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

**(설계 프로젝트 수행 결과)**

과목 명 : [CSE4070] 운영체제

담당 교수 : 교수님

조 / 조원 : 1조, 2013\*\*\*\*, 2016\*\*\*\*

개발 기간 : 2019/10/06 – 2019/11/03

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

**제출일 : 2019년 11월 3일**

**참여 조원 : 팀원1, 서인호**

1. **개발 목표**

* 개발 전 핀토스의 상태는, 메모리에 스택을 쌓는 기능이 구현 되어있지 않아서 파라미터 인자들이 전달이 안되는 상태이고, 따라서 프로그램이 실행되지 않는다. 이번 프로젝트에서 System call, User Memory Access, Argument Passing를 구현하여 pintos의 user program 실행 환경을 구축하고, 동작하도록 하는 것을 목표로 한다. 추가로 기본적인 system call 이외에 피보나치 수열을 구하는 Fibonacci 함수와 네 개의 정수를 더하는 sum\_of\_four\_integers 함수를 system call으로 추가 구현 및 등록한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* Argument Passing

유저가 Argument를 불러오면 유저의 프로그램에 전달이 되도록 구현. System Memory 안의 Stack영역에 그 메모리 정보가 쌓여 있어야 한다. ~/pintos/src/userprog/process.c 안에 있는 bool load()함수를 수정하여, 스택이 쌓이는 기능이 가능하도록 한다.

* User Memory Access

유저 프로그램이 커널 메모리의 영역을 접근하는 것을 막기 위하여 ~/pintos/src/userprog/syscall.c 에 있는 syscall\_handler() 함수를 수정하고, 예외처리를 해 준다. 마찬가지로 커널 프로그램이 유저 메모리의 영역을 접근하는 것을 막기 위하여 ~/pintos/src/userprog/exception.c 에 있는 page\_fault() 함수를 수정하여 예외처리를 해 줄 것이다.

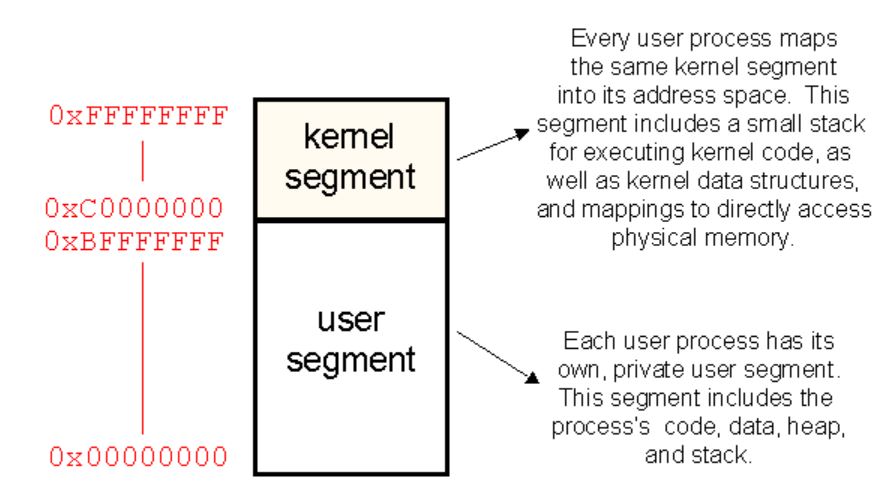
* System Calls

System Call을 하기 위해서 ~/pintos/src/userprog/syscall.c 의 syscall\_handler()를 작성하고, 관련 추가 함수를 구현한다. ~/pintos/src/lib/syscall-nr.h , ~/pintos/src/lib/user 내의 syscall.h와 syscall.c, ~/pintos/src/threads/thread.h를 수정하여 Struct thread에 자식과 부모를 구분 할 수 있는 멤버를 추가하고, process를 실행시키는 exec(), user program을 종료시키는 exit(), pintos를 종료 시키는 halt(), child process가 끝날 때 까지 기다리는 역할을 하는 wait의 기능을 구현한다. 또한 사용자로부터 입력을 받는 read()와, 스텍 영역의 argument를 화면에 출력하는 write() 기능을 표준 입출력 방법으로 구현한다. ~pintos/src/examples 내에 Sum.c.를 추가시켜 n번째 피보나치 수열의 값을 구하는 Fibonacci()와 4개의 숫자들의 합을 구하는 sun\_of\_four\_integers()의 기능을 구현하여 기본 Systemc Call에 추가한다.

