# Project. 2
# Multi-Process Execution with Virtual Memory



(Free day 하루 사용)
32143153 이대훈
32140301 권태완

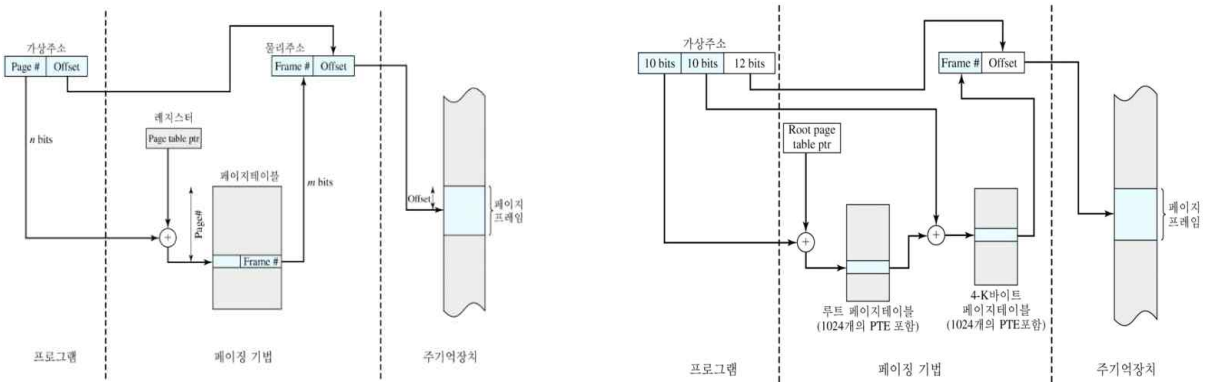
# Ⅰ. Introduction

가상메모리는 RAM을 관리하는 방법의 하나로, 각 프로그램에 실제 메모리 주소가 아닌 가상의 메모리 주소(virtual address)를 주는 방식으로 실제로 우리가 프로그램에서 보여 지는 주소들은 가상의 메모리 주소들입니다. 프로그램의 용량이 커지는 것에 비해 메모리 용량을 늘리는 것은 쉽지 않기 때문에 이 한정된 메모리의 크기를 해결하기 위해 가상 메모리를 사용합니다. 이 가상 메모리는 실제 메모리보다 훨씬 큰 용량을 가질 수 있기 때문에 프로그램 사용자들은 실제 메모리 용량이 아닌 가상 메모리를 활용하여 프로그램을 사용할 수가 있습니다. 그리고 큰 용량의 가상 메모리는 모두 사용 되는 것이 아니라 실행되어야 할 부분만 실제 메모리로 옮겨지면 되는 것입니다.
여기서 가상 메모리의 주소를 실제 메모리의 주소로 옮겨주는 역할을 하는 것을 MMU(Memory Management Unit)라고 하는데 MMU는 메모리를 효율적으로 관리 해주는 하드웨어로 CPU 내에서 프로그램에 사용되는 가상주소를 물리적 주소로 변환해주는 일을 합니다. MMU가 메모리를 변환해주는 방식으로 paging이란 방식이 있는데 가상메모리상의 주소공간을 일정한 크기의 페이지로 분할해주는 방식입니다. 그리고 이 페이지들을 페이지 테이블 안에 알맞게 mapping 시켜 줌으로써 가상메모리를 사용할 수 있게 됩니다.

# Ⅱ. Motivation

이전 프로젝트에서 제작한 Multi-process execution을 각 프로세스에 가상메모리를 추가하고 이를 페이징 방식을 사용하여 실행될 수 있도록 하였습니다. 하나의 페이지테이블을 사용한 1-level 방식과 두 개의 페이지 테이블(directory, table)을 사용한 2-level 방식이 있습니다.

* 1-level : 각 user 프로세스 마다 4MB 크기의 페이지 테이블을 갖는 방식으로 하나의 user 프로세스마다 하나의 테이블을 갖게 됩니다.
* 2-level : 1-level을 사용했을 때 user 수에 맞게 4MB 만큼의 페이지 테이블을 갖고 있어야 하는데 이는 불필요한 메모리의 할당으로 user 수가 많아지면 메모리 낭비가 심해지고 찾는 시간도 오래 걸려 비효율적일 수 있습니다. 이를 해결하기 위해 테이블을 4KB 크기 두 개의 table로 나눠 첫 테이블이 directory이고 두 번째 테이블이 second page table로 directory에서 mapping이 될 때만 그에 맞는 4KB 만큼의 second page table을 2^10개 만들어주는 방식으로 선택적인 메모리 적재를 통해 메모리 효율성을 높인 방식입니다.




출처:
:https://m.blog.naver.com/PostView.nhn?blogId=palyly&logNo=20166427097&proxyReferer=https%3A%2F%2Fwww.google.co.kr%2F

그리고 각 테이블은 어떤 user가 사용하는 테이블인지 알고 있어야 하는데 이는 TTBR로 각 user 프로세스에 해당하는 페이지 테이블을 지정해 줍니다. 또한 MMU가 mapping을 해주는 과정에서 페이지 양이 많아지면 해당 페이지와 이에 맞는 주소를 변환하는 시간이 길어지게 되는데 이를 위해 pagetable에 TLB라는 연관 캐쉬를 사용하여 물리주소에 가지 않고도 TLB에서 확인할 수 있도록 하여 주소 변환을 더욱 원활하게 하였습니다.

## Ⅲ. Concept

Tick이 지날 때 마다 user 프로세스는 10개의 virtual address에 접근하여 mapping 되는 물리 메모리 주소에서 data를 가져올 수 있어야 합니다. 이에 user 프로세스는 tick 마다 32비트의 virtual address를 랜덤으로 10개를 생성하고 생성한 10개의 주소를 kernel 프로세스에게 넘겨줘 kernel의 MMU가 이 10개의 주소를 가지고 물리주소로 변환해주는 일을 하게 됩니다. 그런데 여기서 user 프로세스와 kernel 프로세스는 서로 다른 프로세스 이므로 데이터를 공유할 수 없어 메시지 큐를 사용하여 데이터를 주고받아야 합니다. User 프로세스가 자신의 execution-time이 다 했을 때 커널에게 IO_bursttime과 cpu_bursttime을 보내는 메시지 큐와는 별개로 virtual addresss는 매 tick마다(프로세스가 연산을 수행할 때) 커널에게 보내야하므로 새로운 메시지 큐 하나를 더 생성하여 보내는 user의 pid와 virtual address 10개를 넘겨  주도록 하였습니다. 이제 kernel 에서 이 virtual address 10개를 받을 수 있을 것이고 이제 이 address를 가지고 physical address로 변환을 시켜줘야 할 것입니다. 또한 각 테이블마다 어떤 user가 사용하는 테이블인지 알기 위해 TTBR(Translation Lookaside Buffer)을 사용하는데 이 TTBR은 각 PCB마다 해당 TTBR 값을 가지고 있도록 하였습니다.

### 1. 1-level page table

* Virtual Address

| page number(20bit) | offset(12bit) |
|---|---|

* Physical Address

| page frame number(20bit) | offset(12bit) |
|---|---|

* Free Page Frame List(==qfree)
I = 0 ~ 0xFFFFF
(I << 12) | 0x000 형태
Pageframe number(20bit) + 0x000 형태로 넣어줍니다. --> 0x?????000
12bit를 shift해준 이유는 사용되지 않는 앞의 12bit의 첫bit에 valid bit를 넣어 주기 위해서입니다.

* Valid
Valid는 32bit중에 첫 비트(0x00000001)를 사용하여 mapping의 유무를 확인할 수 있도록 하였습니다. mapping이 된 것은 첫 비트에 valid비트를 추가시켜줌으로써 처음에 확인할 때 첫 비트만 보고 1이면 mapping인 된 것이고 0이면 mapping되지 않은 것을 알 수 있습니다.

* Page Table
Pagetable[TTBR][index] 형태로 TTBR은 현재 user가 사용하는 페이지 테이블이고 index는 페이지 테이블의 index번째를 가리킵니다. 그리고 해당 TTBR과 index에 맞는 page frame number에

valid 비트가 추가 되어 들어가게 될 것입니다. 해당 table에 들어가는 값은 (qfree에서 가져 온 값) | valid bit(0x00000001)입니다.

User에서 보낸 virtual address 값들을 PCB가 pid값을 확인하고 넘겨받습니다. 넘겨받은 virtual address 32비트 중 상위 20비트는 테이블의 index, 나머지 12비트는 offset으로 사용될 것입니다. 그리고 해당 페이지의 index의 값을 보고 valid가 0인지 1인지를 확인합니다. 0이면 아직 mapping 되어 있지 않는 상태인 것이고 1이면 mapping 이 되어있는 상태일 것입니다. valid가 0이면 page frame number 값에 valid 비트 0x1을 추가해서 페이지에 입력한 후에 valid를 1로 바꿔줍니다. valid가 1이면 mapping이 되어 있는 상태이므로 해당 페이지의 page frame number 값을 가져와 offset과 합쳐줌으로써 32비트의 physical address값을 얻을 수 있게 됩니다.

Example)

| qfree값 | Virtual Address | pagetable의 frame num값 (after mapping) | Physical Address |
|---|---|---|---|
| 0x00000000 | 0x12345678 | 0x000000001 | 0x00000678 |
| 0x00001000 | 0x13579333 | 0x000010001 | 0x00001333 |
| 0x00002000 | 0x29391324 | 0x000020001 | 0x00002324 |
| 0x00003000 | 0x47230123 | 0x000030001 | 0x00003123 |
| 0x00004000 | 0x38234576 | 0x000040001 | 0x00004576 |

## 2. 2-level page table

* Virtual Address

| directory index(10bit) | pagetable index(10bit) | offset(12bit) |
|---|---|---|

* Physical Address

| page frame number(20bit) | offset(12bit) |
|---|---|

* Free Page Frame List(==qfree)
I = 0 ~ 0xFFFFF
(I << 12) | 0x000 형태
Pageframe number(20bit) + 0x000 형태로 넣어줍니다. --> 0x?????000
12bit를 shift를 해준 이유는 사용되지 않는 앞의 12bit의 첫 bit에 valid bit를 넣어주기 위해서입니다.

* Dir_valid
Dir_valid는 첫 비트(0x00000001)를 사용하여 directory의 mapping의 유무를 확인할 수 있도록 하였습니다. Mapping이 된 것은 첫 비트에 valid 비트를 추가시켜줌으로써 처음에 확인할 때 첫 비트만 보고 1이면 mapping인 된 것이고 0이면 mapping 되지 않은 것을 알 수 있습니다.

* Page_valid
Page_valid는 첫 비트(0x00000001)를 사용하여 second table의 mapping의 유무를 확인할 수 있도록 하였습니다. Mapping이 된 것은 첫 비트에 valid 비트를 추가시켜줌으로써 처음에 확인할 때 첫 비트만 보고 1이면 mapping인 된 것이고 0이면 mapping 되지 않은 것을 알 수 있습니다.

* Directory

Directory[TTBR][directory_index] 형태로 TTBR은 현재 user가 사용하는 directory 테이블이고 directory index는 directory의 index번째를 가리킵니다. 그리고 해당 TTBR과 directory index에 맞는 값에 valid 비트가 추가되어 들어가게 될 것입니다. 그리고 이 값은 second table의 시작 주소가 될 것입니다.

해당 directory에 들어가는 값은 (qfree 에서 가져온 값) | valid bit(0x00000001)입니다.

* Second Page table

Second table[directory address][index] 형태로 directory address는 Directory의 값으로 Second page table의 시작 주소이고 index는 Second page table의 index 번째를 가리킵니다. 그리고 해당 index에 맞는 값에 valid 비트가 추가되어 들어가게 될 것입니다. 그리고 이 값은 physical address를 얻을 때 page frame number가 될 것입니다. 해당 table에 들어가는 값은 (qfree 에서 가져온 값) | valid bit(0x00000001)입니다.

User에서 보낸 virtual address 값들을 PCB가 pid값을 확인하고 넘겨받습니다. 넘겨받은 virtual address 32비트 중 상위 10비트는 Directory 테이블의 dir_index, 다음 10비트는 Second 테이블의 index, 12비트는 offset으로 사용될 것입니다. 해당 Directory 테이블의 dir_index의 값을 보고 dir_valid가 0인지 1인지를 확인합니다. 0이면 아직 Directory에 mapping 되어있지 않는 상태인 것이고 1이면 mapping 이 되어있는 상태일 것입니다. Dir_valid가 0이면 qfree 에서 가져온 값에 valid 비트 0x1을 추가해서 Directory에 입력한 후에 dir_valid를 1로 바꿔줍니다. 여기서 Directory에 입력한 값이 Second table의 시작 주소가 될 것입니다. Dir_valid가 1이면 이제 해당 Second table로 가서 Second table의 index의 값을 보고 page_valid가 0인지 1인지를 확인합니다. 0이면 아직 Second Table에 mapping 되어있지 않은 상태인 것이고 1이면 mapping 이 되어있는 상태일 것입니다. Page_valid가 0이면 qfree 에서 가져온 값에 valid 비트 0x1을 추가해서 Second Table에 입력한 후에 page_valid를 1로 바꿔줍니다. Page_valid가 1이면 Second Table에 mapping이 되어있는 상태이므로 해당 Second Table의 값의 상위 20비트를 가져와 offset과 합쳐줌으로써 32비트의 physical address 값을 얻을 수 있게 됩니다.

