|  |
| --- |
| Operating Systems  Spring, 2022  School of Software, CAU  **Project #2**  **- Page Allocation and Task Scheduling –**  **[프로젝트 보고서]**  학번: 20182345  이름: 이 건 호 |

**1. 서론**

본 보고서에서는 페이지 할당자와 스케줄러의 구현 방법과 결과에 대해 기술한다.

**1.1 Page Allocation**

Buddy System의 페이지 할당은 먼저 메모리 크기를 2의 n제곱으로 만들고, 할당을 요청한 메모리크기에 적합한 2의n제곱크기의 페이지공간을 할당해줍니다. 그렇기때문에 보통은 요청한 페이지크기보단 큰 페이지가 할당되게 됩니다. (2의 n제곱을 요청했다면 그 크기만큼 할당됨.) 그리고 firt-fit을 변형해야 했기 때문에 처음부터 검사해서 해당 크기 메모리공간이 들어갈 공간을 찾자마자 들어가는 것이 아닌 앞에서 구한 2의n제곱크기의 페이지공간만큼 묶음으로 검사하면서 할당될 자리를 찾고 할당하는 식으로 buddy system을 구현하였습니다. Release는 페이지 주소를 받으면 해당 페이지가 메모리상에서 실제로 차지하고 있는 페이지의 개수갯 구하여서 해제해 주는 방식으로 구현하였습니다. 자세한 방법은 본론에서 서술하도록 하겠습니다.

**2. 본론**

본론에서는 구현 방법에 대해 상세히 기술한다.

**1.1 Page Allocation**

Buddy System과 똑 같은 방식으로 페이지를 할당해주고 free해주기 위해서 palloc.c함수에서 먼저, user와 kernel에 할당된 페이지의 크기를 2의 배수인 256개로 각각 설정해 주었습니다. (2의 승수여야 편한 코딩이 가능하기 때문.)

텍스트, 모니터, 화면, 검은색이(가) 표시된 사진

자동 생성된 설명 (page\_cnt를 257로 해줘야 256개의 페이지를 할당해주기 때문에 위와같이 설정해줬습니다.)

그리고 페이지를 할당한 방법은 palloc\_get\_multiple함수를 수정하여 구현하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

먼저, 요청받은 페이지의 크기인 page\_cnt 이상이 되는 [2의 n승]의 수 중 첫번째로 이상이 되는 수인 size\_val를 찾습니다. 이 수 size\_val는 이제 실제로 할당되는 페이지의 수가 되게 됩니다.

텍스트, 스크린샷, 화면, 은색이(가) 표시된 사진

자동 생성된 설명그리고 메모리풀을 처음부터 size\_val개씩 검색해가면서 해당 범위의 메모리 값이 전부 false인 index를 찾습니다. 전부 false여야만 해당 size\_val크기의 메모리풀에 페이지를 실제로 할당해 줄 수 있다는 의미이기 때문입니다. 예를들면 메모리풀의 인덱스 0~size\_val-1까지 검색하여서 전부 fasle인지 검사하고, 만약 true가 하나라도 이 범위에 있다면 다른 페이지가 할당된것이기 때문에 이제 size\_val~2\*size\_val-1까지 검사하여 전부 false인지 검사하고…이런식으로 전부 false일 때 까지 그러나 메모리 범위를 넘어서지 않는 범위까지 검사하다가, 만약에 전부 false가 맞다면 해당 범위에 페이지를 할당해주고 그 범위의 값을 전부 true로 바꿔준 후 palloc\_get\_multiple함수는 할당된 메모리공간의 실제 시작 주소를 return해 주게 되는 식으로 Buddy System Page Allocation을 구현하였습니다.

그리고 free를 해주는 부분은 처음엔 페이지의 크기마다 어느 인덱스가 사용중인지를 리스트를 사용해서 저장해볼까 하다가 비효율적이라고 생각하여 방법을 바꾸었습니다.

페이지의 free는 palloc\_free\_page를 사용해서 행해지게 됩니다. 매개변수로 해제하고자 하는 페이지의 시작 주소를 주면 이 함수 속에서 해당 페이지가 user풀인지 kernel풀인지, 해당 페이지의 bitmap상에서 시작 index가 몇인지, 그리고 실제로 해제해줘야하는 page의 크기는 얼만큼 되는지를계산하여 palloc\_free\_multiple함수를 부르는 식으로 프로그램을 구성하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명 User,kernel 풀을 구분하는 것은 어렵지않았고 bitmap상에서의 index를 구하는 것도 단순히 ‘(해당 페이지의 주소-해당 메모리의 base주소)/페이지사이즈’ 를 사용하면 쉽게 구할 수 있었기 때문에 문제가 아니었습니다. 그렇기 때문에 여기서 핵심은 과연 해당 페이지는 메모리상에서 실제로 몇 페이지 만큼을 차지하고 있는가 였습니다.

