REPORT

Operating System

* MFQ scheduler & Page allocating algorithm

20133950 공찬형

20132862 김현빈

20143606 이혜성

Index

1. *1-(a) RR 스케줄러 관련 분석*
2. *1-(b) MFQ 스케줄러 구현*
3. *2-(a) 메모리/페이지 할당 관련 분석*
4. *2-(b) 메모리/페이지 할당 알고리즘 구현*
5. *추가/수정 코드 일람표*
6. 1-(a) RR 스케줄러 관련 분석

1) Pintos의 타이머 인터럽트는 1초에 100회 발생한다. 준비 큐에 있다가 Dispatcher에 의해 실행이 시작된 스레드는 다른 스레드가 실행되기 전까지 몇 초(또는 타이머 틱) 동안 실행되는가?

->4틱 만큼 실행된다. (RR TIME\_SLICE = 4)

2) RR 스케줄러는 어떤 자료구조를 이용하여 준비 큐에 머무르고 있는 스레드를 관리하는가?

->List 구조체를 사용한다.

3) thread.c 소스 코드에서 어떤 함수들이 질문 2)에서 언급된 자료구조에 접근하는가?

->thread\_init(), thread\_unblock(), thread\_yield(), next\_thread\_to\_run()

4) RR 스케줄러는 thread.g에 정의된 5가지의 스레드 우선순위를 사용하는가?

->RR 스케줄러는 우선순위를 사용하지 않는다. 생성된 thread는 모두 기본 값을 가지며, 같은 리스트에서 관리되기 때문이다.

1-(b) MFQ 스케줄러 구현

기본적으로 PintOS 상에 구현된 Round-robin 스케줄링을 기반으로, MFQ 스케줄러를 구현하기 위하여, 다음과 같은 조건을 필요로 합니다.

1. 우선순위(0~4)에 따른 개별적인 준비 상태의 스레드 큐 관리.

2. 1. 의 큐를 기반으로 한 우선순위 스케줄링.

3. 우선순위 큐에서 발생하는 기아상태를 방지하기 위한 Aging 정책

조건 1 을 구현하기 위하여, 총 5개의 준비 스레드를 위한 리스트를 선언하였습니다. (thread.c : 28) 각 리스트는 우선순위를 기반하여 Indexing 되어 있습니다.

조건 1로 인한 여러 개의 리스트를 우선순위를 기반으로 하여 스케줄링 하기 위하여, schedule() 함수 내에서 호출되는 next\_thread\_to\_run() 함수를 수정하였습니다. 기존 Round-robin 방식에서는, 우선순위를 염두하지 않고, 하나의 큐로 관리하였기 때문에, MFQ 스케줄러를 구현하기 위해서는 우선순위가 높은 리스트를 우선적으로 검사하여, 만약 준비 상태로 리스트에 있는 스레드가 있다면, 그 스레드를 먼저 실행하도록 해야 합니다. 따라서, 가장 우선순위가 높은 스레드가 있는 리스트(priority = 4) 부터 낮은 우선순위 리스트(priority = 0)를 탐색하면서 우선순위가 높은 스레드의 주소를 반환하여 schedule() 함수에서 다음에 실행할 스레드를 MFQ 스케줄링 정책에 맞게 지정할 수 있게 하였습니다.

또한, MFQ 스케줄링 방식에서는, 스레드에 할당된 Time slice가 우선순위가 높을수록 작게 할당됩니다. 이를 구현하기 위해서, thread\_tick() 함수에서 각 스레드에 할당된 slice를 모두 사용하였는지를 확인하는 구문에서, 우선순위를 고려하여 판단하도록 하였고, 또한 할당된 slice를 모두 사용한 스레드는 강제로 cpu를 양보하며, 그 우선순위가 한단계 하강하도록 구현하였습니다. 이에 수반되는 우선순위 하강과, 준비 리스트의 이동 방법은 slice를 모두 사용하였을 때 호출되는 함수인 intr\_yield\_on\_return() 함수에서 실행하는 thread\_yield() 함수를 수정하여 구현하였습니다. Intr\_yield\_on\_return() 함수가 실행되기 전, 할당된 slice를 모두 사용한 스레드는 그 우선순위가 한단계 하강하며 (최하위 우선순위가 아니라면), thread\_yield() 함수에서는 그 우선순위에 해당하는 준비 리스트에 그 스레드를 넣고, schedule 함수를 통하여 다음 스레드가 진행하도록 하였습니다.