Example of Mapping)

| qfree값 | Virtual Address | Directory Table 값 | Second Table 값 | Physical Address |
|---|---|---|---|---|
| 0x00000000<br>0x00001000<br>0x00002000<br>0x00003000<br>0x00004000<br>..... | 0x00000678 | 0x000000001 | 0x000010001 | 0x000001678 |
| | 0x13579333 | 0x000020001 | 0x000030001 | 0x00003333 |
| | 0x29391324 | 0x000040001 | 0x000050001 | 0x00005324 |
| | 0x47230123 | 0x000060001 | 0x000070001 | 0x00007123 |
| | 0x38234576 | 0x000080001 | 0x000090001 | 0x00009576 |

## 3. Using TLB(Translation Lookaside Buffer)

TLB란 Virtual Address를 Physical Address로 변환하는 속도를 높이기 위해 사용되는 캐시로 최근에 일어난 virtual address에서 physical address로의 mapping 되었던 데이터를 저장하게 됩니다. TLB가 없을 때는 실제 데이터를 읽어내기 전에 memory에 존재하는 페이지 테이블을 읽어야

하지만 TLB가 있을 경우 페이지 테이블을 거치지 않고 바로 memory의 data에 접근이 가능하므로 memory 접속횟수를 줄이고 이를 통해 delay 시간을 줄일 수 있습니다. TLB에 저장시킬 항목들로 page number, page frame number, full, tag 4가지가 사용될 것입니다. 메모리 주소를 얻기 위한 page number, page frame number를 저장시킬 것이고 TLB에 데이터가 들어있는지(유효한 값들인지) 확인하기 위해 full bit를 사용하였으며 어떤 user 프로세스에서 입력된 값들인지 확인하기 위해 tag bit를 사용하여 TTBR 값을 저장하였습니다.

* 1-level using TLB

TLB를 먼저 확인해서 TLB에 원하는 데이터가 있는지 확인해야 합니다. TLB에 원하는 데이터가 있을 경우(HIT)는 첫 번째로 해당 TLB가 full 이여야 하고 두 번째로 실행 중인 TTBR과 TLB의 tag가 같아야 하며 마지막으로 virtual address의 index와 TLB의 page number가 일치하면 Hit이 됩니다. 이 3조건 만족할 시 일치하는 TLB의 index를 저장시켜줍니다. Hit일 경우에는 TLB에 있는 page frame number를 offset과 합쳐 physical address값을 얻어 메모리에 접근합니다. Hit이 아닐 경우 Page Table을 확인해야 하고 우선 valid가 0인지 1인지에 따라 0이면 mapping을 시켜주고 physical address값을 얻고 valid를 1로 바꿔줍니다. valid가 1인 경우에는 physical address값을 얻고 TLB에 업데이트를 시켜줍니다. Page Table의 page number, page frame number를 TLB의 page number, page frame number에 넣어주고 TLB의 tag에 TTBR값을 넣어 줌으로서 어떤 user의 table에서 온 데이터인지 확인할 수 있도록 하였습니다. 그리고 해당 TLB의 full을 1로 업데이트 시켜줍니다. 그리고 64개의 TLB를 모두 채웠을 때 LRU 방식을 사용하여 TLB를 업데이트 합니다.

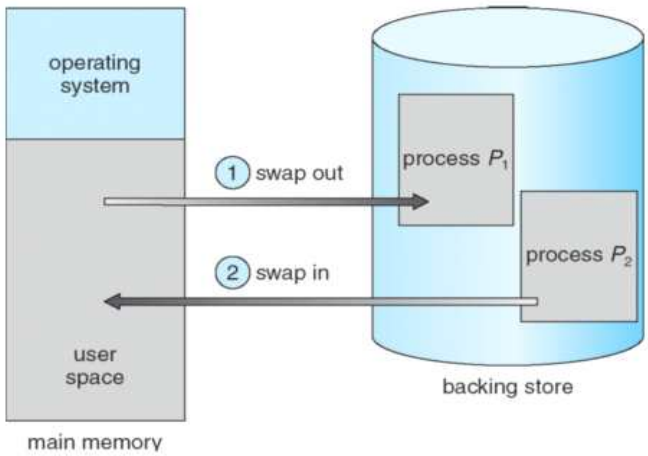
* 2-level using TLB

TLB를 먼저 확인해서 TLB에 원하는 데이터가 있는지 확인해야 합니다. TLB에 원하는 데이터가 있을 경우(HIT)는 full인 동시에 virtual address의 second page table의 index와 TLB의 page number가 일치하는 경우로 조건에 만족할 시 HIT를 1로 바꿔주고 일치하는 TLB의 index를 저장시켜줍니다. Hit일 경우에는 TLB에 있는 page frame number를 offset과 합쳐 physical address값을 얻어 메모리에 접근합니다. Hit이 아닐 경우 Page Table을 확인해야 하고 우선 Directory의 valid가 0인지 1인지에 따라 0이면 mapping을 시켜주고 해당 Second table로 넘어가서 또 mapping을 시켜준 후에 valid값들을 1로 바꿔줍니다. Directory 의 valid가 1이면 해당 Second table로 넘어가 mapping이 되어 있지 않으면 mapping을 시켜주고 mapping이 되어있으면 physical address값을 얻고 TLB에 업데이트를 시켜줍니다. Second Table의 page number, page frame number를 TLB의 page number, page frame number에 넣어주고 TLB의 tag에 TTBR값을 넣어줌으로서 어떤 user의 table에서 온 데이터인지 확인할 수 있도록 하였습니다. 그리고 해당 TLB의 full을 1로 업데이트 시켜줍니다. 그리고 64개의 TLB를 모두 채웠을 때 LRU 방식을 사용하여 TLB를 업데이트 합니다.

## 4. Using Swapping

프로그램을 실행할 때 프로세스들은 한정된 메모리에 한해서 작업을 수행하게 되는데 이 한정된 메모리를 다 사용하게 되고 작업할 것이 남았을 때 원활하게 작업이 가능하도록 사용된 메모리에 대해서 프로세스에게 메모리 공간을 할당해주는 작업을 Swapping이라고 합니다. 메모리 사용이 끝난 프로세스를 메모리 공간에서 내보내는 작업을 Swap Out이라고 하며 메모리를 사용해야 되는 프로세스에게 메모리 공간을 할당해주는 작업을 Swap In이라고 합니다.  Swap Out을 할 때 사용이 완전히 끝난 프로세스에 대한 메모린지 꼭 확인해줘야 할 필요가 있으며 효과적인 CPU사용을 위해서

는 각 프로세스의 실행시간이 swapping 시간보다 충분히 길어야 할 것입니다. 또한 Swapping 이 일어나야 할 시점이 메모리가 다 사용된 시점에서 하게 된다면 그 이후에 프로세스들은 이 Swapping되는 시간을 기다렸다가 수행해야 하므로 메모리가 다 사용된 시점보다 70~80%정도 사용된 시점에서 하는 것이 좋다고 합니다.




출처:

https://m.blog.naver.com/PostView.nhn?blogId=jevida&logNo=140191090013&proxyReferer=https%3A%2F%2Fwww.google.co.kr%2F

## * 1-level using swapping

1-level swapping에서는 VA의 page가 pageframe에 mapping이 되고 mapping을 함으로써 page가 실제 memory에서 사용합니다. 하지만 VA는 각 process마다 가진 크기이고 실제로 사용하는 memory는 제한적이기 때문에 memory의 공간이 부족한 경우에 대해서 지금 현재 사용하지 않는 process에 mapping되어 있는 pageframe을 현재 사용해야 하는 process에 할당 해줍니다. 이때 process에서 가지고 있는 기존의 data를 disk에 저장해 줍니다. Disk에 저장하는 과정에서 block이라는 structure를 사용하여 pageframe과 같은 크기로 할당을 해주고 pageframe 전체를 block에 저장합니다. Data를 disk에 저장함으로써 다음에 swapout 된 process가 들어오면 disk에 저장된 값을 불러와 다시 값을 입력해주는 방식입니다. 이를 구현하기 위해 disk의 어떤 block에 저장했는지를 알기 위한 blocktable을 각 process마다 만들어 주었습니다. blocktable을 통해 다음에 swapout된 VA가 입력되면 pageframe을 새로 할당해주고 blocktable을 확인하여 disk에 저장된 block을 불러와 data를 pageframe에 저장합니다. Swapping이 일어날 때 어떤 pageframe을 mapping 할지에 대해서는 LRU 방식을 사용하였습니다. 처음 mapping한 pageframe의 TTBR과 index를 기억하도록 mapping queue를 만들어 처음 mapping한 pageframe에 대해 저장을 합니다. 이후 swapping이 이루어지게 되면 이때 기억하고 있는 mapping queue에서 값을 가지고 와 pageframe을 찾을 수 있습니다. Swapping이 일어나면 이 pageframe을 다시 free-list queue에 넣어줌으로써 다시 mapping이 진행되도록 하였습니다.

## * 2-level using swapping

2-level swapping을 하기 위해서 두 가지 방법을 사용하였습니다.

1) 2-level 모두에서 swapping이 발생하는 경우 1-level에서 사용한 방법과 마찬가지로 data를 disk block에 저장합니다. 하지만 이런 경우에 대해 first level인 page directory에서 swapping이 이루어지면 second level에서 mapping되어 있는 값들이 전부 사라지게 됩니다. 이를 통해 first level에서 swapping이 이루어지면 second level의 data를 모두 잃어버리는 현상이 발생하였습니

다. 이를 방지하기 위해 다음과 같은 두 번째 방법을 사용하였습니다.
2) Second level에 대해서만 swapping이 발생하도록 하였습니다. Free-list queue에 pageframe을 저장할 때 first level에 mapping한 pageframe은 다시 free-list queue에 저장되지 않습니다. Second level에 대해서만 pageframe을 저장합니다. 이를 통해 swapping이 발생하는 경우 second level에서 pageframe mapping을 풀고 data는 disk block에 저장한 후 free-list queue에 넣어줍니다. 이러한 방식을 사용하여 1-level swapping과 유사한 방식으로 코딩을 하였습니다.

## 5. Program Structure

### (1) 1-level

1. Structure & Global value initialize

2. fork
    - 2-1. parent
        - 2-1-1 get data(pid, timequantum, execution time, TTBR)
    - 2-2. child
        - 2-2-1 make random virtual address
        - 2-2-2 msgsend1
        - 2-2-3 child exit handler
        - 2-2-4 check(exec_time == 0)?
        - 2-2-5 msgsend
3. kernel
    - 3-1 SIGALRM handler
    - 3-2 check ready queue empty?
    - 3-3 do_wait for I/O burst
    - 3-4 Round-Robin(time quantum=0 --> msgrcv)
    - 3-5 msgrcv1
    - 3-6 make_table
        - 3-6-1 get virtual address
        - 3-6-2 In page table check(valid == 0)?
        - 3-6-3 In page table check(valid == 1)?
        - 3-6-4 get physical address

### (2) 2-level

1. Structure & Global value initialize

2. fork
    - 2-1. parent
        - 2-1-1 get data(pid, timequantum, execution time, TTBR)
    - 2-2. child
        - 2-2-1 make random virtual address
        - 2-2-2 msgsend1
        - 2-2-3 child exit handler
        - 2-2-4 check(exec_time == 0)?
        - 2-2-5 msgsend

3. kernel
    3-1 SIGALRM handler
    3-2 check ready queue empty?
    3-3 do_wait for I/O burst
    3-4 Round-Robin(time quantum=0 --> msgrcv)
    3-5 msgrcv1
    3-6 make_table
        3-6-1 get virtual address
        3-6-2 In Directory check(dir_valid == 0)?
        3-6-3 In Directory check(dir_valid == 1)?
        3-6-4 In Second Table check(page_valid == 0)?
        3-6-5 In Second Table check(page_valid == 1)?
        3-6-6 get physical address

(3) 1-level Using TLB

1. Structure & Global value initialize

2. fork
    2-1. parent
        2-1-1 get data(pid, timequantum, execution time, TTBR)
    2-2. child
        2-2-1 make random virtual address
        2-2-2 msgsend1
        2-2-3 child exit handler
        2-2-4 check(exec_time == 0)?
        2-2-5 msgsend

3. kernel
    3-1 SIGALRM handler
    3-2 check ready queue empty?
    3-3 do_wait for I/O burst
    3-4 Round-Robin(time quantum=0 --> msgrcv)
    3-5 msgrcv1
    3-6 make_table
        3-6-1 get virtual address
        3-6-2 check TLB
        3-6-3 check Page Table
        3-6-4 check(valid == 0)?
        3-6-5 check(valid == 1)?
        3-6-6 update TLB
        3-6-7 get physical address

(4) 2-level Using TLB

1. Structure & Global value initialize

2. fork

2-1. parent

2-1-1 get data(pid, timequantum, execution time, TTBR)

2-2. child

2-2-1 make random virtual address

2-2-2 msgsend1

2-2-3 child exit handler

2-2-4 check(exec_time == 0)?

