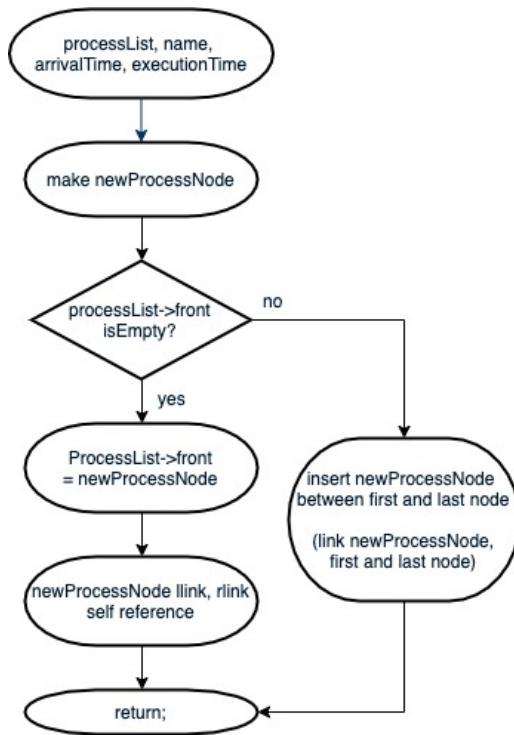
- isEmptyList



- initList



- insertProcess



List head 정보를 포함한 processList와
새로운 노드에 대한 정보인 name, arrivalTime,
executionTime을 입력으로 받는다.

새로운 PrecessNode를 동적할당으로 생성하고 입력받은 값을
할당한다.
name string 포인터 또한 동적할당으로 메모리 공간을 할당하
고, string을 copy하여 값을 넣어준다.

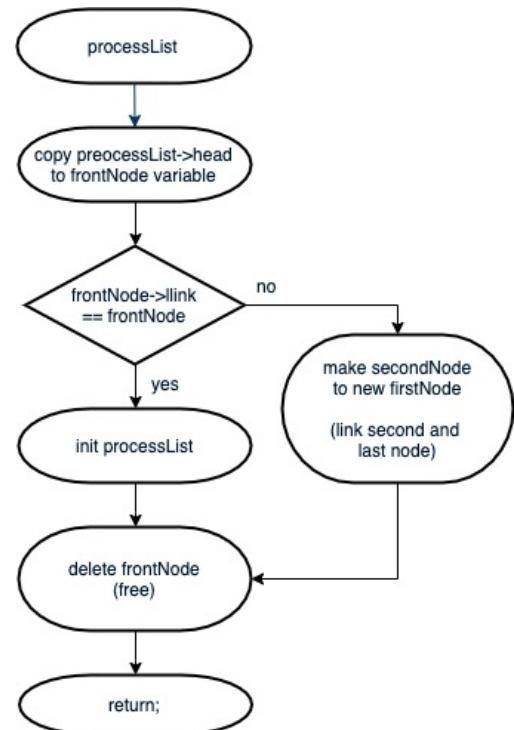
processList->front가 비어있는지 확인한다.
(실제 코드상에서는 앞서 만든 isEmptyList 함수를 사용)

1) isEmptyList: true이면
새로 만든 newProcessNode를 head포인터가 바라보는 first
Node로 설정하고, llink, rlink 포인터 모두 자기 자신을 바라보
도록 초기화 한다.

2) isEmptyList: false이면
이미 List 내에 Node가 존재한다는 의미이므로
newNode를 firstNode와 lastNode 사이에 insert한다.

void 반환 함수이므로 반환할 값이 없고, 함수를 종료한다.

- removeProecess



List head 정보를 포함한 processList를 입력으로 받는다.

frontNode 포인터 변수를 만들어
processList->front의 주소를 넘겨준다.

List의 노드가 하나만 있는지 확인하기 위해
frontNode의 llink가 자기 자신을 가리키는지 확인한다.
(isEmptyList: true 상황에서 함수 불리지 않는다고 가정)

1) List 노드가 하나만 있으면
precessList의 head를 NULL로 초기화한다.
(실제 코드상에서는 앞서 만든 initList 함수를 사용)

2) List 노드가 하나 초과로 있으면
두번째 노드를 head 포인터가 가리키는 첫번째 노드로 만든다.
이 과정에서 마지막 노드와 두번째 노드의 연결과정이 포함된다.

frontNode 주소에 동적할당 된 변수들을 제거하고
해당 주소를 가리키는 포인터 변수도 NULL로 초기화 한다.
(dangling pointer 방지 (지역 포인터 변수가 금방 제거되기에
굳이 NULL 처리할 필요는 없긴하다..))

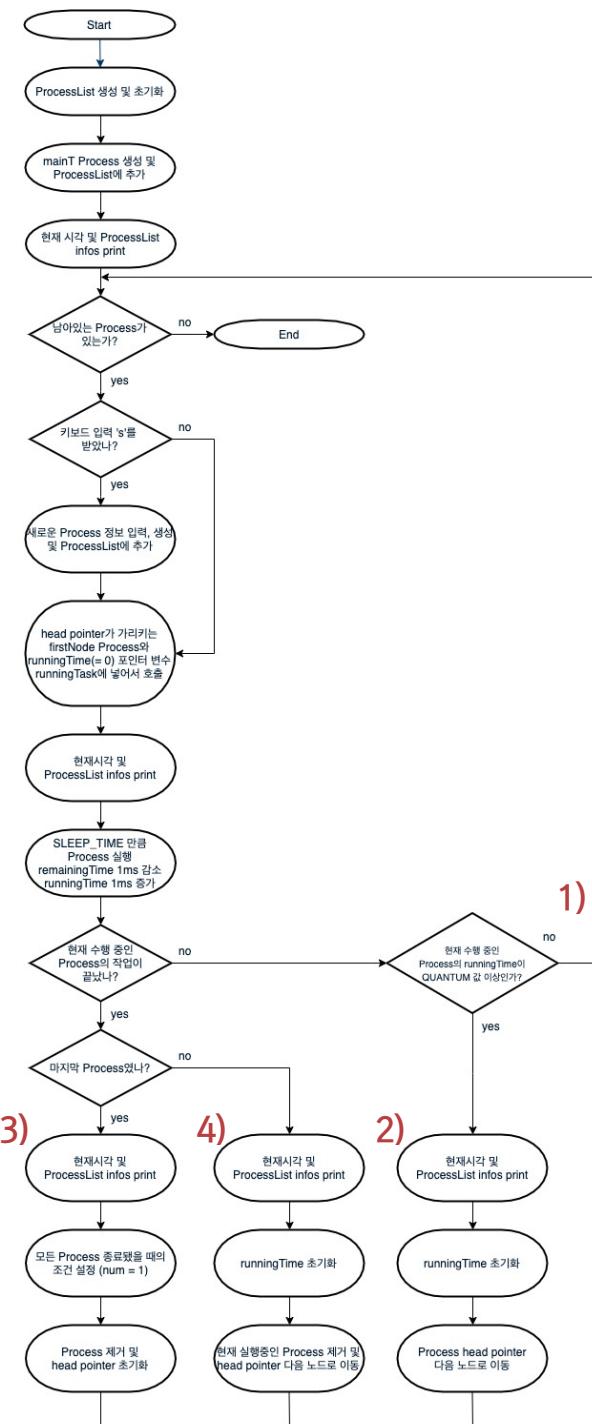
void 반환 함수이므로 반환할 값이 없고, 함수를 종료한다.