* 1. **개발 내용**

Page : pintos가 실행이 되면 메모리에 4GB의 용량을 사용 한다. 그 중에서 3GB가 User 영역이고, 1GB가 Kernel 영역이다. User 영역에서는 프로그램이 메모리 스택영역에 위에서부터 밑으로 쌓이기 시작하게 되는데, 프로그램 하나가 실행되면 그 프로그램이 차지하는 메모리 영역이 4KB이고, 따라서 페이지 하나의 크기가 4KB가 된다. User 프로그램이 메모리상에 상주 할 수 있는 범위가 Page이다.

Process : User 프로그램의 개별적인 부분들을 Process라고 한다. 사용자가 프로그램을 실행하면 Process라는 thread에 할당이 되고, User process가 Kernel process에 접근하는 것을 막기 위하여 0XC0000000를 기준으로, 0x00000000부터 0xC0000000까지가 User virtual memory 공간으로, 그 이상의 범위를 Kernel virtual Memory 만큼의 공간으로 나누어 주어, 서로의 메모리를 참조하는 것에 대하여 예외처리를 발생시켜 오류를 내도록 한다. 그림으로 보면 다음과 같다.



이번 프로잭트에서 구현 할 System call은

|  |
| --- |
| SYS\_HALT : pintos를 종료시킨다.  SYS\_EXIT : 현재 User program을 종료시키고 Kernel에 System call의 상태를 return 시킨다. 이 상태가 0이면 success를, 0이 아니면 error를 뜻한다.  SYS\_EXEC : 주어진 명령에서 얻는 이름을 갖는 process를 수행시킨다.  SYS\_WAIT : pid를 갖는 child process를 기다리고 child의 exit status를 찾아온다.  SYS\_READ : 파일에서 size byte만큼 buffer에 읽어온다.  SYS\_WRITE : 파일에 size byte만큼 쓴다.  SYS\_FIBONACCI : N번째 피보나치 수열의 값을 return  SYS\_SUM : a+b+c+d 의 값을 return |

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

|  |  |
| --- | --- |
| 날짜 | 내용 |
| 2019/10/06 | 매뉴얼 분석 및 pintos 실행방법 숙지 |
| 2019/10/07 | 전체강의 내용을 참고한 매뉴얼 분석 |
| 2019/10/12 | User memory access 구현 |
| 2019/10/13 ~ 2019/10/17 | Argument passing 구현하고,  Argument passing의 출력 결과를 확인 |
| 2019/10/18 ~ 2019/10/25 | Systemcall을 구현, 여러가지 내용 추가  Thread 구조를 수정하는 작업과 System call의 실행 확인 |
| 2019/10/26 ~ 2019/10/29 | Fibonacci , sum\_of\_four\_integers 구현 및 실행 확인  각종 예외처리 및 프로잭트 마무리 |
| 2019/10/30 ~ 2019/11/3 | 보고서 작성 |

* 1. **개발 방법**
     + - 1. Argument Passing :
     1. Argument Parsing: 사용자가 콘솔에서 핀토스에 전달한 명령어를 어절 별로 관리하기 위해 ~/pintos/src/userprog/process.c 의 load 함수에서 입력받은 명령어를 ‘ ‘를 기준으로 tokenize 작업을 수행하는 기능을 추가한다.
     2. stack 영역에 아규먼트를 저장하기 위해 1)와 같은 함수에서 esp를 이용한 메모리 접근 기능을 구현한다.

2. System Call

1) halt: system call의 종류 중 하나인 halt를 구현하기 위해 ~/pintos/src/userprog/syscall.c의 systemcall\_handler 함수에 관련 내용을 추가하고, void halt() 함수를 추가 구현한다.

2) exit: system call의 주요 기능 중 하나인 exit() 기능을 구현하기 위해 ~/pintos/src/userprog/syscall.c의 systemcall\_handler 함수에 관련 내용을 추가하고, int exit(int status) 함수를 추가 구현한다. 또한, 스레드의 정상 종료 여부를 표현하기 위해 ~/pintos/src/threads/thread.h의 struct thread에 정수 형태의 exit 멤버 변수를 추가한다.

