**Pintos Project 1 : User Program (1)**

**(설계 프로젝트 수행 결과)**

과목 명 : [CSE4070] 운영체제

담당 교수 : 박성용 교수님

조 / 조원 : 23조, 20171640, 20171702

개발 기간 : 2019/10/6 ~ 2019/11/3

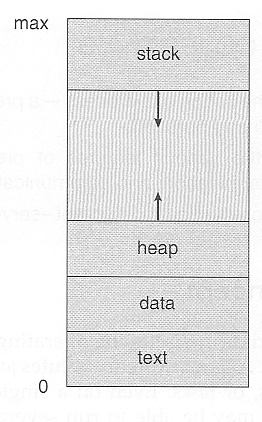
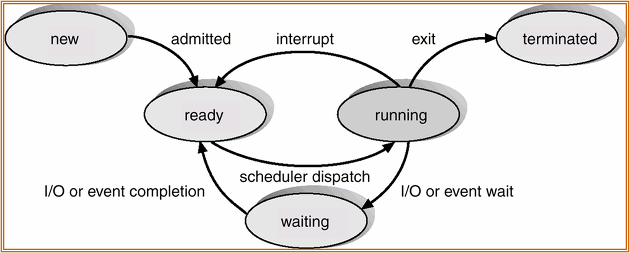
**프로젝트 제목 : Pintos Project 1 User Program (1)**

**제출일 : 2019년 11월 3일**

**참여 조원 : 박수진, 하민주**

1. **개발 목표**

* 이번 프로젝트의 초기 pintos는 argument passing, system call이 구현되어 있지 않아 process를 실행할 수 없다. Argument passing을 통해 Stack을 쌓고 stack pointer(esp)를 System Call handler에 전달하여 명령어에 해당하는 system  call을 수행할 수 있도록 한다. 이 때 메모리가 범위를 벗어나지 않도록 User memory access 작업을 할 수 있도록 한다. 최종적으로 pintos의 user program을 개발한다.

1. **개발 범위 및 내용**
   1. **개발 범위**
2. **Argument Passing** (userprog/process.c)  
   - parsing : load(), process\_execute() 수정  
   - construct esp : load() 수정, stack에 pointer 추가
3. **User Memory Access** (userprog/exception.c)  
   - valid memory check : page\_fault(), syscall\_handler() 수정
4. **System Calls** (userprog/syscall.c, threads/thread.c)  
   - thread : thread struct 수정  
   - system call handler : syscall\_handler() 수정  
   - system call : halt(), exit(), exec(), wait(), write(), read()  
   - additional implement : fibonacci(), sum\_of\_four\_int()
   1. **개발 내용**
      1. **Page**   
         가상 페이지라고도 하는 페이지는 가상 메모리 4,096바이트(페이지 크기)의 연속적인 영역이다. 페이지는 페이지 정렬, 즉 페이지 크기로 균등하게 분할할 수 있는 가상 주소에서 시작해야 한다. 따라서 32비트 가상 주소는 다음과 같이 20비트 페이지 번호와 12비트 페이지 오프셋(또는 그냥 오프셋)으로 나눌 수 있다.  
           
         31 12 11 0  
         +-------------------+-----------+  
         | 페이지 번호 | 오프셋 |  
         +-------------------+-----------+  
         가상 주소  
         각 프로세스에는 가상 주소 PHYS\_BASE 아래의 페이지인 사용자(가상) 페이지 세트가 있으며 일반적으로 0xc0000000(3GB)이 있다. 반면에 커널(가상) 페이지 세트는 어떤 스레드나 프로세스가 활성화되었는지에 관계없이 글로벌하게 남아 있다. 커널은 사용자와 커널 페이지 모두에 접근할 수 있지만, 사용자 프로세스는 자신의 사용자 페이지에만 접근할 수 있다.
      2. **Process**프로세스는 메모리에 적재되어 실행되고 있는 프로그램을 말한다.  
            
