종합시험 공부 내용 정리

빅데이터융합학과 조송현

과목 : 운영체제

1. CPU 스케줄링 기법인 RR(Round Robin) 알고리즘에 대한 물음에 답하시오.

(1-1) RR 알고리즘에 대한 스케줄링 방식에 대해 설명하고, RR 알고리즘에서 설정하는 time quantum 크기 측면에서 FIFO(First In First Out) 알고리즘과 비교 설명하시오.

라운드로빈 (순환할당) 스케쥴링 방식 / 선점식(preemptive) 방식 / FCFS에 의해 프로세스들이 보내지며, 각 프로세스는 같은 크기의 CPU 시간을 할당받음 / 시분할 방식에 효과적, 할당시간 크기가 매우 중요 / RR에서 Time quantum을 무한대로 설정시 FCFS(FIFO)와 동일 / RR에서 Time qunatum이 매우 작다면 Process sharing. 이론적으로 n개의 프로세스 각각 실제 프로세서의 n/1 속도로 실행되는 것처럼 보이게 됨. 문맥교환(context switching) 증가

(1-2) RR 알고리즘에서 time quantum 크기가 작을수록 발생할 수 있는 오버헤드 측면에서 큰 경우와 비교하여 장단점을 설명하시오.

Time quantum은 time slice와 같은 의미 / 운영체제가 프로세스에 CPU의 사용 시간을 할당할 때 그 기준이 되는 **최소 단위의 실행 시간** / **RR에서 Time quantum을 무한대로 설정시 FCFS(FIFO)와 동일** / **RR에서 Time qunatum이 매우 작다면 Process sharing**. 이론적으로 n개의 프로세스 각각 실제 프로세서의 n/1 속도로 실행되는 것처럼 보이게 됨. 문맥교환(context switching) 증가

2. Page replacement 방식 중 하나인 LRU(least-recently-used) 기법에 대한 물음에 답하시오.

(2-1) LRU 기법에서 교체할 page를 결정하기 위한 정책을 설명하고, 그 구현방법 2가지를 자세히 설명하시오

LRU 알고리즘은 가장 오랫동안 사용하지 않은 페이지를 교체하는 알고리즘 / 최적 알고리즘과 비슷한 효과를 낼 수 있음. / 성능이 좋은 편 / 많은 운영체제가 채택하는 알고리즘 / 단점 : 프로세스가 주기억장치에 접근할 때마다 참조된 페이지에 대한 시간을 기록해야함. (큰 오버헤드가 발생) /

(2-1) 정확한 하나의 LRU page를 선택하는 방식이 아닌, LRU-Approximation page를 결정하기 위한 구현 방법 2가지를 자세히 설명하시오.

LRU-Approximation 알고리즘은 기존의 LRU 알고리즘과 다르게 **H/W의 지원 없이 Reference-Bit를 이용하여 구현하는 알고리즘 기법** /

1. Additional-Reference-Bits 알고리즘

이 알고리즘은 Page table 내 각각의 Page에 8 비트 짜리 참조 비트가 존재하여 일정 시간의 간격마다 Access 되었던 페이지들의 비트들을 오른쪽으로 Shift하는 연산을 하게 됩니다. 그렇다면 가장 큰 값의 비트를 가진 페이지가 최근에 Access 되었다는 것을 알 수 있죠. 또한 가장 작은 값의 비트를 가진 페이지들이 Victim Page로 선택이 되며 같은 값들에 대해서는 FIFO 알고리즘을 이용하여 Victim 페이지를 고르게 됩니다.

2. Second-Chance 알고리즘

Second-Chance 알고리즘은 말 그대로 한 번의 기회를 더 주는 알고리즘을 말합니다. 이 알고리즘은 FIFO 알고리즘을 기반으로 합니다. 하지만 Page Table의 각각의 Page에는 1 비트 짜리 Reference bit가 존재하고 초기값은 모두 0이며 Access가 되면 1로 바뀝니다. 하지만 FIFO 알고리즘이 기본 알고리즘이기 때문에 Reference bit가 1이라도 선택이 될 수 있는데 이때 만일

Reference bit가 1이라면 최근에 Access되었다는 것이라고 간주를 하고 0으로 바꾼 후 한 번의 기회를 더 주고 다른 Victim 페이지를 찾습니다. 만일 모든 Page의 Reference bit가 1이라면 결국은 FIFO 방식으로 작동하게 됩니다.

3. 페이징 기법을 이용하는 가상 메모리 구조에서는 메모리에 해당 페이지가 없을 때, 페이지 교체를 통해 원하는 페이지를 메모리에 적재한 후 사용한다. 그러나 이런 페이지 교체가 자주 발생하게 되면, 프로세스의 처리 시간보다 메모리의 페이지 교체 시간이 더 길어지는 스래싱(Thrashing) 문제가 발생할 수 있다. 이와 같은 문제의 원인과 해결책을 쓰시오.

(원인) 부적절한 페이지 교체 정책 : Locality(집중적으로 함께 참조되는 페이지들 집합) 및 페이지 빈도 등을 고려하지 않음 / (Size of Locality > Total Memory Size) - 과도한 Multi-Processing 정책 : 주기억장치 사이즈 대비하여 많은 Multi-Procesing 정책에 따라 각 프로세스마다 필요한 페이지를 요구, 프로세스를 교환때마다 페이지 부재처리 발생 / 주로 주기억장치가 부족할 때 많이 발생.

(해결책) **1) Working Set(작업집합) 모델** : Working Set이란 특정시간에 실행되는 프로그램에 Locality가 포함되는 페이지들의 집합. 프로세스들은 주어진 시간 간격동안 자신의 페이지들 중 일부를 더 자주 접근하려는 지역성이 있으므로 이를 이용하면 페이지 부재율을 감소시킬 수 있음. / 프로그램이 효과적으로 실행되기 위해서는 활동프로세스의 Working Set이 메모리에 적재되어 있어야 함. / 프로세스가 진행되는 동안 Working Set은 계속 변화함.

2) Page-Fault Frequency(PFF) : 페이지 부재 발생시 Frame 수를 조정 / if 페이지부재율 > 상한 : 그 프로세스에 Frame 더 할당 / if 페이지부재율 < 하한 : 그 프로세스로부터 Frame 회수 / 이 결과, 페이지 부재 자주 발생하면 Frame 수가 점점 커지고 페이지부재 빈도 낮아지면 Frame 수가 점점 작아지게 됨.

3. Time quantum (혹은 time slice)을 설명하고, 태스크의 특성과 관련하여 time quantum의 길이와 스케줄러의 성능에 관한 연관 관계를 설명하시오.

Time quantum은 time slice와 동일한 의미로 운영체제가 프로세스에 CPU 사용시간을 할당할 때 기준이 되는 최소단위 실행시간. /

1) Time quantum 주기 짧을 때 : 각 프로세스에게 할당된 CPU 점유시간이 줄었다는 의미 / CPU를 사용하는 프로세스가 매우 빠른 주기로 변경 / 프로세스가 다른 프로세스에게 CPU를 양보하기 위한 문맥교환(Context Switching) 에 따른 부하가 늘어나 성능 저하 /

2) Time quantum 주기 길 때 : 프로세스는 입출력 과정에서도 블록(Blocking) 모드가 됨 / 현재 실행중이던 프로세스가 I/O 작업을 위해 블록 모드가 되었다고 가정하면 자신이 아무리 키보드나 마우스를 입력해도 다른 프로세스의 CPU 점유시간이 다 끝날 때 까지 나의 요청은 처리되지 않을 것. 즉, 시스템 반응이 늦어 문제 발생할 수 있음. /

4. 멀티프로그래밍만을 지원하는 운영체제 A가 있다. A에서 수행 가능한 형태로 적합하지 않은 것은? 정답 4

① 다수의 응용프로그램을 메모리에 적재하여 실행할 수 있다.

