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

담당 교수 : 문의현

학번 / 이름 : 20161250 / 이지호

개발 기간 : 2022.10.14 ~ 2022.10.22

1. **개발 목표**

주어진 PintOS 소스코드에서 System call, System call handler, Argument passing, User stack을 구현하여 user program이 정상적으로 수행되도록 한다. 또한, 두 개의 새로운 system call인 fibonacci()와 max\_of\_four\_int()를 구현하여 정상적으로 동작하는지 확인한다.

1. **개발 범위 및 내용**
   1. **개발 범위**
2. Argument Passing

pintos 명령어로 kernel에 전달하는 argument 정보를 stack에 push한다.

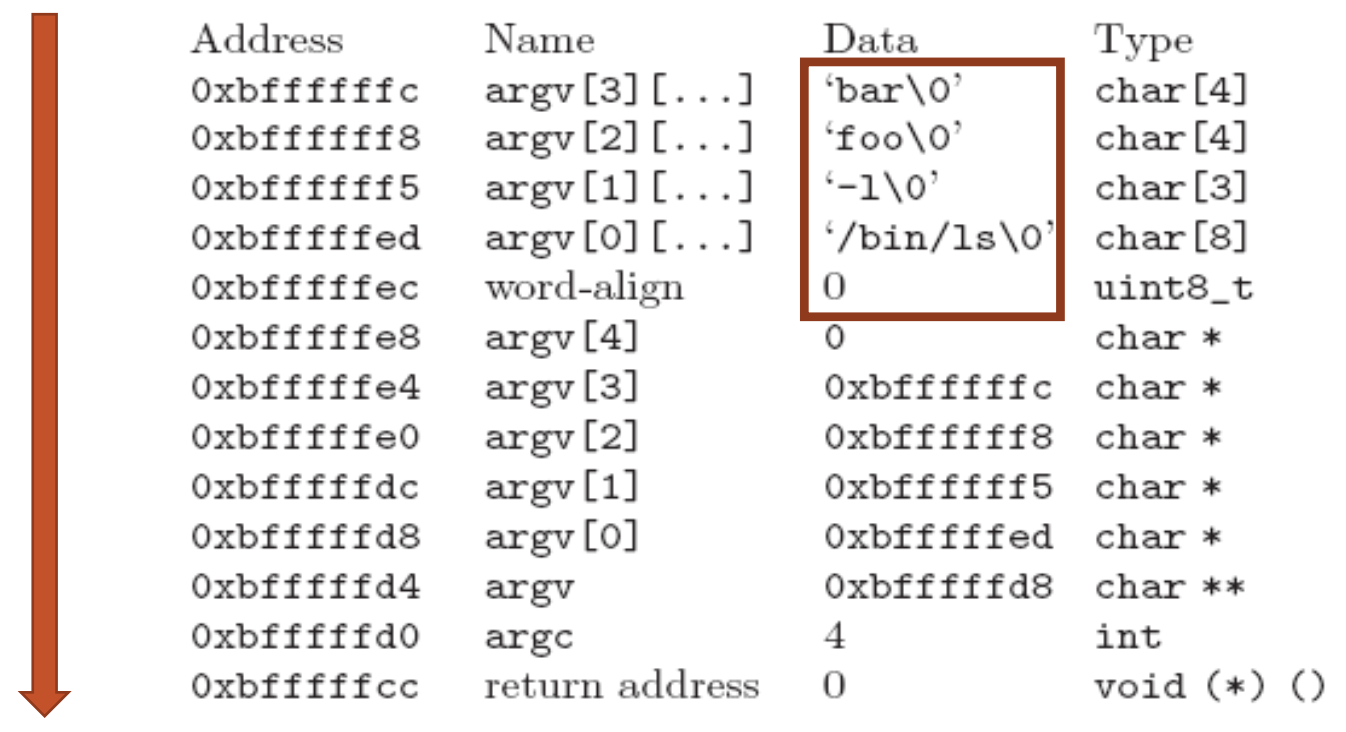
1. User Memory Access

만약 user program에서 유효하지 않은 주소값을 접근하는 경우(null pointer, kernel address 등), 해당 주소값으로의 접근을 제한하고 프로세스를 종료해야 한다. 이를 위해 threads/vaddr.h에 정의된 is\_user\_vaddr(), is\_kernel\_vaddr() 등을 활용한다.