3) write and read: systemcall의 주요 기능인 read와 write를 표준 입출력 방식으로 구현하기 위해 ~/pintos/src/userprog/syscall.c의 systemcall\_handler 함수에 관련 내용을 추가하고, int read(int, char\*, unsigned)와 int write(int, char\*, unsigned) 함수를 추가 구현한다.

4) exec: systemcall의 주요 기능인 exec 기능을 구현하기 위해 ~/pintos/src/userprog/syscall.c의 systemcall\_handler에 관련 내용을 추가하고, int exec(char\*) 함수를 추가 구현한다. 또한, 관련 함수인 ~/pintos/src/userprog/process.c의 process\_execute에 프로그램 파싱 기능을 추가한다.

5) wait: systemcall의 주요 기능 중 하나인 wait 기능을 구현하기 위해 ~/pintos/src/userprog/syscall.c의 systemcall\_handler에 관련 내용을 추가하고, int waint(tid\_t) 함수를 추가 구현한다. 또한, 자식 스레드와 부모 스레드에 접근할 수 있도록 ~/pintos/src/threads/thread.h의 struct thread에 자식 스레드와 부모 스레드, 스레드의 작업을 일시정지할 수 있는 semaphore 멤버를 추가한다. 그리고 ~/pintos/src/userprog/process.c의 process\_wait 함수를 구현하고, process\_exit 함수 내용을 수정한다.

3. Memory Accessing

1) 유저 프로그램이 커널 영역의 메모리에 접근하는 것을 방지하기 위해 ~/pintos/src/userprog/syscall.c의 systemcall\_handler 함수에서 커널 영역 접근 여부를 체크하는 기능을 추가한다.

2) 커널 프로그램이 유저 영역의 메모리에 접근하는 것을 방지하기 위해 ~/pintos/src/userprog/exception.c의 page\_fault 함수에서 프로그램의 오류를 잃으킨 메모리가 커널/유저 영역에 있는지를 체크하는 기능을 추가 구현한다.

4. 추가 Systemcall 구현

새로운 systemcall Fibonacci()와 sum\_of\_four\_int()를 구현한다.

* + - 1. 사용자로부터 4개의 아규먼트를 프로그램에 전달하기 위해 ~/pintos/src/lib/user/syscall.c에 syscall4를 추가 정의하고, int Fibonacci()와 int sum\_of\_four\_int 함수의 프로토타입을 추가한다.
      2. 새로 추가할 기능을 systemcall table에 등록하기 위해 ~/pintos/src/lib/syscall-nr.h에 SYS\_FIBONACCI와 SYS\_SUM을 추가한다.
      3. 앞서 추가한 system call의 table 정보를 이용하여 실질적인 함수 구현을 위해 ~/pintos/src/userprog/syscall.c의 systemcall\_handler 함수에 관련 내용을 추가하고, int Fibonacci(int), int sum\_of\_four\_int(int, int, int, int) 함수를 추가한다.
      4. 지금까지 구현한 systemcall 기능을 이용하는 사용자 프로그램을 sum.c로 작성하여 ~/pintos/src/examples에 저장한다.
      5. 4)에서 추가한 사용자 프로그램을 pintos에 등록하기 위해 ~/pintos/src/examples/Makefile을 수정한다.
  1. **연구원 역할 분담**
* 서인호: 프로젝트 주요 기능 구현
* 팀원1: 시스템 점검 및 보고서 작성

1. **연구 결과**
   1. **합성 내용**
      * + 1. **System call**

|  |
| --- |
|  |

* + - * 1. **Argument passing**

|  |
| --- |
| +)Parsing 및 esp constructing 추가 |

* + - * 1. **User memory access 4. Kernel memory access**

|  |
| --- |
|  |

|  |
| --- |
|  |

* 1. **제작 내용**
     + - 1. **Argument Passing**
     1. 핀토스 실행 배경

처음 핀토스가 실행이 되면 threads 내의 init.c가 먼저 실행이 되면서 Init process가 돌기 시작한다. 여기서 새로운 프로세스가 만들어지고, 스레드를 만드는 작업을 하는데, 여기서 argument passing을 하기 위하여 main에서 run\_task()를 호출하고, 이 함수 내에서 process\_execute 함수를 호출한다. process\_execute 내에서는 load()를 호출하여, 유저가 들어갈 메모리가 비어있는 부분에 t라는 thread를 선언하고, 현재 실행중인 스레드의 정보를 t에 담아온다.