2-2-5 msgsend

3. kernel

3-1 SIGALRM handler

3-2 check ready queue empty?

3-3 do_wait for I/O burst

3-4 Round-Robin(time quantum=0 --> msgrcv)

3-5 msgrcv1

3-6 make_table

3-6-1 get virtual address

3-6-2 check TLB

3-6-3 check Directory Table

3-6-4 check(dir_valid == 0)?

3-6-5 check(dir_valid == 1)?

3-6-6 check Second Table

3-6-7 check(page_valid == 0)?

3-6-8 check(page_valid == 1)?

3-6-9 update TLB

3-6-10 get physical address

(5) 1-level SWAP

1. Structure & Global value initialize

2. fork

2-1. parent

2-1-1 get data(pid, timequantum, execution time, TTBR)

2-2. child

2-2-1 make random virtual address

2-2-2 msgsend1

2-2-3 child exit handler

2-2-4 check(exec_time == 0)?

2-2-5 msgsend

3. kernel

3-1 SIGALRM handler

3-2 check ready queue empty?

3-3 do_wait for I/O burst

3-4 Round-Robin(time quantum=0 --> msgrcv)

3-5 msgrcv1

3-6 make_table

3-6-1 get virtual address
3-6-2 check valid
3-6-3 check free-list queue
3-6-4 swapping event : copy pageframe data to disk block
3-6-5 swapping event : release mapping and mapping to new page
3-6-6 swapin : copy disk block to pageframe data
3-6-7 get data

(6) 2-level SWAP

1. Structure & Global value initialize

2. fork
    2-1. parent
        2-1-1 get data(pid, timequantum, execution time, TTBR)
    2-2. child
        2-2-1 make random virtual address
        2-2-2 msgsend1
        2-2-3 child exit handler
        2-2-4 check(exec_time == 0)?
        2-2-5 msgsend
3. kernel
    3-1 SIGALRM handler
    3-2 check ready queue empty?
    3-3 do_wait for I/O burst
    3-4 Round-Robin(time quantum=0 --> msgrcv)
    3-5 msgrcv1
    3-6 make_table
        3-6-1 get virtual address
        3-6-2 check(dir_valid == 0)?
        3-6-3 check Directory Table
        3-6-4 check free-list queue
        3-6-5 swapping event : copy second level pageframe data to disk block
        3-6-6 swapping event : release mapping and map to new first level page
        3-6-7 check(dir_valid == 1)?
        3-6-8 check(page_valid == 0)?
        3-6-9 check free-list queue
        3-6-10 swapping event : copy second level pageframe data to disk block
        3-6-11 swapping event : release mapping and map to new second level page
        3-6-12 swapin : copy disk block to second level pageframe data
        3-6-13 get data

## Ⅳ. Build environment

Compilation : Linux Assam, with GCC
Header file : msg.h, t.h, queue.h
To compile : gcc 1_level.tlb.c / gcc 2_level.c / gcc 2_level.tlb.c / gcc one_level_swap.c
/ gcc two_level_swap1.c / gcc two_level_swap2.c
To run : ./a.out

## Ⅳ. Problem & Solution

1. problem) user의 virtual address값을 kernel에게 알려야 해서 io/burst time을 넘겨줄 때 같이 넘겨주려고 하였으나 넘겨주는 타이밍이 달라서 같이 넘겨줄 수가 없었습니다.
☞ sol) io/burst time은 execution time이 0이 될 때 넘겨주지만 virtual address는 매 tick마다 받아야 하므로 또 다른 메시지 큐 하나를 더 생성하여 user의 virtual address와 pid를 kernel로 넘겨줬습니다.

2. problem) table의 valid값을 page frame number의 수만큼 생성하였습니다. 이럴 경우 structure 내에 int 형 valid가 추가 되고 모든 structure에 4byte의 memory를 추가로 할당합니다. Memory를 최소한으로 사용하기 위해 다음과 같은 방법으로 처리하였습니다.
☞ sol) Pageframe number는 20 bit만을 사용하고 사용되지 않고 남은 12 bit에 대해 앞의 1bit를 valid bit으로 표현하여 mapping 안 되어있는 것은 0, mapping이 된 것은 1로 표현 하였습니다. 이런 경우에 대해 valid에 대해 추가적으로 memory를 할당할 필요가 없습니다.

3. problem) TLB를 사용하였을 때 가끔씩 hit이 원하지 않을 때 되는 경우가 발생합니다.
☞ sol) TLB는 각 User 프로세스마다 사용되는 TLB가 달라야 함을 알았고 한 User 프로세스당 TLB에 담길 수 있는 최대 수를 알고 그만큼만 TLB를 만들어 줬습니다. 이 방법 말고도 모든 User 프로세스가 한 TLB를 사용할 수 있게 TLB에 tag라는 변수를 추가해서 TTBR을 저장시킴으로써 해당 TLB값들이 어떤 User 프로세스에서 온 값들인지 알 수 있도록 하였습니다.

4. problem) 일반 TLB의 경우 context switching이 발생할 시에 VA가 초기화됩니다. 그리고 VA는 0x00000000~0xFFFFFFFF 사이의 큰 범위 중 랜덤 값으로 생성되기 때문에 TLB에서 Hit이 될 경우가 매우 드물어져 확인이 어려웠습니다.
☞ sol) 한 프로세스가 실행될 때마다 랜덤 VA를 받는 것이 맞지만 TLB의 Hit를 확인하기 위해 한 프로세스가 실행 중이던 일이 끝나기 전(remaining timequantum이 끝나기 전)까지는 VA를 유지시켜줌으로써 TLB의 Hit를 확인할 수 있었습니다.

5. problem) 2-level을 할 때 Page Directory와 Second Page Table에 어떤 값을 넣어줘야 할지 확실하지가 않았습니다.
☞ sol) Page Directory에는 물리 메모리를 얻기 위한 실질적인 pageframe number와는 관련이 없는 두 번째 Page Table을 가리키는 주소의 값이 들어가야 했습니다. 그리고 Second Page Table에는 pageframe number가 들어감으로써 이 값을 가지고 물리 메모리를 얻을 수 있었습니다.

6. problem) 실질적으로 사용한 모든 주소 값들은 physical memory에 할당 된 값들입니다. 그렇

기 때문에 각 user process가 사용하는 page table의 시작 주소를 나타내는 TTBR을 알기 위해서는 page table이 시작하는 첫 physical address의 주소를 가지고 page table의 크기만큼 더해 주어 다음에 있는 page table의 시작 주소를 알 수 있습니다. 이런 식으로 address를 찾을 경우 add를 해주어야 하므로 instruction이 추가되고 소요시간이 더 발생합니다. 이러한 경우 이전의 schedule에서 사용한 time interval을 사용할 경우 error가 발생했습니다.
☞ sol) Simulation을 하는 과정에서 이러한 연산과정을 줄이고 빠르게 보여주기 위해 array를 사용하여 연산을 대신하였습니다. 실질적으로 저장하는 memory의 위치에 대한 주소가 아닌 각 프로세스에서 사용하는 page table을 나타내기 위해 process마다 index로 나타내어 user process가 10개면 pagetable[ttbr][index]라고 표현하여 ttbr에 0부터 9까지 10개를 만들어 주었습니다. 이 외에도 physical address를 구하는 경우에 pageframe number와 virtual address의 offset 부분을 합쳐 연산을 한다면 각각에 대해 array의 index로 사용하여 memory[pageframe number][offset]과 같이 표현해 주었습니다.

7. problem) Swapping 할 때 Physical memory의 크기를 정합니다. Physical memory의 크기가 매우 크다면 pageframe 또한 많아지므로 swapping이 이루어지는 경우가 작아 확인하기가 쉽지 않습니다.
☞ sol) Physical memory의 크기를 임의적으로 정해줍니다. 이를 통해 pageframe의 개수도 정할 수 있습니다. Test를 하기 위해 100 tick을 진행하는 동안 pageframe의 개수를 50개로 제한하였습니다. 이를 통해 매 tick마다 10개의 VA가 들어오고 5 tick이 지나 50개의 pageframe이 모두 찬 이후부터 swapping이 이루어지는 것을 볼 수 있었습니다.

8. problem) Two-level swapping을 구현할 때 first level인 page directory와 second level인 page table 모두에서 swapping이 이루어지는 경우에 대해 first level에서 swapping이 이루어지면 이전에 mapping이 되어있던 second level의 mapping을 모두 잃게 됩니다. 이를 통해 이미 이전에 mapping되었던 VA에 대해 data를 기록하고 있을 수 없습니다.
☞ sol)Second level에서만 swapping이 일어나도록 하였습니다. First level에서 mapping을 필요로 하며 남은 pageframe이 없는 경우에 대해 LRU 방식을 통해 먼저 사용한 pageframe의 mapping을 풀고 data를 disk block에 저장한 후 지금 필요한 page에 pageframe을 새로 mapping 해줍니다. 하지만 이때 다시 LRU로 확인하는 pageframe에 대해 first level의 pageframe은 예외 처리를 합니다. First level에 대해 mapping 된 pageframe들은 계속 mapping 상태를 유지하도록 합니다. 이를 통해 second level에서만 swapping이 이루어지도록 하여 위의 문제와 같이 기존에 mapping 된 page에 대해서 잃어버리는 data를 계속 유지할 수 있습니다.

9. problem) Swapping을 구현하기 전에는 VA에 대해 mapping 된 PA를 구하여 이를 통해 mapping이 되었다는 것을 확인하였습니다. 하지만 swapping을 하면서 swapout된 VA가 다음에 swapin이 되었을 때 값은 PA를 가지고 있지는 않습니다. Mapping 되는 pageframe이 바뀌면서 PA의 값도 바뀌기 때문입니다.
☞ sol) 이러한 문제를 확인하기 위해 pageframe에 data를 직접 입력한 후 그 값에 대해 swapping이 이루어진 후에도 disk를 통해 기존에 가지고 있는 값을 그대로 유지하는지를 확인하였습니다.

10. problem) Swapping이 일어날 때에 대한 free-list에 남아있는 pageframe의 비율이 중요했습니다. 남은 pageframe의 수가 0일 때부터 swapping이 이루어지는 경우에 대해 랜덤하게 들어오는 VA 때문에 계속적으로 swapping이 발생합니다. VA 의 범위가 매우 크기 때문에 mapping을 지속적으로 다시 하기 때문에 문제가 생겼습니다.

☞ sol) 이런 경우를 해결하기 위해 swapping이 이루어지는 경우를 free-list의 pageframe이 대략 30%정도 남아있을 때부터 실행하게 하였습니다. Swapping이 시작하는 경우에 대해 if( emptyqueue(free-list queue))에서 free-list queue->count가 30%보다 작은 경우에 실행 하도록 바꿔주었습니다.

## Ⅴ. Personal Feelings

우선, 컴파일 과정에서 주소 공간과 테이블, 물리 공간 등에서 서로 매칭되는 사이즈가 맞지 않거나 사이즈가 커져서 생기는 오류들이 있었습니다. 이런 것들을 제외하고 오류가 많이 나지는 않았지만, 의도에 맞게 실행되는지 확인하는 것이 힘들었습니다. 잘 실행이 된 것 같더라도 가끔씩 원하지 않는 값들이 나와서 이럴 때 오류를 찾아내는 것이 어려웠습니다. 특히, mapping에 대해서 눈으로 확인하기가 쉽지 않았습니다. 코드 상에서 오류가 없으면 컴파일은 되었지만, 그 때에 대해 잘 된 것인지를 확인하는 일을 직접 해야 했습니다. 이번 가상메모리 구현 과제를 진행하면서, 이번 학기 과제 및 프로젝트를 하는 동안 발생한 segmentation fault에 대해 좀 더 깊고 확실하게 이해할 수 있었고, OS의 메모리 관리 기법을 이해할 수 있는 수업이었습니다. 이렇게 제한된 메모리를 효율적으로 사용하기 위해 1-level, 2-level, TLB, swapping 등 여러 방식들을 적용해봄으로써 메모리 관리의 중요성을 깨닫게 되었습니다.