1. System Calls

System call을 위해 userprog/syscall.c의 syscall\_handler()에서 각 syscall\_num에 대해 처리하는 로직을 작성하여 system call이 user program을 호출할 수 있도록 연결한다. halt(), exit(), exec(), wait(), 그리고 read(), write()를 구현한다. 추가로 fibonacci(), max\_of\_four\_int()도 구현한다.

* 1. **개발 내용**
* Argument Passing
  + 아래 예시는 “/bin/ls -l foo bar”를 실행할 때 argument passing이 적용된 stack의 모습이다.



* + - Stack은 최초에 PHYS\_BASE(0xc0000000)로 할당되어 push가 진행될수록 주소값이 감소하는 방향으로 자란다.
    - 각 argument를 역순으로 push한다.
    - 빠른 포인터 연산을 위해, stack pointer가 4-byte 단위로 움직일 수 있도록 일정 메모리 공간을 0으로 채운다. 이 작업을 word-align이라고 한다.
    - [C standard] Argument 주소값을 push하기 전에, Null pointer를 push하여 실제 argument와 argument 주소값의 경계임을 나타낸다. (위 그림 상에서 argv[4], Null Pointer Sentinel)
    - 각 argument의 주소값을 역순으로 push한다.
    - 첫 번째 arg의 주소값(argv[0])을 push한다.
    - Argument 개수를 나타내는 argc 값을 push한다.
    - Code segment 상에서 다음 instruction을 실행하기 위한 return address를 push한다.
* User Memory Access
  + Null pointer를 참조하려는 경우, 가상 메모리에 매핑되지 않은 주소값을 참조하려는 경우, user program에서 kernel address space를 참조하려는 경우를 invalid memory access로 간주하고, 이를 제한하고 프로세스를 종료해야 한다.
  + 주소값의 null 여부를 검사하고, thread/vaddr.h에 정의된 is\_user\_vaddr(), is\_kernel\_vadder()과 userprog/pagedir.h의 pagedir\_get\_page()를 사용하여 address validity를 검사한다.
* System Calls
  + PintOS는 program이 user mode로 실행되고 있을 때 메모리나 디스크에 접근할 수 없다. 이와 같은 작업을 수행하기 위해서는 kernel mode로 실행되어야 하며 이때 사용되는 것이 system call이다. System call을 통해 user program이 kernel memory에 접근하여 exit, read, write 등 여러 가지 기능을 사용할 수 있다.
  + 이번 프로젝트에서 개발할 system call은 다음과 같다.
    - **halt()** PintOS를 종료한다.
    - **exit()** 실행 중인 프로세스를 종료하고, 종료한 프로세스의 이름과 status를 출력한다.
    - **exec()** 새로운 프로세스를 실행하고 프로세스 ID를 반환한다.
    - **wait()** 인자로 전달받은 tid에 해당하는 thread가 종료될 때까지 기다린다.
    - **read()**, **write()** 이번 프로젝트에서는 STDIN, STDOUT에 대해서만 동작하도록 구현한다.
    - **[+] fibonacci()** N번째 피보나치수를 반환한다.
    - **[+] max\_of\_four\_int()** int형 정수 네 개를 입력받아 그 중 가장 큰 값을 반환한다.
  + User program이 실행되어 system call API를 호출하면 argument와 system call number가 stack에 push되고 interrupt를 발생시킨다. 이때, intr\_handler는 system call에 대한 정보를 intr\_frame이라는 struct에 저장하여 syscall\_handler에게 전달한다. syscall\_handler에서는 전달받은 intr\_frame내의 esp라는 필드를 가지고 stack에 접근할 수 있고, 이로부터 system call number와 argument 정보를 읽어서 적절한 로직을 수행한다. return value는 intr\_frame의 eax 필드에 저장되어 이 값은 user program에 전달된다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* **II. A.의 개발 범위를 포함하여 구현 내용에 대한 일정 작성**

2022.10.14 프로젝트 명세 확인

2022.10.15 - 2022.10.17 Argument passing 구현

2022.10.18 - 2022.10.20 User memory access 체크, system call handler 작성

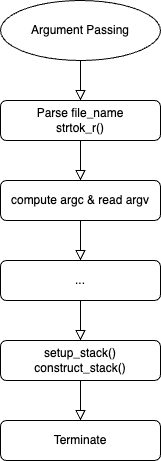
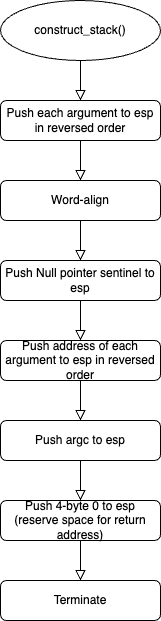
* halt(), exit(), write(), read(), exec(), wait()