          [프로세스의 구성]  
         프로세스의 구성은 위의 그림과 같다. 또한 각 영역을 살펴보면 다음과 같다.  
         스택(Stack) : 일시적인 데이터(파라미터, 복귀주소, 지역 변수)등을 저장하는 곳  
         힙(Heap) : 동적 할당된 메모리   
         데이터(data): 전역변수 등이 저장 되는 곳  
         텍스트(text) : 프로그램 코드  
           
         그리고 다음은 프로세스의 상태도이다.  
           
         new : 프로세스가 최초 생성 중인 상태  
         running : 프로세스가 실행중인 상태  
         waiting : 프로세스가 어떠한 이벤트를 기다리고 있는 상태  
         ready : 프로세스가 프로세서에 할당되기 기다리고 있는 상태  
         terminated : 프로세스의 실행이 종료된 상태  
           
         이러한 관점에서 프로세스는 크게 활동 상태와 지연 상태로 나뉜다.  
         활동상태란 프로세스가 기억장치에 적재된 상태이고, 지연 상태란 프로세스가 기억장치에 적재되지 않은 상태를 의미한다.
      3. **Argument passing**pintos의 user program의 입력으로 다수의 argument가 들어올 수 있어 이를 80x86 Calling Convention에 맞추어 메모리에 배치해야 한다. Program에 입력된 명령어를 strtok\_r을 사용하여 띄어쓰기를 기준으로 token해준다. 그리고 token한 명령어를 임시 변수에 저장한다. 그 후 esp를 PHY\_BASE부터 해서 downward로 명령어와 명령어의 주소값을 저장한다. 명령어 string을 저장한 후엔 꼭 word\_align을 해주어야 한다.
      4. **User process가 Kernel memory에 접근하는 것을 막기 위한 방법  
         -** user program에서 PHY\_BASE와 0xc0000000사이의 주소를 갖는지 system call handler에서 지속적으로 확인  
         - 그 외의 주소들은 page\_fault()에서 user memory가 아니거나 커널일 경우 exit(-1)하도록 조건을 주어 user process가 kernel memory를 침범하지 않도록 관리한다.
      5. **Thread**Thread란 data등을 공유하는 프로세스는 하나만 생성하고 Program counter만 여러 개 생성하여 CPU수행 단위를 여러 개를 두는 것이다.  
         Thread는 program counter 값과 register 값 등을 가지고 있는데 이 section은 다른 thread와 공유하지 않기 때문에 각각의 thread는 각각 다른 Instruction을 수행할 수 있다. Thread는 공유할 수 있는 것을 최대한 공유하여 메모리의 낭비를 줄일 수 있다. (주소 공간, 프로세스가 사용하는 각종 자원 등) 특히, multi-thread로 구성된 프로세스 내에서는 하나의 thread가 waiting 상태인 동안에도 다른 thread를 실행할 수 있다. 즉, process가 끝날 때까지 기다린 후 실행할 필요가 없으므로 빠른 처리가 가능하다.   
           
         운영체제에서 thread의 종류에는 두가지가 있는데, 하나는 kernel thread이고 다른 하나는 user thread이다. Kernel thread는 프로세스 내 thread가 여러 개라는 것을 운영체제가 알고 있는 thread로 스케줄링을 맡아서 하게 된다. 그러나 user thread는 라이브러리 차원에서 지원되는 thread로 프로세스 안에 thread가 여러 개 있다는 것을 운영체제가 모르기 때문에 kernel 입장에서는 하나의 일반적인 프로세스로 인식된다.
5. **추진 일정 및 개발 방법**
   1. **추진 일정**

|  |  |
| --- | --- |
| 기간 | 내용 |
| 1주차 (10/6 ~ 10/13) | 자료구조 및 알고리즘 구상 |
| 2주차 (10/14 ~ 10/21) | Argument passing & construct ESP |
| 3주차 (10/22 ~ 10/29) | Thread struct 수정 및 System Call 구현 |
| 4주차 (10/29 ~ 11/3) | 마무리 작업 및 보고서 작성 |

