|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | |  | | **[ C언어 main함수에서 반환값이 0인 이유 ]** | | **프로세스를 실행하면 부모프로세스에서 커널의 fork()함수를 시스템콜하여**  **자식프로세스로 부모프로세스와 동일한 프로세스를 생성한다.**  **동일한 프로세스에서 exec()함수를 시스템콜하여 원하는 코드로 덮어씌우게된다**  **이때 자식프로세스가 반환하게되는 PID(Process ID)값은 0이다**  **부모프로세스에 0을반환하는건 정상 종료되었다는것을 알려주게된다** | |  | | **[ 선점형 ( Preemptive 프리앰티브 ) ]** | | **Cpu스케줄링 에서 여러프로세스가 실행중에 하나의 프로세스가 독점해서 사용하는것이 아니라**  **Timer의 시간을 두어서 정해진시간이 되면 cpu의 사용권을 강제로 빼앗아온다** | |  | | **[ 비선점형 ( non-Preemptive 넌프리앰티브 ) ]** | | **Cpu스케줄링 에서 여러프로세스가 실행중에 하나의 프로세스가 cpu를 독점하여 프로세스를 완료하고**  **그다음 프로세스로 cpu의 사용권을 스스로 넘긴다** | |  | | **[ 호위효과 ( Convoy Effect 콘보이 이펙트) ]** | | **1차세계대전 때 전쟁물자를 호송을 하는대 물자를 보호하기위해 앞뒤로 탱크가 호위하였던 것에서**  **유래되어 Cpu스케줄링에 있어서 FCFS알고리즘의 문제점으로 먼저들어온 프로세스의 실행시간이**  **아주 길게되면 뒤에들어오는 짧은 프로세스들이 전부 기다리게되어 전체적인 cpu성능을 저해하는 상황이다** | |  | | **[ FCFS ( First-Come First-Served 퍼스트컴 퍼스트서브드) ]** | | **프로세스들이 도착한 순서에 따라 실행한다. 그러나 먼저 온 프로세스가 오랜 시간을 요구하는 경우**  **뒤에 온 프로세스들은 대기 시간이 길어질 수 있는 단점이 있다** | |  | | **[ SJF ( Shortest-Job-First 숏테스트잡퍼스트 ) ]** | | **비선점형 방식으로**  **먼저 도착한 프로세스를 cpu버스트하고 버스트 타임안에 도착한 프로세스들 중에**  **cpu버스트타임이 짧은 순서대로 프로세스를 실행한다** | |  | |  | | **[ 디스패처 ( Dispatcher ) ]** | | **커널에 디스패처 함수가있다 cpu스케줄링으로 결정이된 후 문맥( Context )을저장하고**  **다음프로세스에게 cpu를 할당하는 역활을 해준다** | |  | | **[ 램영역 ( Ram ) ]** | | **RAM----------------------------------mode bit 1**  **| 커널영역 | 텍스트영역 | 스택영역 | 힙영역 |**  **-----------------------------------------------** | |  | | **[ 커널영역 ( Kernel ) ]** | | **커널영역----------------------------------------------------------mode bit 0**  **| 텍스트영역 | 데이터영역 | 스택영역 | 힙영역 | 프로세스 제어 블록 |**  **--------------------------------------------------------------------------** | |  | | **[ SRTF ( Shortest-Remaining-Time-First 숏테스트 리마이닝타임 퍼스트 ) ]** | | **SJF의 선점형 방식 으로**  **짧은 프로세스를 실행중에 새로운 cpu버스트타임이 더짧은 프로세스가 도착하면**  **더짧은 프로세스를 디스패처 한다** | |  | | **[ 기아 ( Starvation 스타베이션 ) ]** | | **SRTF의 방식에서 cpu버스트타임이 짧은 프로세스들이 계속해서 앞쪽에 끼어들어서**  **cpu버스트타임이 긴 프로세스는 실행되지 못하는 현상** | |  | | **[ 우선순위 스케줄링 ( Priority Scheduling 프라이올리 스케줄링 ) ]** | | **우선순위가 제일높은 프로세스에게 cpu를 준다** | |  | | **[ 에이징 ( Aging ) ]** | | **우선순위가 낮은 프로세스라도 기다림에 따라서 우선순위를 높여준다** | |  | |  | | **[ RR ( Round Robin 라운드로빈) ]** | |  | |  | | **[ 타임슬라이스 / 시간할당량 ( Time Quantum 타임 퀀텀) ]** | | **cpu가 하나의 프로세스를 몇초동안 실행하고 다음 프로세스를 디스패치할껀지의 시간이다** | |  | | **[ 멀티레벨큐 MLQ ( MultiLevel Queue ) ]** | | **단일코어를 전재로한다 우선순위큐로 프로세스를 줄을세운다**  **준비큐 ( Ready queue )는 여러 큐가 이있다**  **1) Foreground**  **I/O인터럽트,Trap인터럽트가 많은 프로세스가 있는 큐**  **2) Background**  **Cpu를 오랫동안 사용하는 프로세스가 있는 큐** | |  | | **[ 멀티레벨 피드백큐 MLFQ ( MultiLevel Feedback Queue ) ]** | | **단일코어를 전재로한다**  **1)-먼저 들어온 프로세스는 가장 높은 우선순위 큐에 들어가게 된다**  **이 큐에서는 라운드로빈 방식으로 타임 슬라이스 시간만큼 실행한다**  **만약 프로세스가 해당 타임 슬라이스 시간 안에 끝나지 않았다면**  **2)-다음으로 낮은 우선순위 큐로 이동하여 대기한다**  **그리고 다음 우선순위 큐에서는 더 긴 타임 슬라이스 시간을 할당하여 실행한다**  **만약 프로세스가 여전히 실행이 완료되지 않았다면**  **3)-이번에는 가장 낮은 우선순위 큐에 들어가서 FCFS 방식으로 프로세스를 실행한다** | |  | | **[ 멀티 프로세서 스케줄링 ( Multiple Processor Scheduling ) ]** | | **코어가 여러개인 경우** | |  | | **[ 리얼타임스케줄링 ( Real Time Scheduling ) / 리얼타임잡 ( Hard Real Time Task ) ]** | | **주어진 시간에 데드라인 안에 반드시 끝내야하는 프로세스 스케줄링 방식** | |  | | **[ 스레드 스케줄링 ( Thread Scheduling ) ]** | | **사용자레벨(유저스레드)는 사용자프로그램 안에서 실행흐름인 스레드가 Cpu에 디스패치 할지 결정한다**  **커널스레드는 운영체제가 스레드의 존재를 알고있고 커널의 cpu스케쥴링 방식으로 cpu를 디스패치 한다** | |  | | **[ 프로세스 동기화 ( Process Synchronization 싱크로나이제이션 ) / 병행제어 ( Concurrent Control ) ]** | | **프로세스가 동시에 뭔가를 실행하면서 생기는 문제를 해결하는 방법이다**  **공유데이터( Shared Data )에 동시접근( Concurrent Access ) 하면**  **데이터처리에 시간적 차이가 생겨서 원치않는 결과가 나온다**  **커널모드를 실행중일때는 타임슬라이스가 다되어도 커널모드에서 유저모드로 바뀔때까지는 실행한다**  **커널모드에서 동일한 데이터를 접근하는게 아니라면 허용하고 동일하다면 락을 걸어준다** | |  | | **[ 임계영역 ( Critacal Section 크리티컬 영역 ) ]** | | **공유데이터를 말하는것이 아니라 공유데이터를 접근한 코드를 말한다** | |  | | **[ 상호배제 ( Mutual Exclusion 뮤추얼 익스클루전 ) ]** | | **어떤 프로세스가 