2. Round Robin Scheduling 알고리즘

특징

- CPU 스케줄링 기법 중, 선점형 스케줄링의 한 방식이다.
- 프로세스들 사이에 우선순위를 두지 않고, 순서대로 시간단위로 CPU를 할당하는 방식이다. (시분할 시스템)
- 시간 단위동안 수행한 프로세스는 준비 큐의 끝으로 밀려나게 된다.
- 문맥 전환의 오버헤드가 큰 반면, 응답시간이 짧아지는 장점이 있어 실시간 시스템에 유리하다.

구현한 알고리즘 설명



ProcessList 생성 및 초기화를 한다.

초기에 mainT라는 이름의 10ms짜리 Task를 만들고 ProcessList에 insert한다.

현재 스케줄링이 어떻게 진행되는지 알 수 있는 정보들을 print한다.

수행할 Process가 남아있으면, 수행을 하고 없다면 스케줄링 프로그램을 종료한다.

수행 중 키보드 입력 's'를 받으면

새로운 프로세스 정보를 입력 받고 입력받은 정보들로 프로세스를 생성하여 ProcessList에 추가한다.

ListProcess head pointer가 가리키는 Process와 runningTime(초기값 0)을 인자로 runningTask 함수를 실행한다.

현재 스케줄링이 어떻게 진행되는지 알 수 있는 정보들을 print한다.

SLEEP_TIME 만큼 Process를 (가상) 실행하고 Process의 remainingTime을 1ms 감소하고, runningTime을 1ms 증가한다.

현재 수행 중인 Process의 작업이 끝났는지 확인한다. 끝나지 않았다면, QUANTUM 값 이상으로 현재 Process를 수행했는지 확인하고 끝났다면, 마지막 Process였는지 확인한다.

1) 반복해서 runningTask를 수행하도록 한다.

2, 3, 4 공통) 현재 스케줄링이 어떻게 진행되는지 알 수 있는 정보들을 print한다.

2) runningTime을 초기화하고, ProcessList head pointer를 다음 노드를 바라보도록 이동한다.

3) 모든 Process 종료됐을 때의 조건을 설정하고, Process 제거 및 head pointer를 NULL로 초기화한다.

4) runningTime을 초기화하고, 완료된 Process 제거한다. 또한, ProcessList head pointer를 다음 노드를 바라보도록 이동한다.

프로그램 종료 혹은 runningTask를 수행하도록 한다.

소스코드 분석

기존 제공된 코드

초기 선언 코드들

```
1 #define _CRT_SECURE_NO_WARNINGS
2
3 #include <stdio.h>
4 #include <conio.h>
5 #include <malloc.h>
6 #include <string.h>
7 #include <Windows.h>
8
9
10 #define MAX_LENGTH 20
11 #define BAR
"====="
12 #define QUANTUM 5
13 #define SLEEP_TIME 1000
14
15 //시간을 나타내는 변수
16 int now_time = 0;
17 int num = 0;
18
19 //자료구조 선언
20 typedef struct ProcessElement {
21     char *name;
22
23     int executionTime;
24     int remainingTime;
25     int arrivalTime;
26 }ProcessElement;
27
28
29 typedef struct ProcessNode {
30
31     ProcessElement element;
32
33     struct ProcessNode *llink;
34     struct ProcessNode *rlink;
35
36 }ProcessNode;
37
38 typedef struct ProcessList {
39     struct ProcessNode *front;
40 }ProcessList;
41
42 //횟수 선언
43 int isEmptyList(ProcessList *processList);
44 void initList(ProcessList *processList);
45 void printProcessList(ProcessList *processList);
46 void insertProcess(ProcessList *processList, char *name, int arrivalTime, int executionTime);
47 void removeProcess(ProcessList *processList);
48 void runningTask(ProcessList *processList, int *runningTime);
49
```

프로그램에 사용되는 라이브러리의 헤더파일 include

전처리 매크로 정의

프로세스 name의 최대 길이는 20Byte로 할당,

각 프로세스에 최대 스케줄링 시간을 결정하는 QUANTUM 값은 5ms로 할당,

가상의 스케줄링 실행 환경에서 콘솔 상에서의 프로세스 실행 Delay시간은 1000ms로 할당

전역 변수 now_time, num 초기화 (둘 다 초기 값은 0)

자료구조에 필요한 객체들 선언

이중원형 연결리스트 노드의 값을 담당하는 Element 필드인 ProcessElement 선언

이중원형 연결리스트 노드로 사용될 struct ProcessNode 선언

이중원형 연결리스트 front(head) 노드를 가리키는 포인터를 갖는 Struct ProcessList 선언

스케줄링 및 자료구조 조작을 위한 함수 프로토타입 선언

main 함수

```
49 int main(void)
50 {
51     char name[MAX_LENGTH + 1];
52     int execution_time;
53     int runningTime = 0;
54
55     ProcessList *processList = (ProcessList*)malloc(sizeof(ProcessList));
56     initList(processList);
57
58     system("cls");
59     insertProcess(processList, "mainT", 0, 10);
60
61     printf("%s\n", BAR);
62     printf("현재시각 : %d msec\n\n", now_time);
63     printf("Process List Information\n\n");
64
65     printProcessList(processList);
66     printf("%s\n\n\n", BAR);
67
68     Sleep(SLEEP_TIME);
69
70     while (num != 1)
71     {
72
73         if (_kbhit())
74         {
75             if (_getch() == 's')
76             {
77                 printf("새로운 Process에 대한 정보를 입력해주세요(Name(영문 최대 20자 한글 최대 10자), Execution Time 순서로)\n");
78                 fflush(stdin);
79                 scanf("%s", name);
80
81                 fflush(stdin);
82                 scanf("%d", &execution_time);
83
84                 insertProcess(processList, name, now_time, execution_time);
85             }
86         }
87
88         now_time++;
89         runningTask(processList, &runningTime);
90     }
91
92     return 0;
93 }
```

지역변수 선언

name 배열, execution_time, 0으로 초기화 된 runningTime

ProcessList 포인터 변수에 동적할당 후 할당된 주소값 할당 후 ProcessList 초기화

현재시각, Process List Information을 Print

1000ms (1sec) 만큼 Sleep (맨 첫번째 프로세스 수행 사이클을 여기서 포함)

반복문 While을 통한 프로세스 생성 및 실행관련된 블럭 실행
num == 1인 조건이 실행할 프로세스가 더이상 없다는 의미

keyboard input이 발생하면 _kbhit()에 값이 들어가고, 입력된 값이 's'이면 새로운 Process 정보 입력받도록 한다.
프로세스 name을 인풋 버퍼에 남아있던 쓰레기 값 문제 없이 fflush로 초기화를 반복하며
name과, excution_time을 입력한다.
입력받고 나면, insertProcess 함수를 이용해 ProcessList에 새로운 Process를 insert한다.