* 1. **개발 방법**

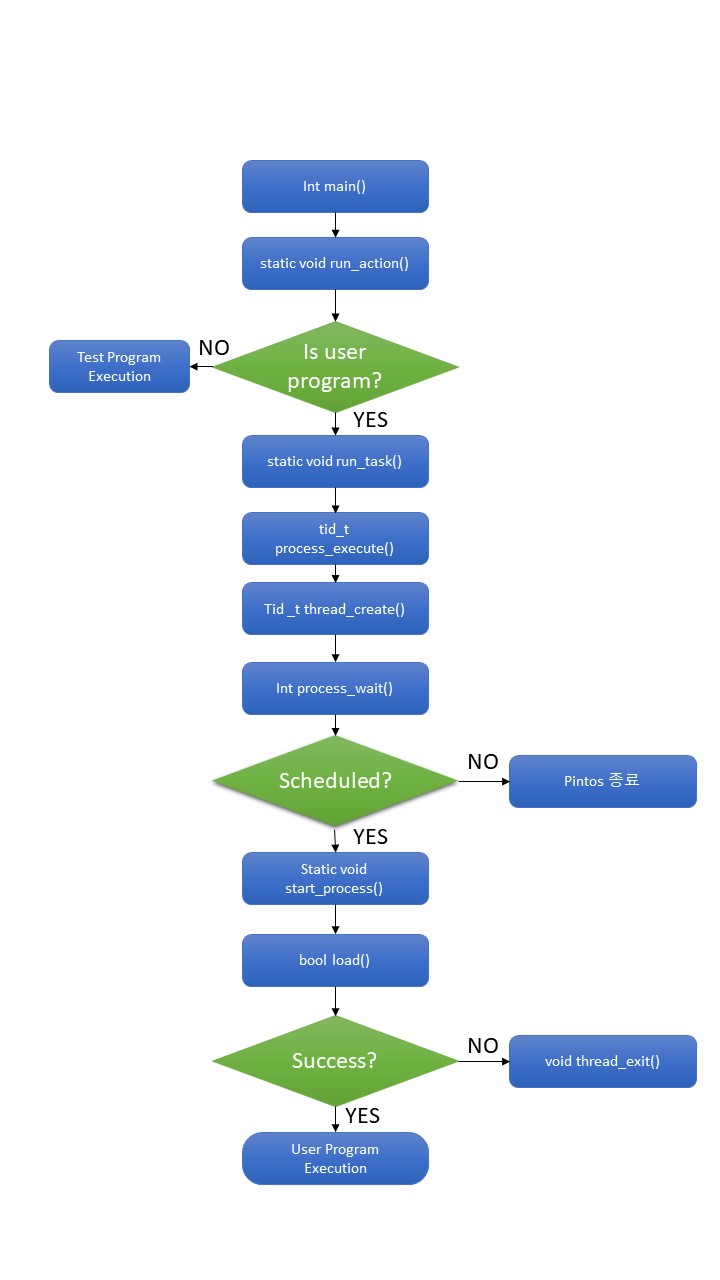
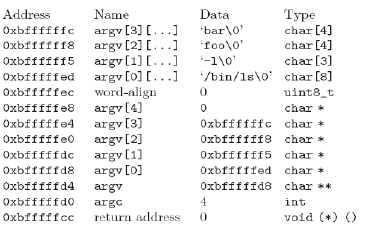
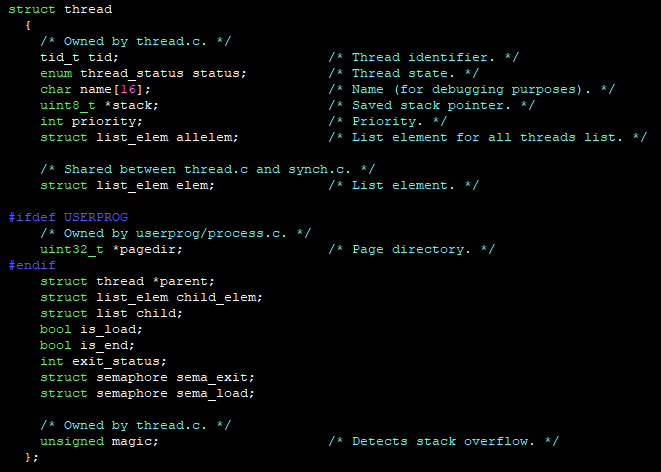
1. **Argument Passing** (userprog/process.c)  
   - Argument Passing을 위하여 userprog/process.c의 load()에서 명령어를 parsing하고 stack을 쌓을 것이다.  
   - userprog/process.c의 process\_execute()에서 파일 이름을 받고, file open했을 때 NULL이면 return 처리해준다.
2. **User Memory Access** (userprog/exception.c,userprog/syscall.c)

