# 2019년 2학기 시스템 프로그래밍 과제 #2

## 과제 개요

### 1 과제 목적

- 리눅스 커널에서 메모리 관리 기법을 이해한다.
- 리눅스 커널의 인터럽트 처리 과정에서 bottom-half에 해당하는 tasklet을 직접 사용해본다.

### 2 제출 기한

- 2019년 12월 13일 낮 12시 (정오)까지 (보고서 출력본과 파일 모두 해당)

### 3 제출 방법

- 보고서 출력본: 제 4공학관 D814호 앞 과제 제출함
- 보고서 파일(pdf) 및 실습 과제 소스 파일(tar.bz2): 과목 홈페이지 과제 제출 게시판

### 4 제한 사항

- 리눅스는 Ubuntu **18.04.3 LTS를 권장**한다.
- 64bit version을 사용하며, VMware, VirtualBox 등의 가상머신에서 작업하는 것을 권장한다.
- 대상 커널은 **반드시 5.0.0 버전**을 이용한다. (Ubuntu 18.04.3 LTS 설치시 기본 커널 버전)
- 1인 1프로젝트이며 각자 환경에서 작업한다.

### 5 유의 사항

- **적절한 사유 없이 Delay 될 경우 절대 받지 않음**
- 그림, 코드 조각을 포함한 모든 참고 자료의 내용은 반드시 출처를 명시
- kernel 을 사용한 시스템 프로그래밍 결과는 사용자 환경에 따라 다르게 적용될 수 있으므로 시스템의 환경을 반드시 명시
    - 실습 과제 수행 보고서에 uname -a 결과 화면 반드시 첨부
- 제출 파일 생성 방법을 지키지 않을 경우 감점 또는 0점 처리
- 타 수강생의 보고서 및 과제를 카피 또는 수정하여 제출하였을 경우 모두 0점 처리
- 과제에 대한 문의 사항이 있을 경우 게시판으로 문의한다.

# 과제 세부 사항

## 보고서 과제

보고서는 사전 조사 보고서와 실습 과제 수행 보고서로 구성한다.

### 1 사전 조사 보고서

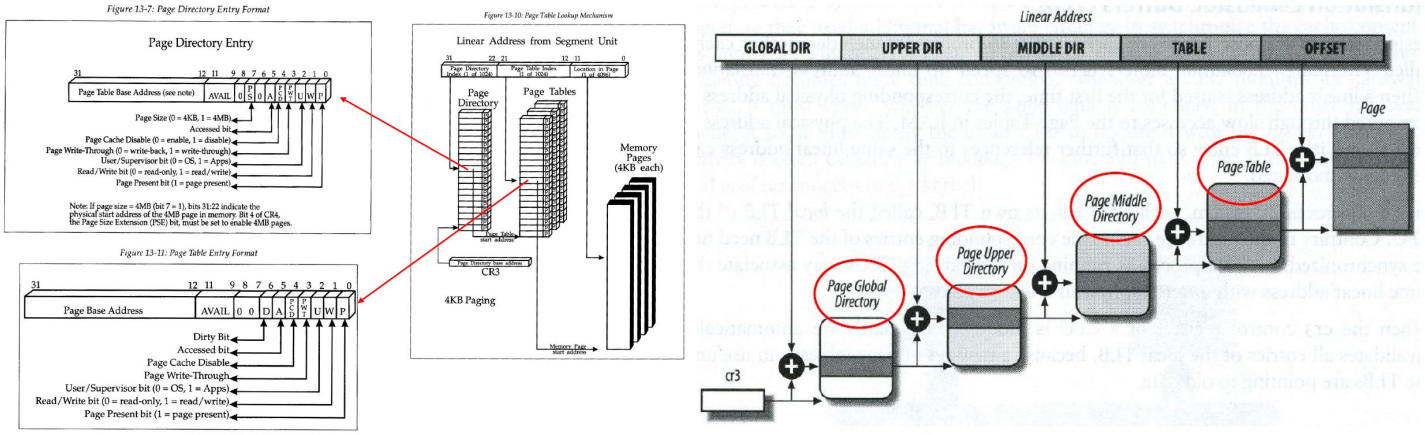
- 리눅스 커널 2.6.10 버전과 5.0.0 버전의 커널 소스를 **메모리 페이징 기법**과 관련된 내용을 중심으로 분석한다.
    - Linux Kernel은 2.6.10 버전 까지는 32bit 물리적 메모리 를 지원하기 위해서 3-level paging 기법이 적용되어 있었으나 이 후 버전에는 64bit 물리 메모리를 지원하기 위해서 4-level paging 기법을 지원하게 되었다.
    - 3-level paging과 4-level paging을 비교하여 분석한다. 그리고 리눅스가 하위 level paging을 어떻게 지원하는지 조사한다.
    - Page Global Directory, Page Upper Directory, Page Middle Directory, Page Table Entry의 관계를 Linear Address와 Physical Address의 관계와 함께 설명한다. 이때, 이들에 관계된 **자료 구조, 매크로, 함수들과 매커니즘의 흐름**등을 **각 항목 별로 나누어** 자세하게 설명한다.
    - Process 별로 할당된 Page 정보를 이용하여 물리 메모리를 참조하는 과정을 조사한다.
    - 이외에 기타 필요하다고 생각되는 정보를 추가로 조사한다.
    - 이를 설명하는데 구체적인 설명과 함께 **반드시 순서도나 그림 또는 도표**로 표현한다.
- 사용자 프로그램에서 일정 크기의 메모리를 할당 했다가 해제 할 시 커널 내부에서 일어나는 동작을 설명한다.
- Linux에서 인터럽트 지연 처리를 위한 **Tasklet의 자료구조**에 대해서 분석하고 이것이 수행되는 매커니즘을 분석하여 설명한다.
- Reference 기재

