DD724: 운영체제

1. FIFO 스케쥴링 알고리즘이 대화형 사용자 (또는 프로그램)에 적합하지 않은 이유를 상세히 기술하시오.

2. X 시스템의 현재 Ready Queue 에 포함된 프로세스의 Arrival Time 과 CPU Burst 시간이 다음과 같다고 가정하자. 이 때, FCFS, Nonpreemptive SJF, 및 RR 스케쥴링 알고리즘을 사용했을 때 각 프로세스의 Waiting Time(WT)과 Turnaround Time(TRT)를 계산하시오. 단, Time Quantum 의 크기는 1 이며, Context Switch 에 소비되는 시간은 0 으로 가정한다. 모든 조건이 동일한 경우 먼저 도착한 프로세스를 먼저 처리하도록 한다.

Process ID	Arrival Time	CPU Burst Time
Process-1	**0**	**4**
Process-2	**2**	**5**
Process-3	**3**	**3**
Process-4	**8**	**4**

Scheduler	Time	Process-1	Process-1	Process-1	Process-1
FCFS	**WT**				
	TRT				
Nonpreemptive SJF	**WT**				
	TRT				
RR	**WT**				
	TRT				

3. 시스템 X는 FCFS 스케쥴링 알고리즘이 적용된 Multiprogramming 시스템이다. 현재 시스템 X의 Ready Queue 에는 프로세스 P1 과 P2 가 있으며 각 프로세스의 실행 내용 및 순서는 다음과 같다.

P1: Computation (6 초) → I/O (3 초) → Computation (5 초)

P2: Computation (5 초) → I/O (2 초) → Computation (3 초)

P1, P2 프로세스의 도착 시간은 둘 다 0 이며, P1 에게 CPU 가 먼저 할당된다고 가정했을 때 각 프로세스의 간트 차트를 그리고 Ready Queue 에서 보내는 시간을 계산하시오.

4. Multi-Threading 기법을 이용하여 Video Streaming 을 위한 YouTube 웹서버를 개발하고자 한다. 사용자가 특정 비디오 파일을 요청하면, 해당 요청을 담당하는 쓰레드는 디스크로부터 파일을 읽어들여 스트리밍을 수행한다. 이 때 디스크 I/O 를 위해 운영체제는 Blocking read 만을 위한 시스템 콜을 제공한다고 가정했을 때 User-Level Thread 와 Kernel-Level Thread 중 어떤 형태의 쓰레드를 이용하여 구현하는 것이 유리한지 상세히 기술하시오.

5. 프로세스간 context switch 에 소요되는 평균 시간이 s 이며, IO-bound process 가 IO Request 를 요청하기 전에 사용하는 평균 시간이 t (s << t)라고 가정하자. Time quantum 의 크기 q 를 다음과 같이 설정할 때 그 효과를 상세히 기술하시오.

1) q = s

2) q 가 매우 클 때

6. Multi-level feedback queue 스케쥴링 알고리즘은 IO-bound process 를 선호하여 입출력 장치의 활용도를 높인다고 볼 수 있는가? Yes 또는 No 로 답하고 그 원인을 상세히 기술하시오.

7. 다음과 같이 Multi-level feedback queue 로 구성된 스케쥴링 알고리즘이 있다고 가정하자.

Level	Time quantum	When dispatched
1	10 msec	
2	40 msec	Only when level 1 is empty
3	Unlimited	Only when levels 1 and 2 are empty

프로세스들은 최초로 Level 1 queue 에 진입한 후 자신에게 할당된 time quantum 내에 작업을 완료하지 못하면 그 다음 레벨의 queue 로 차례대로 진입하게 되며, 반대 방향으로는 (Level 3 → Level 2 → Level 1) 이동하지 못한다. 각각의 queue 내에서는 nonpreemptive 로 동작한다.

다음은 각 프로세스들의 CPU burst time 및 Level 1 queue 에서 실행 순서를 나타낸다.

Execution order	Processing name	CPU burst time
1	A	8
2	B	300
3	C	60
4	D	12
5	E	5
6	F	20

위 정보를 활용하여 "프로세스 실행 순서", "프로세스가 선택되는 queue", "프로세스 실행 시""을 나타내는 아래 표를 완성하시오.

Process	Queue	Run Time
A	1	8

8. 스케쥴링 알고리즘 X 가 있다. X 알고리즘은 시스템에 새롭게 진입한 프로세스들을 위한 queue (NewQ)과 기존의 프로세스들을 위한 queue (OldQ)로 구성되면 동작 과정은 다음과 같다.

① 새로운 프로세스가 시스템에 진입하게 되면 NewQ 의 끝에 삽입된다.