2022.10.21 make check, make grade 확인 & 예외 처리

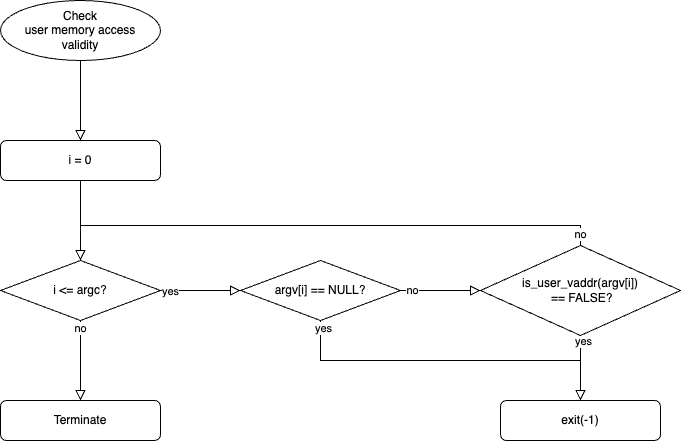
2022.10.22 Additional system call 구현 (fibonacci(), max\_of\_four\_int())

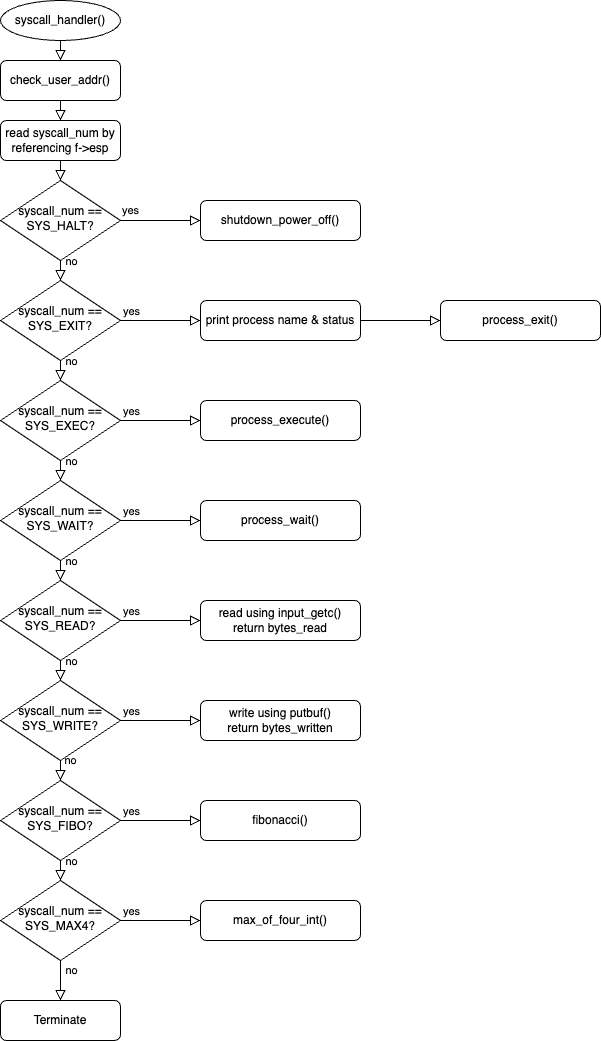
* 1. **개발 방법**
* **II. B.의 개발 내용을 구현하기 위해 어느 소스코드에 어떤 요소를 추가 또는 수정할 것인지 설명. (함수, 구조체 등의 구현이나 수정을 서술)**
* 이번 프로젝트에서는 multi-thread를 고려하지 않으므로, process ID와 thread ID를 동일시하여 구현함.
* Argument passing을 구현하기 위해, userprog/process.c에 construct\_stack() 함수를 정의함. load() 내에서 setup\_stack()이 정상적으로 실행된 이후에 construct\_stack()을 호출하여 argument passing을 처리하고, 최종 stack pointer를 esp에 저장함.
* userprog/syscall.c에 정의된 syscall\_handler()에서는 인자로 전달받은 intr\_frame 내의 esp 필드를 참조하여 실행해야 할 system call의 종류(syscall\_num)와 argument들을 읽는다. syscall\_num의 값으로 switch-case문을 작성하여 각각 적절한 API를 호출한다. 메모리 접근 전에 check\_user\_addr()로 주소값의 validity를 검사한다.
* 두 system call fibonacci(), max\_of\_four\_int()를 userprog/syscall.c에 구현하고, lib/synch-nr.h 내에 정의된 enum(system call number)에 SYS\_FIBO, SYS\_MAX4를 추가로 정의한다. max\_of\_four\_int()는 네 개의 인자를 필요로 하므로, 네 개의 인자를 처리할 수 있는 syscall4()를 lib/user/syscall.c에 정의한다.