* + 1. 파싱

Load 함수에 파라미터로 전달된 file\_name를 ‘ ‘ 를 기준으로 나누어 userstack에 저장한다. 이는 사용자가 입력한 명령어를 argument 단위로 나누어 관리하여 프로그램 이름, 파라미터 개수 및 파라미터 정보를 활용하기 위함이다.

* + 1. Stack 영역에 Argument 정보 쌓기

cnt=0 는 사용자가 입력한 명령이 몇 어절인지 argument의 개수를 체크하는 것이고, stacksize=0 은 argument 하나당 스택영역에 메모리가 저장되는데, argument가 들어 온 개수에 따른 메모리의 용량을 저장하는 역할을 한다.

Userstack[40][20]은 입력한 문자를 어절마다 분석하기 위한 역할이고,

Stackaddress[40]은 스택 주소를 써서 배열을 40개 선언하여, argument들이 저장 되어있는 메모리의 주소 값을 Stackaddress에 저장한다.

Stack에 정보를 쌓는 방법에 대해서 생각해보자. 입력 받은 argument를 토큰에 있는 문자열을 쌓는 경우에 esp에서 토큰한 문자열의 길이 +1 (1은 \0문자)만큼의 포인터를 빼 준다. 이는 스택에 문자열을 쌓기 위한 공간을 만들어내는 작업니다. 그런 다음, memmove() 함수를 통하여 esp 위치에 문자열을 넣는다. 그 다음 4의 배수를 맞추어 word alignment 개수를 센다. 그 다음 alignment 되어야 하는 개수 만큼 공간을 할당하고, 0을 넣어준다. 그 후의 null 값의 크기는 4로 고정되어 있으므로, esp에서 4를 빼 준 후, null 값인 0을 넣어준다. 각 문자열들이 저장되어 있는 주소를 위한 공간은 stackaddress를 이용하는데, 주소는 항상 4바이트의 크기 공간을 가지기 때문에, esp에서 항상 4를 빼어 공간을 생성하고, cnt만큼 for문을 수행하여 주소를 넣어주면 된다. userstack의 첫 주소, cnt의 값, return address는 모두 4byte 크기의 공간을 가지기 때문에 esp에서 4를 빼준 후에 알맞은 정보를 할당해 스택을 쌓아준다. 예로 ‘echo x’ 의 경우를 보자.

시작을 하면 0xC0000000부터 시작이 되는데, 스택이므로 x가 먼저 들어가게 된다. x뒤에는 보이지 않는 null 문자 1바이트가 존재하므로, x는 총 2바이트를 차이하게 되고, 그 밑에 그택에는 echo가 들어갈 것이다. 여기도 또한 null이 포함되어 있으므로 총 5바이트를 차지한다. 그 밑에는 word alignment가 들어가는데, 1바이트를 차지한다. 그 밑에 x를 가리키는 주소의 주소값이 들어가고, 그 밑에 echo를 가리키고 있는 주소의 주소값이 들어간다. 그 밑에는 전체 argument의 개수가 들어간다. 이 내용을 hex\_dump()를 통해 확인하면 다음과 같다.

|  |
| --- |
| bfffffe0 00 00 00 00 02 00 00 00-ec ff ff bf f9 ff ff bf |................|  bffffff0 fe ff ff bf 00 00 00 00-00 65 63 68 6f 00 78 00 |.........echo.x.|  c0000000 53 ff 00 f0 53 ff 00 f0-c3 e2 00 f0 53 ff 00 f0 |S...S.......S...|  c0000010 53 ff 00 f0 53 ff 00 f0-53 ff 00 f0 53 ff 00 f0 |S...S...S...S...|  c0000020 a5 fe 00 f0 87 e9 00 f0-2c d6 00 f0 2c d6 00 f0 |........,...,...|  c0000030 2c d6 00 f0 2c d6 00 f0-57 ef 00 f0 2c d6 00 f0 |,...,...W...,...|  c0000040 26 55 00 c0 |&U.. | |