현재 시간을 1ms 추가하고

현재 frontProcess와 runningTime을 인자로 넣어 runningTask 함수를 호출하여 프로세스 처리를 한다.

isEmptyList, initList, printProcessList 함수들

```
99 int isEmptyList(ProcessList *processList)
100 {
101     return (processList->front == NULL);
102 }
103
104
105
106 void initList(ProcessList *processList)
107 {
108     processList->front = NULL;
109 }
110
111 void printProcessList(ProcessList *processList)
112 {
113     ProcessNode *temp = processList->front;
114
115     printf("%-21s%-20s%-20s%-20s\n", "Name", "Arrive Time", "Execution Time", "Rest Time");
116
117     if (temp == NULL)
118         printf("          (List is empty)      \n");
119     printf("%-21s%-20d%-20d%-20d\n", temp->element.name, temp->element.arrivalTime, temp->element.executionTime, temp->element.remainingTime);
120     temp = temp->rlink;
121     while (temp != processList->front)
122     {
123         printf("%-21s%-20d%-20d%-20d\n", temp->element.name, temp->element.arrivalTime, temp->element.executionTime,
124               temp->element.remainingTime);
125         temp = temp->rlink;
126     }
127 }
```

1) isEmptyList

processList의 front 포인터 필드가 NULL인지 확인해서 비어있는지 확인하여 boolean 값을 반환한다. 비어있으면 true

2) initList

processList의 front 포인터 필드를 NULL로 초기화한다.

3) printProcessList

먼저 processList의 front 포인터가 empty인지 확인하고, empty가 아니면,

processList의 노드들을 반복적으로 꺼내서 Name, Arrive Time, Execution Time, Rest Time 정보를 프린트한다.

runningTask 함수

```
175 void runningTask(ProcessList *processList, int *runningTime)
176 {
177     int nowTask_ResidualTime;
178
179     nowTask_ResidualTime = --(processList->front->element.remainingTime);
180
181     system("cls");
182
183     printf("%s\n", BAR);
184     printf("현재시각 : %d msec\n\n", now_time);
185     printf("Process List Information\n\n");
186
187     printProcessList(processList);
188     printf("%s\n\n\n", BAR);
189
190     printf("현재 Task %s가 수행중입니다!\n\n", processList->front->element.name);
191     Sleep(SLEEP_TIME);
192     (*runningTime)++;
193
194     if (nowTask_ResidualTime == 0)
195     {
196         if (processList->front == processList->front->llink)           //남은 Process가 1개일 때
197         {
198             system("cls");
199
200             printf("%s\n", BAR);
201             printf("현재시각 : %d msec\n\n", now_time);
202             printf("Process List Information\n\n");
203
204             printProcessList(processList);
205             printf("%s\n\n\n", BAR);
206
207             printf("현재 수행되던 Task %s가 종료되었습니다\n", processList->front->element.name);
208             printf("모든 Task가 종료되었습니다\n");
209
210             num = 1;
211
212             removeProcess(processList);
213         }
214     else
215     {
216         system("cls");
217
218         printf("%s\n", BAR);
219         printf("현재시각 : %d msec\n\n", now_time);
220         printf("Process List Information\n\n");
221
222         printProcessList(processList);
223         printf("%s\n\n\n", BAR);
224
225         printf("현재 수행되던 Task %s가 종료되었습니다\n", processList->front->element.name);
226         printf("다음 Task로 넘어갑니다\n");
227         Sleep(SLEEP_TIME);
228
229         *runningTime = 0;
230         removeProcess(processList);
231     }
232
233     else if (*runningTime >= QUANTUM)                                //Time Quantum이 지났을 때
234     {
235         system("cls");
236
237         printf("%s\n", BAR);
238         printf("현재시각 : %d msec\n\n", now_time);
239         printf("Process List Information\n\n");
240
241         printProcessList(processList);
242         printf("%s\n\n\n", BAR);
243
244         printf("%d msec가 흘러 다음 Task로 넘어갑니다.\n", QUANTUM);
245         Sleep(SLEEP_TIME);
246
247         *runningTime = 0;
248         processList->front = processList->front->rlink;
249     }
250 }
```

main 함수에서 반복적으로 호출하여 process를 처리하는 함수

nowTask_ResidualTime이란 변수에 현재 frontNode의 element.remainingTime 값을 1 감소시키면서 받아온다.

현재시각, Process List Information을 Print하고,

1000ms (1sec) 만큼 Sleep한다. (가상으로 프로세스 실행하는 것을 표현)

현재 프로세스의 runningTime을 1 증가시켜준 뒤

nowTask_ResidualTime 값을 이용해 프로세스 남은 시간이 0인지 확인한다. (프로세스 제거 조건)

프로세스 제거 조건이 충족되면, 남은 프로세스가 1개인지 아닌지 확인하고

각각에 상황에 맞게 정보를 프린트해준다.

프로세스 제거 조건은 충족이 안되었지만, 정해진 QUANTUM 이상의 runingTime을 진행하였을 때 정보를 프린트하고 프로세스를 다음 프로세스를 수행하도록 front 포인터의 주소값을 바꿔주는 작업을 진행한다.

본인 작성코드는 아래에 추가적으로 설명하였다.