다음과 같은 상황이 문제였습니다. (사진 참조)



위 사진을 보면 먼저 D에서 페이지 4칸을 할당 받았습니다. 그리고 E는 페이지 2칸을 할당 받았습니다. 이때 release D를 했을 때 오직 D만 release되고 그 외 나머지는 영향을 어떻게 하면 안 받게 하고 free해줄수 있을까 하다가 한 가지 방법을 생각해 냈습니다.

그 방법은 일단, 페이지를 free해줄 때 쓰는 함수는 palloc\_free\_page함수입니다. 이 함수는 매개변수로 page의 시작 주소를 받습니다. 저는 이 시작 주소를 사용해서 커널풀인지 유저풀인지에 따른 pool을 얻어내고, 그 pool에 해당하는 bitmap을 구했습니다. 그리고 bitmap\_all함수를 사용합니다. 이 함수의 기능은 매개변수로 bitmap과 해당 비트맵에서 페이지의 시작 인덱스와 몇 개를 검사할건지 이렇게 3가지를 인자로 받습니다. 그리고 해당 bitmap에서 입력받은 페이지 인덱스부터 시작해서 입력받은 숫자만큼 전부 true가 세팅되어있다면 true를 리턴해줍니다. 저는 이 기능을 이용하였는데, Bitmap은 앞서서 이미 구했고, 인덱스 또한 (해당 페이지의 주소-해당 메모리의 base주소)/페이지사이즈’사용하여 이미 구했습니다. 검사할 숫자를 변수 i로 두고 이것을 늘려가며 검사하는 식으로 해당 페이지가 실제로 보유한 페이지의 크기를 구했습니다. 먼저 i=1로 시작합니다. bitmap\_all (pool->used\_map, pageIndex, i)를 활용해 검사합니다. 만약에 true라면 i를 2배 늘려서 다시 검사합니다. 이때 2배씩 곱해주는 이유는, 페이지의 할당 단위가 2의 n제곱이기 때문에 검사해주는 단위도 마찬가지여야 하기 때문입니다. 언제까지 반복을 하게 되냐면 페이지의 인덱스가 i의 배수가 아니거나, 검사 범위중 false가 하나라도 있을때까지 반복을 하게 됩니다. 여기서 중요한점은 페이지의 인덱스가 i의 배수가 아닐 때 반복을 멈춘다는 것입니다.

이것은 왜 그러냐면



이러한 상황이라고 가정하였을 때 만약 B를 release하고자 하였는데 B의 시작 인덱스부터 i를 2배씩 늘려가며 검사하면 일단은 B까지는 반복문이 정상적으로 도달할 것입니다. 그러나 여기서 i를 2배를 했을 때 바로 인접한 D가 B와 똑같은 크기를 지녔기 때문에 시작 인덱스~i까지는 전부 true여서 이때에도 반복문이 정상으로 돌아가서 실제 B가 갖는 페이지의 크기보다도 2배나 큰 메모리를 갖는다고 인식을 해버리게 됩니다. 그러면 이를 free해줬을 때 B만 release해주고 싶은데 D까지 release해버리는 문제가 발생하게 됩니다. 이러한 상황을 막으려면 매 검사마다 페이지의 시작 인덱스가 i의 배수인지 확인하면 되지 않을까 생각하여서 적용하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

while문을 탈출한 순간은 이미 i가 정상값보다 2배 큰 상태이기 때문에 한번 2로 나눠줍니다. 그리고 이 i는 release를 요청받은 것의 페이지의 갯수가 됩니다.



그리고 최종적으로 palloc\_free\_multiple함수에 해당 페이지의 시작 주소와 그 페이지가 실제로 몇 페이지를 갖는지를 매개변수로 넘겨주면 release구현이 완성됩니다.

**1.2 Page Allocation Test**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

Pa.c에 구현한 테스트 코드는 위와 같습니다. 먼저 5개의 페이지를 요구하는 A 할당을 요청하고, 9개의 페이지를 요구하는 B,C 할당을 요청하고, 4개의 페이지를 요구하는 D의 페이지를 요청하고, 2개의 페이지를 요구하는 E를 요청하고, 18개의 페이지를 요구하는 F를 요청하였습니다. 그 후에 D를 release하고 C를 release하고 B를 release해보았습니다. 그리고 마지막으로 적절한 빈 공간에 잘 들어가는지를 보기 위해 18개의 page를 요구하는 G를 요청하였습니다.