② 응용프로그램이 높은 CPU 활용도(utilization)를 요구하는 경우 효과적이다.

③ 응용프로그램의 입출력 요청이 완료되기까지 대기해야 하는 시간을 최대한 활용할 수 있다.

~~④ 응용프로그램이 짧은 응답시간(response time)을 요구하는 경우 효과적이다.~~

멀티프로그래밍이란 프로세서의 자원 낭비를 최소화하기 위해 낭비되는 시간을 다른 프로그램(프로세스) 수행에 쓰게 하여, 하나의 프로세서에서 여러 프로그램을 교대로 수행할 수 있게 하는 것. (프로세서가 입출력 작업의 응답을 대기할 동안 다른 프로그램을 수행시킬 수 있도록 하는 것)

5. SJF(Shortest Job First) 스케줄링 알고리즘은 실질적으로 구현하기가 어렵다. 그 이유를 설명하고 해결하기 위해 시도할 수 있는 방안에 대해 자세히 설명하시오

SJF(최소작업우선) 스케줄링 알고리즘은 ready큐 내 작업수행시간(CPU burst time)이 가장 짧다고 판단되는 것을 먼저 수행 / FCFS보다 평균 대기시간을 감소 / **구현 어려운 이유 : Next CPU Request의 길이를 미리 알 수 없다.** (I/O를 요청하는 명령어가 나와야, 그때까지가 CPU burst 이므로) / **해결 방안 : 우선순위 스케줄링(Priority Scheduling)** : 프로세스의 어떤 의미(semantic)에 따라 우선순위를 주고 높은 프로세스를 스케줄링하는 방법. / 우선순위란 : 시간제한, 메모리요구량과 같이 측정가능한 내부적 우선순위 또는 사용자선호도와 같이 임의 설정된 외부적 우선순위로 정의될 수 있음 / 단, Starvation문제 (우선순위 낮은 프로세스는 높은 프로세스 있는 한 무한히 실행 못하는 것)발생할 수 있으며 Aging으로 해결가능 /

6. Short-term scheduler와 Long-term scheduler의 차이점을 설명하고, (1)과 (2) 상황이 발생하는 경우, 스케줄러(scheduler)와 큐(queue) 관점에서 자세히 설명하시오.

(1) If all processes in the ready queue are I/O bound

(2) If all processes in the ready queue are CPU bound

Short-term Scheduler(CPU Scheduler)는 메모리에 있는 여러 프로세스 중 하나를 선택하여 프로세서를 할당하는 선택 방법. 즉, process를 Ready state에서 Runnning state로 전이시켜주는 것.

Long-term Scheduler(Job Scheduler)는 spooled된 프로세스들 중에서 어떤 것을 메모리로 Load할지 선택. 즉, New state에서 Ready state으로 상태전이 승인.

일반적으로 I/O Bound 프로세스는 CPU Bound 프로세스보다 우선순위가 높은데, Long-Term Sheduler는 이러한 프로세스를 적절히 선택시켜주는 작업을 함.

I/O bound 프로세스가 많아지면 ready queue가 항상 비어있을 것(I/O or event waiting에 몰리므로) -> Short-Term Scheduler가 할 일이 없어짐.

CPU bound 프로세스가 많아지면 I/O wating queue가 항상 비어있을 것 -> devices가 거의 사용되지 않고(I/O or event가 없어서) 시스템이 언밸런스해짐.

따라서 I/O Bound와 CPU Bound 프로세스가 적절히 조화되어야 최고의 시스템성능 가능.

I/O Bound는 프로세스가 진행될 때, I/O Wating 시간이 많은 경우다. 파일 쓰기, 디스크 작업, 네트워크 통신을 할 때 주로 나타나며 작업에 의한 병목(다른 시스템과 통신할 때 나타남)에 의해 작업 속도가 결정된다. / CPU Bound는 프로세스가 진행될 때, CPU 사용 기간이 I/O Wating 보다 많은 경우다. 주로 행렬 곱이나 고속 연산을 할 때 나타나며 CPU 성능에 의해 작업 속도가 결정

7. CPU scheduling algorithm들인 First In First Out, Shortest Job First, Round Robin을 비교하고 장단점을 분석하시오.

FIFO 스케줄링 : 비선점 / 프로세스들은 ready큐에 도착한 순서대로 cpu할당 / 일괄처리 시스템에서 주로 사용 / 작업완료시간 예측 용이 / 장점 : 알고리즘 간단, 공평 / 단점 : 성능(반환시간)이 좋지 않음. 짧은 작업이 긴 작업을 기다림.

SJF 스케줄링 : 비선점 / ready큐 내 작업수행시간 가장 짧다고 판단되는 것을 먼저 수행 / FCFS보다 평균대기시간 감소 / 큰 작업은 시간예측이 어려움. 짧은 작업에 유리 / 장점 : 평균대기시간이 최소 / 단점 : 일부작업의 무한대기 발생 가능성. Next CPU Request의 길이를 미리 알 수

없다

RR 스케줄링 : 선점 / FCFS에 의해 프로세스 보내짐 / 각 프로세스는 같은 크기의 CPU시간 할당받음 / 시분할방식에 효과적 / 할당시간의 크기가 매우 중요 / 할당시간이 크면 FCFS와 같게 되고 작으면 문맥교환 자주 일어남. / 장점 : 골고루 서비스 가능, 대화식 시스템에 적합 / 단점 : time slice가 작으면 문맥교환에 시간 낭비.

8. 세마포어(semaphore)에 관한 설명 중 틀린 것은? 정답 2

① 상호 배제 문제를 해결하기 위하여 사용된다.

~~② 정수의 변수로 양의 값만을 가진다.~~

③ 여러 개의 프로세스가 동시에 그 값을 수정하지 못한다.

④ 세마포어에 대한 연산을 처리 도중에 인터럽트 되어서는 안 된다.

세마포어는 상호배제 해결안으로 주어짐. / 세마포어의 P연산과 V연산 통해 가능하긴 함(변수값이 음수가 되면 프로세스 대기열에 넣음.) / 세마포어의 가장 중요한 사항은 그들이 원자적으로 실행된다는 점 / 같은 세마포어에 대해 두 프로세스가 동시에 P와 V 연산들을 실행할 수 없도록 반드시 보장해야 함.

9. 다음 자기 디스크의 접근 시간에서 시간이 가장 많이 소요되는 것은?

~~① 탐색 시간~~ ② 회전 지연 시간 ③ 헤드 활성화 시간 ④ 전송 시간

탐색시간은 기계적 동작에 의해 읽기/쓰기 헤드를 원하는 트랙에 위치시켜야 하기 때문에 나머지 요소 중 가장 느림.

10. 페이지 부재(page fault)를 처리하는 순서로 올바른 것은?

a. 운영체제에서 트랩(trap)이 발생한다. 1

b. 페이지 테이블을 재조정한다. 5

c. 현재 사용자 레지스터와 프로그램의 상태를 저장한다. 2

d. 사용 가능한 메모리 프레임을 프레임 리스트에서 찾는다. 3

e. 명령어 수행을 계속한다. 6

f. 디스크와 같은 backing store에 있는 페이지를 물리 메모리로 가져와 적재한다. 4

① a – b – c – d – e – f

② a – c – f – b – d – e

③ a – c – d – f – b – e

④ c – f – b – a – d – e

⑤ e – a – d – b – c – f

11. 마이크로커널(microkernel) 구조와 모놀리틱커널(monolithic kernel) 구조의 차이점에 대해서 커널 서브시스템(kernel subsystem)의 프로그램 실행 레벨과 보호영역(protection domain) 관점에서 서술하시오.