본인 작성 코드

insertProcess 함수

```
127
128 void insertProcess(ProcessList *processList, char *name, int arrivalTime, int executionTime)
129 {
130     ProcessNode *newProcess = (ProcessNode *)malloc(sizeof(ProcessNode));
131     newProcess->element.name = (char *)malloc(sizeof(char) * (strlen(name) + 1));
132     strcpy(newProcess->element.name, name);
133     newProcess->element.arrivalTime = arrivalTime;
134     newProcess->element.executionTime = executionTime;
135     newProcess->element.remainingTime = executionTime;
136
137     ProcessNode *frontNode = processList->front;
138
139     if (isEmptyList(processList)) {
140         processList->front = newProcess;
141         newProcess->llink = newProcess;
142         newProcess->rlink = newProcess;
143     }
144     else {
145         ProcessNode *lastNode = frontNode->llink;
146         newProcess->llink = lastNode;
147         newProcess->rlink = frontNode;
148         lastNode->llink = newProcess;
149         frontNode->llink = newProcess;
150     }
151 }
```

ProcessNode를 동적할당 받은 주소를 newProcess ProcessNode 포인터에 할당해주고
매개변수로 넘겨받은 name, arrivalTime, executionTime 값을 이용해 newProcess 주소값의 필드들에 값을 할당해 주었다.
추가적으로 name의 경우 string pointer로 넘겨받은 name 길이에 필요한 string 메모리 공간만큼 동적할당을 받아 할당하고,
strcpy를 통해 string을 copy하여 저장하였다. (이렇게 안하면 name string pointer가 가리키는 주소 참조를 해버리기 때문에, 새
로운 Process를 생성하여 name을 쓸 때마다 name 주소값을 바라보는 Process의 name들이 모두 함께 변하게 된다.)
공통적으로 frontNode가 많이 참조되므로, 따로 frontNode 포인터 변수를 만들어 processList->front의 주소를 할당해서 재사용
하였다.

먼저 processList가 Empty인지 확인하고

Empty인 경우엔 요구사항에 맞게 front 노드를 newProcess로 할당한 후, llink, rlink가 newProcess를 가리키도록 하였다.
그 외의 경우엔 frontNode의 llink인 마지막 노드를 알아내고 frontNode와 lastNode 사이에 newProcess를 연결하여 newProcess
가 마지막 노드가 되도록 하였다.

removeProcess 함수

```
152
153 void removeProcess(ProcessList *processList)
154 {
155     ProcessNode *frontNode = processList->front;
156
157     if (frontNode == frontNode->llink) {
158         initList(processList);
159     }
160     else {
161         ProcessNode *lastNode = frontNode->llink;
162         ProcessNode *secondNode = frontNode->rlink;
163         lastNode->rlink = secondNode;
164         secondNode->llink = lastNode;
165
166         processList->front = secondNode;
167     }
168
169     free(frontNode->element.name);
170     frontNode->element.name = NULL;
171     free(frontNode);
172     frontNode = NULL;
173 }
```

공통적으로 frontNode가 많이 참조되므로, 따로 frontNode 포인터 변수를 만들어 processList->front의 주소를 할당해서 재사용
하였다.

frontNode와 frontNode->llink 즉, 마지막 노드가 같은 주소값을 갖고있다면 하나의 노드만 있다는 의미이다.

하나의 노드만 있다면 앞서 만들어두었던 initList함수를 사용해 front 포인터 변수를 NULL로 초기화한다.

하나 초과의 노드가 있다면 (Empty일 때엔 안불린다고 가정) frontNode의 llink와 rlink로 lastNode, secondNode를 찾아내고
lastNode, secondNode끼리 연결시킨다. 또한, processList->front 포인터도 secondNode를 바라보도록 한다.

마지막으로 모든 경우에서 함수 진입시 할당하였던 frontNode 변수에 아직 메모리 해제되지 않은 제거되어야 할 이전 frontNode 주
소값이 있고, 이 frontNode 포인터 변수를 이용해

먼저 동적할당 받았던 name을 free하고, name 포인터 변수를 NULL로 초기화 하고

다음으로 동적할당 받았던 frontNode 구조체를 free하고, frontNode 포인터 변수를 NULL로 초기화해주며 removeProcess를 완
료한다.

runningTask 함수

```

175 void runningTask(ProcessList *processList, int *runningTime)
176 {
177     int nowTask_ResidualTime;
178     nowTask_ResidualTime = --(processList->front->element.remainingTime);
179     system("cls");
180
181     printf("%s\n", BAR);
182     printf("현재시각 : %d msec\n\n", now_time);
183     printf("Process List Information\n\n");
184
185     printProcessList(processList);
186     printf("%s\n\n\n", BAR);
187
188     printf("현재 Task %s가 수행중입니다\n\n", processList->front->element.name);
189     Sleep(SLEEP_TIME);
190     (*runningTime)++;
191
192     if (nowTask_ResidualTime == 0)
193     {
194         if (processList->front == processList->front->llink)           //남은 Process가 1개일 때
195         {
196             system("cls");
197
198             printf("%s\n", BAR);
199             printf("현재시각 : %d msec\n\n", now_time);
200             printf("Process List Information\n\n");
201
202             printProcessList(processList);
203             printf("%s\n\n\n", BAR);
204
205             printf("현재 수행되던 Task %s가 종료되었습니다\n", processList->front->element.name);
206             printf("모든 Task가 종료되었습니다\n");
207
208             num = 1;
209
210             removeProcess(processList);
211         }
212         else
213         {
214             system("cls");
215
216             printf("%s\n", BAR);
217             printf("현재시각 : %d msec\n\n", now_time);
218             printf("Process List Information\n\n");
219
220             printProcessList(processList);
221             printf("%s\n\n\n", BAR);
222
223             printf("현재 수행되던 Task %s가 종료되었습니다\n", processList->front->element.name);
224             printf("다음 Task로 넘어갑니다\n");
225             Sleep(SLEEP_TIME);
226
227             *runningTime = 0;
228             removeProcess(processList);
229         }
230     }
231     else if (*runningTime >= QUANTUM) {                                //Time Quantum이 지났을 때
232         system("cls");
233
234         printf("%s\n", BAR);
235         printf("현재시각 : %d msec\n\n", now_time);
236         printf("Process List Information\n\n");
237
238         printProcessList(processList);
239         printf("%s\n\n\n", BAR);
240
241         printf("%d msec가 흘러 다음 Task로 넘어갑니다.\n", QUANTUM);
242         Sleep(SLEEP_TIME);
243
244         *runningTime = 0;
245         processList->front = processList->front->rlink;
246     }
247 }
248 }
```

1) 남은 Process가 1개인지 확인하는 방법은 processList->front 첫번째 노드의 llink나 rlink가 자기 자신의 주소값을 가리키는지 확인하면 알 수 있다. 노드가 1개일 때는 llink, rlink 모두 자기자신을 참조하도록 하였다.

