**프로젝트 3 보고서**

**2014-17952 정재철**

1. 구조체 변경

<eos.h>

**eos\_tcb\_t**에 다음 변수를 추가했다.

int32u\_t period

eos\_alarm\_t alarm

int32u\_t timeout

* timeout에는 항상 (sleep 호출할 때의 tick) + period가 저장되게 된다. 초기화 해주지 않아도 된다.

1. 변경한 함수

<task.c>

**eos\_create\_task**에 다음 항목을 추가했다.

task->period = 0

task->alarm.alarm\_queue\_node.ptr\_data = &(task->alarm)

* 태스크에 포함된 알람에 포함된 노드의 데이터가 알람의 주소를 가리키게 하였다.

**eos\_schedule**을 다음과 같이 수정했다.

* 만약 이 태스크가 sleep을 호출했으면 WAITING 상태로 진입한 후 스케줄 함수를 호출한 것이다. 따라서 RUNNING 상태가 아닌 경우에는 레디 큐에 노드를 넣지 않는다. 레디 큐에 노드를 추가하였을 때 항상 \_os\_set\_ready를 호출하여 큐의 상태를 업데이트한다. 레디 큐에 노드를 제거하였을 때 만약 큐가 비워지면 \_os\_unset\_ready를 호출하여 큐의 상태를 업데이트한다.

**eos\_set\_period**를 다음과 같이 수정했다.

* 인자로 받은 태스크의 주기를 인자로 받은 주기로 변경한다.

**eos\_sleep**을 다음과 같이 수정했다.

* 만약 주기가 0인데 sleep이 호출되었다면 곧바로 스케줄 함수를 호출하고 리턴한다. 그렇지 않으면 현재 수행중인 태스크는 WAITING 상태로 진입한다. 또한 현재 tick에 태스크의 주기를 더한 값을 timeout으로 설정한다. eos\_set\_alarm 함수를 호출하여 알람 큐에 현재 태스크의 알람을 추가한다. 이 때 콜백 함수로 \_os\_wakeup\_sleeping\_task를 사용한다. 마지막으로 스케줄 함수를 호출하고 리턴한다.

**\_os\_wakeup\_sleeping\_task**를 다음과 같이 수정했다.

* 이 함수는 콜백 함수로, 인자인 arg는 깨워줄 태스크의 tcb 주소이다. 이 태스크를 멀티 레벨 레디 큐에 넣고 \_os\_set\_ready를 호출하여 큐의 상태를 업데이트한다.

<timer.c>

**eos\_set\_alarm**을 다음과 같이 수정했다.

alarm->timeout = timeout

alarm->alarm\_queue\_node.priority = timeout

alarm->handler = entry

alarm->arg = arg

\_os\_add\_node\_priority(&(counter->alarm\_queue), &(alarm->alarm\_queue\_node))

* 이 알람의 timeout을 인자로 주어진 timeout으로 설정한다. 알람 큐를 priority queue로 이용하기 위해\_os\_node\_t에 존재하는 priority 변수를 timeout으로 설정한다. 이 때 priority queue란 push\_back/pop\_front 할 때마다 timeout이 제일 작은 노드가 항상 top에 존재하는 자료구조를 뜻한다. 또한 핸들러로는 주어진 entry(wakeup 함수), arg로는 깨워줄 태스크의 tcb 주소를 설정한다. <common.c>에 구현된 \_os\_add\_node\_priority 함수를 이용하여 알람을 푸쉬하면 알람 큐가 priority queue로 기능한다. 이 함수는 priority가 낮은 순서대로 큐를 유지하며 노드를 추가한다.

**eos\_trigger\_counter**를 다음과 같이 수정했다.

counter->tick ++;

PRINT("tick %d\n", counter->tick);

while(counter->alarm\_queue != NULL){

eos\_alarm\_t \* alarm = (eos\_alarm\_t \*) (counter->alarm\_queue->ptr\_data);

if (alarm->timeout != counter->tick) break;

(alarm->handler)(alarm->arg);

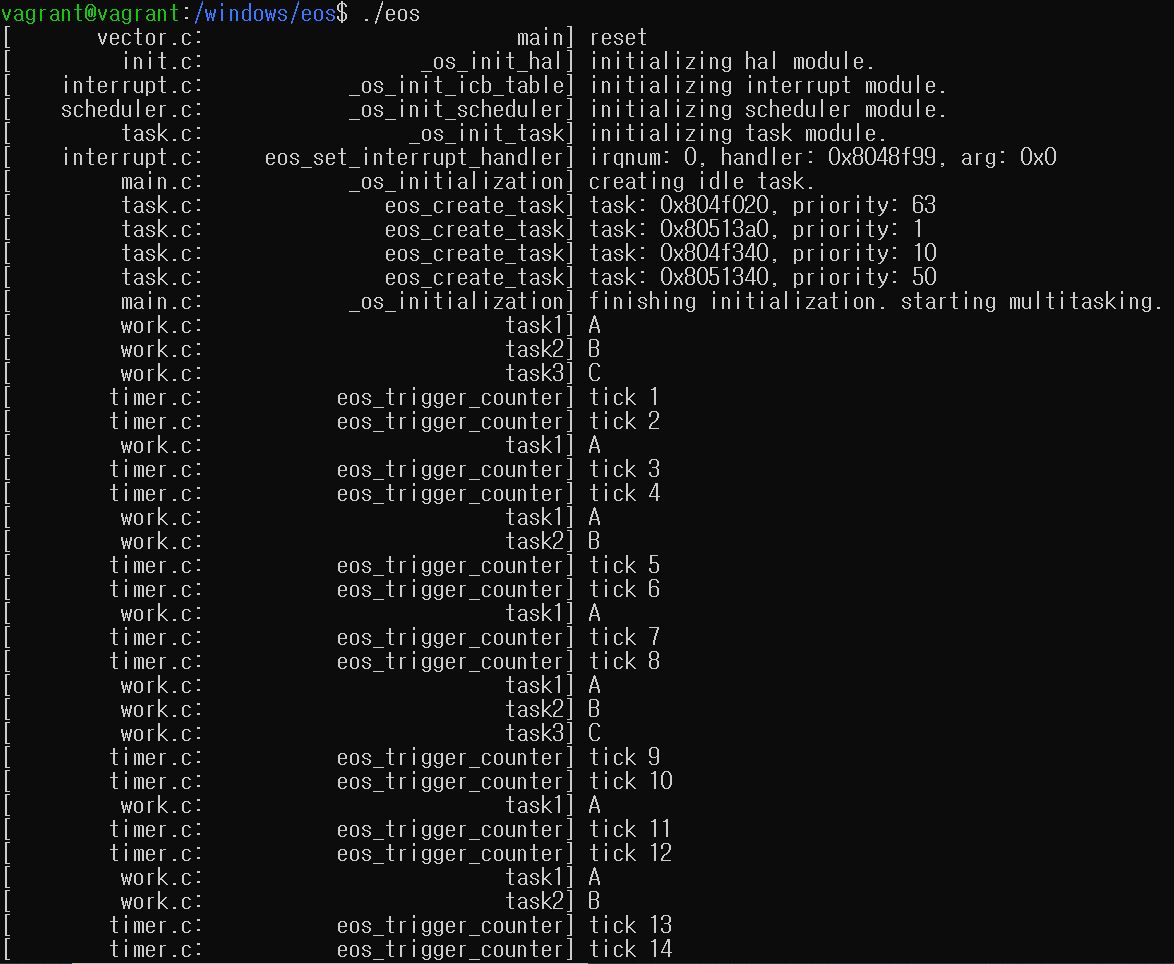
\_os\_remove\_node(&(counter->alarm\_queue), counter->alarm\_queue);

}

eos\_schedule();

* 시스템 카운터의 tick을 1씩 증가시킨다. 만약 알람 큐의 탑의 timeout이 현재 tick과 같은 경우 알람을 울리게 된다. 콜백 함수인 알람 핸들러를 이용해 태스크를 깨운 후 \_os\_remove\_node 함수를 이용하여 알람 큐의 탑을 제거한다. 똑같은 timeout을 가진 알람이 여러 개 있을 수 있으므로 큐의 top에 있는 알람의 timeout이 현재 tick과 달라질 때 까지 반복한다.

1. 테스트 프로그램 수행



다음 코드를 실행 시킨 결과이다. (task1은 A 출력 후 sleep, task2는 B 출력 후 sleep, task3은 C 출력 후 sleep 무한 반복)

void eos\_user\_main(){

eos\_create\_task(&tcb1, stack1, STACK\_SIZE, task1, NULL, 1);

eos\_set\_period(&tcb1, 2);

eos\_create\_task(&tcb2, stack2, STACK\_SIZE, task2, NULL, 10);

eos\_set\_period(&tcb2, 4);

eos\_create\_task(&tcb3, stack3, STACK\_SIZE, task3, NULL, 50);

eos\_set\_period(&tcb3, 8);

}