조건 3을 위한 Aging 정책을 구현하기 위하여, thread.h 내부에 age 변수를 추가하였습니다. 이 age변수는 최초 스레드가 생성될 때, 또는 스레드가 실행되었을 때 0으로 초기화됩니다. 또한, 현재 스레드의 thread\_tick() 함수를 호출할 때마다, 현재 실행되고 있는 스레드의 우선순위보다 낮은 리스트에 존재하는 모든 스레드의 age를 1씩 증가시킵니다. 이는 추가한 코드의 thread\_aging() 함수를 호출함으로써 실행됩니다. 이 함수에서, 만약 스레드의 age가 20 이상이 되었을 경우, 그 스레드를 현재 리스트에서 제거한 후, 한단계 높은 우선순위 리스트에 넣은 후, 스레드의 우선순위를 증가시켜, 기아상태가 발생하지 않도록 합니다. 이 과정에서, 스레드가 종료되거나, 블락되거나 하는 인터럽트를 발생시키지 않기 위하여, 이 과정동안 인터럽트를 금지함으로써, 야기될 수 있는 오류를 방지하였습니다.

수정, 및 추가 코드는 문서 하단 코드 일람표를 참고하시기 바랍니다.

1. 2-(a) 페이지/메모리 할당 관련 분석

1) Pintos 메모리 시스템의 기본 페이지 크기는 몇 바이트인가?

-> 4096 byte, vaddr.h.에 의해 12개의 비트를 offset 비트와 20 pageNum을 사용한다. 따라서 4096 byte이다.

2) malloc.c에서 기본 페이지보다 작은 크기의 메모리 영역을 할당하기 위해 어떤 방법이 사용되고 있는가?

->SIZE 바이트 요청을 만족시키는 가장 작은 디스크립터를 찾는다.

3) palloc.c와 bitmap.c 내의 어떤 함수에 페이지 할당 알고리즘이 구현되어 있는가?

-> palloc\_get\_multiple(), bitmap\_scan\_and\_flip(), bitmap\_scan()

1. 2-(b) 페이지/메모리 할당 알고리즘 구현

PintOS상에 구현된, 최초적합(First Fit) 탐색 코드 분석을 기반으로, Next-fit, Best-fit, Buddy-system을 구현하였습니다.

1) Next-fit을 통한 프레임 할당을 구현하기 위하여, bitmap 내부에 다음 페이지 할당 위치를 저장하기 위한 index를 추가하였습니다. 이 index는 바로 이전에 진행된 할당에서 할당한 프레임들의 다음 프레임을 가리키게 되며, 다음 할당에서는 이 위치를 시작으로 탐색을 진행하게 됩니다. 만약 탐색이 마지막 비트까지 진행되었다면, 비트맵의 첫번째 주소를 시작으로 최초 탐색을 시작한 위치까지 탐색합니다.

2) Best-fit을 통한 프레임 할당을 구현하기 위하여, 가장 작으며, 요청된 프레임 수보다 크거나 같은 빈 공간을 탐색합니다. 비트맵의 시작부터, 연속된 0 의 개수를 저장하여 기록하며 계속 진행합니다. 만약 1이 탐색되면, 현재까지 저장된 0의 개수와 지금까지 찾은 최소 공간의 크기를 비교하여 작은 공간의 시작주소를 저장합니다. 이후 탐색이 종료되면, 현재 저장된 최소 공간의 시작주소를 반환합니다. 모든 비트맵 공간이 비어 있거나, 유휴 공간이 단일 상태로 존재할 경우를 위하여, 최소 공간의 크기를 가지는 시작주소는, 비트맵 상 첫번째 주소로 초기화하고, 최초 최소 공간의 크기 또한 전체 공간의 크기로 초기화 하였습니다.

3) Buddy-system을 통한 프레임 할당을 구현하기 위하여, 요청된 크기에 가장 근접하고, 그것보다 큰 2의 n승 값을 계산합니다. Buddy-system 에서는 실제 요청된 크기보다 크거나 같은 프레임 수를 할당합니다. 할당할 프레임의 시작 주소를 찾기 위하여, 실제 할당할 크기만큼 전체 프레임을 쪼갠 후, 탐색한 뒤, 가능한 첫번째 주소를 반환합니다. 이 방식으로, 기존 Buddy-system에서 구현되는 tree 구조를 사용하지 않을 수 있습니다. 또한, 이 스레드가 어떤 경로에 있든, 사용하고 있던 프레임을 반환해야 할 경우 또한, 실제 스레드가 사용하고 있는 프레임 수 보다 크거나 같은 크기의 프레임을 반환해야 합니다. 따라서 스레드의 프레임 해제 에서도, 할당된 프레임 만큼 할당을 해제해야 합니다.