2. User Memory Access

핀토스는 많은 운영체제와 마찬가지로 메모리를 유저 프로그램 영역과 커널 프로그램 영역으로 나누어 관리한다. 유저 프로그램은 커널 영역 메모리에 접근할 수 없고, 커널 프로그램은 유저 영역 메모리에 접근할 수 없다. 이는 운영체제가 수행하는 일과 유저가 수행하는 일을 나누어 유저 입장에서 사용성을 높이고 운영체제 입장에서 시스템의 안정성을 확보하는 데 도움이 된다. 핀토스에서 위의 기능을 구현하기 위해 ~/pintos/src/threads/vaddr.h의 is\_user\_vaddr() 함수와 is\_kernel\_vaddr() 함수를 사용하였다.

* + - 1. Is\_user\_vaddr()

핀토스에서 유저 영역의 메모리는 0x00000000부터 0xC0000000까지이고, 커널 영역의 메모리는 0xC0000000부터 끝까지이다. 즉, 현재 프로그램이 접근하고자 하는 메모리가 0xC0000000보다 작으면 유저 프로그램의 영역 메모리이고, 그 이상이면 커널 프로그램을 위한 메모리 영역이다. 유저 프로그램이 system call을 사용할 때 해당 system clal을 호출하는 메모리가 유저 영역의 프로그램이 맞는지를 확인해줄 필요가 있다. 따라서 system call 함수를 호출하기 전에 is\_user\_vaddr() 함수를 사용해 이를 확인한다. 만약 현재 접근하고자 하는 메모리가 커널 영역의 메모리라면 프로그램을 강제 종료한다.

* + - 1. Is\_kernel\_vaddr()

마찬가지로, 커널 프로그램이 유저 영역의 메모리에 접근하는 것을 막아야 한다. 이를 위해 ~/pintos/src/userprog/exception.c의 page\_fault에서 문제를 이르키는 주소값이 커널 영역의 메모리인지를 확인한다. 만약 커널 영역이 아니고 유저 영역의 주소값이라면 프로그램을 강제종료한다.

3. System Call

1) halt: ~/pintos/src/lib/syscall-nr.h에 정의되어 있는 SYS\_HALT API를 사용하여 ~/pintos/src/userprog/syscall.c의 systemcall\_handler에서 현재 호출하고자 하는 시스템 콜의 종류를 확인하여 halt()를 실행한다. Halt()의 내용은 ~/pintos/src/devices/shutdown.c에 정의된 shutdown\_power\_off()를 수행한다.

2) exit: halt와 마찬가지로 이미 등록된 SYS\_EXIT API를 사용해 systemcall\_handler의 내용을 채운다. 다만, 스레드가 종료될 때 종료 상태를 저장할 수 있는 공간이 필요하므로, ~/pintos/src/threads/thread.h의 struct thread에 int exit를 추가한다. Int exit(int status) 함수를 추가하되, 내용은 파라미터로 전달받은 status를 thread 구조체의 exit 멤버에 저장하도록 한다. 그 후 thread\_exit() 함수를 수행하여 스레드를 종료한다.

3) write & read: 앞서 구현한 두 개의 시스템콜과 마찬가지 방법으로 API를 이용해 구현한다. 새로 구현해야 하는 int write() 함수는 fd, buffer, size 세 개의 파라미터를 갖는다. 첫 번째 파라미터인 fd는 표준 입출력 방법 중 입력인지 출력인지를 결정한다. Fd가 1이면 화면에 내용을 출력하고, 0이면 내용을 읽는다. 즉, 첫 번째 파라미터가 1이면 두 번째 파라미터인 buffer를 세 번째 파라미터인 size만큼 화면에 출력한다. 반대로 fd가 0이면 size 만큼의 입력을 받아 buffer에 저장한다.

4) exec: 다른 시스템콜과 마찬가지로 기존의 API를 이용해 systemcall\_handler을 채운다. 새롭게 구현해야 하는 int exec()는 파라미터로 문자열을 입력받는다. Exec 함수는 전달받은 이름으로 새로운 프로세스를 실행한다. 즉, process\_execute()의 역할을 수행한다.