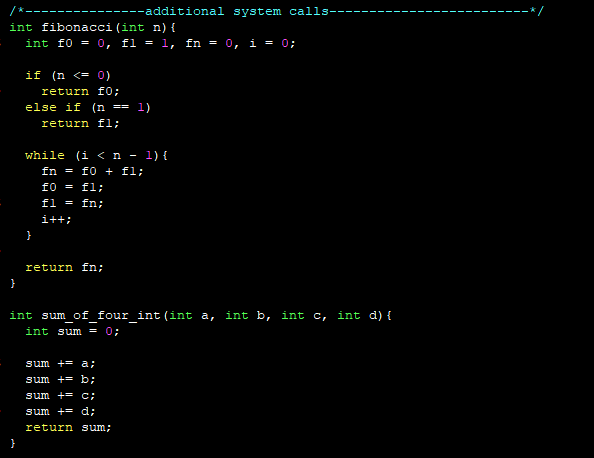
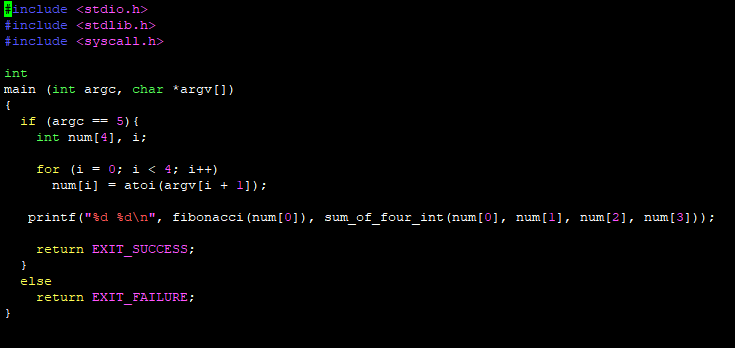
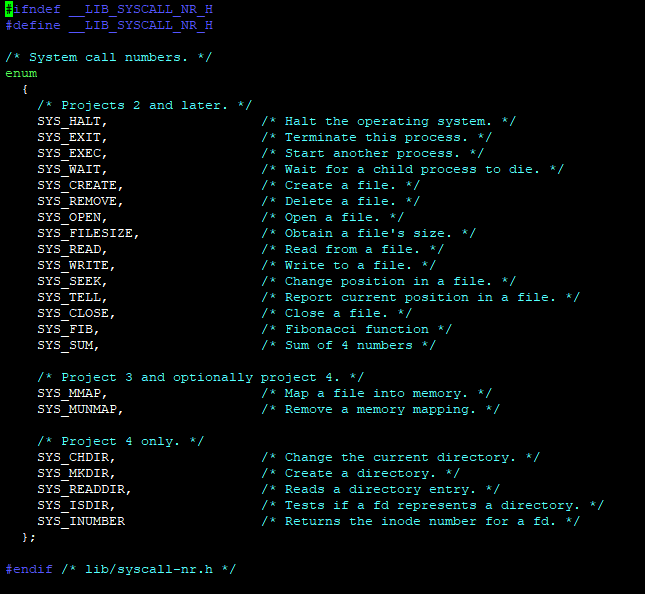
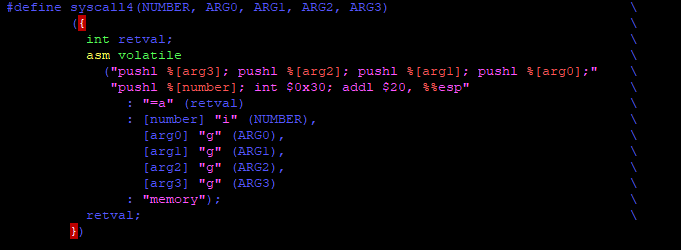
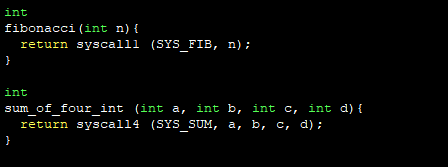
- PHY\_BASE와 0xc0000000사이의 값이 아닌 fault address 는 userprog/exception.c의 page\_fault()에서 user memory가 아니면 exit(-1)를 호출하여 kernel memory에 접근 하는 것을 방지한다.  
- user memory access를 확인하기 위하여, userprog/syscall.c에서 is\_user\_vaddr함수를 통해 user memory인지 지속적으로 체크해준다.