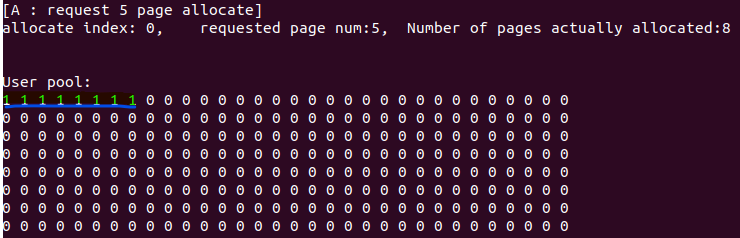
텍스트이(가) 표시된 사진

자동 생성된 설명

Palloc\_get\_status에 구현한 내용은 위와 같습니다.

32개씩 8줄로 메모리에서 페이지가 어떤 상태인지를 표시해줍니다. 1은 사용중, 0은 사용가능이라는 의미입니다. 이 외에도 페이지할당을 요청하였을 때 할당된 페이지의 시작 인덱스, 요청된 페이지의 개수, 실제로 할당된 페이지의 개수, release된 페이지의 인덱스도 추가로 표시가능하게 palloc.c에 구현하였습니다.

테스트 내용은 아래와 같습니다.

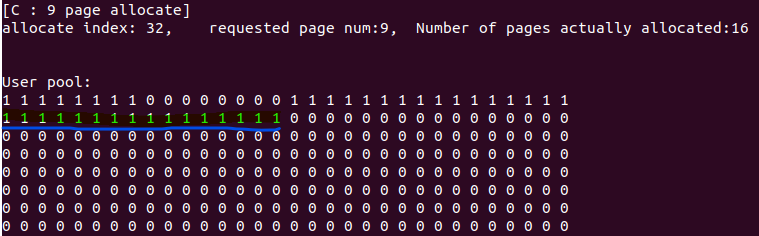


먼저 A를 할당해주는데 5개의 페이지를 요청하였는데 buddy system이므로 8개를 첫번째 빈 공간에 할당해 준준 것 확인 가능합니다.

텍스트, 전자기기, 키보드이(가) 표시된 사진

자동 생성된 설명

그 다음으로는 B를 할당해 주었는데, 9개의 페이지를 요청하였으므로 실제로는 16개의 페이지를 할당해줍니다. 마찬가지로 buddy system의 규칙에 맞춰서 A의 바로 뒤가 아닌 buddy system에 맞는 위치에 페이지가 할당된 모습을 볼 수 있습니다.

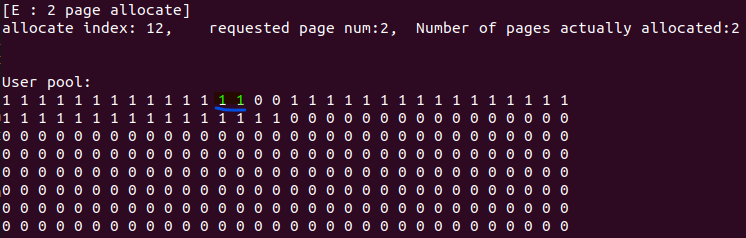


그 다음은 C를 할당해주었습니다. 요청한 페이지는 9개지만 실제로는 16개를 할당해주었습니다. 앞과 동일한 구조이기 때문에 설명은 생략하겠습니다.

텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명

그 다음은 페이지 4개짜리 D를 할당해주었습니다. 2의 배수이기 때문에 요청한 값 만큼 페이지가 할당되었습니다.



그 다음은 페이지 2개짜리 E를 할당해주었습니다. 마찬가지로 정확한 위치에 할당되는 것을 볼 수 있습니다.

텍스트, 전자기기, 키보드이(가) 표시된 사진

자동 생성된 설명

그리고 18페이지를 요청한 F는 36크기의 페이지를 할당해 주게 됩니다.

여기까지가 할당에 관련한 부분이고 이제 release에 대한 결과를 보여드리겠습니다.

텍스트, 전자기기, 키보드이(가) 표시된 사진

자동 생성된 설명

Release D를 하였을 때 정확히 D부분만 release된 것을 알 수 있습니다.

텍스트, 전자기기, 키보드이(가) 표시된 사진

자동 생성된 설명

Release C가 정상적으로 된 걸 볼 수 있습니다.

텍스트, 전자기기, 키보드이(가) 표시된 사진

자동 생성된 설명

Release B가 정상적으로 된 걸 볼 수 있습니다.

텍스트, 전자기기, 키보드, 타이프라이터이(가) 표시된 사진

자동 생성된 설명

페이지 18개를 요구하는 G를 할당해주었는데 실제로는 32개가 할당이 되었고 앞쪽에 연속된 빈 공간이 있음에도 불구하고 Buddy System의 규칙에 따라서 적합한 위치에 할당된 것을 확인할 수 있습니다.

**3. 결론**

사실 아이디어 및 로직을 생각해 내는 것 자체는 복잡하지 않았으나 핀토스의 돌아가는 구조를 파악하는데 대부분의 시간을 쓴 것 같습니다. 하드웨어단을 상세히 컨트롤 해 볼 수 있었던 좋은 기회였던 것 같습니다.