마이크로커널 : 커널의 가장 핵심 기능만 모아놓은 구조 / 커널의 컴포넌트 서버를 사용자모드에서 실행하고, **필수커널 루틴**만을 코어커널이 가지고 있음 / 커널모드에서는 스레드 사이의 메시지 전송과 하드웨어를 직접적으로 다루는 부분을 비롯한 최소한의 기능만을 동작하는 커널 의미. **하드웨어나 메시지전송 같은 부분들만을 커널모드에서 서비스**하고, **그 외 디바이스**

드라이버, 인터럽트핸들러 등의 OS서비스들이 대부분 사용자모드에서 실행.

모놀리틱커널 : 커널이 운영체제가 관장하는 모든 서비스들을 가지고 있는 구조 / 여러 운영체제 서브 시스템들이 하나의 큰 커널 안에 뭉뚱그려져 들어있으며 사용자 운영프로그램과 함께 동작 / 커널과 연결하는 시스템호출 인터페이스를 제공하여 사용자영역 응용프로그램과 커널 사이에 전환 메커니즘 제공. / OS의 주요기능을 모두 커널에 위치시켜 이와 같은 대부분의 동작이 커널모드에서 사용. / 처리기, 프로세스, 메모리, 파일시스템, 입출력관리, 네트워크의 모든 커널의 기능을 커널 내부에 시스템호출과 인터럽트 처리 부분으로 포함하여 제공하는 커널.

12. 선점형 스케줄러와 비선점형 스케줄러를 설명하고, 응답성, 예측 가능성 측면에서 비교/분석 하시오.

선점형 : 하나의 프로세스가 '이미 프로세스를 점유하고 실행중인 다른 프로세스'로부터 프로세서를 선점하여 실행할 수 있는 방법. / 현재 프로세스를 임의로 중단시키고 자기가 대신 차지 가능 / 특정 프로세스가 cpu를 효율적으로 사용할 수 없는 시점에 이를 때마다 cpu의 사용권이 다른 프로세스로 옮겨지는 방식 / 높은 우선순위의 프로세스들이 급하게 실행해야 할 경우 유용.
특징 : 대화식 시분할 시스템, 실시간 시스템에서 빠른 응답시간 유지해야할 때 필요 / 경비가 많이 들고 오버헤드까지 초래 / 효과적인 선점 위해서는 ready상태 프로세스가 많아야 함. / 우선순위 고려해야 함 / 문맥교환 횟수가 비선점형보다 많음 /

비선점형 : 하나의 프로세스가 프로세서 할당받으면 자신에게 할당된 시간 동안에는 다른 작업에 의해 간섭받지 않고 끝까지 프로세서 소유 / 현재 cpu차지하고 있는 프로세스의 작업이 완료되기까지 다른 프로세스가 cpu 빼앗을 수 없음 / 실행종료시까지 cpu 사용권 독점 / 짧은 작업 기다리는 경우 있지만 모든 프로세스관리에 공정 /
특징 : 응답시간 예측 쉬움 / 문맥교환 횟수 적음 / 일괄처리방식에 적합 / 모든 프로세스들의 요구에 공정한 처리 / 처리시간이 짧은 작업이 긴 작업의 수행 끝나기를 기다리는 경우 발생가능

13. 버퍼 캐시를 LRU 정책과 FIFO 정책 두 가지 방식을 사용한다고 할 때, 아래 액세스 패턴에 대해 총 액세스 타임을 계산하시오. 액세스는 블록 단위 로 이루어지며, 버퍼 캐시는 총 세 개의 블록을 저장할 수 있다고 가정한다. 버퍼 캐시에서의 액세스 타임은 0.1ms, 그 외의 경우는 10ms로 계산할 것. 액세스 패턴: A, B, C, D, A, E, C, B, A, D

FIFO 방식

시간 (ms)	10	20	30	40	50	60	70	80	90	100
입력값	A	B	C	D	A	E	C	B	A	D
버퍼1	A	A	A	D	D	D	C	C	C	D
버퍼2		B	B	B	A	A	A	B	B	B
버퍼3			C	C	C	E	E	E	A	A

액세스 패턴: A, B, C, D, A, E, C, B, A, D

LRU 방식

시간 (ms)	10	20	30	40	50	60	70	80	90	100
입력값	A	B	C	D	A	E	C	B	A	D
버퍼1	A	A	A	D	D	D	C	C	C	D
버퍼2		B	B	B	A	A	A	B	B	B
버퍼3			C	C	C	E	E	E	A	A

LRU : 최근으로부터 가장 오래 전에 사용한 페이지를 제거.

14. DMA(direct memory access)를 사용하여 CPU의 실행 부하(execution load)없이 고속 입출력 장치들을 사용하고자 한다. 이때 장치로의 메모리 연산이 완료되었음을 CPU가 알 수 있는 방법이 무엇이며, 그 방법과 트랩(trap)과의 차이에 대해서 서술하시오.

DMA는 (cpu와 상관없이) 모든 데이터 전송을 끝내면, 하드웨어 인터럽트로 cpu에게 알려줌. CPU에서 할당받은 작업이 끝난 경우 디바이스의 작업이 종료되었음을 메인cpu에 알리게 되는데 이 과정에서 발생하는 신호를 인터럽트라 함 / 인터럽트는 하드웨어적인 현상 / 소프트웨어적인 인터럽트는 트랩 / 인터럽트 발생시 추가적인 인터럽트 발생은 설정되어진 값에 따라 허가를 할수도안할수도 /

트랩은 프로그램 내 발생하는 것이며, 내부 인터럽트. cpu로부터 발생하는 운영오류 등이 포함 / 발생시점이 프로그램의 일정 지점이란 점에서 동기적. 즉, **고정된 영역에서 발생** / 트랩은 **운영체제의 루틴을 호출하거나 산술오류를 잡아내는 데 사용**할 수 있음. / 트랩 발생시에는 현재 프로세스 상태를 저장할 필요 없음. / **개념적으로 트랩은 어플리케이션에서 커널을 접근하는 것을 의미**. /

반면 **인터럽트는 프로그램 외부 상황에 따라 발생시점 일정하지 않음 (비동기적)** / 하드웨어 인터럽트는 **cpu외 다른 장치들에서 발생**. 키보드, 디스크, 드라이브, CD-ROM, 사운드카드, 마우스 등 장치들이 포함 / **인터럽트는 입출력 장치의 완료 신호로 사용할 수 있음** / 인터럽트는 디바이스에서 커널로 접근하는 것.

15. 5개의 프로세스들 P0, P1, P2, P3, P4에 할당된 자원의 수(Allocation), 작업 완료시까지 필요한 최대 자원의 수(Max), 그리고 현재 가용한 자원의 수(Available)가 다음과 같다고 하자. Deadlock handling methods 중 하나인 Banker's Algorithm을 사용하여 시스템이 안정 상태(safe state)인지 여부를 판단하시오.

안정 상태이면 처리 순서(safe sequence)를 구하고, 안정 상태가 아니라면 그 이유를 설명하시오. (단, safe sequence를 구할 수 있다면, 매 단계를 자세히 서술.)