## Ⅶ. Code

### <1_level.tlb.c>

```
#include "t.h"
#include "queue.h"
#include "msg.h"
#define NUM 10

QUEUE *qfree;
QUEUE *qwait;
QUEUE *qready;
int ptr;
int wptr;
int msgq;
int msgq1;
int key = 0x32143153;
int key1 = 0x32140301;
int t = 0;

struct sigaction old_sa;
struct sigaction new_sa;
struct msgbuf msg;
struct msgaddr msg1;
```

```
pcb_t PCB[MAXPROC];
user uProc;
table **pagetable;
int *pageframe;

Tlb tlb[64];

int main()
{
	memset(&new_sa, 0, sizeof(new_sa));
	msgq = msgget( key, IPC_CREAT | 0666);
	memset(&msg, 0, sizeof(msg));
	msgq1 = msgget( key1, IPC_CREAT | 0666);
	memset(&msg1, 0, sizeof(msg1));
	pageframe = (int*)malloc(sizeof(int) * 0xA0000);
	pagetable = (table**)malloc(sizeof(table*) * MAXPROC);
	for(int k =0; k<MAXPROC; k++)
	{
		pagetable[k] = (table*)malloc(sizeof(table) * 0x100000);
	}
	qwait = CreateQueue();
	qready = CreateQueue();
	qfree = CreateQueue();
	srand((unsigned)time(NULL));
	pid_t pid;

	for(int i=0; i<0xA0000; i++)
	{
		Enqueue(qfree, i << 12 | 0x000);
	}
	for(int k=0; k<MAXPROC; k++)
	{
		for(int i=0; i<NUM; i++)
		{
			uProc.VA[i] = (rand() % 0x100000000);
		}
		uProc.exec_time = (rand() % 10) + 1;
		uProc.ppid = getpid();
		uProc.IO = (rand() % 10) + 1;
		pid=fork();
		if(pid<0){
			perror("fork fail");
		}
		else if(pid == 0){//child
			new_sa.sa_handler = &signal_handler2;
			sigaction(SIGUSR1, &new_sa, &old_sa);
			while(1);
			exit(0);
		}
		else{//parent
```

```
                        PCB[k].pid = pid;
                        PCB[k].TTBR = k;
                        PCB[k].remaining_tq = DEFAULT_TQ;
                        PCB[k].exec_time = uProc.exec_time;
                }
                Enqueue(qready, k);
        }

        new_sa.sa_handler = &signal_handler;
        sigaction(SIGALRM, &new_sa, &old_sa);
        new_sa.sa_handler = &signal_handler3;
        sigaction(SIGUSR2, &new_sa, &old_sa);
        fire(1);

        while(1);
}

void RR()
{
        if(emptyQueue(qready))return;
        else{
                ptr = qready->front->data;
                if(PCB[ptr].remaining_tq > 0)
                {
                        PCB[ptr].remaining_tq--;
                        PCB[ptr].exec_time--;
                        printf("Tick : %d, PID : %d\tPCB[%d].exec_time : %d\n", tick, PCB[ptr].pid,
ptr, PCB[ptr].exec_time);
                        kill(PCB[ptr].pid, SIGUSR1);
                        if(PCB[ptr].exec_time == 0)
                        {
                                PCB[ptr].remaining_tq = DEFAULT_TQ;
                                Enqueue(qwait, Dequeue(qready));
                                return;
                        }
                }
                if(PCB[ptr].remaining_tq == 0)
                {
                        PCB[ptr].remaining_tq = DEFAULT_TQ;
                        Enqueue(qready, Dequeue(qready));
                        return;
                }
        }
        return;
}

void do_wait()
{
        if(emptyQueue(qwait))
        {
```

```
                return;
        }
        else
        {
                for(int k=0; k<qwait->count; k++)
                {
                        wptr = qwait->front->data;
                        if(PCB[wptr].IO_time > 0)
                        {
                                PCB[wptr].IO_time--;

                                if(PCB[wptr].IO_time == 0)
                                {
                                        Enqueue(qready, Dequeue(qwait));
                                }
                                else if(PCB[wptr].IO_time > 0)
                                {
                                        Enqueue(qwait, Dequeue(qwait));
                                }
                        }
                }
        }
}

void make_table()
{
        int index;
        int offset;
        int valid;
        int Address;
        int HIT;
        int tlbindex;

        for(int k=0; k<MAXPROC; k++)
        {
                if(PCB[k].pid == msg1.pid)
                {
                        memcpy(&PCB[k].VA, &msg1.VA, sizeof(msg1.VA));
                }
        }
        for(int k=0; k<NUM; k++)
        {
                index = (PCB[ptr].VA[k] & 0xFFFFF000) >> 12;//first 20bit
                offset = (PCB[ptr].VA[k] & 0xFFF);//12bit
                valid = pagetable[PCB[ptr].TTBR][index].pfn & 0x00000001;//valid bit
                HIT = 0;

                for(int k=0; k<64; k++){
                        if((tlb[k].tag == PCB[ptr].TTBR)&&(tlb[k].full == 1)&&(tlb[k].pn ==
index)){//check TTBR, full, pagenumber
```

```
                                HIT = 1;//find in TLB
                                tlbindex = k;//check tlb'index when hit
                        }
                }

                if(HIT){//if hit
                        Address = (tlb[tlbindex].pfn & 0xFFFFF000) | offset;//get physical address
                        printf("HIT!!, Tick %d = VA[%d] : 0X%08X -------- Address : 0X%08X\n",
tick, k, PCB[ptr].VA[k], Address);
                        printf("--> VA: 0X%08X, TTBR : %d index : 0X%08X ---------------- PA :
0X%08X tlb[%d], TTBR : %d pn : 0X%08X pfn : 0X%08X\n",PCB[ptr].VA[k], PCB[ptr].TTBR, index,
Address, tlbindex, tlb[tlbindex].tag, tlb[tlbindex].pn, tlb[tlbindex].pfn);
                }

                else if(!HIT){//if miss
                        if(valid == 0)//if not mapping in page table
                        {
                                pagetable[PCB[ptr].TTBR][index].pfn = Dequeue(qfree) | 0x1;//get
page frame number from free list and add valid bit
                                Address = (pagetable[PCB[ptr].TTBR][index].pfn & 0xFFFFF000) |
offset;//get physical address
                                valid = 1;//now mapping
                                printf("Tick %d, TTBR %d = VA[%d] : 0X%08X -------- Address :
0X%08X\n", tick, PCB[ptr].TTBR, k, PCB[ptr].VA[k], Address);
                        }
                        else if(valid == 1)//if already mapping in page table
                        {
                                Address = (pagetable[PCB[ptr].TTBR][index].pfn & 0xFFFFF000)|
offset;//get physical address
                                tlb[t].pn = index;//tlb page number update
                                tlb[t].pfn = pagetable[PCB[ptr].TTBR][index].pfn;//tlb page frame
number update
                                tlb[t].full = 1;//tlb full update
                                tlb[t].tag = PCB[ptr].TTBR;//tlb TTBR tag update
                                t++;//next tlb index
                                if(t==64){
                                        t=0;//go first index
                                }
                                printf("Tick %d, TTBR %d = VA[%d] : 0X%08X -------- Address :
0X%08X\n", tick, PCB[ptr].TTBR, k, PCB[ptr].VA[k], Address);
                        }
                }
        }
}

void signal_handler(int signo)
{
        tick++;
        if(tick == 10000)
        {
```

```
                for(int k=0; k<MAXPROC; k++)
                {
                        kill(PCB[k].pid, SIGINT);
                }
                DestroyQueue(qready);
                DestroyQueue(qwait);
                DestroyQueue(qfree);
                printf("Tick : %d, OS exit\n", tick);
                exit(0);
        }
        do_wait();
        RR();
        msgrecieve1();//get msg (virtual address) from user
        make_table();
}

void signal_handler2(int signo)
{
        uProc.exec_time--;
        /*for(int k = 0; k < NUM; k++)
        {
                uProc.VA[k] = (rand() % 0x100000000);
        }*/
        msgsend1();//send msg(virtual address) to kernel
        if(uProc.exec_time == 0)
        {
                uProc.exec_time = (rand() % 10) + 1;
                msgsend();
                kill(uProc.ppid, SIGUSR2);
        }
}

void signal_handler3(int signo)
{
        msgrecieve();
        for(int k=0; k<MAXPROC; k++)
        {
                if(PCB[k].pid == msg.pid)
                {
                        PCB[k].IO_time = msg.io_time;
                        PCB[k].exec_time = msg.cpu_time;
                }
        }
}

int msgsend()
{
        int ret = -1;
        //struct msgbuf msg;
        msg.mtype = 0;
```

```
        msg.pid = getpid();
        msg.io_time = uProc.IO;
        msg.cpu_time = uProc.exec_time;
        while(ret == -1)
        {
                ret = msgsnd(msgq, &msg, sizeof(msg), 0);
        }
        return 0;
}

int msgsend1()
{
        int ret = -1;
        msg1.mtype = 0;
        msg1.pid = getpid();
        memcpy(&msg1.VA, &uProc.VA, sizeof(uProc.VA));
        while(ret == -1)
        {
                ret = msgsnd(msgq1, &msg1, sizeof(msg1), 0);
        }
        return 0;
}

int msgrecieve()
{
        int ret = -1;
        //struct msgbuf msg;
        while(ret == -1)
        {
                ret = msgrcv(msgq, &msg, sizeof(msg), 0, MSG_NOERROR);
        }
        return 0;
}

int msgrecieve1()
{
        int ret = -1;
        while(ret == -1)
        {
                ret = msgrcv(msgq1, &msg1, sizeof(msg1), 0, MSG_NOERROR);
        }
        return 0;
}

void fire(int interval_sec)
{
        struct itimerval new_itimer, old_itimer;
        new_itimer.it_interval.tv_sec = 0;
        new_itimer.it_interval.tv_usec = 100;
        new_itimer.it_value.tv_sec = 0;//interval_sec;
```

```
        new_itimer.it_value.tv_usec = 100;
        setitimer(ITIMER_REAL, &new_itimer, &old_itimer);
}
```