1. **System Calls** (userprog/syscall.c, threads/thread.c)  
   - exit과 wait를 효과적으로 수행하기 위해 threads/thread.h에서 thread struct 내에 변수를 추가하고 threads/thread.c의 thread\_create()에서 추가한 변수를 초기화한다.  
   - system call을 구현하기 위해 userprog/syscall.c에서 syscall\_handler()에 switch 문을 추가하여 system call number에 따라 해당하는 system call function을 호출하도록 한다. (system call function : halt(), exit(), exec(), wait(), write(), read(), fibonacci(), sum\_of\_four\_int())  
   1. **연구원 역할 분담**

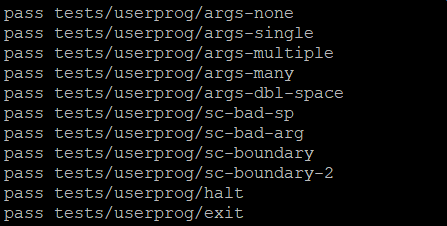
|  |  |
| --- | --- |
| 박수진 | 하민주 |
| * 매뉴얼 및 자료 분석 * 자료구조 및 알고리즘 구상 * thread struct 및 thread function 수정 * system call handler 및 system call 구현 * additional implement 구현 * 테스트 및 디버깅 * 보고서 작성 | * 매뉴얼 및 자료 분석 * 자료구조 및 알고리즘 구상 * argument passing (stack) 구현 * user memory access 구현 * 테스트 및 디버깅 * flow chart 작성 * 보고서 작성 |