	Allocation				Max				Available			
	A	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

Need

	A	B	C	D
P0	0	0	0	0
P1	0	7	5	0
P2	1	0	0	2
P3	0	0	2	0
P4	0	6	4	2

Available 1 5 2 0

P0 반납받기 (remaining need가 모두 0 이므로)

	A	B	C	D
P1	0	7	5	0
P2	1	0	0	2
P3	0	0	2	0
P4	0	6	4	2

=> Available 1 5 3 2

P3에게 빌려주고 반납받기

	A	B	C	D
P1	0	7	5	0
P2	1	0	0	2
P4	0	6	4	2

=> Available 1 11 8 4

P1에게 빌려주고 반납받기

	A	B	C	D
P2	1	0	0	2
P4	0	6	4	2

=> Available 2 18 13 4

P2에게 빌려주고 반납받기

	A	B	C	D
P4	0	6	4	2

=> Available 4 21 18 10

P4에게 빌려주고 반납받기

	A	B	C	D

=> Available 4 27 23 16

safe sequence : <P0,P3,P1,P2,P4>

※참고

· Allocation : 현재 할당 중인 자원
· Max : 프로세스가 작업 끝내기 위해 필요한 총 자원
· Available : 현재 빌려줄 수 있는 자원의 양
· Need : Max - Allocation. 프로세스가 작업을 끝내는데 앞으로 필요한 양

우리가 할 일은 현재 Available한 자원을 각 프로세스들에게 빌려주어 프로세스가 작업을 끝내게 만드는 것. 프로세스는 작업을 끝내면 그동안 가져갔던 모든 자원을 돌려주기 때문에, Available에 있는 자원의 양은 점점 늘어난다.

한마디로 Available을 이용해 / 만만한 Need를 협박해서 프로세스가 작업을 끝내게 하고 / 자원을 상환받는 것.

16. 아래와 같은 프로그램을 실행할 때, 첫 부모 프로세스를 포함해서 몇 개의 프로세스가 생성되는가?

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    fork();
    fork();
    fork();
    return 0;
}
```

fork();
부모
자식

fork();
부모
자식
손자
자식

fork();
부모
자식
손자
증손자
손자
자식
손자
자식
정답 : 총 8개
즉, 매번 fork()마다 각 대별로 자식놈들을 하나씩 더 낳으면 됨.

17. Critical-section problem의 해결책은 Mutual exclusion, Progress 그리고 Bounded waiting의 요구사항을 만족 시켜야 한다. 각 3가지 요구사항을 설명 하시오.
1) Mutual exclusion (상호배제) : 어떤 프로세스가 C/S에서 실행되고 있다면 다른 프로세스들은 해당 C/S에 실행되지 않도록 보장
2) Progress (진행) : C/S에 실행중인 프로세스가 없을 때 몇 개의 프로세스가 C/S에 진입하고자 하면 이들의 진입순서는 이들에 의해서만 결정되어야 함. (접근이 가능해야 함.)
3) Bounded waiting (한정 대기) : 한 프로세스가 자신의 C/S에 진입하고자 요청한 후부터 이 요청이 허용될때까지 다른 프로세스가 그들의 C/S에 진입할 수 있는 횟수가 제한되어야 함.
(기다리는 시간이 한정되어야 함.)

18. 유닉스 I-node가 10개의 직접 접근 블록과 각 1개씩의 1차(single), 2차(double) 간접 접근 블록(indirect block)까지 활용한다고 할 때, 한 파일이 표현할 수 있는 최대 용량을 계산하시오. 단, 하나의 디스크 블록은 1KB 이며, 하나의 디스크 블록 주소는 4 Bytes이다. (계산기 불필요 최종 결과는 수식으로 표현 가능)
주의: 계산 과정을 보이시오. 결과만 쓴 답안은 점수를 받지 못함.
- 직접 접근블록 : 1KB * 10 = 10KB
- 1차 간접 접근블록 : 1KB * 256 = 256KB
- 2차 간접 접근블록 : 1KB * 256 * 256 = 65536KB
10KB + 256KB + 65536KB = 65802KB
정답 : 65802KB

19. 페이징 기법을 이용한 시스템에서 페이지의 크기가 작을 경우와 클 경우에 나타나는 영향을 기술하시오.

페이지가 작으면 : 프로그램 구성하는 마지막 페이지의 단편화가 작아짐 / 실제 프로그램 수행에 필요한 내용만 포함할 수 있고 Locality(국부성)에 더 일치 -> 알찬 Working Set 유지할 수 있음 / 페이지 한 개를 주기억장치로 이동시키는 시간 감소 / 페이지 Mapping Table 크기가 커지고 맵핑속도가 늦어짐 / 디스크 접근 횟수가 늘어나서 총 입출력시간이 늘어남.

페이지가 크면 : 프로그램 구성하는 마지막 페이지의 단편화가 커짐 / 실제 프로그램 수행과 무관한 내용이 포함될 수 있음 / 페이지 한 개를 주기억장치로 이동시키는 시간 증가 / 페이지 Mapping Table 크기가 작아지고 맵핑속도가 빨라짐 / 디스크접근 횟수가 줄어들어 디스크 입출력이 효율적으로 됨.

20. 임계구역(critical-section) 문제의 해결을 위한 3가지 요구사항은 Mutual exclusion, Progress, Bounded waiting이다. 다음에 주어진 Algorithm 1과 2가 두 프로세스에 대한 임계구역 문제 해결을 위한 3가지 요구사항을 만족하는지 여부를 판단하시오. 그리고 만족 여부에 대해 자세히 설명하시오. (설명없이 만족여부만 판단하는 경우 0점)