수정, 및 추가 코드는 문서 하단 코드 일람표를 참고하시기 바랍니다.

1. 수정/추가 코드 일람표

<MFQ Scheduler>

*Thread.h*

|  |  |  |
| --- | --- | --- |
| Function | Line | Desciption |
| Struct thread | 110 | 스레드의 Aging 구현을 위하여, 스레드 구조체 내부에 int형 age 변수 추가. |

*Thread.c*

|  |  |  |
| --- | --- | --- |
| Function | Line | Desciption |
| Static struct List fb\_ready\_list[5] | 28 | 우선순위에 따른 큐 분할을 위하여, 총 5개의 리스트 선언. |
| MFQ\_TIME\_SLICE | 64 | 우선순위에 따른 시분할 차등할당을 위하여, 최소 우선순위에 부여되는 할당값 변경. |
| Void thread\_init | 104 | 여러 개의 큐를 초기화. |
| Void thread\_tick | 157 | Aging 구현과, 우선순위 구현을 위하여, 현재 스레드가 동작중일 때 마다 age를 0으로 초기화하고, thread\_aging 함수를 통하여, 현재 스레드보다 낮은 우선순위 리스트에 대기하고 있는 스레드의 age를 1씩 증가시킨다. 이후, 해당 우선순위에 부여될 시분할을 모두 소모하였는지 검사한 후, 모두 소모하였다면, 현재 스레드의 우선순위를 한단계 감소시킨다. |
| Void thread\_unblock | 268 | 스레드가 블락 상태에서 해제될 경우, 그 스레드의 우선순위에 맞는 리스트에 연결시키도록 수정하였다. |
| Void thread\_yield | 395 | 스레드가 시분할을 모두 소모하거나, 선점 당하였을 경우, 그 스레드의 우선순위에 맞는 (시분할 전소로 인한 선점은 이미 thread\_tick에서 우선순위를 감소시켰다.) 리스트에 연결시키도록 수정하였다. |
| Void init\_thread | 568 | 최초 스레드가 생성 될 때, age를 0으로 초기화하는 부분을 추가하였다. |
| Struct Thread next\_thread\_to\_run | 597 | Schedule 함수에서 다음 실행될 스레드를 가져오는 함수이다. 이 함수의 반환값으로, 우선순위가 높은 리스트부터 우선탐색하여, 다음 실행할 스레드를 반환할수 있도록 수정하였다. |
| Void thread\_aging | 694 | Thread\_tick에서 실행되는 thread\_aging 함수는 현재 이 함수를 호출한 스레드의 우선순위보다 낮은 리스트에 대기하고있는 모든 스레드의 age값을 1씩 증가시킨다. 이때, 스레드의 age값이 20이상이 되었다면, 그 스레드의 우선순위가 한단계 증가하고(가능하다면), 그 우선순위에 맞는 리스트로 스레드를 이동시킨다. |
| Int thread\_get\_age | 737 | 현재 스레드의 age값을 반환한다. |

<Algorithm for various page allocation>

*Palloc.c*

|  |  |  |
| --- | --- | --- |
| Function | Line | Desciption |
| Enum palloc\_allocator pallocator | 45 | kernel에서 받아온 모드값을 받아오기 위하여, 기본 값을 해제하였다. |
| Void \* palloc\_get\_multiple | 75 | 스레드에 페이지를 할당 할 때, 받아온 모드값을 통하여 각각 다른 알고리즘을 가진 할당방식을 사용하기 위하여, 조건문을 통해 분기하도록 수정하였다. |
| Void palloc\_free\_multiple | 149 | 다른 할당방식과 달리, buddy-system에서는 요청된 크기보다 크거나 같은 프레임을 할당하기 때문에, 그 할당을 해제할 때 또한 그와 같은 양을 해제하여야 한다. 이를 위하여, 만약 buddy-system 모드일 경우, 조건문을 통하여 다른 해제방식이 구현된 함수(bitmap\_set\_multiple\_buddy : bitmap.c)를 호출함으로써 할당된 만큼을 해제할 수 있도록 구현하였다. |

*Bitmap.h*

|  |  |  |
| --- | --- | --- |
| Function | Line | Desciption |
| Declare | 29  41  42  43  45  46  47 | Void bitmap\_set\_multiple\_buddy  size\_t bitmap\_scan\_nf  size\_t bitmap\_scan\_bf  size\_t bitmap\_scan\_buddy  size\_t bitmap\_scan\_and\_flip\_nf  size\_t bitmap\_scan\_and\_flip\_bf  size\_t bitmap\_scan\_and\_flip\_buddy |

