GeekOS
Report Guideline

March, 2015

Daeyeon Son

Dept. of Software, Dankook University

E-mail : sonn2@daum.net

최종 과제 수행 방법

- 모든 조는 1~3번째 과제 중 1가지를 선택하여 과제를 수행한다.
  - ✓ 최종 과제 리스트 (아래의 3가지 중에서 하나만 선택 할 것)
    - 1st Homework : **ELF Parsing**
    - 2nd Homework : **EDF Scheduler**
    - 3rd Homework : **Paging**
  - ✓ 과제 제출 기한 : 6월 18일(목요일) 오후 1시 까지 (기한을 꼭 지켜주세요.)
  - ✓ 과제 보고서 발표 일정 : 6월 18일(목요일) 오후 1시 부터
  - ✓ 과제 수행 참고 사항
    - Project 미완성 상태라도 그대로 제출 할 것
      - 제출하지 않을 시에는 0점 처리되니 반드시 제출하도록 한다.
      - Tip : 과제의 완성보다는 Algorithm을 이해하는 것에 초점을 맞추도록 한다.
    - Project 단위의 Source를 압축하여 제출
      - Source에는 제작한 부분마다 **주석을 삽입하여** Algorithm에 대한 설명을 할 수 있도록 한다.
    - 제작 과정에 대한 보고서 (PPT, 양식 자유)
      - 기존 소스에서 **어느 부분을 바꾸어서 제작 하였는지** 설명을 하는 내용으로 작성하도록 한다.
    - 과제 보고서 발표
      - 각 조는 발표자 1인을 선정하여 10분 내외로 발표하도록 한다. (**발표 10분 + 질의 응답 5분**)
      - 각 조는 정해진 순서에 따라 지정된 장소에서 전원 참석하여 '**비공개 발표**'를 진행한다.
    - 과제 채점 방식
      - 보고서 완성도 : 20%, 구현 완성도 : 30%, 보고서 발표 : 20%, 보고서 및 구현 내용에 대한 질의 응답 : 30%
      - **질의 응답의 경우 조원 전체가 준비할 수 있도록 한다.**
  - ✓ **실습조교**에게 E-mail을 통하여 정해진 기간 내에 제출 할 것
    - E-mail 주소 : sonn2@daum.net
    - 제출 내용 : Project 압축 파일, 보고서

최종 과제 발표 일정

- 6월 18일 발표 조 리스트
  - ✓ 1st 발표 : 13:00~13:15
    - 32101918 성민규
    - 32101903 민상원
    - 32101912 박찬영
    - 32101950 이태진
    - 32101962 장홍준
  - ✓ 2nd 발표 : 13:15~13:30
    - 32131715 심재우
    - 32131680 강윤중
    - 32131695 김윤이
    - 32131737 이은빈
    - 32131743 임희정
  - ✓ 3rd 발표 : 13:30~13:45
    - 32111867 윤현도
    - 32111892 최희택
    - 32111846 김진욱
    - 32111847 김철중
    - 32111869 이건하
  - ✓ 4th 발표 : 13:45~14:00
    - 32101942 이영재
    - 32071502 서대웅
    - 32131689 김성윤
    - 32121855 이혜진
    - 32131691 김세은
  - ✓ 5th 발표 : 14:00~14:15
    - 32082924 정용욱
    - 32092036 표성우
    - 32091996 이범수
    - 32091989 유지수
    - 32121857 임한솔

최종 과제 발표 일정

- 6월 18일 발표 조 리스트
  - ✓ 6th 발표 : 14:30~14:45
    - 32101979 최영훈
    - 32101888 김상현
    - 32101978 최영준
    - 32101982 최진영
    - 32121822 박소윤
  - ✓ 7th 발표 : 14:45~15:00
    - 32091944 김상준
    - 32081932 김융
    - 32101891 김성민
    - 32091930 고상훈
  - ✓ 8th 발표 : 15:00~15:15
    - 32111861 신수정
    - 32131701 박주영
    - 32131722 여소영
    - 32131705 백승연
  - ✓ 9th 발표 : 15:15~15:30
    - 32111863 오성진
    - 32111848 김태성
    - 32111841 김인성
    - 32111890 최용재
  - ✓ 10th 발표 : 15:30~15:45
    - 32111874 이재영
    - 32111850 김희건
    - 32111855 백진우
    - 32111838 김수영

GeekOS 환경설정

- 최종 과제는 다음과 같이 환경설정 파일을 수정한다.

```
# An example .bochsrc file.

# You will need to edit these lines to reflect your system.
# vgaromimage: /export/home/daveho/linux/bochs-2.0.2/share/bochs/VGABIOS-lgpl-latest
# romimage: file=/export/home/daveho/linux/bochs-2.0.2/share/bochs/BIOS-bochs-latest, address=0xf0000
vgaromimage: file=/usr/share/bochs/VGABIOS-lgpl-latest
romimage: file=/usr/share/bochs/BIOS-bochs-latest

megs: 8
boot: a

ata0-master: type=disk, path="diskc.img", cylinders=40, heads=8, spt=64
#diskc: file=diskc.img, cyl=40, heads=8, spt=64

floppya: 1_44=fd.img, status=inserted
#floppya: 1_44=fd_aug.img, status=inserted

log: ./bochs.out
keyboard_serial_delay: 200
#floppy_command_delay: 500
vga_update_interval: 300000
#ips:  1000000
mouse: enabled=0
private_colormap: enabled=0
i440fxsupport: enabled=0
#newharddrivesupport: enabled=1

# Uncomment this to write all bochs debugging messages to
# bochs.out.  This produces a lot of output, but can be very
# useful for debugging the kernel.
#debug: action=report
```

/home/(자신의 아이디)/(Project dir)/build/.bochsrc

```
# Flags used for all C source files
GENERAL_OPTS := -O -Wall $(EXTRA_C_OPTS) -fno-stack-protector
CC_GENERAL_OPTS := $(GENERAL_OPTS) #-Werror
```