5) wait:

(1) 다른 시스템 콜과 마찬가지로 기존의 API를 이용해 systemcall\_handler에서 wait 함수를 호출한다. 새로 작성하는 함수인 int wait()는 파라미터로 스레드 아이디 하나를 전달받아 해당 스레드를 wait 한다. 이를 위해 ~/pintos/src/userprog/process.c의 process\_wait() 함수를 호출한다.

(2) wait 기능을 완성하기 위해 process\_wait 함수를 구현한다. Process\_wait는 process\_execute와 함께 수행되어 현재 수행중인 스레드를 정지하고, 새로 생기는 스레드의 작업이 끝날 때까지 기다린다. 이때, 정지되는 스레드를 부모 스레드라고 하고, 새롭게 생성되어 수행되는 스레드를 자식 스레드라고 한다. 즉, process\_wait는 파라미터로 넘어오는 스레드 아이디가 멈출 때까지 멈춰있어야 한다. 그리고 자식 스레드가 작업을 마치고 exit() 함수에 의해 종료되면 자식 스레드의 종료 상태를 리턴한다. 자식 스레드를 찾을 수 없거나 자식 스레드가 정상적으로 종료되지 않았다면 -1을 리턴한다.

(3) process\_wait에서 부모 스레드가 자식 스레드의 수행에 따라 자신의 상태를 조정하고, 자식 스레드가 종료되며 자신의 부모 스레드에게 자신이 종료됨을 알리기 위해 ~/pintos/src/threads/thread.h의 struct thread에 semaphore 자료형의 pause와 list children, list\_elem child\_elem, thread 형태의 parent을 멤버로 추가한다. Pause는 스레드의 작업 상태를 일시정지하는 역할을 하고, children은 스레드의 자식 정보를 리스트 형태로 저장한다. 이때 부모는 thread 형태로 추가했는데, 이는 하나의 스레드가 여러 개의 자식을 가질 수 있지만 모든 스레드는 하나의 부모만을 가질 수 있기 때문이다. 이후, ~/pintos/src/threads/thread.c의 init\_thread 함수에서 새로운 스레드가 생성될 때 자식 리스트를 초기화하는 list\_init()과 semaphore 상태를 초기화하는 sema\_init()을 추가하였다.

(4) process\_wait은 전달받은 파라미터인 스레드 아이디와 같은 스레드가 현재 수행중인 스레드의 자식 중에 있는지 확인하고, 만약 그러한 스레드가 있다면 현재 작업중인 스레드를 일시정지한다. 이후 새롭게 생성된 스레드의 작업이 끝나 exit() 함수가 수행되면 ~/pintos/src/userprog/syscall.c의 exit 함수에서 먼저 종료 상태를 저장하고, ~/pintos/src/userprog/process.c의 process\_exit() 함수가 호출되어 해당 스레드가 제거된다. 이때, process\_exit() 함수에서 스레드를 제거하기 전에 일시 정지되어 있는 부모 스레드를 깨우고, 자기 자신을 일시정지한다. 일시정지 상태에서 해제된 부모 스레드는 자식 스레드의 종료 상태를 로컬 변수에 복사하고 일시정지되어 있던 자식 스레드를 깨워 스레드 정보가 완전히 삭제되도록 한다. 이때, 자식 스레드에서 exit() 함수를 진행하기 전에 부모 스레드를 깨우고 자기 자신을 일시정지하는 이유는 process\_exit() 함수가 끝나면 자식 스레드의 정보가 시스템에서 완전히 삭제되어 부모 스레드에서 자식 스레드의 종료 상태를 복사할 수 없게 되기 때문이다. 즉, 자식 스레드는 자기 자신이 완전히 사라지기 전에 부모 스레드를 깨우고 부모 스레드가 자신으로부터 필요한 작업을 수행할 때까지 잠시 대기했다가 부모가 자식 스레드의 종료를 확인하고 나서야 완전히 삭제되는 것이다.

**4. 추가 시스템콜 구현**

(1) 4개의 파라미터를 전달받기 위해 ~/pintos/src/lib/user/syscall.c에서 필요한 API인 syscall4를 추가 정의한다. 그리고 Fibonacci 함수와 sum\_of\_four\_int의 프로토타입을 정의한다.