### 2 실습 과제 수행 보고서

- 과제 수행 시스템 환경 명시
    - OS, CPU, 메모리 정보 등
    - `uname -a` 명령 실행 결과를 캡쳐하여 서두에 첨부할 것
- 커널 모듈 작성 보고서 **(자세한 내용은 실습 과제 세부 사항 참고)**
    - 프로그램 구현에 사용한 커널 함수 및 자료구조에 대해 설명 (메모리 관련 함수, Tasklet 관련 자료구조)
    - 작성한 프로그램의 전체 구조 및 동작 과정을 설명
    - 동작과정을 순서도나 그림 또는 도표로 표현해 볼 것
- 문제점, 해결방법, 애로사항 등 기재
- Reference 기재

# 실습 과제

- 리눅스 커널에서 프로세스가 실행되면서 할당 받는 메모리 주소공간에 대해 이해하고, 가상메모리가 실제 물리 메모리와 연결되는 페이징 기법을 이해 한다. 이를 위해 시스템 콜과 사용자 프로그램을 이용하여 현재의 프로세스가 할당 받은 메모리 공간을 직접 확인해보고 이 중 하나의 영역이 페이징 된 결과를 실제로 추적하여 1차 과제에서 사용했던  proc 파일 시스템을 통해 출력해 본다.




- 위의 왼쪽 그림은 강의 자료 L08에 소개된 자료로서 가상 주소에서 Page Global Directory Entry, Page Table Entry 까지의 관계를 요약한 그림이다. 위의 관계를 잘 이해한뒤 64bit system 상에서 4-level paging 기법에 따라 특정 프로세스의 메모리 구역을 확인해 본다. 4-level paging 기법의 각 level 별 관계는 강의 자료 L08 18 page에서 확인할 수 있으며 위의 오른쪽 그림과 같다.
- 해당 프로세스의 Code 영역의 가상 메모리 주소를 바탕으로 Page Global Directory, Page Upper Directory, Page Middle Directory, Page Table Entry 를 실제로 조사해 보고 실제 물리 주소의 시작 지점을 찾아 본다. proc 파일 시스템으로 출력할 정보는 다음과 같다.
    1. 프로세스 정보를 가져온 시각 (시스템 시작 후 현재 까지 경과한 시간, 단위: ms)
    2. 선택된 프로세스의 모든 가상 메모리 주소의 맵
    3. Page Global Directory의 시작 부분의 주소와 함께 해당 프로세스의 Code 영역 시작 부분이 존재하는 Page Global Directory 정보
    4. 현재 프로세스의 Code 영역 시작 부분의 Page Upper Directory 정보
    5. 현재 프로세스의 Code 영역 시작 부분의 Page Middle Directory 정보
    6. 현재 프로세스의 Code 영역 시작 부분의 Page Table Entry 정보
    7. 현재 프로세스의 Code 영역 시작 부분의 Page의 Physical Address 정보
    8. 3와 6번에 대한 기타 정보

