OS Project 4

세마포와 메시지 큐

Real-Time Operating Systems Laboratory,
Seoul National University, Korea

Contents

1. 과제 목적

- 과제 목적
  - 태스크 동기화의 필요성과 메커니즘 이해
  - 세마포와 메시지 큐 구현

2. 과제 설명 (1)

- 구현해야 할 API 목록
  - `core/eos.h`
    1. eos_semaphore_t - 세마포 구조체
    2. eos_mqueue_t - 메시지 큐 구조체
  - `core/sync.c`
    3. eos_init_semaphore() - 세마포 초기화
    4. eos_acquire_semaphore() - 세마포 획득 시도
    5. eos_release_semaphore() - 세마포 반환
  - `core/comm.c`
    6. eos_init_mqueue() - 메시지 큐 초기화
    7. eos_send_message() - 메시지 큐로 메시지 전송
    8. eos_receive_message() - 메시지 큐에서 메시지 읽음

2. 과제 설명 (2)

- 세마포의 목적
  - 임계 영역(critical section) 보호
    - 임계 영역이 동시에 하나 혹은 제한된 개수의 태스크에 의해서만 수행되도록 함

- 메커니즘
  - 세마포를 얻는데 성공하면 임계 영역 수행
  - 세마포를 얻는데 실패하면
    - 포기하고 다른 코드를 수행
    - 혹은 세마포를 얻을 때까지 기다림
  - 싱글코어 시스템을 가정하므로, 임계 영역 보호를 위해 eos_disable_interrupt(), eos_restore_interrupt() 사용

2. 과제 설명 (3)

1. eos_semaphore_t – 세마포 구조체

count	공유 자원의 instance 개수
wait_queue	태스크 wait 큐
queue_type	0: FIFO, 1: priority-based

2. 과제 설명 (4)

2. eos_init_semaphore() – 세마포 구조체 초기화
   - 세마포 구조체의 각 필드를 입력된 인자의 값으로 초기화

형	void eos_init_semaphore(eos_semaphore_t *sem, int32u_t initial_count, int8u_t queue_type)
입력	sem: 초기화할 세마포 구조체 initial_count: 공유 자원에 동시에 접근 가능한 태스크 개수 queue_type: 대기 중인 태스크를 깨울 때의 방식 (0: FIFO, 1: priority-based)
출력	없음

2. 과제 설명 (5)

3. eos_acquire_semaphore() – 세마포 획득 시도

형	int eos_acquire_semaphore(eos_semaphore_t *sem, int32s_t timeout)
입력	sem: 획득할 세마포 구조체 timeout: 기다릴 시간 지정 -1: 바로 리턴 0: 무한히 대기 ≥ 1: 명시된 시간만큼 대기
출력	0: 획득 실패, 1: 획득 성공

2. 과제 설명 (6)

3. eos_acquire_semaphore() – 세마포 획득 시도
   1. count > 0 이면, count를 1 감소시키고 리턴(성공)
   2. count ≤ 0 이면 timeout 값에 따라 다음과 같이 구현
      - timeout이 -1인 경우 바로 리턴(실패)
      - timeout이 0인 경우 다른 태스크가 반환할 때까지 wait 큐에서 대기
        - 깨어난 뒤에는 count를 다시 확인
        - count > 0 이면 count를 1 감소 후 리턴하고(성공)
        - count ≤ 0 인 경우 다시 wait 큐에서 대기
      - timeout이 1 이상인 경우 timeout 값의 tick 동안만 wait 큐에서 대기
        - eos_set_alarm 이용

2. 과제 설명 (7)

4. eos_release_semaphore() – 세마포 반환

형	void eos_release_semaphore(eos_semaphore_t *sem)
입력	sem: 반환할 세마포 구조체
출력	없음

2. 과제 설명 (8)

4. eos_release_semaphore() – 세마포 반환
   1. count를 1 증가시킴
   2. wait 큐에 대기 중인 태스크가 있는 경우 하나의 태스크를 선택해 깨움

2. 과제 설명 (9)

- 메시지 큐의 목적
  - 태스크 간의 데이터 교환
  - 데이터를 보내는 태스크와 받는 태스크는 서로를 식별할 필요 없이 메시지 큐와 데이터를 주고 받을 수 있음

- 메커니즘
  - 메시지를 보낼 때 메시지 큐가 full이면 다른 태스크가 메시지를 읽어갈 때까지 wait 큐에서 대기
  - 메시지를 받을 때 메시지 큐가 empty면 다른 태스크가 메시지를 보낼 때까지 wait 큐에서 대기

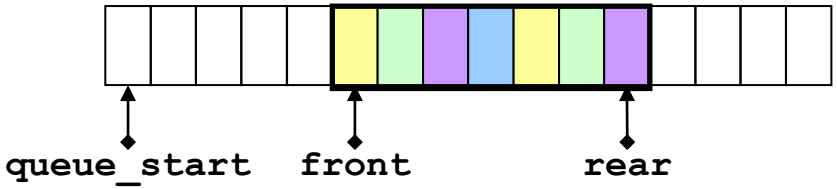
2. 과제 설명 (10)

5. eos_mqueue_t – 메시지 큐 구조체

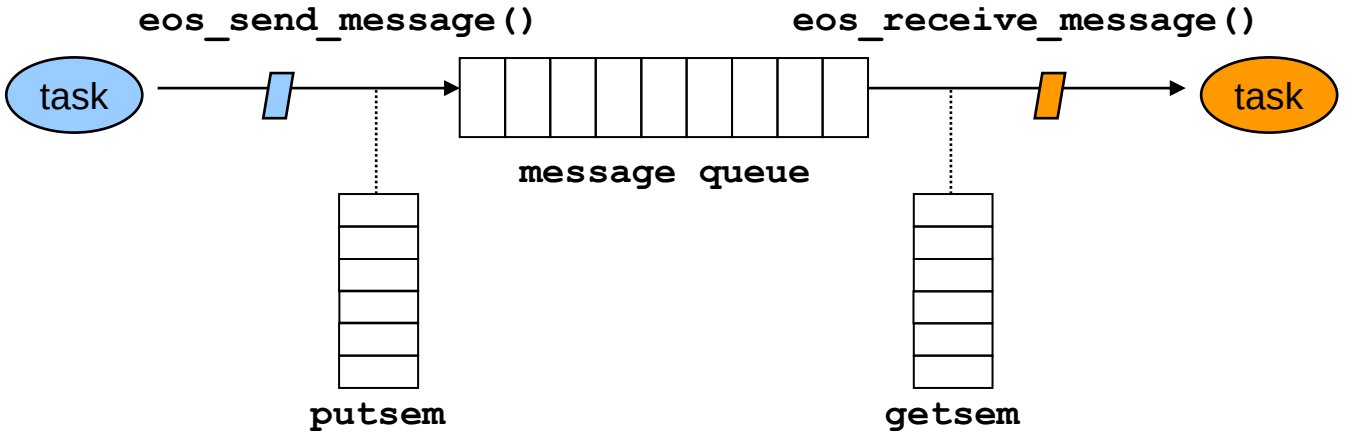
queue_size	메시지 큐에 들어갈 수 있는 최대 메시지 개수
msg_size	한 메시지가 전달할 수 있는 데이터 크기(bytes)
queue_start	메시지 큐에 할당된 메모리의 첫 주소
front	메시지 큐에서 실제로 메시지가 저장된 첫 위치
rear	메시지 큐에서 실제로 메시지가 저장된 마지막 위치
queue_type	0: FIFO, 1: priority-based
putsem	메시지 큐에 메시지를 보낼 수 있는 권한을 관리하는 세마포
getsem	메시지 큐에서 메시지를 받을 수 있는 권한을 관리하는 세마포

2. 과제 설명 (11)

- Memory structure



- Kernel structure



2. 과제 설명 (12)

6. eos_init_mqueue() – 메시지 큐 구조체 초기화
   - 메시지 큐 구조체의 각 필드를 입력된 인자의 값으로 초기화
   - putsem = queue_size, getsem = 0 으로 초기화

형	void eos_init_mqueue(eos_mqueue_t *mq, void *queue_start, int16u_t queue_size, int8u_t msg_size, int8u_t queue_type)
입력	**mq:** 초기화할 메시지 큐 구조체 **queue_start:** 메시지 큐에 할당된 메모리의 첫 주소 **queue_size:** 메시지 큐에 들어갈 수 있는 최대 메시지 개수 **msg_size:** 한 메시지가 전달할 수 있는 데이터 크기 (bytes) **queue_type:** 대기 중인 태스크를 깨울 때의 방식 (0: FIFO, 1: priority-based)
출력	없음

2. 과제 설명 (13)