```
boolean flag[2];  /* initially false */

int turn;  /* i or j */
```

Algorithm 1	Algorithm 2

repeat
flag[i] = true;
while (flag[j]) ;
Critical Section
flag[i] = false;
Remainder Section
until false;
	repeat
	flag[i] = true;
	turn = j;
	while (flag[j] && turn == j) ;
	Critical Section
	flag[i] = false;
	Remainder Section
	until false;

알고리즘 1 (flag사용): 자신의 flag[i]를 true로 변경한 다음, 상대방의 flag[j]를 확인하여 Pj가 c/s에 들어가고자 하는지 확인(만약 flag[j]==true 이면 Pi는 기다리다가 Pj가 수행 후 flag[j]를 false로 바꾸면 그때 Pi가 c/s 수행한다.
문제: Bounded Wating 미충족(Pi는 Pj의 깃발을 보고는, 자신의 깃발만들고 코드실행을 못함. 즉 둘다 들어가지 못하고 깃발만 들게되는 상황이 나옴)

알고리즘 2 (flag & turn 사용): 피터슨알고리즘에 의해 자신의 flag[i]를 먼저 true로 변경한 다음, turn=j로 turn을 다음 프로세스로 바꿔줌. 다음 프로세스 j가 들어가고자 하는 의사표시를 했고 && turn이 j차례인 경우 j프로세스가 c/s를 수행 후 나올때까지 기다림. c/s를 수행한다음 flag를 false로 바꿔준 다음, c/s를 빠져나옴. ->
즉, 상대방의 차례인 경우 대기하고, 아니라면 자신이 들어가서 수행 진행. 만약 자신 수행 후 turn이 상대방으로 바뀌어도 상대방이 더 이상 한다는 의사표시 없다면 조건 두개 다 충족되는 것이 아니므로 자신이 수행할 수 있음. 따라서 끊임없는 양보상황 발생하지 않음. 3가지 요구사항 모두 만족

※Solution for Critical Section Problem

1) mutual exclusion(상호배제) 임의의 한 프로세스가 크리티컬섹션을 수행하는 동안에는 어떤 다른프로세스도 크리티컬섹션을 수행할 수 없다. (only one) (간섭받을 수 없다)

2) progress (진행) 아무도 c/s에 들어가있지 않은 상황이라면 들어가고자 하는 프로세스는 들어갈 수 있어야 한다. 크리티컬섹션을 수행하는 프로세스가 지금 없는데(비어 있는데), 진입(enter)을 하고 싶은데 못한다면 문제가 있다(수행할 수 있어야 한다)

3) Bounded Waiting (제한적 대기. 웨이팅이 무한해선 안되고, 어느 일정시간이 지나면 반드시 차례가 와야 한다.) 이 3가지 만족해야 크리티컬 섹션 프로브럼의 솔루션으로 적합하다

21. CPU 스케줄링 알고리즘들 중 FIFO(First In First Out)와 RR(Round Robin) 기법에 대해 비교 설명하시오. 그리고 이러한 두 알고리즘 간의 관계를 설명하시오.

FIFO 스케줄링 : 비선점 / 프로세스들은 ready큐에 도착한 순서대로 cpu할당 / 일괄처리 시스템에서 주로 사용 / 작업완료시간 예측 용이 / 장점 : 알고리즘 간단, 공평 / 단점 : 성능(반환시간)이 좋지 않음. 짧은 작업이 긴 작업을 기다림.

RR 스케줄링 : 선점 / FCFS에 의해 프로세스 보내짐 / 각 프로세스는 같은 크기의 CPU시간 할당받음 / 시분할방식에 효과적 / 할당시간의 크기가 매우 중요 / 할당시간이 크면 FCFS와 같게 되고 작으면 문맥교환 자주 일어남(RR에서 Time quantum을 무한대로 설정시 FCFS(FIFO)와 동일). / 장점 : 골고루 서비스 가능, 대화식 시스템에 적합 / 단점 : time slice가 작으면 문맥교환에 시간 낭비.

22. 가상메모리 구조에서 페이징 기법을 사용하는 경우 쓰레싱(thrashing) 문제가 발생할 수 있다. 이 문제가 발생할 수 있는 상황에 대해 자세히 설명하고, 해결 방안에 대해 서술하시오.

스래싱(thrasing) (원인) 부적절한 페이지 교체 정책 : Locality(집중적으로 함께 참조되는 페이지들 집합) 및 페이지 빈도 등을 고려하지 않음 / (Size of Locality > Total Memory Size) - 과도한 Multi-Processing 정책 : 주기억장치 사이즈 대비하여 많은 Multi-Procesing 정책에 따라 각 프로세스마다 필요한 페이지를 요구, 프로세스를 교환때마다 페이지 부재처리 발생 / 주로 주기억장치가 부족할 때 많이 발생.

(해결책) 1) Working Set(작업집합) 모델 : Working Set이란 특정시간에 실행되는 프로그램에 Locality가 포함되는 페이지들의 집합. 프로세스들은 주어진 시간 간격동안 자신의 페이지들 중 일부를 더 자주 접근하려는 지역성이 있으므로 이를 이용하면 페이지 부재율을 감소시킬 수 있음. / 프로그램이 효과적으로 실행되기 위해서는 활동프로세스의 Working Set이 메모리에 적재되어 있어야 함. / 프로세스가 진행되는 동안 Working Set은 계속 변화함.