## <2_level.c>

```
#include "t.h"
#include "queue.h"
#include "msg.h"
#define NUM 10

QUEUE *qfree;
QUEUE *qwait;
QUEUE *qready;
int ptr;
int wptr;
int msgq;
int msgq1;
int key = 0x32143153;
int key1 = 0x32140301;

struct sigaction old_sa;
struct sigaction new_sa;
struct msgbuf msg;
struct msgaddr msg1;
pcb_t PCB[MAXPROC];
user uProc;
table **pagetable;
int *pageframe;
/*
typedef struct DIRECTORY_{
        int dn;
}Directory;

typedef struct TABLE2_{
        int pfn;
}secondtable;
*/
Directory **directory;
secondtable **table2;

int main()
{
        memset(&new_sa, 0, sizeof(new_sa));
        msgq = msgget( key, IPC_CREAT | 0666);
        memset(&msg, 0, sizeof(msg));
        msgq1 = msgget( key1, IPC_CREAT | 0666);
        memset(&msg1, 0, sizeof(msg1));
        pageframe = (int*)malloc(sizeof(int) * 0x80000);
        directory = (Directory**)malloc(sizeof(Directory*)*MAXPROC);
```

```
for(int k=0; k<MAXPROC; k++)
{
        directory[k] = (Directory*)malloc(sizeof(Directory)* 0x400);
}
table2 = (secondtable**)malloc(sizeof(secondtable*)*0x400);
for(int k=0; k<0x400; k++)
{
        table2[k] = (secondtable*)malloc(sizeof(secondtable)*0x400);
}
qwait = CreateQueue();
qready = CreateQueue();
qfree = CreateQueue();
srand((unsigned)time(NULL));
pid_t pid;

for(int i=0; i<0x80000; i++)
{
        Enqueue(qfree, i << 12 | 0x000);
}
for(int k=0; k<MAXPROC; k++)
{
        /*for(int i=0; i<NUM; i++)
        {
                uProc.VA[i] = (rand() % 0x100000000);
        }*/
        uProc.exec_time = (rand() % 10) + 1;
        uProc.ppid = getpid();
        uProc.IO = (rand() % 10) + 1;
        pid=fork();
        if(pid<0){
                perror("fork fail");
        }
        else if(pid == 0){//child
                new_sa.sa_handler = &signal_handler2;
                sigaction(SIGUSR1, &new_sa, &old_sa);
                while(1);
                exit(0);
        }
        else{//parent
                PCB[k].pid = pid;
                PCB[k].TTBR = k;
                PCB[k].remaining_tq = DEFAULT_TQ;
                PCB[k].exec_time = uProc.exec_time;
        }
        Enqueue(qready, k);
}

new_sa.sa_handler = &signal_handler;
sigaction(SIGALRM, &new_sa, &old_sa);
new_sa.sa_handler = &signal_handler3;
```

```
	sigaction(SIGUSR2, &new_sa, &old_sa);
	fire(1);

	while(1);
}

void RR()
{
	if(emptyQueue(qready))return;
	else{
		ptr = qready->front->data;
		if(PCB[ptr].remaining_tq > 0)
		{
			PCB[ptr].remaining_tq--;
			PCB[ptr].exec_time--;
			printf("Tick : %d, PID : %d\tPCB[%d].exec_time : %d\n", tick, PCB[ptr].pid,
ptr, PCB[ptr].exec_time);
			kill(PCB[ptr].pid, SIGUSR1);
			if(PCB[ptr].exec_time == 0)
			{
				PCB[ptr].remaining_tq = DEFAULT_TQ;
				Enqueue(qwait, Dequeue(qready));
				return;
			}
		}
		if(PCB[ptr].remaining_tq == 0)
		{
			PCB[ptr].remaining_tq = DEFAULT_TQ;
			Enqueue(qready, Dequeue(qready));
			return;
		}
	}
	return;
}

void do_wait()
{
	if(emptyQueue(qwait))
	{
		return;
	}
	else
	{
		for(int k=0; k<qwait->count; k++)
		{
			wptr = qwait->front->data;
			if(PCB[wptr].IO_time > 0)
			{
				PCB[wptr].IO_time--;
				//printf("PCB[%d].IOtime : %d\n", wptr, PCB[wptr].IO_time);
```

```
                                if(PCB[wptr].IO_time == 0)
                                {
                                        Enqueue(qready, Dequeue(qwait));
                                }
                                else if(PCB[wptr].IO_time > 0)
                                {
                                        Enqueue(qwait, Dequeue(qwait));
                                }
                        }
                }
        }
}

void make_table()
{
        int dirindex;
        int index;
        int offset;
        int dir_valid;
        int page_valid;
        int Address;

        for(int k=0; k<MAXPROC; k++)
        {
                if(PCB[k].pid == msg1.pid)
                {
                        memcpy(&PCB[k].VA, &msg1.VA, sizeof(msg1.VA));
                }
        }

        for(int k=0; k<NUM; k++)
        {
                dirindex = (PCB[ptr].VA[k] & 0xFFC00000) >> 22;
                index = (PCB[ptr].VA[k] & 0x3FF000) >> 12;
                offset = (PCB[ptr].VA[k] & 0xFFF);
                dir_valid = directory[PCB[ptr].TTBR][dirindex].dn & 0x00000001;
                page_valid = table2[directory[PCB[ptr].TTBR][dirindex].dn >> 22][index].pfn &
0x00000001;
                if(dir_valid == 0)
                {
                        //need mapping
                        directory[PCB[ptr].TTBR][dirindex].dn = (Dequeue(qfree) & 0xFFC00000) | 0x1;
                        dir_valid = 1;
                }
                if(dir_valid == 1)
                {
                        if(page_valid == 0){
                                printf("no mapping\t");
                                table2[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].pfn =
Dequeue(qfree) | 0x1;
```

```
                                page_valid = 1;
                        }
                        if(page_valid == 1){
                                Address = (table2[(directory[PCB[ptr].TTBR][dirindex].dn) >>
22][index].pfn & 0xFFFFF000) | offset;
                                printf("Tick %d, TTBR %d = VA[%d] : 0X%08X -------- Address :
0X%08X\n", tick, PCB[ptr].TTBR, k, PCB[ptr].VA[k], Address);
                        }
                }
        }
}

void signal_handler(int signo)
{
        tick++;
        if(tick == 10000)
        {
                for(int k=0; k<MAXPROC; k++)
                {
                        kill(PCB[k].pid, SIGINT);
                }
                DestroyQueue(qready);
                DestroyQueue(qwait);
                DestroyQueue(qfree);
                printf("Tick : %d, OS exit\n", tick);
                exit(0);
        }
        do_wait();
        RR();
        msgrecieve1();
        make_table();
}

void signal_handler2(int signo)
{
        uProc.exec_time--;
        for(int k = 0; k < NUM; k++)
        {
                uProc.VA[k] = (rand() % 0x100000000);
        }
        msgsend1();
        if(uProc.exec_time == 0)
        {
                uProc.exec_time = (rand() % 10) + 1;
                msgsend();
                kill(uProc.ppid, SIGUSR2);
        }
}

void signal_handler3(int signo)
```

```
{

        msgrecieve();
        for(int k=0; k<MAXPROC; k++)
        {
                if(PCB[k].pid == msg.pid)
                {
                        PCB[k].IO_time = msg.io_time;
                        PCB[k].exec_time = msg.cpu_time;
                        //printf("mother CPU : %d IO : %d, index : %d\n", PCB[k].exec_time,
PCB[k].IO_time, k);
                }
        }
}

int msgsend()
{
        int ret = -1;
        //struct msgbuf msg;
        msg.mtype = 0;
        msg.pid = getpid();
        msg.io_time = uProc.IO;
        msg.cpu_time = uProc.exec_time;
        while(ret == -1)
        {
                ret = msgsnd(msgq, &msg, sizeof(msg), 0);
        }
        return 0;
}

int msgsend1()
{
        int ret = -1;
        msg1.mtype = 0;
        msg1.pid = getpid();
        memcpy(&msg1.VA, &uProc.VA, sizeof(uProc.VA));
        while(ret == -1)
        {
                ret = msgsnd(msgq1, &msg1, sizeof(msg1), 0);
        }
        return 0;
}

int msgrecieve()
{
        int ret = -1;
        //struct msgbuf msg;
        while(ret == -1)
        {
                ret = msgrcv(msgq, &msg, sizeof(msg), 0, MSG_NOERROR);
```

```
        }
        return 0;
}

int msgrecieve1()
{
        int ret = -1;
        while(ret == -1)
        {
                ret = msgrcv(msgq1, &msg1, sizeof(msg1), 0, MSG_NOERROR);
        }
        return 0;
}

void fire(int interval_sec)
{
        struct itimerval new_itimer, old_itimer;
        new_itimer.it_interval.tv_sec = 0;
        new_itimer.it_interval.tv_usec = 100;
        new_itimer.it_value.tv_sec = 0;//interval_sec;
        new_itimer.it_value.tv_usec = 100;
        setitimer(ITIMER_REAL, &new_itimer, &old_itimer);
}
```

## <2_level.tlb.c>

```
#include "t.h"
#include "queue.h"
#include "msg.h"
#define NUM 10

QUEUE *qfree;
QUEUE *qwait;
QUEUE *qready;
int ptr;
int wptr;
int msgq;
int msgq1;
int key = 0x32143153;
int key1 = 0x32140301;
int t=0;

struct sigaction old_sa;
struct sigaction new_sa;
struct msgbuf msg;
struct msgaddr msg1;
pcb_t PCB[MAXPROC];
user uProc;
table **pagetable;
```

```
int *pageframe;

Tlb tlb[64];
Directory **directory;
secondtable **table2;

int main()
{
	memset(&new_sa, 0, sizeof(new_sa));
	msgq = msgget( key, IPC_CREAT | 0666);
	memset(&msg, 0, sizeof(msg));
	msgq1 = msgget( key1, IPC_CREAT | 0666);
	memset(&msg1, 0, sizeof(msg1));
	pageframe = (int*)malloc(sizeof(int) * 0x80000);
	directory = (Directory**)malloc(sizeof(Directory*)*MAXPROC);
	for(int k=0; k<MAXPROC; k++)
	{
		directory[k] = (Directory*)malloc(sizeof(Directory)* 0x400);
	}
	table2 = (secondtable**)malloc(sizeof(secondtable*)*0x400);
	for(int k=0; k<0x400; k++)
	{
		table2[k] = (secondtable*)malloc(sizeof(secondtable)*0x400);
	}
	qwait = CreateQueue();
	qready = CreateQueue();
	qfree = CreateQueue();
	srand((unsigned)time(NULL));
	pid_t pid;

	for(int i=0; i<0x80000; i++)
	{
		Enqueue(qfree, i << 12 | 0x000);
	}
	for(int k=0; k<MAXPROC; k++)
	{
		for(int i=0; i<NUM; i++)
		{
			uProc.VA[i] = (rand() % 0x100000000);
		}
		uProc.exec_time = (rand() % 10) + 1;
		uProc.ppid = getpid();
		uProc.IO = (rand() % 10) + 1;
		pid=fork();
		if(pid<0){
			perror("fork fail");
		}
		else if(pid == 0){//child
			new_sa.sa_handler = &signal_handler2;
			sigaction(SIGUSR1, &new_sa, &old_sa);
```

```
                        while(1);
                        exit(0);
                }
                else{//parent
                        PCB[k].pid = pid;
                        PCB[k].TTBR = k;
                        PCB[k].remaining_tq = DEFAULT_TQ;
                        PCB[k].exec_time = uProc.exec_time;
                }
                Enqueue(qready, k);
        }

        new_sa.sa_handler = &signal_handler;
        sigaction(SIGALRM, &new_sa, &old_sa);
        new_sa.sa_handler = &signal_handler3;
        sigaction(SIGUSR2, &new_sa, &old_sa);
        fire(1);

        while(1);
}

void RR()
{
        if(emptyQueue(qready))return;
        else{
                ptr = qready->front->data;
                if(PCB[ptr].remaining_tq > 0)
                {
                        PCB[ptr].remaining_tq--;
                        PCB[ptr].exec_time--;
                        printf("Tick : %d, PID : %d\tPCB[%d].exec_time : %d\n", tick, PCB[ptr].pid,
ptr, PCB[ptr].exec_time);
                        kill(PCB[ptr].pid, SIGUSR1);
                        if(PCB[ptr].exec_time == 0)
                        {
                                PCB[ptr].remaining_tq = DEFAULT_TQ;
                                Enqueue(qwait, Dequeue(qready));
                                return;
                        }
                }
                if(PCB[ptr].remaining_tq == 0)
                {
                        PCB[ptr].remaining_tq = DEFAULT_TQ;
                        Enqueue(qready, Dequeue(qready));
                        return;
                }
        }
        return;
}
```

```
void do_wait()
{
	if(emptyQueue(qwait))
	{
		return;
	}
	else
	{
		for(int k=0; k<qwait->count; k++)
		{
			wptr = qwait->front->data;
			if(PCB[wptr].IO_time > 0)
			{
				PCB[wptr].IO_time--;
				//printf("PCB[%d].IOtime : %d\n", wptr, PCB[wptr].IO_time);
				if(PCB[wptr].IO_time == 0)
				{
					Enqueue(qready, Dequeue(qwait));
				}
				else if(PCB[wptr].IO_time > 0)
				{
					Enqueue(qwait, Dequeue(qwait));
				}
			}
		}
	}
}

void make_table()
{
	int dirindex;
	int index;
	int offset;
	int dir_valid;
	int page_valid;
	int Address;
	int HIT = 0;
	int tlbindex;

	for(int k=0; k<MAXPROC; k++)
	{
		if(PCB[k].pid == msg1.pid)
		{
			memcpy(&PCB[k].VA, &msg1.VA, sizeof(msg1.VA));
		}
	}

	for(int k=0; k<NUM; k++)
	{
		dirindex = (PCB[ptr].VA[k] & 0xFFC00000) >> 22;//Directory's index
```

```
		index = (PCB[ptr].VA[k] & 0x3FF000) >> 12;//Second table's index
		offset = (PCB[ptr].VA[k] & 0xFFF);
		dir_valid = directory[PCB[ptr].TTBR][dirindex].dn & 0x00000001;//Directory's valid
		page_valid = table2[directory[PCB[ptr].TTBR][dirindex].dn >> 22][index].pfn & 0x00000001;//Second table's valid

		HIT = 0;
		for(int k=0; k<64; k++){//check hit
			if((tlb[k].tag == PCB[ptr].TTBR)&&(tlb[k].full == 1) && (tlb[k].pn == index)){//check TTBR, full, pagenumber
				HIT = 1;//find in TLB
				tlbindex = k;//check tlb' index when hit
			}
		}

		if(HIT){//if hit
			Address = (tlb[tlbindex].pfn << 12) | offset;//get physical address
			printf("HIT! Tick %d, TTBR %d = VA[%d] : 0X%08X -------- Address : 0X%08X\n", tick, PCB[ptr].TTBR, k, PCB[ptr].VA[k], Address);
		}

		else if(!HIT){//if miss
			if(dir_valid == 0)//if not mapping in Directory
			{
				directory[PCB[ptr].TTBR][dirindex].dn = (Dequeue(qfree) & 0xFFC00000) | 0x1;//get Directory value from free list and add valid bit

				if(page_valid == 0){//if not mapping in Second table
					table2[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].pfn = Dequeue(qfree) | 0x1;// get page frame number from free list and add valid bit
					Address = (table2[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].pfn & 0xFFFFF000) | offset;//get physical address
					page_valid = 1;
					printf("Tick %d, TTBR %d = VA[%d] : 0X%08X -------- Address : 0X%08X\n", tick, PCB[ptr].TTBR, k, PCB[ptr].VA[k], Address);
				}
				dir_valid = 1;
			}

			else if(dir_valid == 1)//if mapping in Directory
			{
				if(page_valid == 0){//if not mapping in Second table
					table2[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].pfn = Dequeue(qfree) | 0x1;
					Address = (table2[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].pfn & 0xFFFFF000) | offset;//get physical address
					page_valid = 1;
					printf("Tick %d, TTBR %d = VA[%d] : 0X%08X -------- Address : 0X%08X\n", tick, PCB[ptr].TTBR, k, PCB[ptr].VA[k], Address);
				}
```

```
                                else if(page_valid == 1){//if mapping in Second table
                                        Address = (table2[(directory[PCB[ptr].TTBR][dirindex].dn) >>
22][index].pfn & 0xFFFFF000) | offset;
                                        tlb[t].pn = index;//update tlb page number
                                        tlb[t].pfn = (table2[(directory[PCB[ptr].TTBR][dirindex].dn) >>
22][index].pfn & 0xFFFFF000) >> 12;//update tlb page frame number
                                        tlb[t].full = 1;//update tlb full
                                        tlb[t].tag = PCB[ptr].TTBR;//update tlb TTBR tag
                                        t++;//next tlb index
                                        if(t == 64){
                                                t=0;//go first index
                                        }
                                        printf("Tick %d, TTBR %d = VA[%d] : 0X%08X --------
Address : 0X%08X\n", tick, PCB[ptr].TTBR, k, PCB[ptr].VA[k], Address);
                                }
                        }
                }
        }
}

void signal_handler(int signo)
{
        tick++;
        if(tick == 10000)
        {
                for(int k=0; k<MAXPROC; k++)
                {
                        kill(PCB[k].pid, SIGINT);
                }
                DestroyQueue(qready);
                DestroyQueue(qwait);
                DestroyQueue(qfree);
                printf("Tick : %d, OS exit\n", tick);
                exit(0);
        }
        do_wait();
        RR();
        msgrecieve1();
        make_table();
}

void signal_handler2(int signo)
{
        uProc.exec_time--;
/*      for(int k = 0; k < NUM; k++)
        {
                uProc.VA[k] = (rand() % 0x100000000);
        }
*/      msgsend1();
```

```
	if(uProc.exec_time == 0)
	{
		uProc.exec_time = (rand() % 10) + 1;
		msgsend();
		kill(uProc.ppid, SIGUSR2);
	}
}

void signal_handler3(int signo)
{

	msgrecieve();
	for(int k=0; k<MAXPROC; k++)
	{
		if(PCB[k].pid == msg.pid)
		{
			PCB[k].IO_time = msg.io_time;
			PCB[k].exec_time = msg.cpu_time;
			//printf("mother CPU : %d IO : %d, index : %d\n", PCB[k].exec_time,
PCB[k].IO_time, k);
		}
	}
}

int msgsend()
{
	int ret = -1;
	//struct msgbuf msg;
	msg.mtype = 0;
	msg.pid = getpid();
	msg.io_time = uProc.IO;
	msg.cpu_time = uProc.exec_time;
	while(ret == -1)
	{
		ret = msgsnd(msgq, &msg, sizeof(msg), 0);
	}
	return 0;
}

int msgsend1()
{
	int ret = -1;
	msg1.mtype = 0;
	msg1.pid = getpid();
	memcpy(&msg1.VA, &uProc.VA, sizeof(uProc.VA));
	while(ret == -1)
	{
		ret = msgsnd(msgq1, &msg1, sizeof(msg1), 0);
	}
	return 0;
```

```
}

int msgrecieve()
{
        int ret = -1;
        //struct msgbuf msg;
        while(ret == -1)
        {
                ret = msgrcv(msgq, &msg, sizeof(msg), 0, MSG_NOERROR);
        }
        return 0;
}

int msgrecieve1()
{
        int ret = -1;
        while(ret == -1)
        {
                ret = msgrcv(msgq1, &msg1, sizeof(msg1), 0, MSG_NOERROR);
        }
        return 0;
}

void fire(int interval_sec)
{
        struct itimerval new_itimer, old_itimer;
        new_itimer.it_interval.tv_sec = 0;
        new_itimer.it_interval.tv_usec = 100;
        new_itimer.it_value.tv_sec = 0;//interval_sec;
        new_itimer.it_value.tv_usec = 100;
        setitimer(ITIMER_REAL, &new_itimer, &old_itimer);
}
```