/home/(자신의 아이디)/(Project dir)/build/Makefile

ELF Parsing

- GeekOS
  - ✓ (Project dir)/src/geekos/userseg.c
    - Int Load_User_Program()
      - 실행하고자 하는 파일의 Data와 Length를 기반으로 하여 ELF Header를 읽어 들인 뒤 관련 내용을 바탕으로 struct Exe_Format의 내용을 입력한다.
  - ✓ (Project dir)/include/geekos/elf.h
    - struct Exe_Format
      - struct Exe_Segment segmentList[EXE_MAX_SEGMENTS]
        : Segment 구조체를 담는 공간
      - Int numSegments : ELF 파일의 Segment 개수
      - ulong_t entryAddr
        : ELF 파일의 Entry Point 주소 (프로그램 시작 코드가 있는 주소)
    - struct Exe_Segment
      - ulong_t offsetInFile : Segment 위치
      - ulong_t lengthInFile : Segment 길이
      - ulong_t startAddress : Segment의 메모리 시작 주소
      - ulong_t sizeInMemory : Segment의 메모리 사용 크기

ELF Parsing

- GeekOS
  - ✓ (Project dir)/include/geekos/user.h
    - struct User_Context
      - char *memory : Process를 적재 시켜야 하는 메모리 공간
      - ulong_t size : Process를 적재하는 메모리 크기
      - ulong_t entryAddr : Process의 시작 주소(Entry Point)
      - ulong_t argBlockAddr : Process의 인자 값(Argument)을 넣는 주소
      - ulong_t stackPointerAddr : Process의 Stack 공간 시작 주소
  - ✓ Process의 Stack 공간은 모든 Segment를 적재한 이후(!)의 주소를 입력해야 이후에 남는 공간을 모두 활용할 수 있다.
  - ✓ 앞서 배웠던 Process의 적재 순서를 생각해보면 쉽게 알 수 있으며, Stack 공간에 대한 사용 원리를 알고 있다면 stackPointerAddr의 사용 목적에 대하여 이해할 수 있다.

ELF Parsing

- GeekOS
  - ✓ (Project dir)/src/geekos/userseg.c
    - struct User_Context* Create_User_Context(**ulong_t size**)
      - 실질적으로 Process 생성 이전에 User Context에 대한 생성을 담당하는 중요한 부분
      - Input 인자 값은 Memory Size를 뜻한다.
      - Process의 Segment에 대한 초기화를 담당하고 있으며, Process가 사용하고자 하는 Memory의 크기를 여기서 결정해야 한다.
      - User_Context에 대한 변수를 먼저 생성하여 초기화 과정을 마친 뒤에 User_Context 구조체의 memory 변수를 size 크기만큼 할당한다.
      - User_Context 구조체의 ldtDescriptor 변수는 Allocate_Segment_Descriptor() 함수를 이용하여 할당한다.

Process 생성에 있어서 ELF Parsing은 시작에 불과한 단계입니다.
Process에 대한 Context 생성 단계가 제일 중요합니다.

ELF Parsing

- GeekOS
  - ✓ Key Point
    - Create_User_Context() 제작 과정 요약
      1. struct User_Context 변수 생성하여 할당
      2. struct User_Context의 memory 변수에 대하여 빈 공간 할당
      3. struct User_Context의 size 변수에 메모리 크기 입력
      4. struct User_Context의 ldtDescriptor를 Allocate_Segment_Descriptor() 함수를 통하여 할당
      5. Init_LDT_Descriptor()를 통하여 Process에 대한 LDT 초기화
      6. Init_Code_Segment_Descriptor(), Init_Data_Segment_Descriptor()를 통하여 Process의 LDT 안에 Segment Descriptor 초기화
      7. struct User_Context의 ldtSelector, csSelector, dsSelector는 Selector()를 통하여 LDT 및 Segment Descriptor에 대한 Index를 입력
      8. struct User_Context 변수를 반환

ELF Parsing

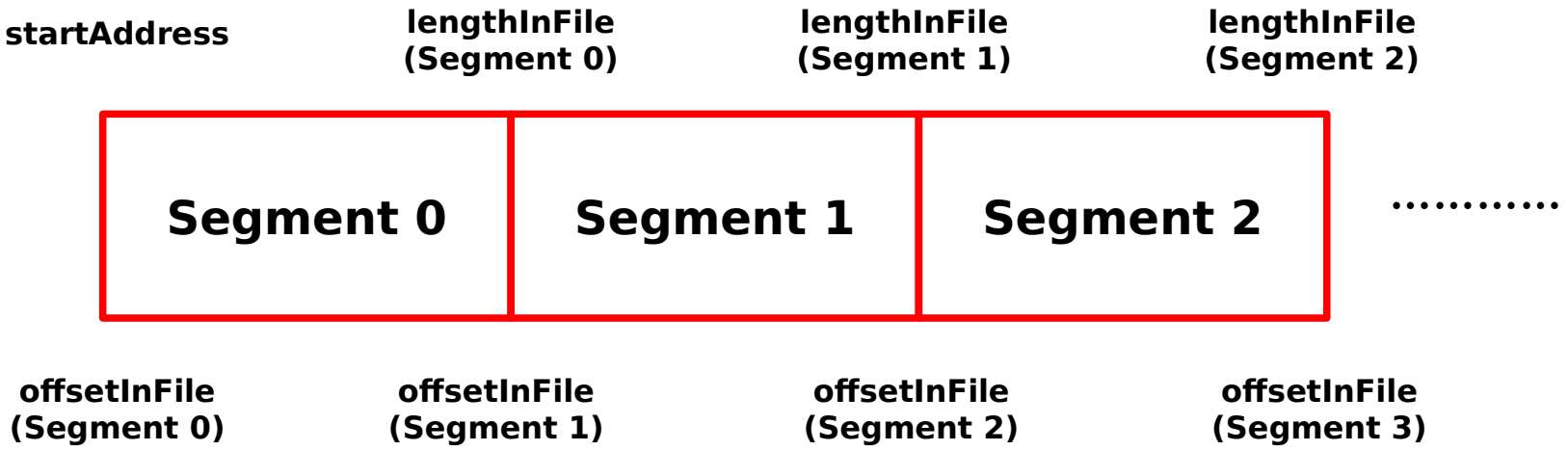
- GeekOS - Hint

```
static struct User_Context* Create_User_Context(ulong_t size)
{
    struct User_Context* pUserContext;
    struct Segment_Descriptor* desc;
    bool iflag;

    iflag = Begin_Int_Atomic();

    End_Int_Atomic(iflag);

    pUserContext->ldtSelector = Selector(0, true, Get_Descriptor_Index(pUserContext->ldtDescriptor));
    pUserContext->csSelector = Selector(3, false, 0);
    pUserContext->dsSelector = Selector(3, false, 1);

    pUserContext->refCount = 0;

    return pUserContext;
}
```

여기 빈칸은 무슨 과정에 해당될까요?
직접 구현 하세요.

ELF Parsing

- GeekOS
  - ✓ Key Point
    - ELF Format에 기초하여 만들어진 Executable File은 기본적인 정보를 ELF Header에 담고 있기 때문에 초기에 Parsing 하는 작업이 매우 중요함.
    - ELF Header에 대한 정보는 표준화 되어 있기 때문에 ELF Format 형태로 되어 있는 Executable File은 모두 동일한 Header를 지니고 있음.
    - Segment의 '시작 주소'와 '크기'를 정확하게 가져와야 하고, 각 Segment의 크기를 합산하여 Process가 사용하는 메모리 크기를 결정해야 함.
    - User Context를 할당 받고, 앞서 Parsing하여 가져온 Segment의 Offset을 이용하여 메모리를 적재 시켜야 한다.



ELF Parsing

- GeekOS - Hint

```
int Load_User_Program(char *exeFileData, ulong_t exeFileLength,
    struct Exe_Format *exeFormat, const char *command,
    struct User_Context **pUserContext)
{
    int i;
    ulong_t maxsegsize = 0; // maximum segment size
    ulong_t stackvaddr = 0; // start virtual address of stack
    ulong_t argvaddr = 0; // start virtual address of argument
    ulong_t totvaddrsize = 0; // user context's total size
    int numargs = 0; // the number of arguments
    ulong_t argblocksize = 0; // arguments block size
```

해당 공간은 직접 구현하세요.

```
    Get_Argument_Block_Size(command, &numargs, &argblocksize);
    stackvaddr = Round_Up_To_Page(maxsegsize);
    argvaddr = stackvaddr + DEFAULT_USER_STACK_SIZE;
    totvaddrsize = argvaddr + Round_Up_To_Page(argblocksize);
```

해당 공간은 직접 구현하세요.

```
    Format_Argument_Block((*pUserContext)->memory + argvaddr, numargs,
            argvaddr, command);

    (*pUserContext)->entryAddr = exeFormat->entryAddr;
    (*pUserContext)->argBlockAddr = argvaddr;
    (*pUserContext)->stackPointerAddr = totvaddrsize;

    return 0;
}
```