## 프로그램 세부 요구사항

- **실습 프로그램은 Linux module programming 방법으로 작성한다 (커널 소스 수정 X)**
- **실습 결과의 출력은 proc 파일 시스템을 사용한다.**
    - 리눅스에서 제공하는 proc API를 통해 /proc/hw2에 proc 파일을 생성해서 실습 결과를 출력할 수 있게 구현한다.
    - 여기에서 hw2는 디렉토리이다. /proc/hw2의 하위 파일들이 프로세스에 대한 정보를 담게 된다.
      예를 들어 cat /proc/hw2/1234 명령어를 통해 1234번 프로세스의 정보를 확인할 수 있다.

```
hjlee@ubuntu:~/hw2$ ls /proc/hw2
1     1034   1468  148   1596  1617  203    20370  27   29   39  414  7    780
10    11     1470  1482  16    1632  20335  20398  270  294  4   439  749  784
1029  11287  1473  1486  1615  1643  2036   204    28   3    41  44   755  8
```

    - 새로운 proc 파일을 생성해서 결과물을 출력하는 것이며 **이미 존재하는 proc 파일의 값을 사용하는 것은 허용하지 않는다.**
- **입력받은 주기(초 단위)로 현재 실행 중인 전체 프로세스 중에서 커널 스레드를 제외한 모든 프로세스에 대한 메모리 정보를 읽어 와서 proc 파일 시스템을 통해 출력하도록 구현한다. 즉, /proc/hw2 디렉토리를 만들고, 그 안에 정보를 읽어 올 프로세스에 대한 파일들을 생성해야 한다.**
    - 예시) insmod 시에 주기를 5초로 설정 `$ sudo insmod hw2.ko period=5`
    - task_struct를 순회하는 과정에서 생길 수 있는 타이밍 오차는 고려하지 않아도 된다.
    - 커널 모듈 insmod 시 주기를 입력받는 방법은 http://tldp.org/LDP/lkmpg/2.6/html/lkmpg.html의 parameter passing 부분을 참고
- **해당 모듈에서 프로세스 정보를 가져오는 부분은 반드시 tasklet을 사용한다.**
- **가상 메모리 출력 시 인접한 메모리 공간이 같은 구역인 경우 묶어서 출력한다.**
- **결과 화면은 제공되는 예제 소스를 사용하여 아래 출력 양식과 완전히 동일하게 출력하도록 한다.**
- **주소영역과 그 크기 및 정보들이 정확한지 검증하여 보고서에 그 내용을 작성한다.**
- **실습과제 소스 파일에는 Makefile이 포함되어야 한다.**
    - make 명령시 **hw2.ko**를 포함한 각종 모듈 관련 파일이 생성되어야 한다. **(hw1과 모듈명 양식이 다름에 유의하기 바람)**
    - make clean 명령시 make의 결과로 생성된 파일들이 모두 지워져야 한다.
- **insmod/rmmod 관련**
    - sudo insmod hw2.ko period=... 명령을 통해 커널 모듈을 등록할 수 있어야 한다.
    - 커널 모듈이 등록된 후, cat /proc/hw2/<pid> 명령을 통해 정보를 얻을 수 있어야 한다.
      예시) cat /proc/hw2/1234
    - sudo rmmod hw2.ko 명령을 통해 커널 모듈을 삭제할 수 있어야 한다.