## <one_level_swap.c>

```
#include "t.h"
#include "queue.h"
#include "msg.h"
#define NUM 10

QUEUE *qfree;
QUEUE *qmapping;
QUEUE *qmapping1;
QUEUE *qwait;
QUEUE *qready;
int ptr;
int wptr;
int msgq;
int msgq1;
```

```
int key = 0x32143153;
int key1 = 0x32140301;

struct sigaction old_sa;
struct sigaction new_sa;
struct msgbuf msg;
struct msgaddr msg1;
pcb_t PCB[MAXPROC];
user uProc;
table **pagetable;
frame *pageframe;
disk **blocktable;
frame *block;
int value = 0;
int where = 1;

int main()
{
	memset(&new_sa, 0, sizeof(new_sa));
	msgq = msgget( key, IPC_CREAT | 0666);
	memset(&msg, 0, sizeof(msg));
	msgq1 = msgget( key1, IPC_CREAT | 0666);
	memset(&msg1, 0, sizeof(msg1));
	pageframe = (frame*)malloc(sizeof(frame) * 0x32);
	pagetable = (table**)malloc(sizeof(table*) * MAXPROC);
	for(int i=0; i<MAXPROC; i++)
	{
		pagetable[i] = (table*)malloc(sizeof(table)*0x100000);
	}
	blocktable = (disk**)malloc(sizeof(disk*) * MAXPROC);
	for(int i=0; i<MAXPROC; i++)
	{
		blocktable[i] = (disk*)malloc(sizeof(disk)*0x100000);
	}
	block = (frame*)malloc(sizeof(frame) * 0x10000);
	qwait = CreateQueue();
	qready = CreateQueue();
	qfree = CreateQueue();
	qmapping = CreateQueue();
	qmapping1 = CreateQueue();
	srand((unsigned)time(NULL));
	pid_t pid;

	for(int i=0; i<0x32; i++)
	{
		Enqueue(qfree, i << 12 | 0x000);
	}
	for(int k=0; k<MAXPROC; k++)
	{
		for(int i=0; i<NUM; i++)
```

```
                {
                        uProc.VA[i] = (rand() % 0x100000000);
                }
                uProc.exec_time = (rand() % 10) + 1;
                uProc.ppid = getpid();
                uProc.IO = (rand() % 10) + 1;
                pid=fork();
                if(pid<0){
                        perror("fork fail");
                }
                else if(pid == 0){//child
                        new_sa.sa_handler = &signal_handler2;
                        sigaction(SIGUSR1, &new_sa, &old_sa);
                        while(1);
                        exit(0);
                }
                else{//parent
                        PCB[k].pid = pid;
                        PCB[k].TTBR = k;
                        PCB[k].remaining_tq = DEFAULT_TQ;
                        PCB[k].exec_time = uProc.exec_time;
                }
                Enqueue(qready, k);
        }

        new_sa.sa_handler = &signal_handler;
        sigaction(SIGALRM, &new_sa, &old_sa);
        new_sa.sa_handler = &signal_handler3;
        sigaction(SIGUSR2, &new_sa, &old_sa);
        fire(1);

        while(1);
}

void RR()
{
        if(emptyQueue(qready))return;
        else{
                ptr = qready->front->data;
                if(PCB[ptr].remaining_tq > 0)
                {
                        PCB[ptr].remaining_tq--;
                        PCB[ptr].exec_time--;
                        printf("Tick : %d, PID : %d\tPCB[%d].exec_time : %d\n", tick, PCB[ptr].pid,
ptr, PCB[ptr].exec_time);
                        kill(PCB[ptr].pid, SIGUSR1);
                        if(PCB[ptr].exec_time == 0)
                        {
                                PCB[ptr].remaining_tq = DEFAULT_TQ;
                                Enqueue(qwait, Dequeue(qready));
```

```
                                return;
                        }
                }
                if(PCB[ptr].remaining_tq == 0)
                {
                        PCB[ptr].remaining_tq = DEFAULT_TQ;
                        Enqueue(qready, Dequeue(qready));
                        return;
                }
        }
        return;
}

void do_wait()
{
        if(emptyQueue(qwait))
        {
                return;
        }
        else
        {
                for(int k=0; k<qwait->count; k++)
                {
                        wptr = qwait->front->data;
                        if(PCB[wptr].IO_time > 0)
                        {
                                PCB[wptr].IO_time--;
                                //printf("PCB[%d].IOtime : %d\n", wptr, PCB[wptr].IO_time);
                                if(PCB[wptr].IO_time == 0)
                                {
                                        Enqueue(qready, Dequeue(qwait));
                                }
                                else if(PCB[wptr].IO_time > 0)
                                {
                                        Enqueue(qwait, Dequeue(qwait));
                                }
                        }
                }
        }
}

void make_table()
{
        int index = 0;
        int offset = 0;
        int valid = 0;
        int Address = 0;
        int swap_ttbr = 0;
        int swap_index = 0;
        //int i = 1;
```

```
	for(int k=0; k<MAXPROC; k++)
	{
		if(PCB[k].pid == msg1.pid)
		{
			memcpy(&PCB[k].VA, &msg1.VA, sizeof(msg1.VA));
		}
	}

	for(int k=0; k<NUM; k++)
	{
		index = (PCB[ptr].VA[k] & 0xFFFFF000) >> 12;
		offset = (PCB[ptr].VA[k] & 0xFFF);
		valid = pagetable[PCB[ptr].TTBR][index].pfn & 0x00000001;

		if(valid == 0x0)
		{
			if(qfree->count < 0x20)//emptyQueue(qfree))
			{
				//swapping
				swap_ttbr = Dequeue(qmapping);
				swap_index = Dequeue(qmapping1);
				Enqueue(qfree, (pagetable[swap_ttbr][swap_index].pfn & 0xFFFFF000));
				memcpy(&block[where].data, &pageframe[pagetable[swap_ttbr][swap_index].pfn >> 12].data, sizeof(pageframe[pagetable[swap_ttbr][swap_index].pfn >> 12].data));
				pagetable[swap_ttbr][swap_index].pfn = 0x0;
				blocktable[swap_ttbr][swap_index].where = where;
				where++;
				printf("SWAPPING EVENT!!!!	");
			}
			//need mapping
			pagetable[PCB[ptr].TTBR][index].pfn = Dequeue(qfree) | 0x1;
			Enqueue(qmapping, PCB[ptr].TTBR);
			Enqueue(qmapping1, index);
			if(blocktable[PCB[ptr].TTBR][index].where != 0x0)
			{
				memcpy(&pageframe[pagetable[PCB[ptr].TTBR][index].pfn >> 12].data, &block[blocktable[PCB[ptr].TTBR][index].where].data, sizeof(block[blocktable[PCB[ptr].TTBR][index].where].data));
				printf("swapin %x Tick %d, TTBR %d = VA[%d] : 0X%08X -------- pageframe[0X%08X] = %d\n", blocktable[PCB[ptr].TTBR][index].where, tick, PCB[ptr].TTBR, k, PCB[ptr].VA[k], pagetable[PCB[ptr].TTBR][index].pfn >> 12, pageframe[pagetable[PCB[ptr].TTBR][index].pfn >> 12].data[offset]);
			}
			else
			{
				pageframe[pagetable[PCB[ptr].TTBR][index].pfn >> 12].data[offset] = value;
```

```
                                value = value + 1;
                                printf("Tick %d, TTBR %d = VA[%d] : 0X%08X -------- pageframe[0X%05X] = %d\n", tick, PCB[ptr].TTBR, k, PCB[ptr].VA[k], pagetable[PCB[ptr].TTBR][index].pfn >> 12, pageframe[pagetable[PCB[ptr].TTBR][index].pfn >> 12].data[offset]);
                        }
                }
                else
                {
                        printf("Tick %d, TTBR %d = VA[%d] : 0X%08X -------- pageframe[0X%05X] = %d\n", tick, PCB[ptr].TTBR, k, PCB[ptr].VA[k], pagetable[PCB[ptr].TTBR][index].pfn >> 12, pageframe[pagetable[PCB[ptr].TTBR][index].pfn >> 12].data[offset]);
                }
        }
}

void signal_handler(int signo)
{
        tick++;
        if(tick == 10000)
        {
                for(int k=0; k<MAXPROC; k++)
                {
                        kill(PCB[k].pid, SIGINT);
                }
                DestroyQueue(qready);
                DestroyQueue(qwait);
                DestroyQueue(qfree);
                printf("Tick : %d, OS exit\n", tick);
                exit(0);
        }
        do_wait();
        RR();
        msgrecieve1();
        make_table();
}

void signal_handler2(int signo)
{
        uProc.exec_time--;
        //for(int k = 0; k < NUM; k++)
        //{
        //      uProc.VA[k] = (rand() % 0x100000000);
        //}
        msgsend1();
        if(uProc.exec_time == 0)
        {
                uProc.exec_time = (rand() % 10) + 1;
                msgsend();
                kill(uProc.ppid, SIGUSR2);
```

```
        }
}

void signal_handler3(int signo)
{

        msgrecieve();
        for(int k=0; k<MAXPROC; k++)
        {
                if(PCB[k].pid == msg.pid)
                {
                        PCB[k].IO_time = msg.io_time;
                        PCB[k].exec_time = msg.cpu_time;
                        //printf("mother CPU : %d IO : %d, index : %d\n", PCB[k].exec_time,
PCB[k].IO_time, k);
                }
        }
}

int msgsend()
{
        int ret = -1;
        //struct msgbuf msg;
        msg.mtype = 0;
        msg.pid = getpid();
        msg.io_time = uProc.IO;
        msg.cpu_time = uProc.exec_time;
        while(ret == -1)
        {
                ret = msgsnd(msgq, &msg, sizeof(msg), 0);
        }
        return 0;
}

int msgsend1()
{
        int ret = -1;
        msg1.mtype = 0;
        msg1.pid = getpid();
        memcpy(&msg1.VA, &uProc.VA, sizeof(uProc.VA));
        while(ret == -1)
        {
                ret = msgsnd(msgq1, &msg1, sizeof(msg1), 0);
        }
        return 0;
}

int msgrecieve()
{
        int ret = -1;
```

```
        //struct msgbuf msg;
        while(ret == -1)
        {
                ret = msgrcv(msgq, &msg, sizeof(msg), 0, MSG_NOERROR);
        }
        return 0;
}

int msgrecieve1()
{
        int ret = -1;
        while(ret == -1)
        {
                ret = msgrcv(msgq1, &msg1, sizeof(msg1), 0, MSG_NOERROR);
        }
        return 0;
}

void fire(int interval_sec)
{
        struct itimerval new_itimer, old_itimer;
        new_itimer.it_interval.tv_sec = 0;
        new_itimer.it_interval.tv_usec = 1000;
        new_itimer.it_value.tv_sec = 0;//interval_sec;
        new_itimer.it_value.tv_usec = 1000;
        setitimer(ITIMER_REAL, &new_itimer, &old_itimer);
}
```