1. 변수 초기화

2. Segment 크기 **Parsing**

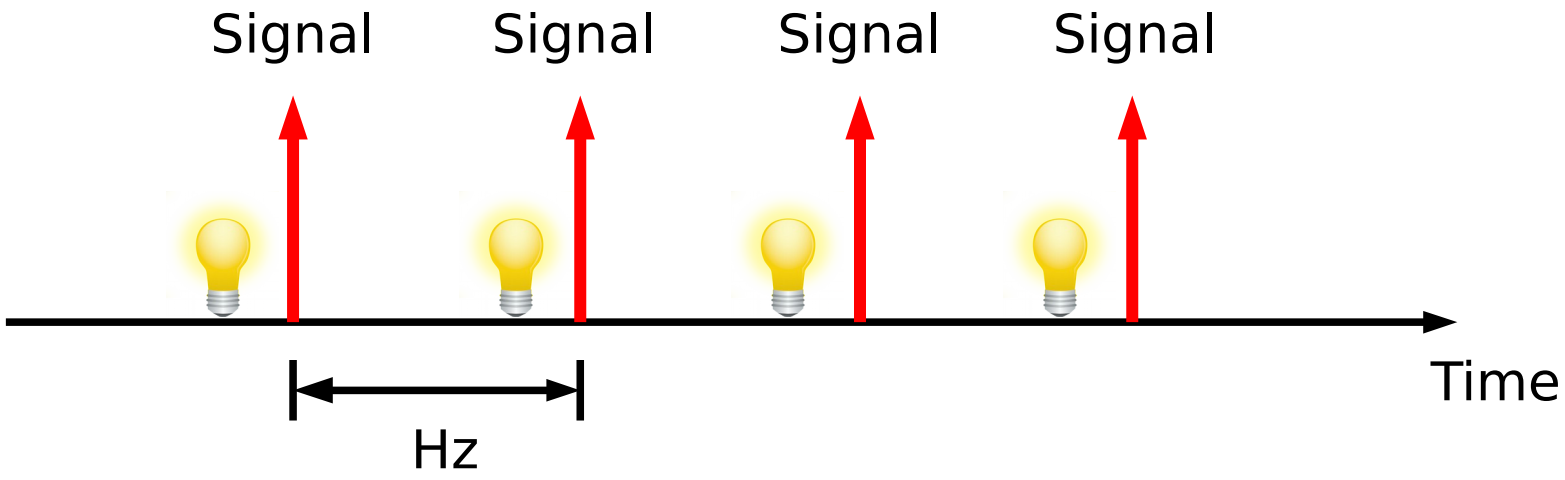
3. 메모리 주소 계산

4. User Context 생성 및 메모리 적재

5. User Context 설정

EDF Scheduler

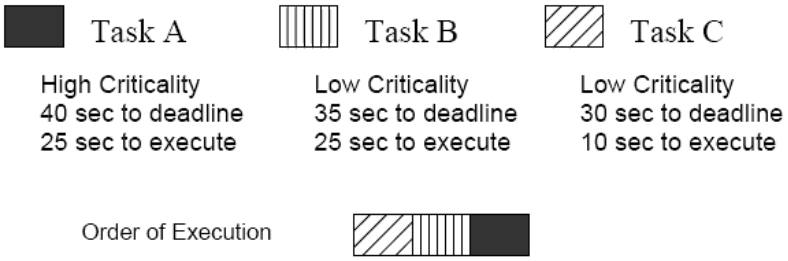
- Scheduling
  - ✓ CPU에서 Context Switching을 해주는 근본적인 이유는 각각의 Core마다 여러가지 Process를 작동 시키기 위한 것이다.
  - ✓ 또한, CPU에서 Hz라고 불리는 주파수 단위는 각 시간 당 Process에 대한 Code를 얼마만큼 처리 할 수 있는 지에 대해서 나타내며, Scheduling의 Time Tick과도 연관된다.

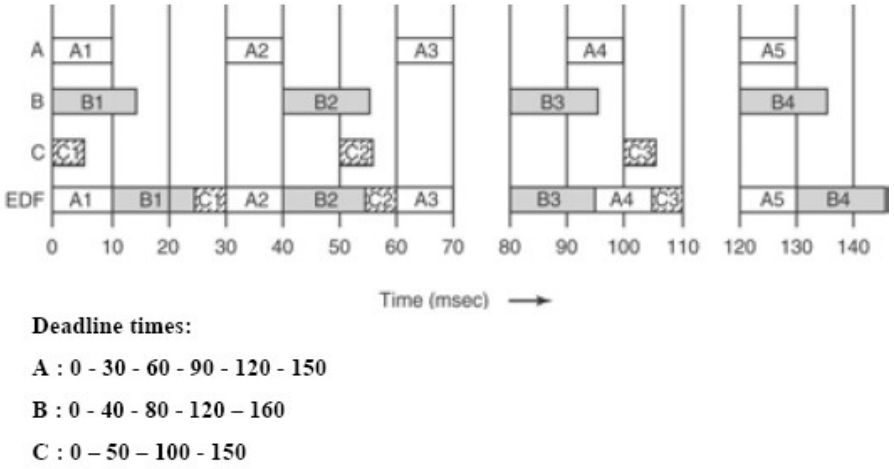


CPU의 Hz가 높아질 수록 Task의 전환 속도가 빠른 이유는 Tick이 그만큼 짧기에 즉각적인 반응을 보이기 때문이다.

EDF Scheduler

- Scheduling Policy
  - ✓ Earliest Deadline First Scheduling
    - 각각의 Task가 가지고 있는 우선순위에 따라서 최초의 실행 순서가 결정된다.
    - Scheduler에서 실행되는 Task의 Execution Time과 Deadline에 의해서 실행 순서가 바뀌게 되며 대체적으로 각각의 Task가 균등하게 실행되는 모습을 확인할 수 있다.





Scheduler가 매번 체크 할 때마다 Deadline이 가장 빠른 Task부터 먼저 처리하는 것이 EDF Scheduler의 최대 관건이다.

EDF Scheduler

- GeekOS
  - ✓ EDF Scheduler의 경우에는 기존의 Round-Robin에서 사용되었던 변수들을 분석한 이후 응용하는 과정을 거쳐서 제작해야 합니다.
  - ✓ EDF Scheduler를 제작하기 전에 알아야 할 것
    - Context Switching에 대한 부분
    - GeekOS에서 Round-Robin Scheduler를 위해 사용된 변수
    - EDF Scheduler를 만들기 위하여 바꿔야 할 부분
    - EDF Scheduler의 Algorithm
  - ✓ EDF Scheduler에서 중요한 부분
    - 각각의 Process마다 Execution time, Deadline에 대한 정보를 저장 할 것
  - ✓ 앞서 보았던 ELF Parsing 부분을 참고해서 User Context 구조체에 관련 정보를 삽입할 수 있도록 수정합니다.

EDF Scheduler

- GeekOS
  - ✓ (Project dir)/src/geekos/timer.c
    - void Timer_Interrupt_Handler()
      - Scheduling 작업을 수행하기 위하여 실행되는 함수로서, Scheduler Policy를 적용하기 위해 수정해야 하는 부분
  - ✓ (Project dir)/include/geekos/kthread.h
    - struct Kernel_Thread
      - GeekOS에서는 일반적으로 Thread라고 칭하지만, Kernel_Thread는 각각 하나의 Process를 뜻하기 때문에 Process에 대한 Schedule을 관리 하기 위한 용도의 자료구조라고 생각합시다.
      - volatile ulong_t numTicks : Process마다 할당해주기 위한 Tick 개수
      - int priority : Process에 대한 우선 순위
      - int currentReadyQueue : Process가 현재 위치하고 있는 Queue의 번호

Scheduler 부분은 내부 Algorithm에 대한 구현 문제라서 다루는 변수의 숫자가 다른 과제에 비해서 상대적으로 적습니다.

EDF Scheduler

- GeekOS
  - ✓ (Project dir)/src/geekos/kthread.c
    - void Schedule()
      - Scheduler에 의하여 Process를 실행하는 함수이며, 내부의 Switch_To_Thread()에 의해서 현재 작동하고 있는 Process에 대한 Context를 저장한 이후 다른 Process를 실행 시키는 주요 역할을 맡는다.
    - struct Kernel_Thread* Get_Next_Runnable()
      - 함수 내부에서 Scheduling 정책을 선택할 수 있고 이후에 실행될 Process를 선택하는 역할을 담당한다.
      - 기본적인 실행 과정은 Find_Best()를 통하여 ReadyQueue 내부의 Process를 찾은 뒤 Process에 해당되는 Kernel_Thread 변수를 반환해주는 역할을 수행한다.
      - 또는, Remove_Thread()를 통하여 ReadyQueue에서 Process에 해당되는 Kernel_Thread 변수를 제거하는 역할을 수행하기도 한다.
    - struct Kernel_Thread* Find_Best()
      - 현재 Queue 내부에 존재하는 Kernel_Thread 변수 중에서 Priority가 제일 높은 변수를 선택해서 반환해준다.