크리티컬섹션( 임계영역 )에 들어가있다면**  **다른프로세스는 임계영역에 들어가지 못하게 해줘야 한다** | |  | | **[ 진행 ( Progress 프로그래스) ]** | | **임계영역에 어떤 프로세스도 접근하고 있지않다면**  **접근하고 싶어하는 프로세스가 있다면 접근할 수 있게 해줘야 한다** | |  | | **[ 유한대기 ( Bounded Waiting 바운디드 웨이팅 ) ]** | | **임계영역에 접근하기위해 대기하는시간은 유한해야 한다** | |  | | **[ 스핀락 ( Spin Lock ) / Busy Wating ( 비지웨이팅 ) ]** | | **while문을 돌면서 프로세스를 기다리는 상황에 cpu와 메모리를 낭비하는 상황** | |  | | **[ Test And Set ]** | | **고급언어에서의 하나의 명령어는**  **cpu의 여러 인스트럭션 으로 만들어져있는대 cpu의 하나의 인스트럭션 을 실행하고**  **다음 프로세스에게 cpu사용권을 디스패치 할수도 있기 때문에**  **이러한 문제를 cpu명령어를 만들어서 중간에 cpu를 뺏기지않는 Lock인스트럭션을 만들었다**  **Test\_And\_Set(Lock) Lock의 값을 읽어오고 그다음 1로 셋팅하는 인스트럭션** | |  | | **[ 세마포어 ( Semaphore ) ]** | | **동기화 문제를 해결하기위한 첫번째 방법**  **Lock에 대한 추상화 방식 공유자원을 획득(P)=1하고 반납(V)=0하는 Lock에대한 방식**  **세마포어 변수는 정수이고 변수의 갯수가 자원(스레드)의 갯수이다** | |  | | **[ ADT ( Abstract Data Type 앱스트럭 데이타 타입 ) ]** | |  | |  | | **[ Block & Wakeup / Sleep Lock ]** | | **Lock상태가 되었을때 while문에서 공회전을 하고있는것이 아니라**  **커널의 block함수를 호출하여 cpu를 반납하고 pcb를 저장하고 대기큐에 넣는다**  **커널의 Wakeup함수를 호출하여 pcb를 준비큐로 이동시킨다** | |  | | **[ 데드락 ( DeadLock ) / 교착상태 ]** | | **여러 프로세스가 서로의 자원이 필요로할때 더이상 진행이 안되는Block상태**  **그 조건으로는 상호배제,비선점,보유대기,순환대기 가 전부 만족한다** | |  | | **[ 모니터 ( Monitor ) ]** | | **동기화 문제를 해결하기위한 두번째 방법**  **Monitor Class 안에 Lock에대한 정의가 되어있어 안전하게 변수에 접근할수있다**  **Condition Variable이 존재하는대 wait() 과 signal()이 있다**  **wait은 이미 사용중인 프로세스가 있다면 대기큐에 대기시킨다**  **signal은 프로세스를 빠져나오면서 대기큐에 자고있는 프로세스를 불러온다** | |  | | **[ 보유대기 ( Hold and wait ) ]** | | **자원애 접근한 프로세스가 다른 자원을 기다릴때 보유자원을 반환하지않고 계속 가지고있는 상태** | |  | | **[ 순환대기 ( Circular wait 서큘러웨잇 ) ]** | | **서로의 프로세스가 서로의 자원을 필요로하며 사이클을 형성하여 데드락이 되는것** | |  | | **[ 데드락 처리방법 ]** | | **-** | |  | | **[ 가상주소 ( Virtual Address ) / 논리주소 ( Logical Address ) ]** | | **자식프로세스를 생성하면 자식프로세스는 각자의 스택,힙,데이터,코드 영역을 가지고 있고**  **그 주소는 0번지부터 시작한다**  **각자의 자식프로세스의 주소를 가상주소 라고 한다** | |  | | **[ 물리주소 ( Physical Address )]** | | **가상주소가 실제Ram주소에 