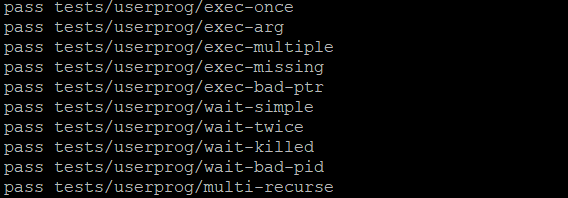
1. **연구 결과**
   1. **합성 내용**

* [pintos user program flow chart]
  1. **제작 내용**
* **Argument Passing**  
  process\_execute()에서 명령어를 파싱하여 파일이름을 넘겨주고, 그 이름을 가진 thread를 생성한다. 그리고 process\_load()에서는 명령어의 주소와, string을 저장하는 구조체를 만들어서 명령어를 파싱하여 구조체의 string에 저장하였다. 그리고  esp 포인터를 PHY\_BASE부터 시작해서 반복문을 통하여 마지막 명령어 인덱스부터 명령어의 길이 + null포인트 사이즈 만큼 내린 후, 그 주소에 명령어를 저장하는 것을 반복하였다. 또한 이 반복문이 실행될 때 명령어를 저장하는 esp포인터의 주소를 구조체의 주소에 저장하였다. 그 후 명령어 전체 길이를 저장한 변수를 통하여 word\_alignment를 해주고, 다시 반복문을 이용하여 구조체에 저장한 명령어의 주소값을 차례대로 stack저장한다. 명령어의 주소와 string을 모두 stack에 쌓은 후 주소, argument 개수, return address를 주소값을 내려가며 차례로 저장하였다. 최종적으로 구현한 구조는 다음과 같다.  
  
* **User Memory Access**  
  page\_fault에서 fault address가 user이 아니거나 kernel memory영역이면 exit(-1)을 해주었다.  또한 system call handler에서도 지속적으로 user memory인지 확인하여 메모리가 이상한 곳에 접근하는 것을 방지하였다.
* **System Calls**
  + 1. Thread  
       user program에서 thread의 exit과 wait를 효율적으로 관리하기 위해 세마 포어를 사용한다. 따라서 thread struct를 다음과 같이 수정하였다.  
       
    2. Halt()  
       shutdown\_power\_off()를 호출하여 pintos를 종료하는 system call이다.
    3. Exit()  
       현재 process를 종료하고 입력된 status를 kernel로 반환하는 system call function이다. Syscall에서 exit함수를 구현할 때 현재 thread의 is\_end를 set하고 child\_elem을 remove하였다. 그리고 status 번호를 출력하고 thread\_exit을 호출하였다. Thread\_exit에서는 세마 포어를 이용하여 부모 프로세스의 wait를 멈추고 thread의 상태를 THREAD\_DYING으로 표시해주었다.
    4. Exec()  
       파라미터로 주어진 cmd\_line을 run하고 새로운 프로세스의 pid를 반환하는 system call이다. Eax에 이 새로운 프로세스의 pid를 저장한다. syscall에서 exec은 process\_execute를 호출하여 구현하였다. 성공적으로 execute되면 thread의 is\_load를 1로 set한다.
    5. Wait()  
       parent process는 child process가 모두 종료될 때 까지 wait해야한다. syscall에서 wait는 process\_wait를 호출하여 구현하였다. 입력으로는 child의 pid가 들어오는데 해당하는 pid를 가진 child thread를 찾아 child thread의 is\_end가 set될 때까지 parent thread를 세마 포어를 이용하여 sleep state로 유지시킨다. Child process가 모두 종료된 것을 확인후 child process의 status를 반환한다.
* **Additional System call**

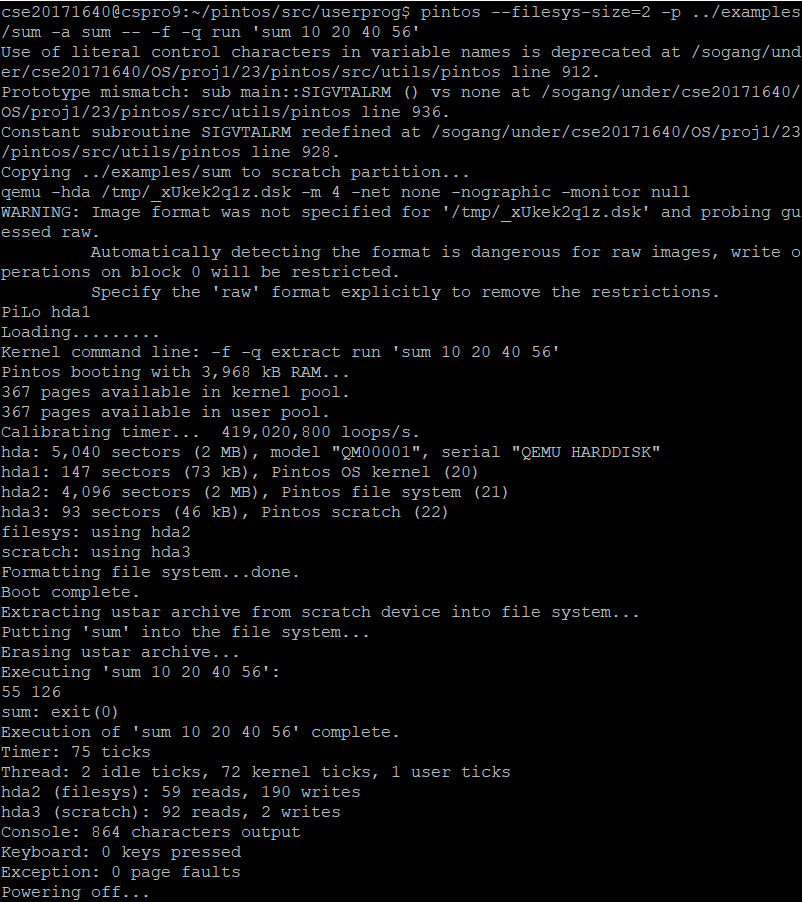
1. Userprog/syscall.c  
     
   syscall.c에서 additional system call의 동작을 정의한다. Fibonacci()는 n이 들어왔을 때 피보나치 수열 n번째 값을 반환하고 sum\_of\_four\_int()는 입력으로 들어온 네 개의 integer를 더한 값을 반환한다. Syscall.h에는 함수의 프로토타입을 작성하였다.
2. Examples/sum.c  
     
   sum 명령어의 결과를 출력하는 sum.c 함수를 examples 폴더에 생성한 후 make 해주었다.
3. Lib/syscall-nr.h  
     
   새로운 system call인 fibonacci와 sum\_of\_four\_int에 상응하는 system call number를 등록해준다 (Fibonacci : 13, sum\_of\_four\_int : 14)
4. Lib/user/syscall.c  
     
     
   argument의 개수가 4개인 경우가 define되지 않았으므로 새롭게 define해주고 syscall.c과 syscall.h에 Fibonacci()와 sum\_of\_four\_int()를 저장한다.  
   1. **시험 및 평가 내용**

* **Test case (21 pass)**





* **sum 수행 결과 (fibonacci(), sum\_of\_four\_int())**



1. **기타**
   1. **연구 조원 기여도**

* 박수진(50%), 하민주(50%)
  1. **소감**
* **박수진 :** pintos가 평소에 흔히 쓰던 운영체제와 달라 생소해서 처음에 어떻게 짜야할지 막막하고 힘들었지만 계속해서 기본 코드와 매뉴얼을 보다 보니 어떻게 코드를 짜야할지 감이 잡혀서 재밌게 코딩할 수 있었습니다. 1번 프로젝트는 접근하기 힘들어서 머리가 아팠지만 1번 프로젝트를 토대로 2번 프로젝트는 더 잘할 수 있을 것 같습니다.
* **하민주 :** 프로젝트를 시작할 때 어떻게 해야할지 너무 막막하고, 컴공과 3학년이 되도록 이거 하나 못하고있는 저 자신에게 자괴감이 들기도 하였습니다. 그러나 강의 자료를 반복해서 읽다보니 조금씩 이해가 되기 시작하였습니다.  이번 프로젝트를 하면서 자료구조에 대해서 많이 배우고, 운영체제의 흐름에 대해 조금 더 알게 된 것 같습니다. 앞으로 남은 프로젝트 또한 험난하겠지만, 열심히 해보려고 합니다.