2) Page-Fault Frequency(PFF) : 페이지 부재 발생시 Frame 수를 조정 / if 페이지부재율 > 상한 : 그 프로세스에 Frame 더 할당 / if 페이지부재율 < 하한 : 그 프로세스로부터 Frame 회수 / 이 결과, 페이지 부재 자주 발생하면 Frame 수가 점점 많아지고 페이지부재 빈도 낮아지면 Frame 수가 점점 적어지게 됨.

23. 가상메모리(virtual memory)의 크기가 256 MB인 시스템에서, 가상메모리의 크기는 변동없이 페이지 크기가 4,096 bytes에서 256 bytes 로 변경되었다고 가정하자. 이 때, 논리 주소 간에서 페이지크기를 의미하는 영역의 크기(bits의 개수)는 어떻게 변화하는지 서술하시오. 또한, 이와 같이 페이지 크기의 변화가 내부단편화(internal fragmentation) 및 전체 입출력시간에 미치는 영향에 대해 설명하시오.

가상메모리 크기 : 256MB (2^8 * 2^10 * 2^10 bytes)
페이지 크기 : 4096 (2^2 * 2^10 bytes -> 256 (2^8 bytes)
즉, 페이지크기가 1/2^4 bytes 크기로 줄어들었음.

페이지 크기를 작게 자를수록 내부 단편화를 줄일 수 있음.
페이징 기법에서 내부 단편화는 프로세스에게 할당된 마지막 페이지에 남은 영역에서 발생.
페이지 크기가 작으면 총 입출력 시간은 늘어난다.
페이지 크기가 작을수록 더 많은 페이지 사상 테이블 공간이 필요하다.
페이지 크기가 작을수록 내부 단편화는 줄어들게 된다.
페이지 크기가 작을수록 사용하는 페이지의 집합을 효율적으로 운영할 수 있다.
페이지 크기가 작을수록 특정한 참조 구역성만을 포함하기 때문에 기억 장치 효율이 좋을 수 있다.

24. Realtime operating system의 스케줄링 정책인 priority-based 방식으로 프로세스들 X, Y를 스케줄링한다고 가정하자. X와 Y의 periods인 px=50, py=80이고 processing times인 tx=30, ty=25 일 때, 다음 물음에 답하시오.

※ 우선순위 스케줄링 : 각 프로세스에 우선순위 번호를 부여하고 CPU는 우선순위가 높은 프로세스를 실행
두 개의 프로세스 px, py
각각의 주기는 px = 50, py = 80 //
수행 시간은 tx = 30, ty =25
그리고 우리가 스케줄링에 대해 논할 때는 이 프로세스가 데드라인을 만족시키는 지 살펴봐야함.
일단은 CPU utilization 에 대해 먼저
P1 = 30/50 = 0.60
P2 = 25/80 = 0.3125
이 둘을 단순 합산 하면 91.25% 의 utilization을 사용합니다. 1보다 작기 때문에 이 두 프로세서를 실행을 시킬 경우 데드라인을 지킬 수 있음

(24-1) 프로세스들 X, Y의 periods와 processing time을 사용하여 프로세스들 X, Y, 그리고 전체 CPU Utilization을 구하시오
P1 = 30/50 = 0.60
P2 = 25/80 = 0.3125
전체 = 0.9125

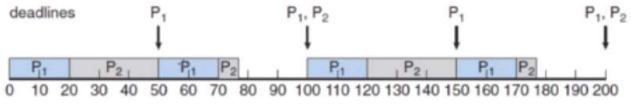
(24-2) X, Y가 rate-monotonic 스케줄링 방식을 사용하여 스케줄링된다고 할 때, 수행되는 과정을 Gantt chart로 표현하고 스케줄링 가능 여부를 설명하시오. (스케줄링 정책은 "the shorter the period, the higher the priority" 라 가정)
rate-monotonic : 발생율이 클수록, 즉 주기가 짧을 수록 우선순위를 높히는 스케줄링 기법

각각의 주기는 p1 = 50, p2 = 100 //이 기법에 의하면 P1의 우선순위가 높은 겁니다.
수행 시간은 t1 = 20, t2 =35

그리고 우리가 스케줄링에 대해 논할 때는 이 프로세스가 데드라인을 만족시키는 지를 살펴보아야합니다.
일단은 CPU utilization 에 대해 먼저 이야기 해봅시다.
P1 = 20/50 = 0.40
P2 = 35/100 = 0.35

이 둘을 단순 합산 하면 75% 의 utilization을 사용합니다. 1보다 작기 때문에 이 두 프로세서를 실행을 시킬 경우 데드라인을 지킬 수 있을 겁니다.

방금 상황의 실제 예시를 보겠습니다.



< Fig 17. Rate-monotonic scheduling >

P1의 주기는 50입니다. 그러니 50,100,150 에서 실행이 되어야합니다. 주기가 50이기 때문에 50마다 P1 프로세스가 실행된다는 걸 알 수 있습니다. P2는 100이 주기이기 때문에 100 그리고 200 에서 실행이 되고 있습니다.

source : https://com24everyday.tistory.com/168

(24-3) X, Y가 EDF(earliest-deadline-first) 스케줄링 방식을 사용하여 스케줄링이 가능한지 Gantt chart를 그려서 그 과정을 자세히 설명하시오.

25. Page replacement 기법에 대한 물음에 답하시오.