(2) ~/pintos/src/userprog/syscall.c에서 기존의 system call과 마찬가지로 systemcall\_handler에서 함수 호출 내용을 추가하고 관련 함수를 구현한다. Fibonacci 함수는 파라미터로 하나의 정수를 입력받아 파라미터로 넘어온 항의 피보나치 수열을 반환하고, sum\_of\_four\_int 함수는 입력받은 4개의 정수 합을 반환한다.

(3) 새로 구현한 시스템 콜을 이용하는 프로그램인 sum.c를 작성하여 ~/pintos/src/examples에 추가한다. Sum.c는 메인 함수에서 아규먼트 개수와 아규먼트를 문자열 형태로 전달받는다. 이 프로그램에서는 숫자 네 개를 입력받아 피보나치 함수와 정수의 합 함수의 결과를 얻을 것이기 때문에 입력받은 문자열을 정수 형태로 변환하는 작업이 필요하다. 이를 위해 문자열의 문자에 하나씩 접근하여 해당 문자의 아스키 코드 값을 정수로 변환하였다. 문자열을 정수로 바꾸는 작업이 완료되면, 변환된 정수를 피보나치 수열과 정수의 합 함수로 넘겨 답을 구해 화면에 출력한다.

(4) 새로 만든 sum.c 프로그램을 핀토스에 등록하기 위해 ~/pintos/src/examples/Makefile의 PROGS에 sum을 추가하고, sum\_SRC를 추가하였다.

(5) 위의 과정을 거쳐 구현한 sum 프로그램의 실행 결과는 다음과 같다.

|  |
| --- |
| **~/pintos/src/userprog $ pintos --filesys-size=2 -p ../examples/sum -a sum -- -f -q run 'sum 10 20 40 56'**  **Loading.........**  **Kernel command line: -f -q extract run 'sum 10 20 40 56'**  **Pintos booting with 3,968 kB RAM...**  **367 pages available in kernel pool.**  **367 pages available in user pool.**  **Calibrating timer... 419,020,800 loops/s.**  **hda: 5,040 sectors (2 MB), model "QM00001", serial "QEMU HARDDISK"**  **hda1: 147 sectors (73 kB), Pintos OS kernel (20)**  **hda2: 4,096 sectors (2 MB), Pintos file system (21)**  **hda3: 82 sectors (41 kB), Pintos scratch (22)**  **filesys: using hda2**  **scratch: using hda3**  **Formatting file system...done.**  **Boot complete.**  **Extracting ustar archive from scratch device into file system...**  **Putting 'sum' into the file system...**  **Erasing ustar archive...**  **Executing 'sum 10 20 40 56':**  **55 126**  **sum: exit(0)**  **Execution of 'sum' complete.**  **Timer: 75 ticks**  **Thread: 3 idle ticks, 72 kernel ticks, 0 user ticks**  **hda2 (filesys): 53 reads, 168 writes**  **hda3 (scratch): 81 reads, 2 writes**  **Console: 852 characters output**  **Keyboard: 0 keys pressed**  **Exception: 0 page faults**  **Powering off...** |

여기서 56은 피보나치 수열 중 첫 번째 파라미터인 10번째 항의 값이고, 126은 네 개의 파라미터의 총 합을 의미한다.

* 1. **시험 및 평가 내용**
     + - 1. sum\_of\_four\_int() 수행결과

|  |
| --- |
|  |

* + - * 1. make check 수행결과

|  |
| --- |
|  |

* + - * 1. make grage 수행결과

|  |
| --- |
|  |

1. **기타**
   1. **연구 조원 기여도**

* **서인호 : 90% , 팀원1 : 10%**
  1. **소감**
* 평소 너무나도 익숙하게 사용하는 운영체제가 기술적으로 복잡한 과정을 거쳐 구현되었다는 것을 본 프로젝트를 통해 직접 배웠습니다. 본 프로젝트 경험을 통해 모든 application을 수행하는 플랫폼인 운영체제에 대해 깊게 이해하게 되어 향후 사용자 프로그램을 개발하거나 서버 등을 다룰 때 해당 프로그램이 실행되는 운영체제를 참고하여 더욱 효율적인 설계가 가능할 것으로 기대됩니다. 또한, 사용자 입장에서 운영체제의 복잡성을 인지하지 못할 정도의 완성도 있는 운영체제가 좋은 운영체제라는 인식을 잊지 않을 것입니다.