EDF Scheduler

- GeekOS
  - ✓ (Project dir)/include/geekos/kthread.h
    - void Enqueue_Thread()
      - 특정 Thread_Queue에 Kernel_Thread 변수를 추가해주는 함수
    - void Remove_Thread()
      - 특정 Thread_Queue의 Kernel_Thread 변수를 제거해주는 함수
    - void Switch_To_Thread()
      - Process의 Kernel_Thread를 선택하여 Context_Switch 작업을 수행하는 함수
      - 실제 함수 구현 부분은 Assembly로 작성되어 lowlevel.asm에 정의되어 있음
  - ✓ (Project dir)/src/geekos/syscall.c
    - static int Sys_SetSchedulingPolicy() → SetSchedulingPolicy()
      - Scheduling Policy 및 Tick의 개수를 정하기 위해서 사용되는 함수
      - struct Interrupt_State의 ebx, ecx 변수에 숫자를 입력한 뒤 인자 값으로 사용 (자세한 부분은 직접 확인할 것)
  - ✓ Scheduler에 있어서 Queue를 관리하는 부분은 상당히 중요한 부분으로서 Policy의 수행 단계에 따라서 Kernel_Thread를 Priority에 따라서 적절하게 배치를 해야 할 필요성이 있다.

EDF Scheduler

- GeekOS - Semaphore
  - ✓ (Kernel dir)/src/geekos/syscall.c
    - static int Sys_CreateSemaphore() → CreateSemaphore()
      - Kernel에서 사용하는 Semaphore를 생성하는 함수
    - static int Sys_P() → P()
      - Kernel에서 Semaphore를 취득하는 함수
    - static int Sys_V() → V()
      - Kernel에서 Semaphore를 반납하는 함수
    - static int Sys_DestroySemaphore() → DestroySemaphore()
      - Kernel에서 Semaphore를 없애는 함수
    - static int Sys_Wait() → Wait()
      - Kernel에서 Process가 종료되기를 기다리는 함수
  - ✓ Scheduler의 기능에 있어서 Semaphore가 필요한 경우 System Call을 직접 제작하여 사용하는 것이 가능하다.

EDF Scheduler

- GeekOS - Hint

```
static int Sys_CreateSemaphore(struct Interrupt_State* state)
{
    char sem_name[25];
    int length = state->ecx;
    int sem_id;
    int i;

    memset(sem_name, 0, 25);
    Copy_From_User(sem_name, state->ebx, length);

    if (sem == NULL) { // create 20 semaphores
        sem = (struct semaphore**)Malloc(NUM_SEMAPHORE * sizeof(struct semaphore*));
        for(i=0 ; i<NUM_SEMAPHORE ; i++) {
            sem[i] = (struct semaphore*)Malloc(sizeof(struct semaphore));
            sem[i]->count = 0;
            sem[i]->avail = 1;
            Clear_Thread_Queue(&sem[i]->waitQueue);
        }
    }

    if (sem != NULL) { // take a semaphore
        for(i=0 ; i<NUM_SEMAPHORE ; i++) { // search whether the same semaphore name exist or not
            if(strcmp(sem_name, sem[i]->sem_name) == 0) { // in that semaphore already exist
                return i;
            }
        }

        if(i == 20) {// in that the same semaphore name don't exist
            i = 0;
            while(1) {
                if(sem[i]->avail == 1) { // search an available semaphore
                    sem[i]->count = state->edx;
                    sem[i]->avail = 0;
                    memcpy(sem[i]->sem_name, sem_name, length+1);
                    break;
                }
                i++;
                if(i == 20) {// in that all semaphore has been allocated
                    return -1;
                }
            }
        }
    }

    return i; // return semaphore id
}
```