$ cat /proc/hw2/9634 출력 예시 (단순 예시임)

```
************************************************************
Virtual Memory Address Information
Process (             vi:9634)
Last update time 39607692 ms
************************************************************
0x5576084ec000 - 0x55760871c78c : Code Area, 561 page(s)
0x55760891c8e8 - 0x55760893e890 : Data Area, 34 page(s)
0x55760893f000 - 0x55760894b000 : BSS Area, 12 page(s)
0x557608972000 - 0x557608e98000 : Heap Area, 1318 page(s)
0x7fc1f8225000 - 0x7fc1fbadb000 : Shared Libraries Area, 14518 page(s)
0x7ffe999e1000 - 0x7ffe99a03000 : Stack Area, 34 page(s)
************************************************************
1 Level Paging: Page Global Directory Entry Information
************************************************************
PGD     Base Address            : 0xffff8801e39d6000
code    PGD Address             : 0xffff8801e39d6550
        PGD Value               : 0x1e39c7067
        +PFN Address            : 0x001e39c7
        +Page Size              : 4KB
        +Accessed Bit           : 1
        +Cache Disable Bit      : false
        +Page Write-Through     : write-back
        +User/Supervisor Bit    : user
        +Read/Write Bit         : read-write
        +Page Present Bit       : 1
************************************************************
2 Level Paging: Page Upper Directory Entry Information
************************************************************
code    PUD Address             : 0xffff8801e39c7ec0
        PUD Value               : 0x1e39dd067
        +PFN Address            : 0x001e39dd
************************************************************
3 Level Paging: Page Middle Directory Entry Information
************************************************************
code    PMD Address             : 0xffff8801e39dd210
        PMD Value               : 0x1e39c3067
        +PFN Address            : 0x001e39c3
************************************************************
4 Level Paging: Page Table Entry Information
************************************************************
code    PTE Address             : 0xffff8801e39c3760
        PTE Value               : 0x1ff601025
        +Page Base Address      : 0x001ff601
        +Dirty Bit              : 0
        +Accessed Bit           : 1
        +Cache Disable Bit      : false
        +Page Write-Through     : write-back
        +User/Supervisor        : user
        +Read/Write Bit         : read-only
        +Page Present Bit       : 1
************************************************************
Start of Physical Address       : 0x1ff601000
************************************************************
```

```c
#include <stdio.h>

void printf_bar()
{
        // *의 개수는 채점과 무관함, 적당히 구분할 수 있는 모양이기만 하면 됨.
        printf("**********************************************************₩n");
}

void print_example(void){
        printf_bar();
        printf("Virtual Memory Address Information₩n");
        printf("Process (%15s:%lu)₩n", "vi", 9634);
        printf("Last update time %llu ms₩n", 39607692);
        printf_bar();

        // print info about each area
        printf("0x%08lx - 0x%08lx : Code Area, %lu page(s)₩n",
                        0x5576084ec000, 0x55760871c78c, 561);
        printf("0x%08lx - 0x%08lx : Data Area, %lu page(s)₩n",
                        0x55760891c8e8, 0x55760893e890, 34);
        printf("0x%08lx - 0x%08lx : BSS Area, %lu page(s)₩n",
                        0x55760893f000, 0x55760894b000, 12);
        printf("0x%08lx - 0x%08lx : Heap Area, %lu page(s)₩n",
                        0x557608972000, 0x557608e98000, 1318);
        printf("0x%08lx - 0x%08lx : Shared Libraries Area, %lu page(s)₩n",
                        0x7fc1f8225000, 0x7fc1fbadb000, 14518);
        printf("0x%08lx - 0x%08lx : Stack Area, %lu page(s)₩n",
                        0x7ffe999e1000, 0x7ffe99a03000, 34);

        // 1 level paging (PGD Info)
        printf_bar();
        printf("1 Level Paging: Page Global Directory Entry Information ₩n");
        printf_bar();

        printf("PGD     Base Address          : 0x%08lx₩n", 0xffff8801e39d6000);
        printf("code    PGD Address           : 0x%08lx₩n", 0xffff8801e39d6550);
        printf("        PGD Value             : 0x%08lx₩n", 0x1e39c7067);
        printf("        +PFN Address          : 0x%08lx₩n", 0x001e39c7);

        printf("        +Page Size            : %s₩n", "4KB");
```

```
        printf("       +Accessed Bit          : %s₩n", "1");
        printf("       +Cache Disable Bit     : %s₩n", "false");
        printf("       +Page Write-Through    : %s₩n", "write-back");
        printf("       +User/Supervisor Bit   : %s₩n", "user");
        printf("       +Read/Write Bit        : %s₩n", "read-write");
        printf("       +Page Present Bit      : %s₩n", "1");

        // 2 level paging (PUD Info)
        printf_bar();
        printf("2 Level Paging: Page Upper Directory Entry Information ₩n");
        printf_bar();

        printf("code    PUD Address            : 0x%08lx₩n", 0xffff8801e39c7ec0);
        printf("       PUD Value              : 0x%08lx₩n", 0x1e39dd067);
        printf("       +PFN Address           : 0x%08lx₩n", 0x001e39dd);

        // 3 level paging (PMD Info)
        printf_bar();
        printf("3 Level Paging: Page Middle Directory Entry Information ₩n");
        printf_bar();

        printf("code    PMD Address            : 0x%08lx₩n", 0xffff8801e39dd210);
        printf("       PMD Value              : 0x%08lx₩n", 0x1e39c3067);
        printf("       +PFN Address           : 0x%08lx₩n", 0x001e39c3);

        // 4 level paging (PTE Info)
        printf_bar();
        printf("4 Level Paging: Page Table Entry Information ₩n");
        printf_bar();

        printf("code    PTE Address            : 0x%08lx₩n", 0xffff8801e39c3760);
        printf("       PTE Value              : 0x%08lx₩n", 0x1ff601025);
        printf("       +Page Base Address     : 0x%08lx₩n", 0x001ff601);

        printf("       +Dirty Bit             : %s₩n", "0");
        printf("       +Accessed Bit          : %s₩n", "1");
        printf("       +Cache Disable Bit     : %s₩n", "false");
        printf("       +Page Write-Through    : %s₩n", "write-back");
        printf("       +User/Supervisor       : %s₩n", "user");
        printf("       +Read/Write Bit        : %s₩n", "read-only");
        printf("       +Page Present Bit      : %s₩n", "1");
```

```
        printf_bar();
        printf("Start of Physical Address      : 0x%08lx₩n", 0x1ff601000);
        printf_bar();
}

int main(void)
{
        print_example();
        return 0;
}
```