7. eos_send_message() – 메시지 큐에 메시지를 보냄

형	int8u_t eos_send_message(eos_mqueue_t *mq, void *message, int32s_t timeout)
입력	mq: 메시지를 보낼 메시지 큐 message: 메시지가 위치한 주소 timeout: 메시지 큐가 full일 때 기다릴 시간 -1: 바로 리턴 0: 다른 태스크가 메시지 큐에서 메시지를 읽어갈 때까지 대기 ≥ 1: 명시된 시간만큼 대기
출력	없음

2. 과제 설명 (14)

7. eos_send_message() – 메시지 큐에 메시지를 보냄
   1. putsem 획득 시도
   2. 획득 실패 시 timeout 값에 따라 바로 리턴하거나 wait 큐에서 대기
   3. 획득 성공 시 메시지 큐의 마지막 위치(rear)에 메시지 copy
   4. rear 1 증가
   5. getsem 반환

2. 과제 설명 (15)

8. eos_receive_message() – 메시지 큐에서 메시지를 읽음

형	int8u_t eos_receive_message(eos_mqueue_t *mq, void *message, int32s_t timeout)
입력	mq: 메시지를 읽어올 메시지 큐 message: 메시지를 저장할 주소 timeout: 메시지 큐가 empty일 때 기다릴 시간 -1: 바로 리턴 0: 다른 태스크가 메시지 큐에 메시지를 보낼 때까지 대기 ≥ 1: 명시된 시간만큼 대기
출력	없음

2. 과제 설명 (16)

8. eos_receive_message() – 메시지 큐에서 메시지를 읽음
   1. getsem 획득 시도
   2. 획득 실패 시 timeout 값에 따라 바로 리턴하거나 wait 큐에서 대기
   3. 획득 성공 시 메시지 큐의 첫 위치(front)에서 메시지 copy
   4. front 1 증가
   5. putsem 반환

3. 제출물 (1)

- 제출물
  - 수정한 EOS 코드
  - 보고서
    - 정의한 구조체에 대한 설명
    - 구현한 함수들에 대한 설명
    - 테스트 프로그램 수행 결과

- 주의사항
  - 주어진 함수 prototype을 변경하지 말 것
  - 가능한 새로운 함수나 글로벌 변수 추가를 하지 말 것
  - 보고서 분량은 5페이지 이하

3. 제출물 (2)

❖ 테스트 프로그램

```
static void sender_task(void *arg) {
    int8u_t *data = "xy";

    while (1) {
        PRINT("send message to mq1\n");
        eos_send_message(&mq1, data, 0) ;
        PRINT("send message to mq2\n");
        eos_send_message(&mq2, data, 0) ;
        eos_sleep(0);
    }
}

static void receiver_task1(void *arg) {
    int8u_t data[2];

    while (1) {
        PRINT("receive message from mq1\n");
        eos_receive_message(&mq1, data, 0);
        PRINT("received message: %s\n", data);
        eos_sleep(0);
    }
}
```

```
static void receiver_task2(void *arg) {
    int8u_t data[2];

    while (1) {
        PRINT("receive message from mq2\n");
        eos_receive_message(&mq2, data, 0);
        PRINT("received message: %s\n", data);
        eos_sleep(0);
    }
}
```

3. 제출물 (3)

```
#include <core/eos.h>

static eos_tcb_t tcb1;
static eos_tcb_t tcb2;
static eos_tcb_t tcb3;
static int8u_t stack1[8096];
static int8u_t stack2[8096];
static int8u_t stack3[8096];
static int8u_t queue1[10];
static int8u_t queue2[10];
eos_mqueue_t mq1;
eos_mqueue_t mq2;

void eos_user_main() {
    eos_create_task(&tcb1, (addr_t)stack1, 8096, sender_task, NULL, 50);
    eos_create_task(&tcb2, (addr_t)stack2, 8096, receiver_task1, NULL, 10);
    eos_create_task(&tcb3, (addr_t)stack3, 8096, receiver_task2, NULL, 20);
    eos_set_period(&tcb1, 2);
    eos_set_period(&tcb2, 4);
    eos_set_period(&tcb3, 6);

    eos_init_mqueue(&mq1, queue1, 5, 2, 1);
    eos_init_mqueue(&mq2, queue2, 5, 2, 1);
}
```

3. 제출물 (4)

- 제출 기한
  - 6/16(화) 자정까지
- 제출 방법
  - ETL에 업로드
    - 보고서는 워드 파일 또는 pdf
    - 소스 코드는 make clean을 하여 정리 후 압축
      - hal/current를 삭제 후 압축하는 것을 권장
      - 압축 에러의 원인이 되는 경우가 있기 때문
      - symbolic link일 뿐이기에 삭제해도 make all을 할 때 다시 생성

Thank You!!!