EDF Scheduler

- GeekOS - Hint

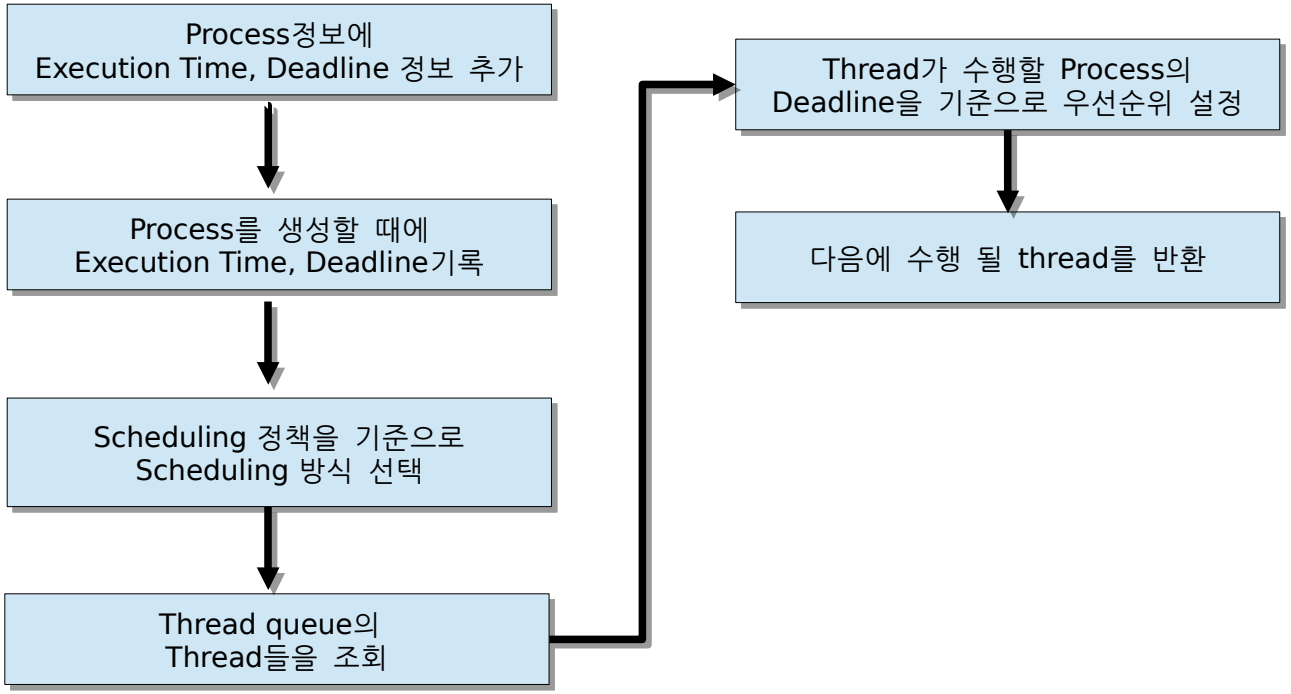
```
static int Sys_P(struct Interrupt_State* state)
{
    int sem_id = state->ebx;

    if(sem[sem_id]->count <= 0)
    {
        Wait(&sem[sem_id]->waitQueue);
    }
    if(sem[sem_id]->count > 0)
    {
        sem[sem_id]->count--;
    }

    return 0;
}
```

```
static int Sys_P(struct Interrupt_State* state)
{
    int sem_id = state->ebx;

    if(sem[sem_id]->count <= 0)
    {
        Wait(&sem[sem_id]->waitQueue);
    }
    if(sem[sem_id]->count > 0)
    {
        sem[sem_id]->count--;
    }

    return 0;
}
```

```
static int Sys_DestroySemaphore(struct Interrupt_State* state)
{
    int sem_id = state->ebx;

    if(sem[sem_id]->avail == 0)
    {
        return -1;
    }
    else
    {
        sem[sem_id]->count = 0;
        sem[sem_id]->avail = 1;
        Clear_Thread_Queue(&sem[sem_id]->waitQueue);
    }

    return 0;
}
```

EDF Scheduler

- GeekOS
  - ✓ EDF Scheduler Operating Sequence



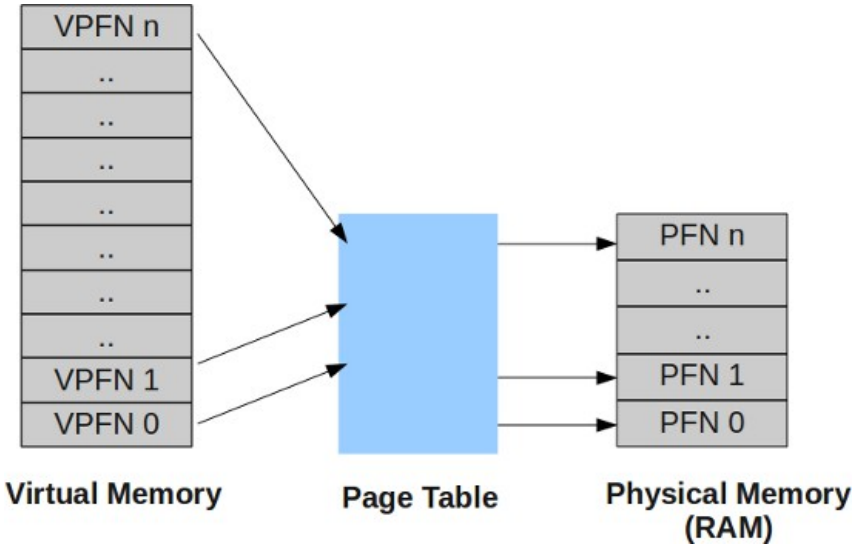
Paging

- Memory
  - ✓ Virtual Memory
    - 각각의 Process마다 존재하며, Hardware와 Software 사이에 위치한 가상 주소 공간이다.
    - Memory의 공간이 여유가 있을 경우에는 Physical Memory와 연결 되지만, 그렇지 않을 경우에는 외부 Storage와 연결이 되기도 한다. (Swap 공간)
  - ✓ Physical Memory
    - 실제 Computer에서 사용되는 Memory이며 OS의 32bit, 64bit Mode의 차이에 따라서 인식이 가능한 최대 Capacity가 제한적이다.

- Operating System
  - ✓ OS에서는 Virtual Memory에 대한 Physical Memory의 할당을 관리하게 된다.
  - ✓ CPU는 MMU(Memory Management Unit)를 통하여 Virtual Address를 Physical Address로 변환한다.

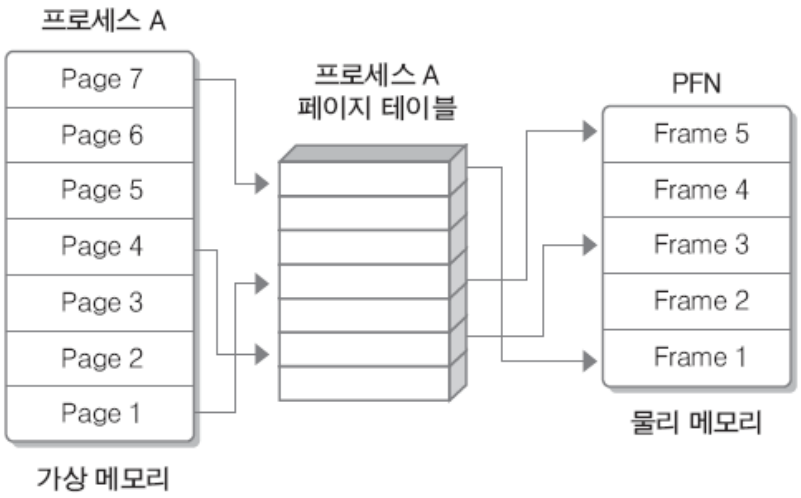
Paging

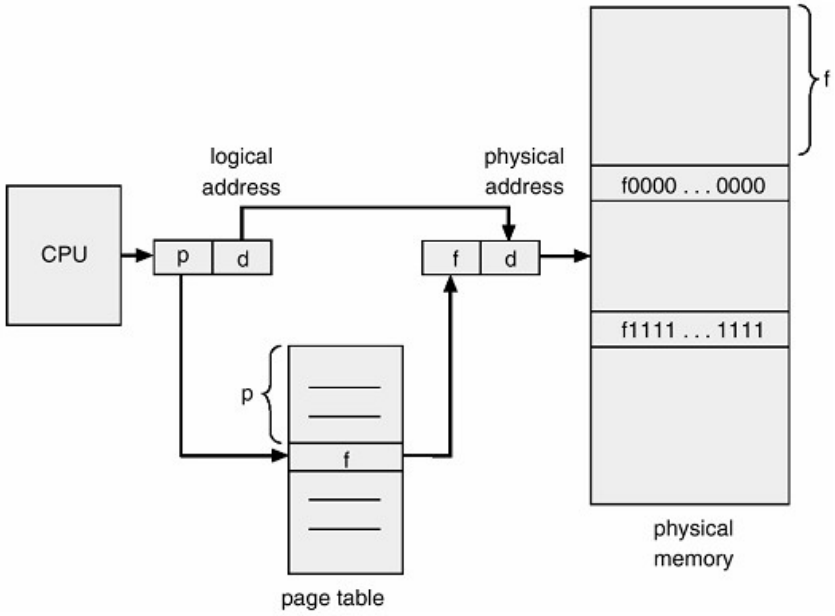
- Paging
  - ✓ Paging은 기본적으로 Memory를 모두 같은 크기의 Block으로 편성하여 운용하는 기법
  - ✓ 일정한 크기를 가진 Block을 Page라고 이야기하며, Virtual Memory를 Page 단위로 나누고 Physical Memory는 Page 크기와 같은 Frame으로 나누어서 사용한다.
  - ✓ 각각의 Page는 Frame과 쌍을 이루게 되며 동일한 Frame과 연결되는 여러 개의 Page가 존재할 수 있다.



Paging

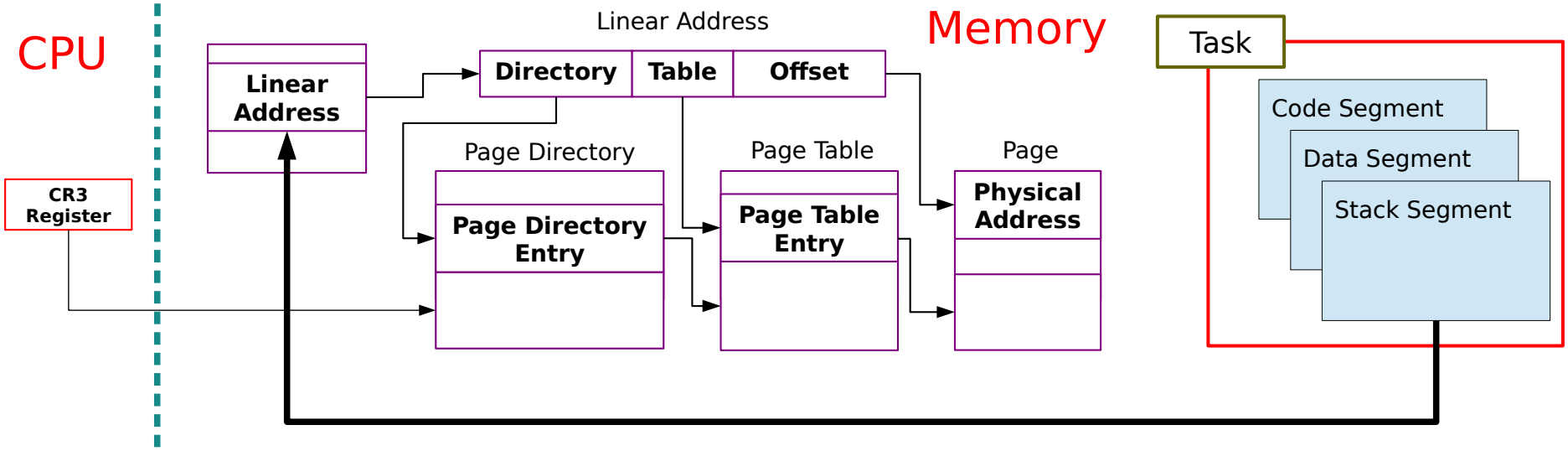
- Page Table
  - ✓ Page Table은 Process의 Page 정보를 저장하고 각 Process의 Virtual Address가 Physical Address와 어떻게 연결되어 있는 지를 나타내는 테이블이다.
  - ✓ 각각의 Process는 한 개의 Page Table 소유한다.





Paging

- Page Table
  - ✓ Page Table은 Index로는 Page Number를 나타내며, 내용은 Page에 할당된 Frame의 시작 주소를 나타낸다.
  - ✓ GeekOS에서는 2단계 Page Table을 구현하게 된다.
    - Page Directory Entry, Page Table Entry



Paging

- GeekOS
  - ✓ (Project dir)/src/geekos/paging.c
    - void Init_VM(struct Boot_Info *bootInfo)
      - GeekOS의 Paging 기능 추가의 시작과 끝을 담당하는 함수이며, 해당 함수를 제작하고 Virtual Memory의 초기화 과정이 완료되면 프로젝트는 종료된다.
      - Page Directory, Page Table의 Index에 따른 Page 할당과 Address 설정에 대해서 정확한 계산이 요구되는 부분이다.
      - GeekOS에서는 2단계만 처리하면 되기 때문에 다소 쉬워 보일 수 있으나 Paging 기능을 처리하는 다른 함수들을 모두 파악해야 종합적인 Mechanism을 이해할 수 있다.
  - ✓ (Project dir)/include/geekos/paging.h
    - typedef struct pde_t
      - Kernel의 Page Directory Entry의 자료구조
    - typedef struct pte_t
      - Kernel의 Page Table Entry의 자료구조

Paging

- GeekOS
  - ✓ (Project dir)/include/geekos/paging.h
    - typedef struct pde_t

```
/*
 * Page directory entry datatype.
 * If marked as present, it specifies the physical address
 * and permissions of a page table.
 */
typedef struct {
    uint_t present:1;
    uint_t flags:4;
    uint_t accesed:1;
    uint_t reserved:1;
    uint_t largePages:1;
    uint_t globalPage:1;
    uint_t kernelInfo:3;
    uint_t pageTableBaseAddr:20;
} pde_t;
```

    - typedef struct pte_t

```
/*
 * Page table entry datatype.
 * If marked as present, it specifies the physical address
 * and permissions of a page of memory.
 */
typedef struct {
    uint_t present:1;
    uint_t flags:4;
    uint_t accesed:1;
    uint_t dirty:1;
    uint_t pteAttribute:1;
    uint_t globalPage:1;
    uint_t kernelInfo:3;
    uint_t pageBaseAddr:20;
} pte_t;
```

pde_t와 pte_t는
구성 요소만 따지고 보면
사실상 동일한 자료구조입니다.

Paging

- GeekOS
  - ✓ (Project dir)/include/geekos/paging.h
    - #define NUM_PAGE_TABLE_ENTRIES
      - GeekOS 소스에서 정해진 최대 Page Table 개수
    - #define NUM_PAGE_DIR_ENTRIES
      - GeekOS 소스에서 정해진 최대 Page Directory 개수
    - #define PAGE_ALLIGNED_ADDR(x)
      - Page의 Address를 오른쪽으로 12칸 Shift를 하여 나타낸 값
    - ulong_t Get_Page_Fault_Address()
      - Page Fault가 발생했을 때의 Memory Address를 반환
    - #define VM_WRITE
      - pde_t 또는 pte_t의 Flag 변수에 선언하여 해당 Page 공간을 Write가 가능하도록 속성을 변경할 수 있다.
    - #define VM_READ
      - pde_t 또는 pte_t의 Flag 변수에 선언하여 해당 Page 공간을 Read가 가능하도록 속성을 변경할 수 있다.

Paging

- GeekOS
  - ✓ (Project dir)/include/geekos/paging.c
    - void Page_Fault_Handler()
      - Page Fault가 발생하였을 때 Handling하는 함수
      - Interrupt가 발생한 이후 Page Address를 가져오는 역할과 Error code를 판별하여 처리하는 역할을 수행한다.
    - void Init_Paging()
      - Paging File에 대한 관리를 위하여 자료구조를 초기화 시켜준다.
      - Paging File에 대한 자료구조는 Get_Paging_Devices() 함수를 참고하여 Paging File을 구성하는 Device, Disk Block의 범위 등을 살펴보고 제작 할 수 있도록 한다.
    - int Find_Space_On_Paging_File()
      - 입력하는 크기 만큼의 Page 여유 공간을 Paging File 안에서 확인한 뒤 존재하는 경우 에는 해당 Page의 Index를 반환하고, 여유 공간이 모자란 경우에는 -1을 반환한다.
    - void Free_Space_On_Paging_File()
      - Find_Space_On_Paging_File() 함수를 통하여 할당 받은 Page 여유 공간을 할당 해제 시켜준다.

Paging

- GeekOS - Hint

```
void Init_VM(struct Boot_Info *bootInfo)
{
    //num_of_pages는 3MB내 페이지의 개수
    int num_of_pages = bootInfo->memSizeKB >> 2; // the number of page frame
    int num_of_dir_entries = Round_Up_To_Page(bootInfo->memSizeKB) / PAGE_SIZE; // the

    // 16진수로 변환 후 몇 개의 페이지를 사용하는지 구한 후, 4096으로 나눈다
    int i, j;
    struct Page* page;

    //Print("*****memSizeKB*****:%d\n", bootInfo->memSizeKB);       //3072 = 1024 * 3
    //Print("*****num_of_pages*****:%d\n", num_of_pages);       //768
    //Print("****num_of_dir_entries****:%d\n", num_of_dir_entries); //1

    //페이지 디렉토리 엔트리를 위한 페이지 할당
    kpde = (pde_t*)Alloc_Page(); // allocate page directory entry
    memset(kpde, 0, PAGE_SIZE);
    pte_t* kpte;

    //i는 페이지 디렉토리의 인덱스
    for(i=0 ; i<NUM_PAGE_DIR_ENTRIES ; i++)
    {
```

해당 공간은 직접 구현하세요.

```
        //j는 페이지 테이블의 인덱스
        for(j=0 ; j<NUM_PAGE_TABLE_ENTRIES ; j++)
        {
```

해당 공간은 직접 구현하세요.

```
        }

        (kpde+i)->present = 1;
        (kpde+i)->flags = VM_READ | VM_WRITE;
        (kpde+i)->accesed = 0;
        (kpde+i)->reserved = 0;
        (kpde+i)->largePages = 0;
        (kpde+i)->globalPage = 0;
        (kpde+i)->kernelInfo = 0;
        //한 페이지 디렉토리 엔트리에 할당 된 한 페이지 테이블 엔트리의 주소를 넣어줌
        (kpde+i)->pageTableBaseAddr = PAGE_ALLIGNED_ADDR(kpte);
    }

    Enable_Paging(kpde);
    Install_Interrupt_Handler(PAGEFAULT_INT, &Page_Fault_Handler);
}
```