**예제 출력 소스**

## 기타 유의사항

- 컴파일 결과 생성된 모듈 파일의 이름은 반드시 “**hw2.ko**”로 한다. (감점 대상)
- 컴파일에 사용한 **Makefile**을 반드시 소스가 있는 폴더에 함께 첨부하도록 한다.
- 반드시 SMP 모드로 커널을 컴파일하여 수행하며, **CONFIG_SPARSE_IRQ=y**로 컴파일한다. (단, 시스템이 SMP를 지원 하지 않는 경우 명시할 것)
    - uname -a 를 했을 때, SMP가 나오는 것을 통해 확인할 수 있다.
- 반드시 64bit system 에서 작업한다. 다만, 본인의 장비 문제로 불가능할 경우 이를 보고서에 명시하고 그에 적합하게 프로그래밍을 수행한다.
- 상세하게 주석을 달 것. (제 3자가 보고 이해 할 수 있는 수준)

# 과제 제출 방법

보고서 pdf 파일(hw2_[학번].pdf)과 소스 압축 파일(hw2_[학번].tar.bz2)을 제출한다.

**보고서 파일(hw2_[학번].pdf)**

PDF 파일로 변환하여 제출한다. 파일의 이름은 반드시 hw2_[학번].pdf로 한다.

**소스 압축 파일(hw2_[학번].tar.bz2)**

반드시 다음 명령어를 사용해서 압축 파일을 생성한다. **다른 형식의 압축 또는 이중 압축은 절대 허용하지 않으며 보고서 파일을 함께 압축해서는 안된다.**

```
# 본인의 학번이 2019123123 이라고 가정
# 최상위 폴더의 이름을 hw2_2019123123으로 한다.
# 모듈 프로그래밍 소스 코드를 첨부한다.
# 커맨드에 따라 생긴 압축 파일을 해제하면 아래와 같이 구성된 폴더가 생성되어야 한다.
```

```
hw1_2019123123
    ㄴ module
        ㄴ Makefile
        ㄴ (기타 구현물)
```

```
$ tar cjf hw2_2019123123.tar.bz2 hw2_2019123123
```

# 참고 사이트

ctags 사용법 http://bowbowbow.tistory.com/15

Linux cross reference http://lxr.free-electrons.com

Linux 한글 문서 프로젝트 http://kldp.org/

Linux kernel guide http://kldp.org/KoreanDoc/html/Kernel-KLDP