② NewQ 에서 2 msec 경과 후 OldQ 로 옮겨진다

③ 새로운 프로세스에게 CPU 를 할당하게 될 경우 X 알고리즘은 두 개의 queue 를 번갈아가면 각 queue 의 가장 먼저 저장된 프로세스를 선택하게 된다. (예를 들어, NewQ 에 가장 먼저 저장된 프로세스에게 CPU 를 할당하고, 그 다음 번에는 OldQ 에서 가장 먼저 저장된 프로세스에게 CPU 를 할당)

④ X 알고리즘은 nonpreemptive 로 동작

⑤ 프로세스들은 임의의 시간에 시스템에 진입할 수 있으며, 대부분의 프로세스들은 2msec 보다 훨씬 많은 시간을 필요로 한다고 가정한다. 단 프로세스들은 무한루프와 같은 비정상 동작은 하지 않는다고 가정한다.

1) X 알고리즘은 신규 프로세스에게 가장 높은 우선순위를 부여하는가? Yes 또는 No 로 대답하고 그 이유를 상세히 기술하시오.

2) X 알고리즘을 사용할 경우 starvation 문제가 발생하는가? Yes 또는 No 로 대답하고 그 이유를 상세히 기술하시오.

3) X 알고리즘은 공평한가? 즉, 모드 프로세스들이 평균적으로 유사한 waiting time 을 가지는가? Yes 또는 No 로 대답하고 그 이유를 상세히 기술하시오.

9. Lottery 스케쥴링 알고리즘은 각 프로세스에게 복권(Lottery Ticket)을 발급하며, 무작위로 복권을 선택하여 해당 복권을 보유한 프로세스에게 CPU 를 할당한다. 이 때 복권의 개수를 차등 지급하여 우선 순위가 높은 프로세스는 우선 순위가 낮은 프로세스보다 더 높은 당첨 가능성을 확보하도록 한다. Lottery 스케쥴링 알고리즘은 기존의 단순히 우선 순위를 비교하여 우선 순위가 높은 프로세스에게 CPU 를 할당하는 우선순위 스케쥴링 알고리즘에 비해 어떠한 장점을 가지고 있는지 기술하시오. 만약 특별한 장점이 없다면 "없음"으로 작성하시오.

10. CPU Scheduler 는 프로세스의 특성 (IO-bound 프로세스와 CPU-bound 프로세스)을 고려하여 CPU 를 차등적으로 할당하는 것이 중요하다. 그러한 이유를 설명하시오.

11. 모니터에 진입하려고 처음 시도하는 프로세스보다 대기 중인 프로세스에게 우선권을 줘야 하는가? 그렇지 않으면 그 반대로 우선권을 줘야 하는가? **어떤 프로세스에게 우선권을 줘야 하는지를 기술하고 그 이유를 상세히 설명하시오**.

12. 임의의 시스템 X 의 현재 상태는 다음과 같다고 가정하자. **현재 시스템이 deadlock 상태인지 기술하고 그 이유를 상세히 설명**하시오.

(1) 자원 유형은 2 가지가 존재한다: R1, R2
(2) 각 자원 유형별로 2 개의 instance 가 존재한다.
(3) 현재 시스템에는 총 4 개의 프로세스가 존재한다: P1, P2, P3, P4
(4) 현재 자원 할당 현황은 다음과 같다.
- R1 유형의 instance 는 각각 P2, P3 프로세스에게 할당
- R2 유형의 instance 는 각각 P1, P4 프로세스에게 할당

(5) 현재 자원 요청 현황은 다음과 같다.
- P1 프로세스는 R1 유형의 자원 요청
- P3 프로세스는 R2 유형의 자원 요청

13. Condition 변수를 Semaphore 를 이용하여 구현하시오. 단 semaphore 는 음수 값을 가질 수 없다. (Hint: 2 개의 Semaphore 를 가지고 구현할 것)

<table>
<tr><td colspan="2">class ConditionVariable {
// variable declaration</td></tr>
<tr><td>// Suspend the calling process
void wait() {

}</td><td>// wake-up every processes
void signal() {

}</td></tr>
</table>

14. 다음 Pseudo 코드는 다양한 버그를 포함하고 있다. **버그를 수정하시오**.

```
Semaphore mutex = 0; // semaphore initialization

// returns true if the item was added, false otherwise
Bool add_item_to_queue(queue_t queue, item_t item) {
    Wait(mutex);
    if (is_full(queue)) { // check if the queue is full
        return false;
    }
    append_to_queue(queue, item); // add an item to the queue
    Signal(mutex);
    return true;
    Signal(mutex);
}
```

15. 다음은 2 개의 프로세스(e.g., P0, P1)에 대한 Critical Section 문제를 해결하기 위한 솔루션을 나타낸다. **아래 코드는 정상적으로 동작하는가? 만약 그렇지 않다는 그 원인을 상세히 기술하시오.** (단, LOAD/STORE 명령은 Atomic 하게 동작한다고 가정한다).

enum { P0, P1 } turn = P0; // turn is a global shared variable	
P0's code	P1's code

// functions to enter the critical section
void critical_section_enter_P0() {
  while (turn != P0) continue;
}
// functions to leave the critical section
void critical_section_leave_P0() {
  turn = P1;
}
	// functions to enter the critical section
	void critical_section_enter_P1() {
	  while (turn != P1) continue;
	}
	// functions to leave the critical section
	void critical_section_leave_P1() {
	  turn = P0;
	}

16. 프로세스 P1, P2, P3 의 우선순위는 P1 = 1, P2 = 5, P3 = 10 (값이 클수록 우선 순위가 높음)로 설정되어 있으며, 각 프로세스가 수행하는 코드 및 주요 특성은 다음과 같다.

P1	**P2**	**P3**
<code sequence A> **Wait(X)** **<Critical Section>** **Signal(X)** **<code sequence B>**	**<code sequence A>** **Wait(X)** **Wait(Y)** **<Critical Section>** **Signal(Y)** **Signal(X)** **<code sequence B>**	**<code sequence A>** **Wait(Y)** **Wait(X)** **<Critical Section>** **Signal(X)** **Signal(Y)** **<code sequence B>**
- Semaphore X 와 Y 는 각각 "1"로 설정됨 - <code sequence A> 실행에 걸리는 시간: 2 time units - <code sequence B> 실행에 걸리는 시간: 3 time units - <Critical Section> 실행에 걸리는 시간: 4 time units - Wait(), Signal() 및 Context Switch 에 소비되는 시간: 0 time unit - P1, P2, P3 은 time 0, time 3, time 10 에서 실행을 시작함		

Preemptive Priority Scheduling 알고리즘을 사용한다고 가정했을 때 각 프로세스의 실행 Diagram 을 작성하시오. Diagram 에서 사용될 notation 은 다음과 같다.

- A: <code sequence A> 실행 중
- B: <code sequence B> 실행 중
- X: 프로세스가 Semaphore X 를 획득하여 현재 Critical Section 을 수행하고 있음

- Y: 프로세스가 Semaphore Y 를 획득하여 현재 Critical Section 을 수행하고 있음
- 2: 프로세스가 Semaphore X, Y 를 모두 획득하여 현재 Critical Section 을 수행하고 있음
- Blank: 프로세스가 현재 실행되고 있지 않음 (for any reason)

Time	00	01	02	03	04	05	06	07	08	09
P1										
P2										
P3										

Time	10	11	12	13	14	15	16	17	18	19
P1										
P2										
P3										

Time	20	21	22	23	24	25	26	27	28	29
P1										
P2										
P3										

17. 프로세스 A 와 B 는 세마포어 R 과 S 를 이용하여 상호 동기화를 수행한다. R = 0, S = 1 로 초기화되어 있으며, 프로세스 A 와 B 는 각각 다음과 같은 코드를 수행한다고 했을 때 **다음 물에 대해 True 또는 False 를 선택하고 그 이유를 상세히 기술하시오**.

Process A	**Process B**
Wait(R)	**Operation B.1**
Wait(S)	**Signal(R)**
Operation A.1	**Operation B.2**
Signal(S)	**Wait(S)**
Operation A.2	**Operation B.3**
Signal(R)	**Signal(S)**

a. **[5]** (True/False) Operation A.1 은 Operation B.1 이 끝나기 전에는 시작될 수 없다

b. **[5]** (True/False) Operation A.1 과 Operation B.2 는 동시에 수행될 수 없다.

c. **[5]** (True/False) Operation A.1 과 Operation B.3 는 동시에 수행될 수 없다.

d. **[5]** (True/False) Operation A.2 와 Operation B.3 은 동시에 수행될 수 없다

e. **[5]** (True/False) 프로세스 A 와 B 는 Deadlock 에 빠진다.

18. [10] 다음은 Peterson's Solution 과 같이 2 개의 프로세스를 대상으로 Critical Section 문제를 해결하기 위한 솔루션의 Pseudo-Code 이다. 다음 코드의 **정상 동작 여부를 기술하고 그 이유를 상세히 기술**하시오

```
boolean flag[2];
while(1) {
    flag[i] = true;
    while (flag[j]);
    …
    Critical Section
    …

    flag[i] = false;
    …
    Remainder Section
    …
}
```