*Bitmap.c*

|  |  |  |
| --- | --- | --- |
| Function | Line | Desciption |
| Struct bitmap | 35 | Next-fit 알고리즘을 위하여, bitmap 상에 다음 할당시에 탐색을 시작할 주소를 저장하도록 하였다. |
| Struct bitmap \* Bitmap\_create | 91 | Bitmap 생성시 Next-fit을 위한 변수 cur\_index를 0으로 초기화시킨다. |
| Void bitmap\_set\_multiple\_buddy | 237 | 프레임을 할당받은 스레드는 요청한 프레임의 수만 알고 있을뿐, 자신에게 얼마나 할당됬는지 알수 없기 때문에, 요청한 프레임 수를 토대로 할당된 프레임을 연산한다. 그 연산 결과(2^n) 만큼 비트맵상에 set된 비트를 해제한다. |
| Size\_t bitmap\_scan | 328 | -수정사항 없음. (First-fit 알고리즘) |
| Size\_t bitmap\_scan\_nf | 349 | 현재 비트맵에 포함된 탐색을 시작할 주소를 바탕으로 탐색을 시작한다. 만약 비트맵의 끝까지 탐색이 진행되었지만, 적절한 공간을 찾지 못한 경우 비트맵의 시작부터 비트맵에 포함된 탐색을 시작한 주소 이전까지 탐색을 재개한다. 성공적으로 할당 위치를 탐색하였을 경우, 비트맵에 지정된 탐색 시작 주소(cur\_index)를 할당한 주소와 요청된 크기를 더하여 저장하여, 다음 탐색 위치를 지정할 수 있도록 구현하였다. |
| Size\_t bitmap\_scan\_bf | 387 | 최초 512 크기와 비트맵의 시작주소를 최소 크기 공간의 시작 주소값과, 크기로 초기화한다. 전체 비트맵을 순회하며 빈 공간을 찾기 시작하다가, 그 크기(size)가 결정될 시(set된 비트를 만날시) 현재 저장된 최소 크기와 비교하여, 현재 찾은 공간의 크기가 요청된 공간의 크기보다 크거나 같고, 비교 대상에 비해 작다면, 직전에 찾은 공간의 시작 주소와 크기를 현재까지 찾은 최소 공간(min\_size, min\_index)에 저장하고, 다음 빈 공간을 탐색한다. 탐색이 종료되면 현재 저장된 최소 크기 공간의 시작 주소를 반환한다. |
| Size\_t bitmap\_scan\_buddy | 435 | 요청된 크기에 가장 맞는 할당 크기를 구한다.(Buddy-system 규칙에 의거) 이후 그 크기만큼 비트맵을 뛰어넘으며 탐색을 시작하여, 최초 탐색된 공간의 시작 주소를 반환한다. 이 과정에서 할당 공간의 크기(2^n)에 기반해 탐색을 시작하면, 그 크기를 가지는 leaf\_node 만을 탐색하기 때문에, 일반적인 tree의 구조를 필요로 하지 않는다. 이 할당 방식은, 스레드에게 어떠한 정보도 주지 않고 요청된 크기보다 크거나 같은 공간을 논리적으로 할당하기 때문에(bitmap 상에서), buddy-system을 사용하여 시스템이 로드된 경우 특별한 해제함수(bitmap\_set\_multiple\_buddy)를 사용하여야만 한다. |
| Size\_t bitmap\_scan\_and\_flip | 469 | -수정사항 없음. (First-fit 알고리즘) |
| Size\_t  Bitmap\_scan\_and\_flip\_nf | 479 | 위의 함수에서 bitmap\_scan 대신 bitmap\_scan\_nf를 호출하여 시작주소를 설정한다. |
| Size\_t  Bitmap\_scan\_and\_flip\_bf | 489 | 위의 함수에서 bitmap\_scan 대신 bitmap\_scan\_bf를 호출하여 시작주소를 설정한다. |
| Size\_t  Bitmap\_scan\_and\_flip\_buddy | 500 | 위의 함수에서 bitmap\_scan 대신 bitmap\_scan\_buddy를 호출하여 시작주소를 설정한 뒤, 그 할당 크기를 재연산하여 논리적으로 (bitmap 상으로) 그 크기만큼 할당해준다. |