

2015 년 2 학기 운영체제 과제 #3

2015/10/16 (금)

1. 과제 내용

세마포어를 스레드를 대상으로 구현하고 제시된 동기화 문제를 해결하시오.

2. 요구 사항

- A. C 언어와 기본 C 라이브러리(glibc)와 pthread API만 이용하고, 소스 파일 `sem.c` 한 개에 모든 함수를 구현합니다.
- B. 세마포어 함수를 pthread 라이브러리를 이용해 구현합니다.
 - 구현해야 할 API 목록과 기능은 다음과 같습니다.
 - `struct test_semaphore {}` : 세마포어 객체 자료 구조
 - `tsem_t *tsem_new(int value)` : 세마포어 객체 생성
`value` 값으로 초기화된 세마포어 객체를 만들어서 반환합니다. 메모리 할당은 실패하지 않는다고 가정합니다.
 - `void tsem_free(tsem_t *sem)` : 세마포어 객체 해제
`sem`이 가리키는 세마포어 객체를 해제합니다.
 - `void tsem_wait(tsem_t *sem)` : 세마포어 연산 P 실행
세마포어 값을 1 감소하고 0 보다 작을 경우 해당 스레드 실행을 지연합니다.
 - `void tsem_signal(tsem_t *sem)` : 세마포어 연산 V 실행
세마포어 값을 1 증가하고 0 이하 일 경우 대기중인 스레드 중 하나의 실행을 재개합니다.
 - `int tsem_try_wait(tsem_t *sem)` : 세마포어 연산 P 실행 또는 가능 여부 확인
세마포어 값이 1 이상일 경우 1 을 감소하고 0 을 돌려주지만, 세마포어 값이 0 이하일 경우 해당 스레드를 지연하지 않고 1 을 돌려줍니다.
 - 구현한 함수와 자료구조는 샘플로 제공되는 `sem.h` 헤더파일의 함수 선언(declaration)과 정확히 일치해야 합니다.

- 샘플로 제공되는 `test.c` 테스트 파일이 정상적으로 컴파일 되어야 하고, 실행하면 다음과 같이 출력하고 종료해야 합니다.

```
$ ./test
thread[0] entered...
thread[1] entered...
thread[3] entered...
thread[2] entered...
thread[4] entered...
thread[5] entered...
thread[6] entered...
thread[7] entered...
thread[8] entered...
thread[9] entered...
thread[0] leaved...
thread[1] leaved...
thread[2] leaved...
thread[3] leaved...
thread[5] leaved...
thread[6] leaved...
thread[4] leaved...
thread[7] leaved...
thread[8] leaved...
thread[9] leaved...
```

- 'entered...' / 'leaved...' 메시지를 출력하는 스레드 번호는 순서가 섞일 수 있지만, 한꺼번에 들어가고(entered) / 한꺼번에 나와야(leaved) 합니다.

C. 구현한 세마포어 라이브러리를 이용해 Dining Philosophers 문제를 해결합니다.

- 샘플로 제공되는 소스 코드(`dining.c`)는 교착 상태(deadlock)가 발생합니다.
- 이 소스 코드를 교착 상태(deadlock)가 발생하지 않도록 수정해서 프로그램이 멈추지 않고 계속 실행되도록 합니다.

```
$ ./dining
...      EAT      ...      ...      ...
...      EAT      ...      EAT      ...
EAT      ...      ...      EAT      ...
EAT      ...      EAT      ...      ...
EAT      ...      EAT      ...      ...
...      ...      EAT      ...      EAT
...      EAT      ...      ...      EAT
...      EAT      ...      ...      EAT
```

```

...      EAT      ...      EAT      ...
...      EAT      ...      EAT      ...
...      EAT      ...      EAT      ...
EAT      ...      ...      EAT      ...
EAT      ...      ...      EAT      ...
EAT      ...      EAT      ...      ...
...      ...      EAT      ...      EAT
...      ...      EAT      ...      EAT
...      ...      EAT      ...      EAT
...      EAT      ...      ...      EAT
...      EAT      ...      ...      EAT
...      EAT      ...      EAT      ...
[...snip...]

```

- 반드시 직접 구현한 세마포어 API를 사용해야 합니다.
- 5 명의 철학자가 반드시 돌아가면서 공평하게 식사하도록 구현하지 않아도 됩니다.
- 당연하지만, 3 명 이상의 철학자가 동시에 식사를 하면 안되고 가까이 있는 철학자가 동시에 식사하면 안됩니다.

3. 구현 방법

- ✓ 세마포어 값(value)을 접근할 때 `pthread_mutex_*`() API와 같은 상호 배제 기법을 이용합니다.
- ✓ 스레드 목록을 직접 유지하는 방식으로 구현해도 되지만, 더 간단하고 정확하게 `pthread_cond_*`() API를 사용해서 구현해도 됩니다.
- ✓ Dining Philosophers 문제의 교착 상태 해결 방법은 교재에도 언급되어 있듯이 여러 가지 방법이 있습니다. 그 중 어느 방법을 사용해도 무방합니다.

4. 채점 기준

- ✓ 보고서 (20 점) : 아래 목차를 따라 작성합니다.
 1. 소개
 2. 관련 연구 (세마포어 및 동기화 기법 / pthread API 함수 조사 포함) (5)
 3. 문제 해결 방법 (5)
 4. 실행 결과 (3)
 5. 추가 및 개선할 부분과 결론 (5)
 6. 참고 자료 / 문헌 (2)

7. 소스 코드 (없을 경우 보고서 점수 0 점 처리)

✓ 구현 (30 점)

- 요구사항 A 준수 (5 점)
- 요구사항 B 구현 (10 점)
- 요구사항 C 구현 (10 점)
- 소스 코드 가독성(readability) 및 주석(comments) (5 점)

5. 제출 방법

✓ 마감시간 : **2015/11/06 오후 6 시 (2 주)**

✓ 감점 : 늦게 제출할 경우 1 일 10 점 감점

✓ 제출 방법

- 보고서 파일(PDF)과 소스 파일, Makefile, README 파일을 압축하고, 파일 이름은 '반_이름_과제번호.zip' 형식의 이름을 사용 (예: 가_홍길동_hw1.zip)
- vanilla@realtime.ssu.ac.kr 로 제출

✓ Dining Philosophers 문제에 대한 접근 방법 및 해결 방법은 보고서에 작성하고, 소스 코드는 샘플로 제공된 `dining.c` 소스 파일을 수정해서 `sem.c` 소스 파일과 함께 제출해야 합니다.

6. 기타

✓ 과제 해결 방법에 대한 논의나 도움은 서로간에 주고 받을 수 있겠지만 과제 자체는 전적으로 본인이 직접 스스로 해야 합니다. 만일 복사본이 발견되면 복사를 한 학생과 더불어 원본 제공자 또한 해당 과제 점수를 전체 0 점 처리합니다.

✓ 과제 관련 궁금한 점이나 문의 사항은 수업 게시판을 이용하시기 바랍니다.

<https://groups.google.com/forum/#!forum/ssu-smyang>