## <two_level_swap1.c>

```
#include "t.h"
#include "queue.h"
#include "msg.h"
#define NUM 10

QUEUE *qfree;
QUEUE *qframe;
QUEUE *qwait;
QUEUE *qready;
QUEUE *qmapping;
QUEUE *qmapping1;
QUEUE *mapping;
QUEUE *mapping1;
int ptr;
int wptr;
int msgq;
int msgq1;
int key = 0x32143153;
int key1 = 0x32140301;
```

```
struct sigaction old_sa;
struct sigaction new_sa;
struct msgbuf msg;
struct msgaddr msg1;
pcb_t PCB[MAXPROC];
user uProc;
frame *pageframe;
disk **blocktable;
frame *block;
Directory **directory;
secondtable **table2;

int value = 0;
int where = 1;

int main()
{
	memset(&new_sa, 0, sizeof(new_sa));
	msgq = msgget( key, IPC_CREAT | 0666);
	memset(&msg, 0, sizeof(msg));
	msgq1 = msgget( key1, IPC_CREAT | 0666);
	memset(&msg1, 0, sizeof(msg1));
	pageframe = (frame*)malloc(sizeof(frame) * 0x100);
	directory = (Directory**)malloc(sizeof(Directory*)*MAXPROC);
	for(int k=0; k<MAXPROC; k++)
	{
		directory[k] = (Directory*)malloc(sizeof(Directory)* 0x400);
	}
	table2 = (secondtable**)malloc(sizeof(secondtable*)*0x400);
	for(int k=0; k<0x400; k++)
	{
		table2[k] = (secondtable*)malloc(sizeof(secondtable)*0x400);
	}
	blocktable = (disk**)malloc(sizeof(disk*) * MAXPROC);
	for(int i=0; i<MAXPROC; i++)
	{
		blocktable[i] = (disk*)malloc(sizeof(disk)*0x10000000);
	}
	block = (frame*)malloc(sizeof(frame) * 0x10000);
	qwait = CreateQueue();
	qready = CreateQueue();
	qfree = CreateQueue();
	qframe = CreateQueue();
	qmapping = CreateQueue();
	qmapping1 = CreateQueue();
	mapping = CreateQueue();
	mapping1 = CreateQueue();
	srand((unsigned)time(NULL));
	pid_t pid;
```

```
	for(int i=0; i<0x32; i++)
	{
		if(i%4)
		Enqueue(qfree, i << 12 | 0x000);
		else
		Enqueue(qframe, i << 12 | 0x000);
	}
	for(int k=0; k<MAXPROC; k++)
	{
		for(int i=0; i<NUM; i++)
		{
			uProc.VA[i] = (rand() % 0x100000000);
		}
		uProc.exec_time = (rand() % 10) + 1;
		uProc.ppid = getpid();
		uProc.IO = (rand() % 10) + 1;
		pid=fork();
		if(pid<0){
			perror("fork fail");
		}
		else if(pid == 0){//child
			new_sa.sa_handler = &signal_handler2;
			sigaction(SIGUSR1, &new_sa, &old_sa);
			while(1);
			exit(0);
		}
		else{//parent
			PCB[k].pid = pid;
			PCB[k].TTBR = k;
			PCB[k].remaining_tq = DEFAULT_TQ;
			PCB[k].exec_time = uProc.exec_time;
		}
		Enqueue(qready, k);
	}

	new_sa.sa_handler = &signal_handler;
	sigaction(SIGALRM, &new_sa, &old_sa);
	new_sa.sa_handler = &signal_handler3;
	sigaction(SIGUSR2, &new_sa, &old_sa);
	fire(1);

	while(1);
}

void RR()
{
	if(emptyQueue(qready))return;
	else{
		ptr = qready->front->data;
```

```
		if(PCB[ptr].remaining_tq > 0)
		{
			PCB[ptr].remaining_tq--;
			PCB[ptr].exec_time--;
			printf("Tick : %d, PID : %d\tPCB[%d].exec_time : %d\n", tick, PCB[ptr].pid,
ptr, PCB[ptr].exec_time);
			kill(PCB[ptr].pid, SIGUSR1);
			if(PCB[ptr].exec_time == 0)
			{
				PCB[ptr].remaining_tq = DEFAULT_TQ;
				Enqueue(qwait, Dequeue(qready));
				return;
			}
		}
		if(PCB[ptr].remaining_tq == 0)
		{
			PCB[ptr].remaining_tq = DEFAULT_TQ;
			Enqueue(qready, Dequeue(qready));
			return;
		}
	}
	return;
}

void do_wait()
{
	if(emptyQueue(qwait))
	{
		return;
	}
	else
	{
		for(int k=0; k<qwait->count; k++)
		{
			wptr = qwait->front->data;
			if(PCB[wptr].IO_time > 0)
			{
				PCB[wptr].IO_time--;
				//printf("PCB[%d].IOtime : %d\n", wptr, PCB[wptr].IO_time);
				if(PCB[wptr].IO_time == 0)
				{
					Enqueue(qready, Dequeue(qwait));
				}
				else if(PCB[wptr].IO_time > 0)
				{
					Enqueue(qwait, Dequeue(qwait));
				}
			}
		}
	}
```

```
}

void make_table()
{
	int dirindex = 0;
	int index = 0;
	int offset = 0;
	int dir_valid = 0;
	int page_valid = 0;
	//int Address = 0;
	int swap_ttbr = 0;
	int swap_index = 0;
	int swap1_ttbr = 0;
	int swap1_index = 0;

	for(int k=0; k<MAXPROC; k++)
	{
		if(PCB[k].pid == msg1.pid)
		{
			memcpy(&PCB[k].VA, &msg1.VA, sizeof(msg1.VA));
		}
	}

	for(int k=0; k<NUM; k++)
	{
		dirindex = (PCB[ptr].VA[k] & 0xFFC00000) >> 22;
		index = (PCB[ptr].VA[k] & 0x3FF000) >> 12;
		offset = (PCB[ptr].VA[k] & 0xFFF);
		dir_valid = directory[PCB[ptr].TTBR][dirindex].dn & 0x00000001;
		page_valid = table2[directory[PCB[ptr].TTBR][dirindex].dn >> 22][index].pfn &
0x00000001;

		if(dir_valid == 0x0)
		{
			//need mapping
			if(emptyQueue(qframe))
			{
				//swapping
				swap1_ttbr = Dequeue(mapping);
				swap1_index = Dequeue(mapping1);
				Enqueue(qframe, (directory[swap1_ttbr][swap1_index].dn &
0xFFFFF000));
				memcpy(&block[where].data,
&pageframe[directory[swap1_ttbr][swap1_index].dn >> 12].data,
sizeof(pageframe[directory[swap1_ttbr][swap1_index].dn >> 12].data));
				directory[swap1_ttbr][swap1_index].dn = 0x0;
				blocktable[swap1_ttbr][swap1_index].where = where;
				where++;
				printf("SWAPPING EVENT directory!!!! ");
			}
```

```
                        directory[PCB[ptr].TTBR][dirindex].dn = Dequeue(qframe) | 0x1;
                        Enqueue(mapping, PCB[ptr].TTBR);
                        Enqueue(mapping1, dirindex);
                        dir_valid = 0x1;
                        if(blocktable[PCB[ptr].TTBR][index].where != 0x0)
                        {
                                memcpy(&pageframe[directory[PCB[ptr].TTBR][dirindex].dn >>
12].data, &block[blocktable[PCB[ptr].TTBR][dirindex].where].data,
sizeof(block[blocktable[PCB[ptr].TTBR][dirindex].where].data));
                        }
                        else
                        {
                                pageframe[(directory[PCB[ptr].TTBR][dirindex].dn) >> 12].data[offset] =
table2[(directory[PCB[ptr].TTBR][dirindex].dn) >> 12][index].pfn;
                        }
                }
                if(dir_valid == 0x1)
                {
                        if(page_valid == 0x0)
                        {
                                if(emptyQueue(qfree))
                                {
                                        //swapping
                                        swap_ttbr = Dequeue(qmapping);
                                        swap_index = Dequeue(qmapping1);
                                        Enqueue(qfree, (table2[swap_ttbr][swap_index].pfn &
0xFFFFF000));
                                        memcpy(&block[where].data,
&pageframe[table2[swap_ttbr][swap_index].pfn >> 12].data,
sizeof(pageframe[table2[swap_ttbr][swap_index].pfn >> 12].data));
                                        table2[swap_ttbr][swap_index].pfn = 0x0;
                                        blocktable[swap_ttbr][swap_index].where = where;
                                        where++;
                                        printf("SWAPPING EVENT pagetable!!!! ");
                                }
                                //printf("no mapping\t");
                                table2[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].pfn =
Dequeue(qfree) | 0x1;
                                Enqueue(qmapping, (directory[PCB[ptr].TTBR][dirindex].dn) >> 22);
                                Enqueue(qmapping1, index);
                                page_valid = 0x1;
                                if(blocktable[(directory[PCB[ptr].TTBR][dirindex].dn) >>
22][index].where != 0x0)
                                {

memcpy(&pageframe[table2[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].pfn >> 12].data,
&block[blocktable[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].where].data,
sizeof(block[blocktable[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].where].data));
                                        printf("swapin %x Tick %d, TTBR %d = VA[%d] : 0X%08X
-------- pageframe[0X%05X] = %d\n", blocktable[(directory[PCB[ptr].TTBR][dirindex].dn) >>
```

```
22][index].where, tick, PCB[ptr].TTBR, k, PCB[ptr].VA[k], table2[(directory[PCB[ptr].TTBR][dirindex].dn)
>> 22][index].pfn >> 12, pageframe[table2[PCB[ptr].TTBR][index].pfn >> 12].data[offset]);
                                        }
                                        else
                                        {
                                                pageframe[table2[(directory[PCB[ptr].TTBR][dirindex].dn)      >>
22][index].pfn >> 12].data[offset] = value;
                                                value = value + 1;
                                                printf("Tick %d, TTBR %d = VA[%d] : 0X%08X --------
pageframe[0X%05X] = %d\n", tick, PCB[ptr].TTBR, k, PCB[ptr].VA[k],
table2[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].pfn >> 12,
pageframe[table2[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].pfn >> 12].data[offset]);
                                        }
                                }
                                else if(page_valid == 0x1)
                                {
                                        printf("Tick %d, TTBR %d = VA[%d] : 0X%08X --------
pageframe[0X%05X] = %d\n", tick, PCB[ptr].TTBR, k, PCB[ptr].VA[k],
table2[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].pfn >> 12,
pageframe[table2[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].pfn >> 12].data[offset]);
                                }
                        }
                }
}

void signal_handler(int signo)
{
        tick++;
        if(tick == 10000)
        {
                for(int k=0; k<MAXPROC; k++)
                {
                        kill(PCB[k].pid, SIGINT);
                }
                DestroyQueue(qready);
                DestroyQueue(qwait);
                DestroyQueue(qfree);
                printf("Tick : %d, OS exit\n", tick);
                exit(0);
        }
        do_wait();
        RR();
        msgrecieve1();
        make_table();
}

void signal_handler2(int signo)
{
        uProc.exec_time--;
        //for(int k = 0; k < NUM; k++)
```

```
	//{
	//	uProc.VA[k] = (rand() % 0x100000000);
	//}
	msgsend1();
	if(uProc.exec_time == 0)
	{
		uProc.exec_time = (rand() % 10) + 1;
		msgsend();
		kill(uProc.ppid, SIGUSR2);
	}
}

void signal_handler3(int signo)
{

	msgrecieve();
	for(int k=0; k<MAXPROC; k++)
	{
		if(PCB[k].pid == msg.pid)
		{
			PCB[k].IO_time = msg.io_time;
			PCB[k].exec_time = msg.cpu_time;
			//printf("mother CPU : %d IO : %d, index : %d\n", PCB[k].exec_time,
PCB[k].IO_time, k);
		}
	}
}

int msgsend()
{
	int ret = -1;
	//struct msgbuf msg;
	msg.mtype = 0;
	msg.pid = getpid();
	msg.io_time = uProc.IO;
	msg.cpu_time = uProc.exec_time;
	while(ret == -1)
	{
		ret = msgsnd(msgq, &msg, sizeof(msg), 0);
	}
	return 0;
}

int msgsend1()
{
	int ret = -1;
	msg1.mtype = 0;
	msg1.pid = getpid();
	memcpy(&msg1.VA, &uProc.VA, sizeof(uProc.VA));
	while(ret == -1)
```

```
        {
                ret = msgsnd(msgq1, &msg1, sizeof(msg1), 0);
        }
        return 0;
}

int msgrecieve()
{
        int ret = -1;
        //struct msgbuf msg;
        while(ret == -1)
        {
                ret = msgrcv(msgq, &msg, sizeof(msg), 0, MSG_NOERROR);
        }
        return 0;
}

int msgrecieve1()
{
        int ret = -1;
        while(ret == -1)
        {
                ret = msgrcv(msgq1, &msg1, sizeof(msg1), 0, MSG_NOERROR);
        }
        return 0;
}

void fire(int interval_sec)
{
        struct itimerval new_itimer, old_itimer;
        new_itimer.it_interval.tv_sec = 0;
        new_itimer.it_interval.tv_usec = 100;
        new_itimer.it_value.tv_sec = 0;//interval_sec;
        new_itimer.it_value.tv_usec = 100;
        setitimer(ITIMER_REAL, &new_itimer, &old_itimer);
}
```