1. **연구 결과**
   1. **Flow Chart**
2. Argument Passing

1. User Memory Access



1. System Calls  
   
   1. **제작 내용**

* **II. B. 개발 내용의 실질적인 구현에 대해 코드 관점에서 작성.**
* **구현에 있어 Pintos에 내장된 라이브러리나 자체 제작한 함수를 사용한 경우 이에 대해서도 설명.**
* **개발상 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결책에 대해 설명.**

1. Argument Passing

* load()의 인자로 전달하는 file\_name을 parse하여 호출할 프로그램명과 argument를 구하였다. Parsing을 위해 strtok\_r()을 반복 호출하였다.
* */\* Argument Parsing \*/*
* int argc = 0;
* char \*argv[128], file\_name\_copy[128], \*ptr, \*ptr\_next;
* strlcpy(file\_name\_copy, file\_name, 128);
* ptr = strtok\_r(file\_name\_copy, delimiter, &ptr\_next);
* while (ptr)
* {
* argv[argc++] = ptr;
* ptr = strtok\_r(NULL, delimiter, &ptr\_next);
* }
* construct\_stack(argc, argv, esp)를 호출한다. 내부적으로 다음과 같이 동작한다.
* static void construct\_stack(int argc, char \*\*argv, void \*\*esp)
* {
* */\* argv[argc-1], ... , argv[0] \*/*
* void \*ptr\_argv = \*esp;
* int bytes = 0;
* for (int i = argc - 1; i >= 0; i--)
* {
* int len = strlen(argv[i]) + 1;
* bytes += len;
* \*esp -= len;
* strlcpy(\*esp, argv[i], len);
* }
* */\* word-align \*/*
* int word\_align\_len = (4 - bytes % 4) % 4;
* \*esp -= word\_align\_len;
* memset(\*esp, 0, word\_align\_len);
* */\* NULL pointer sentinel \*/*
* \*esp -= 4;
* \*\*(uint32\_t \*\*)esp = 0;
* */\* Address' of argv[argc-1], ... , argv[0] to stack. \*/*
* for (int i = argc - 1; i >= 0; i--)
* {
* int len = strlen(argv[i]) + 1;
* ptr\_argv -= len;
* \*esp -= 4;
* \*\*(uint32\_t \*\*)esp = (uint32\_t)ptr\_argv;
* }
* */\* Address of argv \*/*
* \*esp -= 4;
* \*\*(uint32\_t \*\*)esp = (uint32\_t)\*esp + 4;
* */\* argc \*/*
* \*esp -= 4;
* \*\*(uint32\_t \*\*)esp = argc;
* */\* return address \*/*
* \*esp -= 4;
* \*\*(uint32\_t \*\*)esp = 0;
* *// hex\_dump((uintptr\_t)\*esp, \*esp, PHYS\_BASE - \*esp, true);*
* }

1. User Memory Access

* userprog/syscall.c에 int syscall\_argc[]를 static한 전역 배열 변수로 선언한다. syscall\_argc[syscall\_num]은 syscall\_num에 해당하는 system call의 argc를 의미하고, 이는 syscall\_init() 내에서 초기화된다.
* static int syscall\_argc[SYS\_MAX] = {
* 0,
* };
* void syscall\_init(void)
* {
* intr\_register\_int(0x30, 3, INTR\_ON, syscall\_handler, "syscall");
* */\* [PROJECT-1] Initialize argc of each syscall function. \*/*
* syscall\_argc[SYS\_HALT] = 0;
* syscall\_argc[SYS\_EXIT] = 1;
* syscall\_argc[SYS\_EXEC] = 1;
* syscall\_argc[SYS\_WAIT] = 1;
* …
* }
* syscall\_handler()에서 4-byte pointer esp를 f->esp로 초기화하여 syscall\_num을 읽는다. 해당 system call의 argument 개수는 syscall\_argc[syscall\_num]에 저장되어 있으므로, 이를 참조하여 argument들의 메모리 주소에 대한 validity를 검사한다.
* static void
* syscall\_handler(struct intr\_frame \*f UNUSED)
* {
* int syscall\_num = 0;
* uint32\_t \*esp = f->esp;
* check\_user\_addr(esp); */\* syscall\_num \*/*
* syscall\_num = \*esp;
* if (syscall\_num < 0 || syscall\_num >= SYS\_MAX)
* syscall\_exit(-1);
* */\* Check if address of each argument is valid. \*/*
* for (int i = 1; i <= syscall\_argc[syscall\_num]; i++)
* check\_user\_addr(&esp[i]);
* …