(25-1) Page replacement가 필요한 상황에 대해 설명하고, Page들의 교체(replacement)가 수행되는 과정을 자세히 서술하시오.

페이지 부재(CPU가 참조하려는 페이지가 현재 메모리에 올라와 있지 않은 경우)가 발생하면 요청된 페이지를 디스크(backing store)에서 메모리로 읽어와야 한다.
이 때 물리적 메모리에 빈 프레임이 존재하지 않을 수 있는데 이 경우에 메모리에 올라와 있는 페이지 중 하나를 디스크로 쫓아내고 메모리에 빈 공간을 확보하는 작업이 필요하다.
이를 페이지 교체라고 하며 페이지 교체를 할 때 어떤 프레임에 있는 페이지를 쫓아낼 것이지 결정하는 알고리즘이 페이지 교체 알고리즘이다.

1) 페이지 테이블을 통해 페이지를 참조(reference)하려고 하는 해당 페이지가 invalid(0) 하다면, 즉 페이지가 메모리에 없다면 MMU가 OS에 페이지 부재 트랩 (Page Fault Trap)을 발생시킨다.
2) OS는 내부 테이블(주로 PCB에 있음)을 체크하여

if 부적절한 참조(invalid reference)라면 프로세스를 중지(abort)한다.
if 그저 메모리에 없는 것이라면 3번과정으로 넘어간다.

3) 페이지가 Backing Store에 존재한다면
4) 물리 메모리에 비어있는 프레임(Free Frame)이 존재한다면 그곳에 페이지를 읽어온다. 이때 만약 비어있는 프레임이 없다면 페이지 교체 알고리즘(Page Replace Algorithm)을 통해 교체할 페이지를 골라야 한다.
5) 페이지 테이블을 리셋한다. (페이지 테이블의 valid-invalid bit를 valid로 수정한다)
6) 다시 프로세스를 실행한다.

(25-2) Page replacement를 위한 LRU(least-recently-used) 및 Optimal 기법 각각의 특징에 대해 서술하고 두 기법을 비교 설명하시오.

LRU(least-recently-used)기법
최적 알고리즘의 방식과 비슷한 효과를 낼 수 있음. (최적 알고리즘은 페이지가 사용될 시간을 미리 알고 있는데, 미리 아는 것이 불가능하다면, 과거의 데이터를 바탕으로 페이지가 사용될 시간을 예측하여 교체하는 것은 가능함) /
LRU 알고리즘은 **가장 오랫동안 사용하지 않은 페이지를 교체**하는 알고리즘 /
최적 알고리즘과 비슷한 효과를 낼 수 있음. / **성능이 좋은 편** / **많은 운영체제가 채택**하는 알고리즘 / 단점 : **프로세스가 주기억장치에 접근할 때마다 참조된 페이지에 대한 시간을 기록해야함. 큰 오버헤드가 발생** /

OPT(Optimal)기법(최적 페이지교체 알고리즘)
OPT 알고리즘은 **앞으로 가장 오랫동안 사용하지 않을 페이지를 교체하는 알고리즘** /
모든 페이지 교체 알고리즘 중 **page-fault 발생이 가장 적음** /
Belady`s Anomaly 현상(메모리 공간이 늘어나는데 페이지 폴트가 늘어나는 이상현상)이 발생하지 않음 /
프로세스가 앞으로 사용할 페이지를 미리 알아야 함 /
실제로 구현하기 거의 불가능한 알고리즘. 실제로 사용하기 보다는 연구 목적을 위해 사용된다.

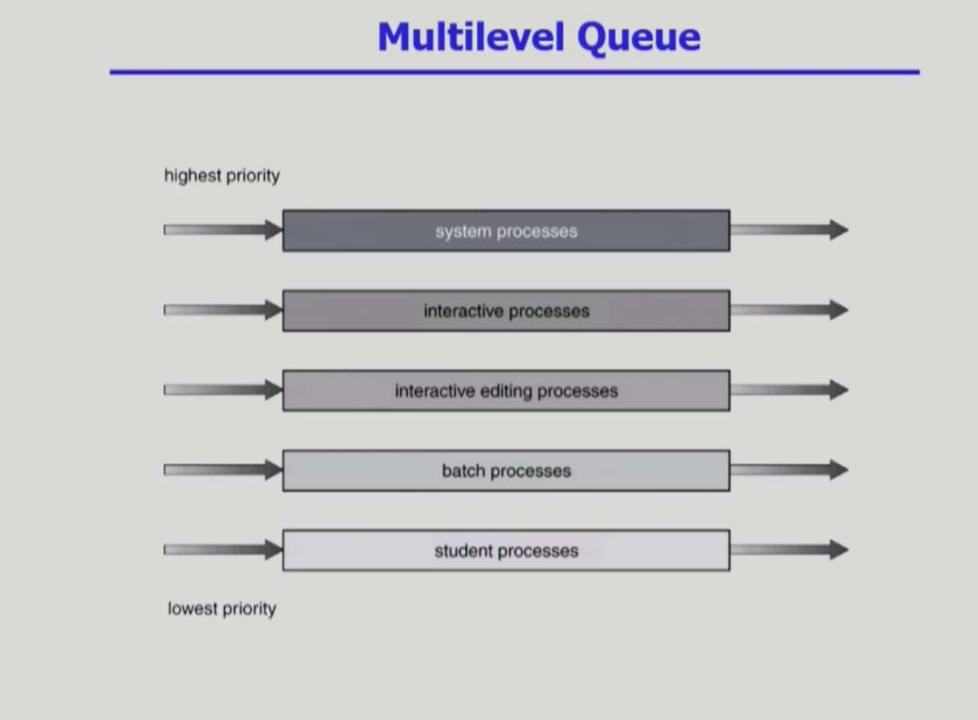
26. CPU스케줄링 기법에 대한 물음에 답하시오.

(26-1) Round robin 기법에 대해 설명하고, time quantum의 크기에 따라 스케줄링 성능에 영향을 주는 부분에 대해 서술하시오.

라운드로빈 (순활할당) 스케쥴링 방식 / 선점식(preemptive) 방식 / FCFS에 의해 프로세스들이 보내지며, 각 프로세스는 같은 크기의 CPU 시간을 할당받음 / 시분할 방식에 효과적, 할당시간 크기가 매우 중요 / RR에서 Time quantum을 무한대로 설정시 FCFS(FIFO)와 동일 / RR에서 Time qunatum이 매우 작다면 Process sharing. 이론적으로 n개의 프로세스 각각 실제 프로세서의 n/1 속도로 실행되는 것처럼 보이게 됨. 문맥교환(context switching) 증가

Time quantum은 time slice와 같은 의미 / 운영체제가 프로세스에 CPU의 사용 시간을 할당할 때 그 기준이 되는 최소 단위의 실행 시간 / RR에서 Time quantum을 무한대로 설정시 FCFS(FIFO)와 동일 / RR에서 Time qunatum이 매우 작다면 Process sharing. 이론적으로 n개의 프로세스 각각 실제 프로세서의 n/1 속도로 실행되는 것처럼 보이게 됨. 문맥교환(context switching) 증가

(26-2) Multi-level Queue 스케줄링 기법의 특징을 설명하고 그 구현 방법에 대해 서술하시오.



- 1) Ready큐를 여러 개로 분할하고, 2) 각각의 큐의 우선순위를 정하고, 3) 각각의 프로세스는 해당 프로세스의 우선 순위에 따라 각각의 큐에 배치되고, 4) 큐 간 경쟁을 통해 하나의 큐가 cpu를 점유하는 형태.

- foreground 큐와 background 큐로 분할. foreground 큐는 interactive하며 cpu burst가 짧은 queue로서 우선 순위가 높다. background 큐는 batch 등 긴 시간을 필요로 하는 작업이다.