## <two_level_swap2.c>

```
#include "t.h"
#include "queue.h"
#include "msg.h"
#define NUM 10

QUEUE *qfree;
QUEUE *qframe;
QUEUE *qwait;
QUEUE *qready;
QUEUE *qmapping;
```

```
QUEUE *qmapping1;
int ptr;
int wptr;
int msgq;
int msgq1;
int key = 0x32143153;
int key1 = 0x32140301;

struct sigaction old_sa;
struct sigaction new_sa;
struct msgbuf msg;
struct msgaddr msg1;
pcb_t PCB[MAXPROC];
user uProc;
frame *pageframe;
disk **blocktable;
frame *block;
Directory **directory;
secondtable **table2;

int value = 0;
int where = 1;

int main()
{
        memset(&new_sa, 0, sizeof(new_sa));
        msgq = msgget( key, IPC_CREAT | 0666);
        memset(&msg, 0, sizeof(msg));
        msgq1 = msgget( key1, IPC_CREAT | 0666);
        memset(&msg1, 0, sizeof(msg1));
        pageframe = (frame*)malloc(sizeof(frame) * 0x100);
        directory = (Directory**)malloc(sizeof(Directory*)*MAXPROC);
        for(int k=0; k<MAXPROC; k++)
        {
                directory[k] = (Directory*)malloc(sizeof(Directory)* 0x400);
        }
        table2 = (secondtable**)malloc(sizeof(secondtable*)*0x400);
        for(int k=0; k<0x400; k++)
        {
                table2[k] = (secondtable*)malloc(sizeof(secondtable)*0x400);
        }
        blocktable = (disk**)malloc(sizeof(disk*) * MAXPROC);
        for(int i=0; i<MAXPROC; i++)
        {
                blocktable[i] = (disk*)malloc(sizeof(disk)*0x100000);
        }
        block = (frame*)malloc(sizeof(frame) * 0x10000);
        qwait = CreateQueue();
        qready = CreateQueue();
        qfree = CreateQueue();
```

```
	qframe = CreateQueue();
	qmapping = CreateQueue();
	qmapping1 = CreateQueue();
	srand((unsigned)time(NULL));
	pid_t pid;

	for(int i=0; i<0x100; i++)
	{
		if(i%4)
		Enqueue(qfree, i << 12 | 0x000);
		else
		Enqueue(qframe, i << 12 | 0x000);
	}
	for(int k=0; k<MAXPROC; k++)
	{
		for(int i=0; i<NUM; i++)
		{
			uProc.VA[i] = (rand() % 0x100000000);
		}
		uProc.exec_time = (rand() % 10) + 1;
		uProc.ppid = getpid();
		uProc.IO = (rand() % 10) + 1;
		pid=fork();
		if(pid<0){
			perror("fork fail");
		}
		else if(pid == 0){//child
			new_sa.sa_handler = &signal_handler2;
			sigaction(SIGUSR1, &new_sa, &old_sa);
			while(1);
			exit(0);
		}
		else{//parent
			PCB[k].pid = pid;
			PCB[k].TTBR = k;
			PCB[k].remaining_tq = DEFAULT_TQ;
			PCB[k].exec_time = uProc.exec_time;
		}
		Enqueue(qready, k);
	}

	new_sa.sa_handler = &signal_handler;
	sigaction(SIGALRM, &new_sa, &old_sa);
	new_sa.sa_handler = &signal_handler3;
	sigaction(SIGUSR2, &new_sa, &old_sa);
	fire(1);

	while(1);
}
```

```
void RR()
{
	if(emptyQueue(qready))return;
	else{
		ptr = qready->front->data;
		if(PCB[ptr].remaining_tq > 0)
		{
			PCB[ptr].remaining_tq--;
			PCB[ptr].exec_time--;
			printf("Tick : %d, PID : %d\tPCB[%d].exec_time : %d\n", tick, PCB[ptr].pid,
ptr, PCB[ptr].exec_time);
			kill(PCB[ptr].pid, SIGUSR1);
			if(PCB[ptr].exec_time == 0)
			{
				PCB[ptr].remaining_tq = DEFAULT_TQ;
				Enqueue(qwait, Dequeue(qready));
				return;
			}
		}
		if(PCB[ptr].remaining_tq == 0)
		{
			PCB[ptr].remaining_tq = DEFAULT_TQ;
			Enqueue(qready, Dequeue(qready));
			return;
		}
	}
	return;
}

void do_wait()
{
	if(emptyQueue(qwait))
	{
		return;
	}
	else
	{
		for(int k=0; k<qwait->count; k++)
		{
			wptr = qwait->front->data;
			if(PCB[wptr].IO_time > 0)
			{
				PCB[wptr].IO_time--;
				//printf("PCB[%d].IOtime : %d\n", wptr, PCB[wptr].IO_time);
				if(PCB[wptr].IO_time == 0)
				{
					Enqueue(qready, Dequeue(qwait));
				}
				else if(PCB[wptr].IO_time > 0)
				{
```

```
                                        Enqueue(qwait, Dequeue(qwait));
                                }
                        }
                }
        }
}

void make_table()
{
        int dirindex = 0;
        int index = 0;
        int offset = 0;
        int dir_valid = 0;
        int page_valid = 0;
        int swap_ttbr = 0;
        int swap_index = 0;

        for(int k=0; k<MAXPROC; k++)
        {
                if(PCB[k].pid == msg1.pid)
                {
                        memcpy(&PCB[k].VA, &msg1.VA, sizeof(msg1.VA));
                }
        }

        for(int k=0; k<NUM; k++)
        {
                dirindex = (PCB[ptr].VA[k] & 0xFFC00000) >> 22;
                index = (PCB[ptr].VA[k] & 0x3FF000) >> 12;
                offset = (PCB[ptr].VA[k] & 0xFFF);
                dir_valid = directory[PCB[ptr].TTBR][dirindex].dn & 0x00000001;
                page_valid = table2[directory[PCB[ptr].TTBR][dirindex].dn >> 22][index].pfn &
0x00000001;

                if(dir_valid == 0x0)
                {
                        //need mapping
                        if(emptyQueue(qframe))
                        {
                                //swapping
                                swap_ttbr = Dequeue(qmapping);
                                swap_index = Dequeue(qmapping1);
                                Enqueue(qframe, (table2[swap_ttbr][swap_index].pfn & 0xFFFFF000));
                                memcpy(&block[where].data,
&pageframe[table2[swap_ttbr][swap_index].pfn >> 12].data,
sizeof(pageframe[table2[swap_ttbr][swap_index].pfn >> 12].data));
                                table2[swap_ttbr][swap_index].pfn = 0x0;
                                blocktable[swap_ttbr][swap_index].where = where;
                                where++;
                                printf("SWAPPING EVENT directory!!!!      ");
```

```
                        }
                        directory[PCB[ptr].TTBR][dirindex].dn = Dequeue(qframe) | 0x1;
                        dir_valid = 0x1;
                        pageframe[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22].data[offset] =
table2[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].pfn;
                }
                if(dir_valid == 0x1)
                {
                        if(page_valid == 0x0)
                        {
                                if(emptyQueue(qfree))
                                {
                                        //swapping
                                        swap_ttbr = Dequeue(qmapping);
                                        swap_index = Dequeue(qmapping1);
                                        Enqueue(qfree, (table2[swap_ttbr][swap_index].pfn &
0xFFFFF000));
                                        memcpy(&block[where].data,
&pageframe[table2[swap_ttbr][swap_index].pfn >> 12].data,
sizeof(pageframe[table2[swap_ttbr][swap_index].pfn >> 12].data));
                                        table2[swap_ttbr][swap_index].pfn = 0x0;
                                        blocktable[swap_ttbr][swap_index].where = where;
                                        where++;
                                        printf("SWAPPING EVENT pagetable!!!! ");
                                }
                                //printf("no mapping\t");
                                table2[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].pfn =
Dequeue(qfree) | 0x1;
                                Enqueue(qmapping, ((directory[PCB[ptr].TTBR][dirindex].dn) >> 22));
                                Enqueue(qmapping1, index);
                                page_valid = 0x1;
                                if(blocktable[(directory[PCB[ptr].TTBR][dirindex].dn) >>
22][index].where != 0x0)
                                {

memcpy(&pageframe[table2[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].pfn >> 12].data,
&block[blocktable[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].where].data,
sizeof(block[blocktable[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].where].data));
                                }
                                else
                                {
                                        pageframe[table2[(directory[PCB[ptr].TTBR][dirindex].dn) >>
22][index].pfn >> 12].data[offset] = value;
                                        value = value + 1;
                                }
                                printf("Tick %d, TTBR %d = VA[%d] : 0X%08X --------
pageframe[0X%05X] = %d\n", tick, PCB[ptr].TTBR, k, PCB[ptr].VA[k],
table2[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].pfn >> 12,
pageframe[table2[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].pfn >> 12].data[offset]);
                        }
```

```
                        else if(page_valid == 0x1)
                        {
                                printf("Tick %d, TTBR %d = VA[%d] : 0X%08X --------
pageframe[0X%05X] = %d\n", tick, PCB[ptr].TTBR, k, PCB[ptr].VA[k],
table2[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].pfn >> 12,
pageframe[table2[(directory[PCB[ptr].TTBR][dirindex].dn) >> 22][index].pfn >> 12].data[offset]);
                        }
                }
        }
}

void signal_handler(int signo)
{
        tick++;
        if(tick == 10000)
        {
                for(int k=0; k<MAXPROC; k++)
                {
                        kill(PCB[k].pid, SIGINT);
                }
                DestroyQueue(qready);
                DestroyQueue(qwait);
                DestroyQueue(qfree);
                printf("Tick : %d, OS exit\n", tick);
                exit(0);
        }
        do_wait();
        RR();
        msgrecieve1();
        make_table();
}

void signal_handler2(int signo)
{
        uProc.exec_time--;
        //for(int k = 0; k < NUM; k++)
        //{
        //      uProc.VA[k] = (rand() % 0x100000000);
        //}
        msgsend1();
        if(uProc.exec_time == 0)
        {
                uProc.exec_time = (rand() % 10) + 1;
                msgsend();
                kill(uProc.ppid, SIGUSR2);
        }
}

void signal_handler3(int signo)
{
```

```
		msgrecieve();
		for(int k=0; k<MAXPROC; k++)
		{
			if(PCB[k].pid == msg.pid)
			{
				PCB[k].IO_time = msg.io_time;
				PCB[k].exec_time = msg.cpu_time;
				//printf("mother CPU : %d IO : %d, index : %d\n", PCB[k].exec_time,
PCB[k].IO_time, k);
			}
		}
}

int msgsend()
{
		int ret = -1;
		//struct msgbuf msg;
		msg.mtype = 0;
		msg.pid = getpid();
		msg.io_time = uProc.IO;
		msg.cpu_time = uProc.exec_time;
		while(ret == -1)
		{
			ret = msgsnd(msgq, &msg, sizeof(msg), 0);
		}
		return 0;
}

int msgsend1()
{
		int ret = -1;
		msg1.mtype = 0;
		msg1.pid = getpid();
		memcpy(&msg1.VA, &uProc.VA, sizeof(uProc.VA));
		while(ret == -1)
		{
			ret = msgsnd(msgq1, &msg1, sizeof(msg1), 0);
		}
		return 0;
}

int msgrecieve()
{
		int ret = -1;
		//struct msgbuf msg;
		while(ret == -1)
		{
			ret = msgrcv(msgq, &msg, sizeof(msg), 0, MSG_NOERROR);
		}
```

```
        return 0;
}

int msgrecieve1()
{
        int ret = -1;
        while(ret == -1)
        {
                ret = msgrcv(msgq1, &msg1, sizeof(msg1), 0, MSG_NOERROR);
        }
        return 0;
}

void fire(int interval_sec)
{
        struct itimerval new_itimer, old_itimer;
        new_itimer.it_interval.tv_sec = 0;
        new_itimer.it_interval.tv_usec = 1000;
        new_itimer.it_value.tv_sec = 0;//interval_sec;
        new_itimer.it_value.tv_usec = 1000;
        setitimer(ITIMER_REAL, &new_itimer, &old_itimer);
}
```