운영체제 기말시험

(54 점 만점) 2017.6.10

1. 다음 빈 칸을 기술하라. (12 개 x 2 점 = 24 점)

(1) 실시간 시스템의 성능은 인터럽트 지연시간과 (디스패치(dispatch)) 지연시간에 의해 성능이 좌우된다.

(2) (Earliest-deadline-first (EDF)) 실시간 스케줄링 기법은 마감시간에 따라 우선순위를 동적으로 부여한다. 마감시간이 빠를수록 우선순위는 높고, 늦을수록 낮아진다.

(3) 공유 자료를 병행 접근(concurrent access)하면 자료의 불일치(data inconsistency)를 초래하여, 자료의 일관성(data consistency)을 보장하는 프로세스의 (동기화(synchronization)) 메커니즘이 필요하다.

(4) 임계구역 문제 해결을 위한 세가지 조건으로 상호배제, 진행, (한정된 대기)가 있다.

(5) (모니터) 구조물은 효율적인 프로세스 동기화 수단을 제공하는 고급 언어수준의 동기화 구조물로, 그 안에 항상 하나의 프로세스만이 활성화되도록 보장한다.

(6) 한 시스템에서 동시에 성립되어 교착상태를 발생시키는 4 가지 필요조건으로는 상호배제, (점유하며 대기), 비선점, 순환대기 가 있다.

(7) 주소는 소스 코드에서는 심볼, 컴파일된 코드에서는 (재배치 가능한) 주소, 링커나 로더에서는 절대 주소로 표시된다.

(8) 페이징 적용 시 페이지 테이블의 크기가 커지는 문제점을 해결하는 페이지 테이블 구조로는 계층적 페이징, 해시형 페이징 테이블, (역 페이지 테이블) 등이 있다.

(9) 프로세스가 메모리에 올라와 있지 않는 페이지를 접근하는 경우 (페이지 부재(page fault)) 트랩이 발생한다.

(10) 오픈파일과 관련된 정보(메타데이터)로는 오픈파일 테이블, 파일포인터, (파일 오픈 계수), 파일의 디스크 위치, 접근 권한 등이 있다.

(11) (디렉터리)는 모든 파일들에 대한 정보를 갖는 노드들의 집합이다.

(12) 디스크 공간은 파일 시스템, (스왑 공간), 포맷되지 않은 (비가공) 디스크 공간 등으로 파티션 된다.

2. 아래와 같이 세마포를 정의할 때 바쁜 대기(busy waiting)이 없는 세마포의 다음 연산을 기술하라.

```
typedef struct {
    int   value;
    struct process *list;
} semaphore;
```

(1) wait() 연산

```
void wait(semaphore *S)
{
    S->value--;
    if (S->value < 0) {
        S->list 가 가리키는 리스트에 현재 프로세스를 삽입;
        block();   // 현재 프로세스 블록킹되어 대기상태 전환
    }
}
```

(2) signal() 연산

```
void signal(semaphore *S)
{
    S->value++;
    if (S->value <= 0) {
        S->list 가 가리키는 리스트에서 한 프로세스 P 를 제거;
        wakeup(P);  // 프로세스 P 를 대기상태에서 준비 완료 상태로 전환
    }
}
```

3. 은행원 알고리즘에서 현재 시스템 상태가 아래와 같다. (10 점)

	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	2	0	0	1	4	2	1	2	3	3	2	1
P1	3	1	2	1	5	2	5	2				
P2	2	1	0	3	2	3	1	6				
P3	1	3	1	2	1	4	2	4				
P4	1	4	3	2	3	6	6	5				

(1) Need 행렬을 기술하라

	Allocation				Max			
	A	B	C	D	A	B	C	D
P0	2	0	0	1	4	2	1	2
P1	3	1	2	1	5	2	5	2
P2	2	1	0	3	2	3	1	6
P3	1	3	1	2	1	4	2	4
P4	1	4	3	2	3	6	6	5

	Need(Max-Allocation)			
	A	B	C	D
P0	2	2	1	1
P1	2	1	3	1
P2	0	2	1	3
P3	0	1	1	2
P4	2	2	3	3