1. System Calls

* **syscall\_halt()**  
  shutdown\_power\_off()를 호출하여 PintOS를 종료한다.
* static void syscall\_halt(void)
* {
* shutdown\_power\_off();
* }
* **syscall\_exit()**  
  현재 thread의 이름과 status를 출력하고, thread를 종료한다. 내부적으로 thread\_exit()을 호출한다.
* static void syscall\_exit(int status)
* {
* struct thread \*cur = thread\_current();
* printf("%s: exit(%d)\n", cur->name, status);
* thread\_exit();
* }
* **syscall\_exec()**  
  해당 프로세스를 실행한다. 내부적으로 process\_execute()를 호출한다.
* static pid\_t syscall\_exec(const char \*file)
* {
* return (pid\_t)process\_execute(file);
* }
* **syscall\_wait()**  
  해당 프로세스가 종료되기를 기다린다. 내부적으로 process\_wait()를 호출한다.
* static int syscall\_wait(pid\_t pid)
* {
* return process\_wait((tid\_t)pid);
* }
* **syscall\_read()**  
  이번 프로젝트에서는 STDIN에 대한 입력만 처리한다. buffer와 size를 인자로 받아, byte 단위로 데이터를 읽는 input\_getc()를 반복 호출하여 buffer에 저장한다. 성공 시 읽어들인 byte 수를 반환한다.
* static int syscall\_read(int fd, void \*buffer, unsigned size)
* {
* int bytes\_read = 0;
* if (fd == 0) */\* STDIN \*/*
* {
* for (bytes\_read = 0; bytes\_read < (int)size; bytes\_read++)
* {
* uint8\_t input = input\_getc();
* \*(uint8\_t \*)(buffer + bytes\_read) = input;
* }
* }
* */\* TODO implementation for other input stream \*/*
* return bytes\_read;
* }
* **syscall\_write()**  
  이번 프로젝트에서는 STDOUT에 대한 출력만 처리한다. buffer와 size를 인자로 받아, buffer로부터 size bytes를 읽어서 putbuf()를 호출한다. 성공 시 출력한 byte 수를 반환한다.
* static int syscall\_write(int fd, const void \*buffer, unsigned size)
* {
* int bytes\_written = 0;
* if (fd == 1) */\* STDOUT \*/*
* {
* putbuf(buffer, size);
* bytes\_written = size;
* }
* */\* TODO implementation for other output stream \*/*
* return bytes\_written;
* }

1. Additional System calls

* **공통**  
  lib/syscall-nr.h에 fibonacci(), max\_of\_four\_int()를 위한 system call number를 새롭게 정의한다. 기존에 정의한 SYS\_MAX 위에 정의하면, 기존 코드에 지장 없이 새로운 enum 변수를 추가할 수 있다.
* */\* System call numbers. \*/*
* enum
* {
* *…*
* */\* 20161250 Jiho Rhee \*/*
* SYS\_FIBO,
* SYS\_MAXOF4INT,
* SYS\_MAX */\* End of syscall enum.\*/*
* };

userprog/syscall.c의 syscall\_handler()에 위 두 가지 system call에 대한 handling logic을 추가한다.

* switch (syscall\_num)
* {
* *…*
* */\* [PROJECT-1] Additional System Calls. \*/*
* case SYS\_FIBO:
* f->eax = syscall\_fibonacci((int)esp[1]);
* break;
* case SYS\_MAXOF4INT:
* f->eax = syscall\_max\_of\_four\_int(
* (int)esp[1],
* (int)esp[2],
* (int)esp[3],
* (int)esp[4]);
* break;
* default:
* NOT\_REACHED();
* break;
* }
* **fibonacci()**  
  인자 하나만을 필요로 하는 system call이므로 syscall1()을 호출한다.
* int fibonacci(int num)
* {
* return syscall1(SYS\_FIBO, num);
* }

userprog/syscall.c에는 아래와 같이 구현한다.