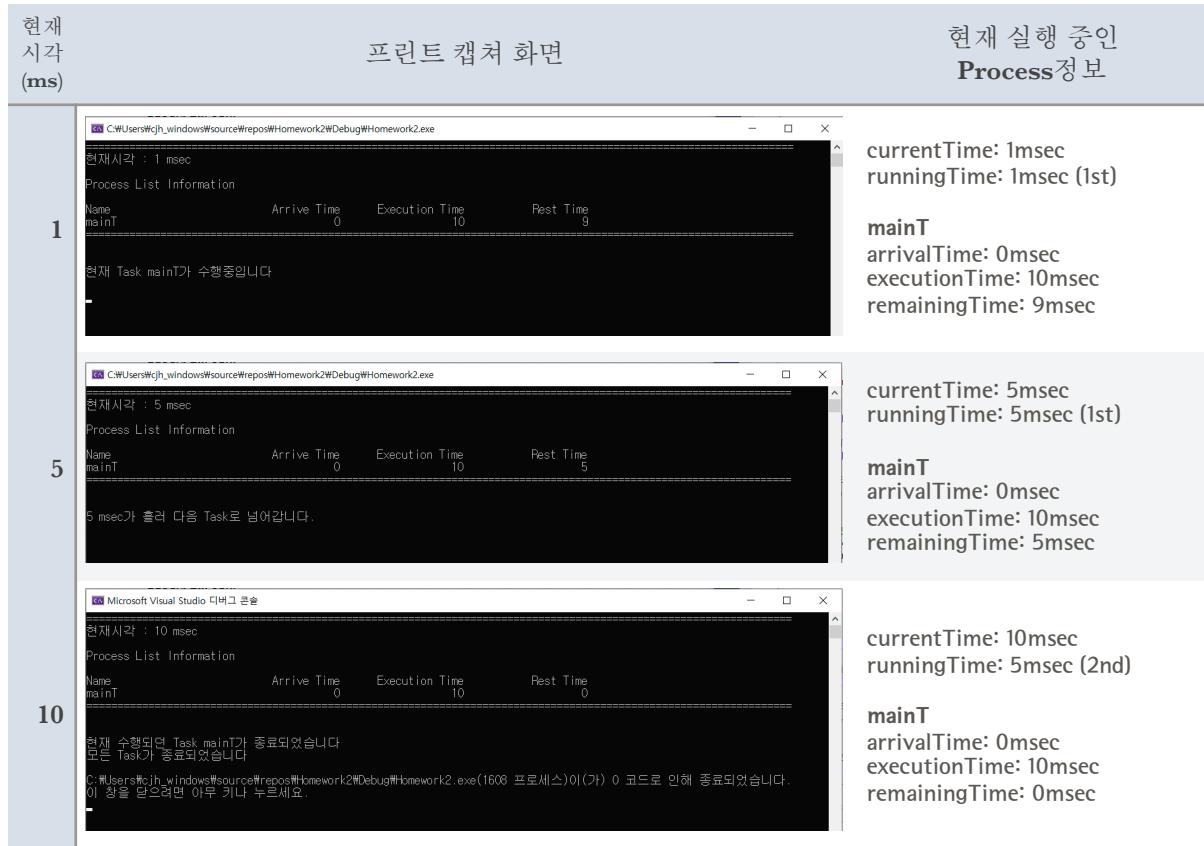
2) 종료조건인 num = 1은 이미 작업이 되어있으므로, 남은 작업인 프로세스 제거만 해주면 된다.
옵션으로 runningTime도 0으로 초기화해주어야 되는데, 바로 종료되는 프로그램이므로 추가적인 초기화를 하지는 않았다.

3) 실행해야 할 프로세스가 아직 남아있으므로 공용변수로 사용되고 있는 runningTime을 0으로 초기화해주고,
현재의 프로세스를 제거해주었다.

4) Time Quantum이 지나서 프로세스를 바꿔주어야 하는 상황이므로
runningTime을 0으로 초기화해주고, processList->front 포인터 주소를 다음으로 실행되어야 할 second node인 processList->front->rlink로 할당해주었다.

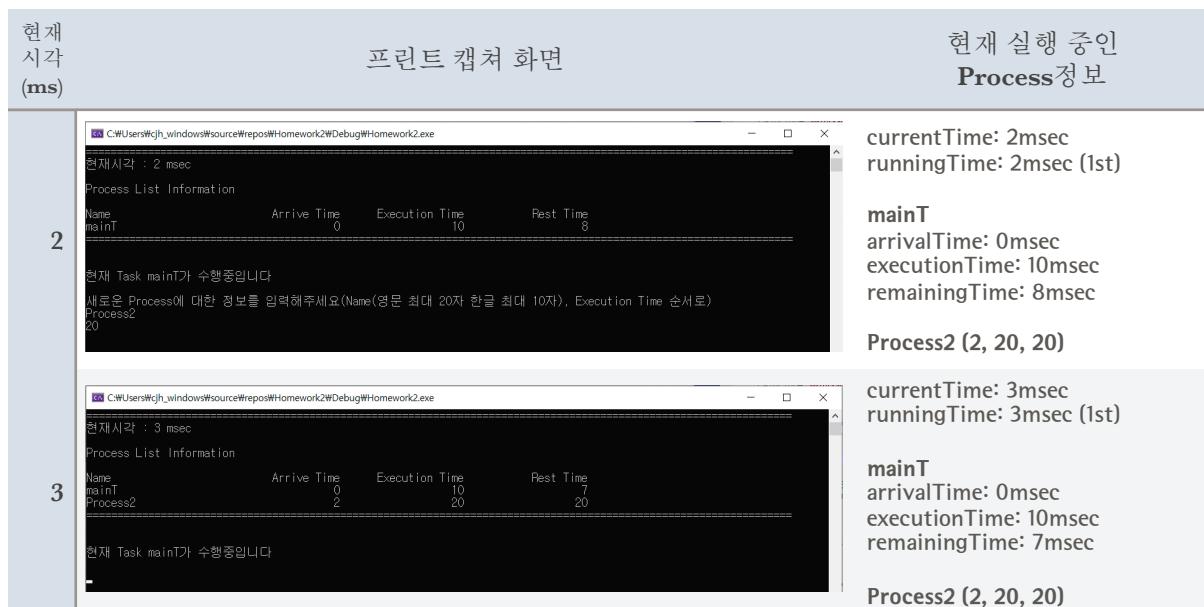
출력된 결과화면 캡쳐 및 결과에 대한 분석

mainT Process만 단독 실행 (QUANTUM=5)



스케줄링의 한 TASK의 최대 QUANTUM이 5ms이기 때문에, 1번의 프로세스 스위칭을 진행하여 mainT를 마친 것을 볼 수 있다. 프로세스는 하나만 있었기 때문에 mainT를 계속하여 처리하였고, 종료시의 currentTime을 executionTime - arrivalTime = 10msec로 예상할 수 있었고, 예상대로 결과가 나왔다.