(2) 프로세스의 종료 가능한 순서를 보여서 시스템이 안전한 상태에 있다는 것을 설명하시오

P0 → Available이 P0의 Need를 만족함
P0이 프로세스 종료 → Available = Available + Allocation(P0)
(3, 3, 2, 1) + (2, 0, 0, 1) = **(5, 3, 2, 2)**
P1 → Available이 P1의 Need(C)보다 작음
P2 → Available이 P2의 Need(D)보다 작음
P3 → Available이 P3의 Need를 만족함
P3가 프로세스 종료 → Available = Available + Allocation(P3)
(5, 3, 2, 2) + (1, 3, 1, 2) = **(6, 6, 3, 4)**
P4 → Available이 P4의 Need를 만족함
P4가 프로세스 종료 → Available = Available + Allocation(P4)
(6, 6, 3, 4) + (1, 4, 3, 2) = **(7, 10, 6, 6)**
P1 → Available이 P1의 Need를 만족함
P1이 프로세스 종료 → Available = Available + Allocation(P1)
(7, 10, 6, 6) + (3, 1, 2, 1) = **(10, 11, 8, 7)**
P2 → Available이 P2의 Need를 만족함
P2이 프로세스 종료 → Available = Available + Allocation(P2)
(10, 11, 8, 7) + (2, 1, 0, 3) = **(12, 12, 8, 10)**

시스템은 안전하고 < P0, P3, P4, P1, P2 > 순서로 종료가 가능하다.

(3) 프로세스 P4 의 요청이 (0.0.2.0) 이라면 요청을 즉시 들어 줄 수 있는가?
(요청을 들어 줄 수 있다면 프로세스안 안전한 종료 순서를 기술하고, 요청을 들어 줄 수 없다면 요청을 들어 줄 수 없는 이유를 기술하라)

Request(P4) = (0, 0, 2, 0)이 Need(P4) = (2, 2, 3, 3)보다 작음
Request(P4) = (0, 0, 2, 0)이 Available (3, 3, 2, 1)보다 작음

Available = Available − Request(P4) → (3, 3, 0, 1)
Allocation(P4) = Allocation(P4) + Request(P4) → (1, 4, 5, 2)
Need(P4) = Need(P4) − Request(P4) → (2, 2, 1, 3)

	Allocation				Max				Need				Available				
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	할당 가능
P0	2	0	0	1	4	2	1	2	2	2	1	1	3	3	0	1	X
P1	3	1	2	1	5	2	5	2	2	1	3	1	3	3	0	1	X
P2	2	1	0	3	2	3	1	6	0	2	1	3	3	3	0	1	X
P3	1	3	1	2	1	4	2	4	0	1	1	2	3	3	0	1	X
P4	1	4	5	2	3	6	6	5	2	2	1	3	3	3	0	1	X

C 의 가용한 자원이 0 개이어서, 자원 C 에 대한 모든 프로세스의 Need 보다 커서 프로세스가 종료할 수 없게 되어서 안전한 상태가 될 수 없으므로 요청을 들어 줄 수 없다.

4. 3 개의 페이지 프레임의 메모리에서 아래와 같은 참조열인 경우 다음 교체 알고리즘을 적용하고 각 참조 시 마다 프레임의 변화 단계와 총 페이지 부재(page fault) 횟수를 기술하라. (10 점)

참조열: 7 1 4 2 0 1 7 2 3 0 7

(1)LRU 페이지 교체

7	(7, ,)
1	(7, 1,)
4	(7, 1, 4)
2	(2, 1, 4)
0	(2, 0, 4)
1	(2, 0, 1)
7	(7, 0, 1)
2	(7, 2, 1)
3	(7, 2, 3)
0	(0, 2, 3)
7	(0, 7, 3)

총 11 번의 페이지 부재 발생

(2) 최적 페이지 교체

7	(7, ,)
1	(7, 1,)
4	(7, 1, 4)
2	(7, 1, 2)
0	(7, 1, 0)
1	
7	
2	(7, 2, 0)
3	(7, 3, 0)
0	
7	

총 7번의 페이지 부재 발생