* */\* Return nth Fibonacci number. \*/*
* int syscall\_fibonacci(int num)
* {
* if (num < 0) */\* input error \*/*
* return -1;
* if (num == 0 || num == 1) */\* base case \*/*
* return num;
* int prev = 1, ret = 1;
* for (int i = 3; i <= num; i++)
* {
* int tmp = ret;
* ret += prev;
* prev = tmp;
* }
* return ret;
* }
* **max\_of\_four\_int()**  
  인자 네 개를 필요로 하는 system call인데, 인자 네 개를 stack에 저장하는 system call API가 정의되어 있지 않으므로 lib/user/syscall.c에 syscall4()를 새로 정의한다.
* */\* Invokes syscall NUMBER, passing arguments ARG0, ARG1, ARG2,*
* *and ARG3, and returns the return value as an `int'. \*/*
* #define syscall4(NUMBER, ARG0, ARG1, ARG2, ARG3)
* ({
* int retval;
* asm volatile("pushl %[arg3]; pushl %[arg2]; pushl %[arg1]; pushl %[arg0]; "
* "pushl %[number]; int $0x30; addl $20, %%esp"
* : "=a"(retval)
* : [number] "i"(NUMBER),
* [arg0] "r"(ARG0),
* [arg1] "r"(ARG1),
* [arg2] "r"(ARG2),
* [arg3] "r"(ARG3)
* : "memory");
* retval;
* })

userprog/syscall.c에는 다음과 같이 정의한다.

* int syscall\_max\_of\_four\_int(int num1, int num2, int num3, int num4)
* {
* int ret = num1 > num2 ? num1 : num2;
* if (ret < num3)
* ret = num3;
* if (ret < num4)
* ret = num4;
* return ret;
* }
* examples/additional.c  
  작성한 system call의 정상 동작 여부를 확인하기 위한 user program이다. main()에 전달되는 인자는 모두 문자열이므로 이를, int형으로 변환해야 한다. C에서 제공하는 strtol()을 PintOS에서 사용할 수 없으므로 문자열을 int형으로 변환하는 함수를 직접 구현하였다.
* int str\_to\_long(char \*str)
* {
* int read = 0;
* int sign = 1;
* int s\_idx = 0;
* int noting\_to\_convert = 1;
* if (str[0] == '-')
* {
* sign = -1;
* s\_idx++;
* }
* for (int i = s\_idx; str[i] != '\0'; i++)
* {
* if (str[i] < '0' || '9' < str[i])
* {
* printf("format error: '%c' in '%s' is not a digit.\n", str[i], str);
* exit(EXIT\_FAILURE);
* }
* noting\_to\_convert = 0;
* read \*= 10;
* read += sign \* (str[i] - '0');
* if (read < INT\_MIN || INT\_MAX < read)
* {
* printf("Out-of-range: Input '%s' is out of range(INT).\n", str);
* exit(EXIT\_FAILURE);
* }
* }
* if (noting\_to\_convert)
* {
* printf("format error: nothing to convert from '%s'.\n", str);
* }
* return read;
* }

main()에서는 int형 인자 4개를 받아서, 첫 번째 인자로 fibonacci()를 수행한 결과와 네 인자로 max\_of\_four\_int()를 수행한 결과를 printf()로 출력한다.

* int main(int argc, char \*argv[])
* {
* if (argc != 5)
* {
* printf("Usage: ./additional <num1> <num2> <num3> <num4>\n");
* return EXIT\_FAILURE;
* }
* int num[4];
* int fibo\_num = 0, max = INT\_MIN;
* */\* Read 4 integers.\*/*
* for (int i = 0; i < 4; i++)
* num[i] = str\_to\_long(argv[i + 1]);
* */\* Compute fibonacci number. \*/*
* fibo\_num = fibonacci(num[0]);
* */\* Get maximum integer. \*/*
* max = max\_of\_four\_int(num[0], num[1], num[2], num[3]);
* */\* Print result. \*/*
* printf("%d %d\n", fibo\_num, max);
* return EXIT\_SUCCESS;
* }
  1. **시험 및 평가 내용**
* **additional.c 실행 결과**  
    
  **명령어** (userprog/build에서 실행)  
  *pintos --filesys-size=2 -p ../../examples/additional -a additional -- -f -q run 'additional 10 20 36 64'*  
    
  **출력**  
  