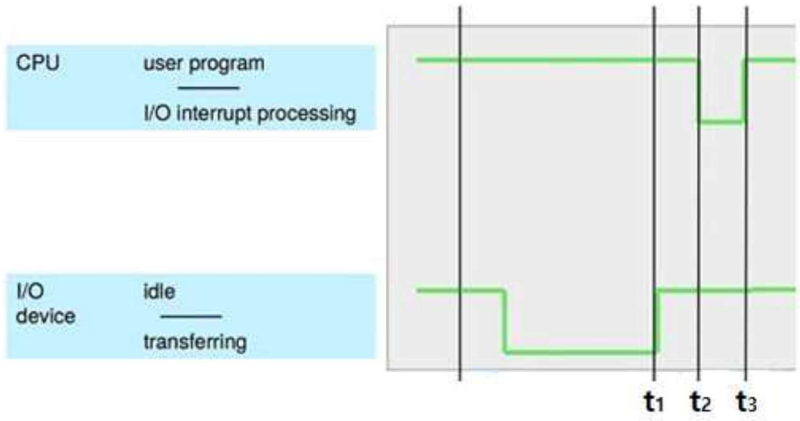
- 각 큐마다 사용하는 스케줄링 알고리즘이 다르다. foreground의 경우 RR 방식, Backgournd의 경우 FCFS를 사용한다.

- 큐 간의 시간 점유 시간의 격차가 크면 클수록 Starvation 현상이 일어날 가능성이 높다. 그러므로 Time Slice를 통해 각 큐의 CPU time을 적절하게 할당해야 한다.

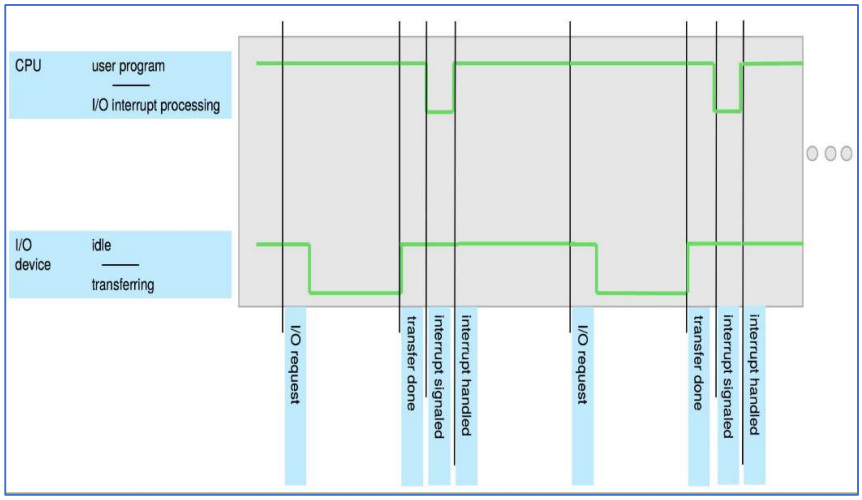
(26-3) Shortest Job First 와 Priority 기법간의 관계를 설명하시오.

SJF(최소작업우선) 스케줄링 알고리즘은 준비큐 내 작업수행시간(CPU burst time)이 가장 짧다고 판단되는 것을 먼저 수행 / FCFS보다 평균 대기시간을 감소 / 구현 어려운 이유 : Next CPU Request의 길이를 미리 알 수 없다. (I/O를 요청하는 명령어가 나와야, 그때까지가 CPU burst 이므로) / 해결 방안 : 우선순위 스케줄링(Priority Scheduling) : 프로세스의 어떤 의미(semantic)에 따라 우선순위를 주고 높은 프로세스를 스케줄링하는 방법. / 우선순위란 : 시간제한, 메모리요구량과 같이 측정가능한 내부적 우선순위 또는 사용자선호도와 같이 임의 설정된 외부적 우선순위로 정의될 수 있음 / Starvation문제(우선순위 낮은 프로세스는 높은 프로세스 있는 한 무한히 실행 못하는 것)는 Aging으로 해결 /

27. 다음 그림은 CPU와 I/O device 간 Timeline을 표현하고 있다. 물음에 답하시오.



(참고그림)



(27-1) 시간 t1에서 I/O device의 상태가 바뀌는데, 그 상황을 I/O device 측면에서 설명하시오.

I/O transfer done
I/O 시작하고 난후 I/O끝나지 않아도 유저프로그램이 컨트롤을 받음: asynchronous I/O
시스템콜(입출력완료됐다고 알림)

(27-2) 시간 t2에서 CPU의 상태 변화 이유를 설명하고, CPU가 처리하는 과정에 대해 자세히 서술하시오.

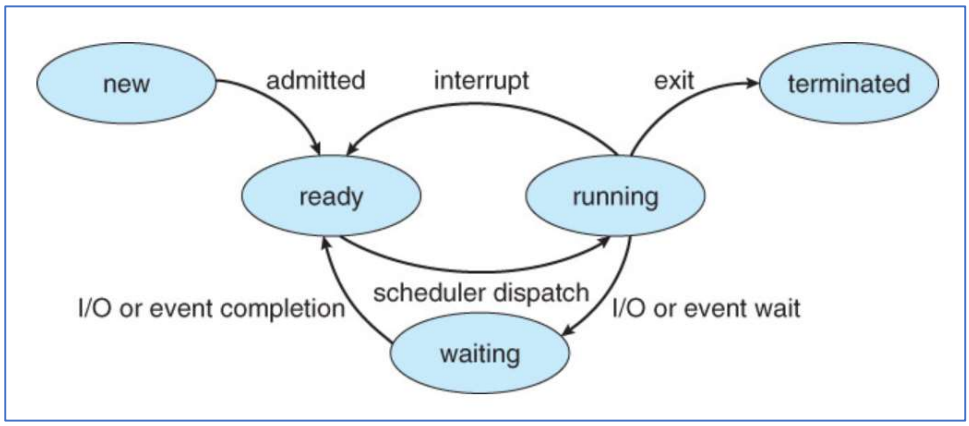
t2에서 인터럽트를 발생, CPU 인터럽트 처리
인터럽트 발생 시 처리과정
1. 프로그램 실행을 중단
2. 현재의 프로그램 상태를 보존
3. 인터럽트 처리 루틴을 실행
4. 인터럽트 서비스 루틴을 실행

(27-3) 시간 t3에서 CPU가 수행하는 과정에 대해 자세히 설명하시오.

5, 인터럽트 요청 신호가 발생했을 때 보관한 PC의 값을 다시 PC에 저장.(the saved return address is loaded into the program counter)
6. PC의 값을 이용하여 인터럽트 발생 이전에 수행중이던 프로그램을 계속 실행.

[기타 참고개념정리]

※ 프로세스 상태



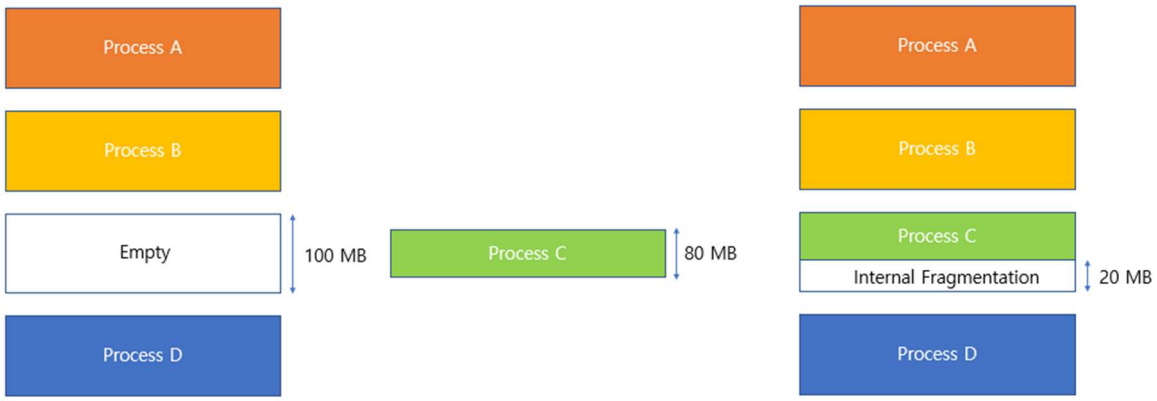
※ 오버헤드

오버헤드(overhead)는 어떤 처리를 하기 위해 들어가는 간접적인 처리 시간 · 메모리 등

예를 들어 A라는 처리를 단순하게 실행한다면 10초 걸리는데, 안전성을 고려하고 부가적인 B라는 처리를 추가한 결과 처리시간이 15초 걸렸다면, 오버헤드는 5초가 된다. 또한 이 처리 B를 개선해 B'라는 처리를 한 결과, 처리시간이 12초가 되었다면, 이 경우 오버헤드가 3초 단축되었다고 말한다

※ 내부단편화

내부 단편화란 주기억장치 내 사용자 영역이 실행 프로그램보다 커서 프로그램의 사용 공간을 할당 후 사용되지 않고 남게 되는 현상을 말한다.

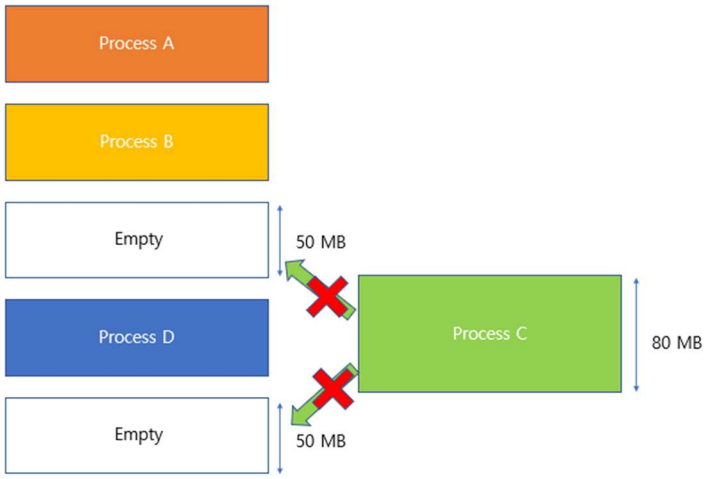


따라서 남아있는 메모리 공간이 낭비되게 되는 문제가 발생한다.

출처: https://code-lab1.tistory.com/54 [코드 연구소:티스토리]

※외부단편화

외부 단편화란 남아있는 총 메모리 공간이 요청한 메모리 공간보다 크지만, 남아있는 공간이 연속적(contiguous)이지 않아 발생하는 현상이다.

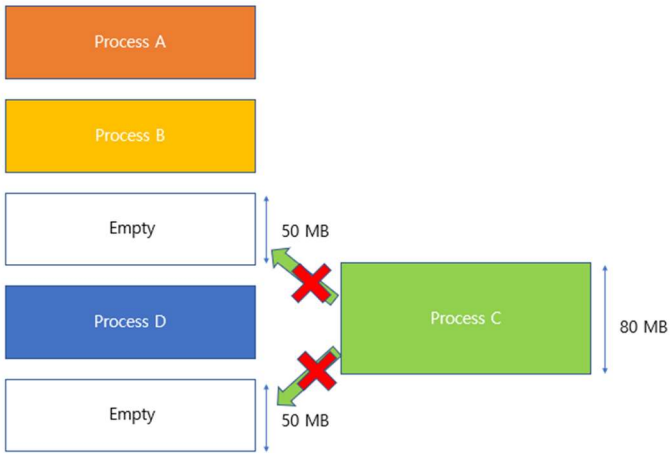


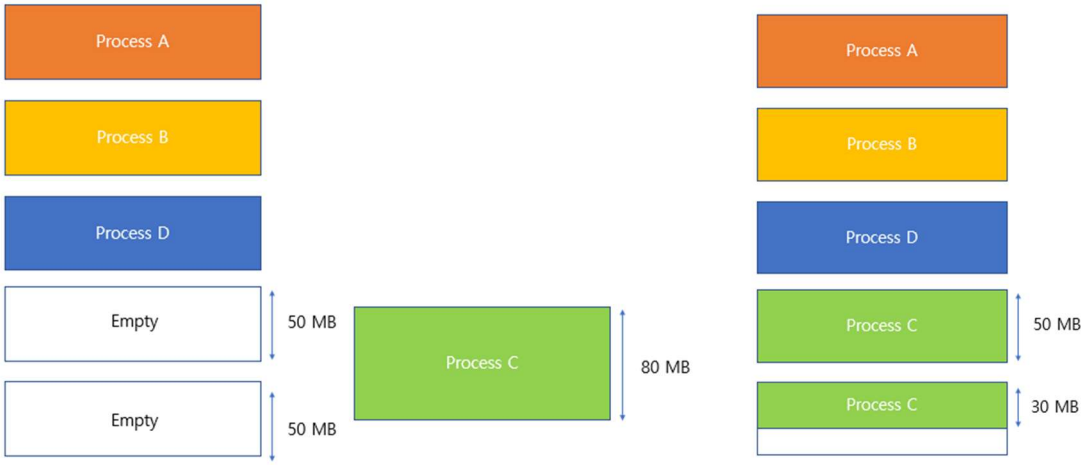
위와 같이 남아있는 메모리 공간은 50MB+50MB =100MB로 요청한 메모리 공간 80MB보다 크지만, 남아있는 공간이 연속적이지 않아 Process C를 할당할 수가 없게 된다. 따라서 남아있는 메모리 공간이 낭비되게 되는 문제가 발생한다.

출처: https://code-lab1.tistory.com/54 [코드 연구소:티스토리]

※ 외부 단편화 해결방법 - 압축 (Compaction)

외부 단편화 문제를 해결하기 위해서 압축 기법을 사용할 수 있다. 압축 기법은 주기억장치 내 분산되어 있는 단편화된 공간들을 통합하여 하나의 커다란 빈 공간을 만드는 작업을 의미한다.





출처: https://code-lab1.tistory.com/54 [코드 연구소:티스토리]