올라가는 곳이다** | |  | | **[ 주소바인딩 ]** | | **가상주소가 물리주소에 저장하기위해 가상주소를 변환하고 물리주소 어디에 저장할지 결정하는 역활**  **Symbolic Address -> Logical Address -> [Address Binding] -> Physical Address** | |  | | **[]** | | **각 프로세스는 실행을 시키면 자식프로세스 로 생성이되고 주소0번지부터 시작하는 가상주소를 갖게되고**  **이것이 실행이 되려면 물리주소에 바인딩 되어야한다** | |  | | **[ 심볼주소 ( Symbolic Address ) ]** | | **물리메모리 주소를 특정하여 접근하는것이 아니라**  **변수,함수,객체 의 심볼을 통해 메모리주소를 사용한다** | |  | | **[ 주소바인딩 (Address Binding) 3가지 시점 ]** | | **주소바인딩이 결정되는 방법 3가지**  **소스코드 -> (컴파일) -> 실행파일(기계어) -> 실행시작 -> (물리메모리) -> 실행중**  **1) 컴파일타임 ( Compile Time Binding )**  **컴파일시점에 항상특정위치인 0번째 물리적주소가 결정되어 실행파일이 물리적주소가 되는 방식**  **한번 결정된 물리주소는 고정주소로 변경하려면 재컴파일을 해야한다 absolute code 절대코드 라고 한다**  **2) 로드타임 ( Load Time Binding )**  **실행이 될때 비어있는 물리메모리공간에주소가 결정된다**  **비어있는 물리공간에 배치하는 방식으로 relocatable code 리로케터블코드 재배치가능코드 라고 한다**  **3) 익스큐션타임 ( Execution Time Binding ) / 런타임 ( Run Time Binding ) ★**  **프로세스가 실행이 될때 비어있는 물리메모리공간에주소가 결정되어**  **프로그램이 실행이 되는 도중에 물리적메모리가 바뀔수 있는방식**  **Cpu가 주소를 참조할떄 바뀐 주소가 있는지 체크하기위한**  **하드웨어의 지원을 통해 바인딩체크( MMU ) 를 해야한다** | |  | | **[ Cpu의 주소접근방식 ]** | | **Cpu는 실행파일의 논리주소를 읽어서 MMU주소변환을 통해**  **논리주소에 바인딩된 물리주소에있는 값을 읽는다** | |  | | **[ MMU ( Memory Management Unit 메모리 메니저먼트 유닛 ) ]** | | **Limit register 논리프로세스의 최대( 마지막 ) 주소를 저장한다**  **cpu가 요청하는 주소가 Limit register보다 큰경우를 먼저 체크한다**  **논리주소 크기를 넘는 주소에 접근하면 trap하여 운영체제에 제어권을 뺏겨**  **다른프로그램을 변경할수 있는 것을 방지한다**  **Base register에 물리주소의 시작위치를 정장한다**  **주소변환을 할때 가상주소에 Base register를 더해서 물리주소를 구한다**  **가상주소 -> 가상주소 + Base register => 물리메모리 주소** | |  | | **[ 동적로딩 ( Dynamic Loading ) ]** | | **필요한해당 루틴이 실행될때마다 OS라이브러리를 통해 필요한 부분만을 메모리에 올리는 방식** | |  | | **[ 스와핑 ( Swapping ) ]** | | **-**  **원칙상 프로그램을 메모리에서 HDD로 전부통째로 쫒아내는것**  **중기스케줄러 방식에서 cpu우선순위가 낮은거 부터 쫒아낸다**  **쫒아났다가 메모리로 다시 올라올때는 메모리의 빈공간에 올라가게 된다**  **페이징에서의 프로세스의 cpu우선순위가 낮은 부분이 SwapOut되는 것도 Swap이라고 한다**  **이공간을 Backing Storage빽킹스토리지 라고 한다**  **[ transfer time 트랜스퍼타임 ]**  **프로세스의 크기가크면 swap할떄 걸리는 비용 디스크로 옮겨지는 시간** | |  | | **[ 동적링킹 ( Dynamic Lingking ) / DLL ( Dynamic Lingking Library 동적링킹라이브러리 ) ]** | | **라이브러리 함수 사용시 라이브러리 위치를 표시해서 불러와 사용하는것**  **라이브러리 위치를 찾아 메모리에 올려 사용하게된다**  **각기 다른 프로세스들이 동일한 라이브러리를 공유해서 사용할수 있다** | |  | | **[ 라이브러리 ( Library ) ]** | | **누군가 만들어놓은 유용한 함수,메서드,클래스 들의 집합이다** | |  | | **[ 정적링킹 ( Static Lingking ) ]** | | **-**  **#include 와 같은 외부라이브러리를 내프로그램안에 포함시켜 실행파일을 만들게 되는것**  **그렇기때문에 동일한 라이브러리를 포함한 각기다른 프로그램을 실행하면**  **동일한 라이브러리 여러개가 메모리에 포함되어 메모리낭비** | |  | | **[ 연속할당 ( Contiguous allocation 컨티규어스 얼로케이션 ) ]** | | **메모리에 프로세스가 통째로 저장하는것**  **1) 고정분할 방식**  **메모리공간을 미리 나누어 놓은 방식**  **2) 가변분할 방식**  **프로그램이 실행될때 메모리에 차곡차곡 저장하는 방식** | |  | | **[ 불연속할당 ( NonContiguous allocation 넌컨티규어스 얼로케이션 ) ]** | | **프로세스의 주소공간을 잘게나누어 페이지단위로 메모리 여러군대 나누어 저장하는것**  **1) 페이징 기법**  **2) 세그먼테이션 기법**  **3 )페이지 세그먼테이션 기법** | |  | | **[ 외부조각 ( External Fragmentation 익스터널 프래그먼테이션 ) ]** | | **-**  **고정분할 방식에서 메모리공간을 고정해서 분할해놓은공간이 크기가 작아 남게되는 공간** | |  | | **[ 내부조각 ( Internal Fragmentation 인터날 프래그먼테이션 ) ]** | | **고정적으로 분할해높은 공간이 프로그램크기보다 커서 남게되는 공간** | |  | | **[ 홀 ( Hole ) ]** | | **사용가능한 비어있는 메모리공간** | |  | | **[ 컴팩션 ( Compaction ) ]** | | **여러 홀 들을 하나로 합치는과정**  **바인딩을 전부 점검해야하기 때문에 매우비용이 많이 든다** | |  | | **[ 페이징 ]** | | **가상논리주소메모리 프로그램을 페이지단위( 4KB )로 잘게 나누어 놓은 방식이다**  **일부는 메모리에 올라가고 일부는 Backing Storage에 저장한다**  **페이지별로 주소변환을 해줘야하기 때문에 페이지테이블 기법을 통해 주소변환을 하게된다** | |  | | **[ 페이지 프레임 ]** | | **물리적주소공간 메모리를 페이지단위로 잘게 나누어 놓은방식이다** | |  | | **[ 세그먼테이션 ( Segmentation ) ]** | | **-**  **프로그램을 코드,스택,데이터 영역별로 물리메모리에 올리는 방법**  **논리주소 공간을 물리주소로 바꾸는 세그먼트테이블 SegmentTable 이 있다** | |  | | **[ 페이징 기법 ]** | | **페이지테이블은 물리메모리Ram에 저장되있는대 어디에 저장되있는지 시작위치를**  **PTBR ( Page Table Base Register )레지스터가 가르키고있다**  **PTLR ( Page Table Length Register )레지스터가 페이지테이블의 길이를 가지고있다**  **페이지테이블이 램에 있기때문에 물리주소에 접근하기위해서는 메모리에 2번 접근해야한다**  **이것은 일반적인 메모리 접근방식이고**  **성능개선을 위해 주소변환을 