Process 2개 실행 (QUANTUM=5)



현재 시각 (ms)	프린트 캡쳐 화면	현재 실행 중인 Process 정보
5		currentTime: 5msec runningTime: 5msec (1st) mainT arrivalTime: 0msec executionTime: 10msec remainingTime: 5msec Process2 (2, 20, 20)
6		currentTime: 6msec runningTime: 1msec (2nd) Process2 arrivalTime: 2msec executionTime: 20msec remainingTime: 19msec mainT (0, 10, 5)
11		currentTime: 11msec runningTime: 1msec (3rd) mainT arrivalTime: 0msec executionTime: 10msec remainingTime: 4msec Process2 (2, 20, 15)
15		currentTime: 15msec runningTime: 5msec (3rd) mainT arrivalTime: 0msec executionTime: 10msec remainingTime: 0msec Process2 (2, 20, 15)
16		currentTime: 15msec runningTime: 1msec (4th) Process2 arrivalTime: 2msec executionTime: 20msec remainingTime: 14msec mainT arrivalTime: 0msec executionTime: 10msec remainingTime: 0msec Process2 (2, 20, 14)
30		currentTime: 30msec runningTime: 5msec (6th) Process2 arrivalTime: 2msec executionTime: 20msec remainingTime: 0msec

mainT, Process2 프로세스의 총 executionTime은 30ms인데, 스케줄링의 한 TASK의 최대 QUANTUM이 5ms이기 때문에, 총 5번의 프로세스 스위칭을 진행하며 mainT, Process2를 마친 것을 볼 수 있다. mainT 프로세스가 먼저 종료된 후, 상대적으로 실행해야 할 양이 많은 Process2를 단독으로 계속하여 처리하는 사이클이 있었고, 예상하던 총 수행시간 currentTime은 (프로세스의 총 executionTime) - (초기 arrivalTime) = (10 + 20) - 0 = 30msec로 예상할 수 있었고, 예상대로 결과가 나왔다.

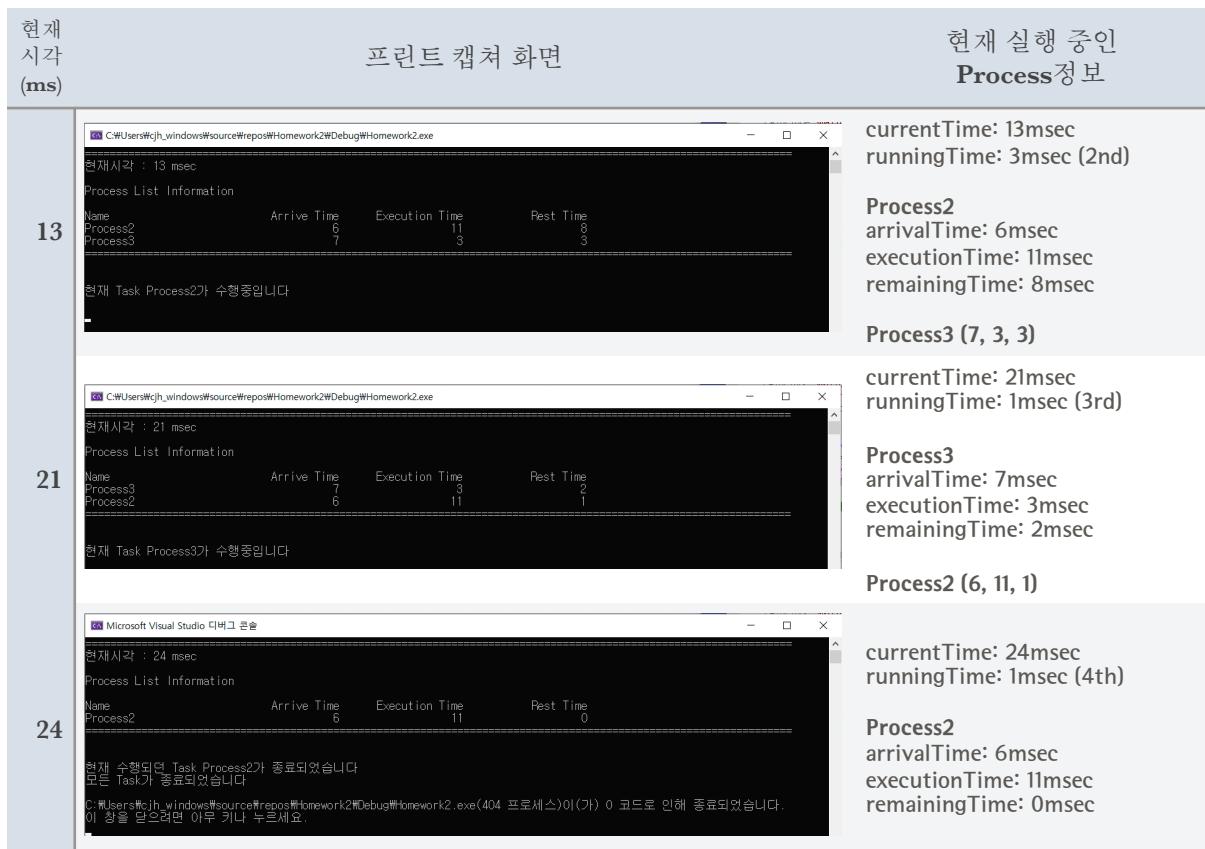
여러개의 Process 실행 (QUANTUM=5)

현재 시각 (ms)	프린트 캡쳐 화면	현재 실행 중인 Process 정보																				
4	<p>현재시각 : 4 msec</p> <p>Process List Information</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Arrive Time</th> <th>Execution Time</th> <th>Rest Time</th> </tr> </thead> <tbody> <tr> <td>mainT</td> <td>0</td> <td>10</td> <td>6</td> </tr> <tr> <td>Process3</td> <td>1</td> <td>20</td> <td>3</td> </tr> <tr> <td>Process4</td> <td>3</td> <td>3</td> <td>20</td> </tr> </tbody> </table> <p>현재 Task mainT가 수행중입니다.</p>	Name	Arrive Time	Execution Time	Rest Time	mainT	0	10	6	Process3	1	20	3	Process4	3	3	20	<p>currentTime: 1msec runningTime: 4msec (1st)</p> <p>mainT arrivalTime: 0msec executionTime: 10msec remainingTime: 6msec</p> <p>Process3 (1, 20, 20) Process4 (3, 3, 3)</p>				
Name	Arrive Time	Execution Time	Rest Time																			
mainT	0	10	6																			
Process3	1	20	3																			
Process4	3	3	20																			
7	<p>현재시각 : 7 msec</p> <p>Process List Information</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Arrive Time</th> <th>Execution Time</th> <th>Rest Time</th> </tr> </thead> <tbody> <tr> <td>Process3</td> <td>1</td> <td>20</td> <td>18</td> </tr> <tr> <td>Process4</td> <td>3</td> <td>3</td> <td>3</td> </tr> <tr> <td>mainT</td> <td>0</td> <td>10</td> <td>5</td> </tr> <tr> <td>Process5</td> <td>6</td> <td>31</td> <td>31</td> </tr> </tbody> </table> <p>현재 Task Process3가 수행중입니다.</p>	Name	Arrive Time	Execution Time	Rest Time	Process3	1	20	18	Process4	3	3	3	mainT	0	10	5	Process5	6	31	31	<p>currentTime: 7msec runningTime: 2msec (2nd)</p> <p>Process3 arrivalTime: 1msec executionTime: 20msec remainingTime: 18msec</p> <p>Process4 (3, 3, 3) mainT (0, 10, 5) main5 (6, 31, 31)</p>
Name	Arrive Time	Execution Time	Rest Time																			
Process3	1	20	18																			
Process4	3	3	3																			
mainT	0	10	5																			
Process5	6	31	31																			
64	<p>현재시각 : 64 msec</p> <p>Process List Information</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Arrive Time</th> <th>Execution Time</th> <th>Rest Time</th> </tr> </thead> <tbody> <tr> <td>Process5</td> <td>6</td> <td>31</td> <td>0</td> </tr> </tbody> </table> <p>현재 수행되는 Task Process5가 종료되었습니다. 모든 Task가 종료되었습니다.</p> <p>C:\Users\chj_windows\source\repos\Homework2\Debug\Homework2.exe(9272 프로세스)이 (가) 0 코드로 인해 종료되었습니다. 이 창을 닫으려면 아무 키나 누르세요.</p>	Name	Arrive Time	Execution Time	Rest Time	Process5	6	31	0	<p>currentTime: 64msec runningTime: 1msec (14th)</p> <p>Process5 arrivalTime: 6msec executionTime: 31msec remainingTime: 0msec</p>												
Name	Arrive Time	Execution Time	Rest Time																			
Process5	6	31	0																			

스케줄링의 한 TASK의 최대 QUANTUM이 5ms인데, Process4가 3ms만으로도 충분하고, Process5도 마지막 프로세스 실행시 10ms으로 총 64ms동안 실행하였지만 총 13번의 프로세스 스위칭(프로세스 실행은 14번)을 진행하여 모든 프로세스를 마친 것을 볼 수 있다. 이 역시 예상하던 총 수행시간 currentTime은 (프로세스의 총 executionTime) - (초기 arrivalTime) = (10 + 20 + 3 + 31) - 0 = 64msec로 예상할 수 있었고, 예상대로 결과가 나왔다.

여러개의 Process 실행2 (QUANTUM=10)

현재 시각 (ms)	프린트 캡쳐 화면	현재 실행 중인 Process 정보																
9	<p>현재시각 : 9 msec</p> <p>Process List Information</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Arrive Time</th> <th>Execution Time</th> <th>Rest Time</th> </tr> </thead> <tbody> <tr> <td>mainT</td> <td>0</td> <td>10</td> <td>11</td> </tr> <tr> <td>Process2</td> <td>6</td> <td>11</td> <td>11</td> </tr> <tr> <td>Process3</td> <td>7</td> <td>3</td> <td>3</td> </tr> </tbody> </table> <p>현재 Task mainT가 수행중입니다.</p>	Name	Arrive Time	Execution Time	Rest Time	mainT	0	10	11	Process2	6	11	11	Process3	7	3	3	<p>currentTime: 9msec runningTime: 9msec (1st)</p> <p>mainT arrivalTime: 0msec executionTime: 10msec remainingTime: 1msec</p> <p>Process2 (6, 11, 11) Process3 (7, 3, 3)</p>
Name	Arrive Time	Execution Time	Rest Time															
mainT	0	10	11															
Process2	6	11	11															
Process3	7	3	3															



QUANTUM을 10msec로 변경하였고, 5msec에 비해 각 task에 할당된 시간이 다소 길어져 한 프로세스에만 오래 묶여있는 것을 볼 수 있다. 총 24ms동안 실행하였지만 총 3번의 프로세스 스위칭(프로세스 실행은 4번)을 진행하며 모든 프로세스를 마친 것을 볼 수 있으며, 이 역시 예상하던 총 수행시간 currentTIme은 (프로세스의 총 executionTime) - (초기 arrivalTime) = $(10 + 11 + 3) - 0 = 24\text{msec}$ 로 예상할 수 있었고, 예상대로 결과가 나왔다.

Round Robin Scheduling 같은 시분할 스케줄링이 있기에 여러개의 거대한 Task Process가 있을 때에, 각 하나하나만을 처리하는 데에 CPU가 묶여있는게 아니라, 사람이 인지할 수 없을 정도로 잘게잘게 시분할로 나눠 실행하여 동시에 실행하는 듯한 느낌이 들도록 OS가 스케줄링을 하며 멀티 캐스팅을 구현했다는 것을 알게 되었다. 위의 모든 시뮬레이션 과정들을 통해, QUANTUM time을 길게 잡을 수록, 하나에 Task Process에 오래 묶여있게 되지만 컨텍스트 스위칭이 덜 일어나게되어 전체 실행시간에는 긍정적인 영향을 미칠 수 있지만, CPU가 특정 작업에 묶여있음을 사람이 인지할 수 있으면 사용자 경험적으로 불편을 느끼기에 이 트레이드 오프를 적절히 잘 설정해야된다는 것도 알 수 있었다.

참고 자료

- 자료구조및알고리즘이해, 과제2 PDF
- 자료구조및알고리즘이해, 강의노트 4장 List
- source code image formatter, <https://carbon.now.sh/>

전체 소스코드

```
1 #define _CRT_SECURE_NO_WARNINGS
2
3 #include <stdio.h>
4 #include <conio.h>
5 #include <malloc.h>
6 #include <string.h>
7 #include <Windows.h>
8
9
10#define MAX_LENGTH 20
11#define BAR
12"=====
13#define QUANTUM 5
14#define SLEEP_TIME 1000
15//시간을 나타내는 변수
16int now_time = 0;
17int num = 0;
18
19//자료구조 선언
20typedef struct ProcessElement {
21    char *name;
22
23    int executionTime;
24    int remainingTime;
25    int arrivalTime;
26}ProcessElement;
27
28
29typedef struct ProcessNode {
30
31    ProcessElement element;
32
33    struct ProcessNode *llink;
34    struct ProcessNode *rlink;
35
36}ProcessNode;
37
38typedef struct ProcessList {
39    struct ProcessNode *front;
40}ProcessList;
41
42//함수 선언
43int isEmptyList(ProcessList *processList);
44void initList(ProcessList *processList);
45void printProcessList(ProcessList *processList);
46void insertProcess(ProcessList *processList, char *name, int arrivalTime, int executionTime);
47void removeProcess(ProcessList *processList);
48void runningTask(ProcessList *processList, int *runningTime);
49
50int main(void)
51{
52    char name[MAX_LENGTH + 1];
53
54    int execution_time;
55    int runningTime = 0;
56
57    ProcessList *processList = (ProcessList*)malloc(sizeof(ProcessList));
58    initList(processList);
59
60    system("cls");
61    insertProcess(processList, "mainT", 0, 10);
62
63    printf("ss\n", BAR);
64    printf("현재시각 : %d msec\n\n", now_time);
65    printf("Process List Information\n\n");
66
67    printProcessList(processList);
68    printf("ss\n\n\n", BAR);
69
70    Sleep(SLEEP_TIME);
71
72    while (num != 1)
73    {
74
75        if (_kbhit())
76        {
77            if (_getch() == 's')
78            {
79                printf("새로운 Process에 대한 정보를 입력해주세요(Name(영문 최대 20자 한글 최대 10자), Execution Time 순서로)\n");
80                fflush(stdin);
81                scanf("%s", name);
82
83                fflush(stdin);
84                scanf("%d", &execution_time);
85
86                insertProcess(processList, name, now_time, execution_time);
87            }
88        }
89
90
91        now_time++;
92        runningTask(processList, &runningTime);
93    }
94
95    return 0;
96 }
```

```

97
98
99
100 int isEmptyList(ProcessList *processList)
101 {
102     return (processList->front == NULL);
103 }
104
105
106 void initList(ProcessList *processList)
107 {
108     processList->front = NULL;
109 }
110
111 void printProcessList(ProcessList *processList)
112 {
113     ProcessNode *temp = processList->front;
114
115     printf("%-21s%20s%20s%20s\n", "Name", "Arrive Time", "Execution Time", "Rest Time");
116
117     if (temp == NULL)
118         printf("          (List is empty)      \n");
119     printf("%-21s%20d%20d%20d\n", temp->element.name, temp->element.arrivalTime, temp->element.executionTime, temp->element.remainingTime);
120     temp = temp->rlink;
121     while (temp != processList->front)
122     {
123         printf("%-21s%20d%20d%20d\n", temp->element.name, temp->element.arrivalTime, temp->element.executionTime, temp->element.remainingTime);
124         temp = temp->rlink;
125     }
126 }
127
128 void insertProcess(ProcessList *processList, char *name, int arrivalTime, int executionTime)
129 {
130     ProcessNode *newProcess = (ProcessNode *)malloc(sizeof(ProcessNode));
131     newProcess->element.name = (char *)malloc(sizeof(char) * (strlen(name) + 1));
132     strcpy(newProcess->element.name, name);
133     newProcess->element.arrivalTime = arrivalTime;
134     newProcess->element.executionTime = executionTime;
135     newProcess->element.remainingTime = executionTime;
136
137     ProcessNode *frontNode = processList->front;
138
139     if (isEmptyList(processList)) {
140         processList->front = newProcess;
141         newProcess->llink = newProcess;
142         newProcess->rlink = newProcess;
143     }
144     else {
145         ProcessNode *lastNode = frontNode->llink;
146         newProcess->llink = lastNode;
147         newProcess->rlink = frontNode;
148         lastNode->rlink = newProcess;
149         frontNode->llink = newProcess;
150     }
151 }
152
153 void removeProcess(ProcessList *processList)
154 {
155     ProcessNode *frontNode = processList->front;
156
157     if (frontNode == frontNode->llink) {
158         initList(processList);
159     }
160     else {
161         ProcessNode *lastNode = frontNode->llink;
162         ProcessNode *secondNode = frontNode->rlink;
163         lastNode->rlink = secondNode;
164         secondNode->llink = lastNode;
165
166         processList->front = secondNode;
167     }
168
169     free(frontNode->element.name);
170     frontNode->element.name = NULL;
171     free(frontNode);
172     frontNode = NULL;
173 }
174
175 void runningTask(ProcessList *processList, int *runningTime)
176 {
177     int nowTask_ResidualTime;
178
179     nowTask_ResidualTime = --(processList->front->element.remainingTime);
180
181     system("cls");
182
183     printf("%s\n", BAR);
184     printf("현재시간 : %d msec\n\n", now_time);
185     printf("Process List Information\n\n");
186
187     printProcessList(processList);
188     printf("%s\n\n\n", BAR);
189
190     printf("현재 Task %s가 수행중입니다\n\n", processList->front->element.name);
191     Sleep(SLEEP_TIME);
192     (*runningTime)++;

```

```

193     if (nowTask_ResidualTime == 0)
194     {
195         if (processList->front == processList->front->llink)           //남은 Process가 1개일 때
196         {
197             system("cls");
198
199             printf("%s\n", BAR);
200             printf("현재시각 : %d msec\n\n", now_time);
201             printf("Process List Information\n\n");
202
203             printProcessList(processList);
204             printf("%s\n\n\n", BAR);
205
206             printf("현재 수행되던 Task %s가 종료되었습니다\n", processList->front->element.name);
207             printf("모든 Task가 종료되었습니다\n");
208
209             num = 1;
210
211             removeProcess(processList);
212         }
213     else
214     {
215         system("cls");
216
217         printf("%s\n", BAR);
218         printf("현재시각 : %d msec\n\n", now_time);
219         printf("Process List Information\n\n");
220
221         printProcessList(processList);
222         printf("%s\n\n\n", BAR);
223
224         printf("현재 수행되던 Task %s가 종료되었습니다\n", processList->front->element.name);
225         printf("다음 Task로 넘어갑니다\n");
226         Sleep(SLEEP_TIME);
227
228         *runningTime = 0;
229         removeProcess(processList);
230     }
231 }
232
233 else if (*runningTime >= QUANTUM) {                                //Time Quantum이 지났을 때
234     system("cls");
235
236     printf("%s\n", BAR);
237     printf("현재시각 : %d msec\n\n", now_time);
238     printf("Process List Information\n\n");
239
240     printProcessList(processList);
241     printf("%s\n\n\n", BAR);
242
243     printf("%d msec가 흘러 다음 Task로 넘어갑니다.\n", QUANTUM);
244     Sleep(SLEEP_TIME);
245
246     *runningTime = 0;
247     processList->front = processList->front->rlink;
248 }
249 }
```

carbon
carbon.now.sh