위한 TLB캐쉬를 통해**  **일부의 변환된 주소를 캐쉬메모리에 저장해놓고**  **원하는 논리주소에 해당하는 위치를 병렬적으로 검색하고**  **찾았다면 물리메모리에 1번만 접근하게 된다**  **캐쉬에 없다면 캐쉬미스 라하고 페이지테이블 기법으로 주소접근을 하게된다** | |  | | **[ 페이지테이블 ( Page Table ) ]** | | **각각의 프로세스마다 램에 저장된다**  **각각의 논리주소 페이지가 물리주소 페이지프레임 어디에 저장되는지**  **주소변환을 해주는 테이블( 배열 ) 이다**  **Valid 밸리드 비트가 있고 Back Storege에 있는 페이지는 i 물리메모리에 맵핑된 페이지는 v 로 구분한다** | |  | | **[ 2단계 페이지 테이블 ( Two Level Page Table ) ]** | | **사용되지않는 메모리공간을 페이지테이블로 만들지않게하여 페이지테이블의 공간을 줄인다** | |  | | **[ 32비트 주소체계 ]** | | **2의 10승 = KB**  **2의 20승 = MB**  **2의 30승 = GB**  **32비트 주소체계에서 만들거나 사용할 수 있는 최대프로그램의 용량은 4GB이다** | |  | | **[ 주소 ]** | | **메모리주소 한줄은 1바이트 이고 32비트체계에서는 4바이트가 하나의 주소이다** | |  | | **[ 공유페이지 ( Shared Page ) / 재진입 가능코드 ( Re-Entry Code ) / 퓨어코드 ( Pure Code ) ]** | | **1) 읽기전용**  **2) 완전히 동일한 논리주소공간이 같아야한다 / 완전히 같은 프로세스**  **동일한 프로세스 여러개를 실행할경우 동일한 물리주소공간에 저장된다**  **여러개의 프로세스가 전부 올라오는것이 아니다**  **각각이 별도로 가져야하는 데이터페이지들은 각각이 다른 프레임으로 맵핑된다** | |  | | **[ 페이지 세그먼테이션 기법 ]** | | **페이지와 세그먼테이션의 혼합방식**  **세그먼트 하나를 여러개의 페이지로 나누는 기법이다**  **세그먼트 주소 [ 세그먼트 주소 | 세그먼트에서 얼마나 떨어져있는지 나타내는 주소 ]** | |  | | **[ 요청페이지 ( Demand Paging 디멘드 페이징 ) ]** | | **페이징 기법을 사용한다**  **특정 페이지에 디멘드페이지 요청이 있을때 메모리에 할당한다**  **필요한 부분만 메모리에 할당하고 나머지는 디스크(뺵킹스토어)에 보관하는 방식이다**  **프로세스를 최초 실행시 Valid비트가 i였다가 페이지에 올라간 valud는 v로 바뀌게된다**  **두번째 접근시 페이지테이블을 통해 접근할수있다**    **여기서 Swap영역에 들어있는 i밸리드 페이지를 로드하는일을 I/O 작업이기때문에**  **운영체제가 인터럽트를 통해 처리하게된다** | |  | | **[ 페이지폴트 ( Page Fault ) ]** | | **cpu가 주소변환을 요청한 페이지가 물리적메모리에 없는경우**  **자동적으로 trap이 걸려 cpu의 제어권은 운영체제로 넘어간다** | |  | | **[ 트랩 ( trap )]** | | **소프트웨어 코드상의 인터럽트 상황** | |  | | **[]** | | **-**  **물리적인 메모리의 주소변환은 운영체제가 관여하지 않는다** | |  | | **[ 페이지 리플레이스먼트 ( Page Replacement ) ]** | | **메모리에 할당된 페이지를 빽킹스토어로 내쫒는일** | |  | | **[ 옵티멀 알고리즘 ( Optimal Algorithm ) / 오프라인 알고리즘 ( Offline Algorithm ) ]** | | **미래를 예측하여 마지막에 참조될 메모리를 빽킹스토어로 쫒아낸다** | |  | | **[ LRU ( Least Recently Used 레스트 리센틀리 유즈드 ) 알고리즘 ]** | | **리플레이스먼트 알고리즘1 개념적**  **페이지폴드 구간에서 가장 최근에 참조되지않았던 메모리를 빽킹스토어로 쫒아낸다** | |  | | **[ LFU ( Least Frequently Used 리스트 프리퀀틀리 유즈드 ) ]** | | **리플레이스먼트 알고리즘2 개념적**  **참조횟수가 가장 적은 페이지를 뺵킹스토어로 쫒아낸다** | |  | | **[ 클락 알고리즘 ( CLock Algorithm ) ]** | | **디멘드페이징 기법에서 사용되는 알고리즘 이다**  **페이지가 참조될때마다 cpu는 래퍼런스비트를 1로 세팅하고 운영체제는 주기적으로**  **래퍼런스비트를 0으로 클리어시킨다**  **래퍼런스비트가 0인걸 빽킹스토어로 쫒아낸다** | |  | | **[ 모디파이드 비트 ( Modified Bit ) ]** | | **페이지 참조시 쓰기 일경우 1 읽기일경우 0으로 세팅한다** | |  | | **[ 인스트럭션 ]** | | **명령어** | |  | | **[ 쓰레싱 ( Thrashing ) ]** | | **프로세스가 필요로하는 최소한의 페이지수가 적어서 폴트현상이 많이 일어나는것** | |  | | **[ 워킹셋 ( Working Set ) ]** | | **프로세스가 실행되기위해선 어느정도의 최소한의 공간이 있어여한다**  **함수,반복문 처럼 특정 기간에 특정한 부분을 비번히 집중적으로 참조하는것** | |  | | **[ 로컬리티셋 ( Locality Set ) ]** | | **함수,반복문 처럼 특정 기간에 특정한 부분을 비번히 집중적으로 참조하는 집합** | |  | | **[ 서스펜드 ( Suspend ) ]** | | **프로세스의 중단** | |  | | **[ 논리디스크 ( Logincal Disk ) / 파티션 ( Partition ) ]** | | **운영체제가 바라보는 디스크는 논리디스크이고 파티션이라고도 한다**  **하나의 물리디스크를 여러 논리디스크로 나눌수있고 각각 파일시스템을 갖게된다** | |  | | **[ 파일오픈 ( Open() ) ]** | | **파일의 메타데이터를 메모리에 올리는 함수이다** | |  | | **[ 파일 ]** | | **하드디스크에 저장되는 기본 단위이다** | |  | | **[ 메타데이터 ]** | | **파일을 관리하기위한 정보들**  **파일이름 ,유형 ,저장된위치 ,파일사이즈 ,접근권한 ,생성수정시간 ,소유자 등등** | |  | | **[ 파일시스템 ( File System ) ]** | | **운영체제에서 파일을 관리하는 부분 메타데이터,디렉토리정보 ,파일보호 등** | |  | | **[ 디렉토리 ( Directory ) ]** | | **하나의 파일인데 디렉토리 아래에 존재하는 파일들의 메타데이터를 가지는 파일이다** | |  | | **[ 디스크용도 ]** | | **파일시스탬 용도 또는 스왑영역 용도로 사용될수있다** | |  | | **[ 파일읽기 ( Read ) ]** | | **커널영역의 버퍼공간에 메타데이터를 캐싱하여 사용하게된다** | |  | | **[ 루트 ( / ) ]** | | **각각 나뉜 파티션이** | |  | | **[]** | | **-** | |  | | **[]** | | **-** | |  | | **[]** | | **-** | |  | | **[]** | | **-** | |  | | **[]** | | **-** | |  | | **[]** | | **-** | |  | | **[]** | | **-** | |  | | **[]** | | **-** | |  | | **[]** | | **-** | |  | | **[]** | | **-** | |  | | **[]** | | **-** | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | | | |