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

담당 교수 : 김영재 교수님

조 / 조원 : 정은교 / 정은교

개발 기간 : 2023.9.18 ~ 2023.10.7

1. **개발 목표**
   * pintos 환경에서 실행시킬 argument를 입력 받아 command와 paramenter로 parsing하고 user stack을 구현하여 passing 한다.
   * 각각 kernel memory, user memory를 고려하며 systemcall(halt, exit, exec, wait, read(stdin), write(stdout))을 구현한다. 추가적으로 fibonacci, max\_of\_four\_int라는 새로운 system call도 구현한다.
2. **개발 범위 및 내용**
   1. **개발 범위**
3. Argument Passing

: Pintos 환경에서 입력 받은 file\_name를 command와 parameter로 parsing하고, user stack을 구현하여 stack에 쌓는다. 이때 제대로 쌓였는 지 확인하기 위해 hex\_dump() 명령어를 사용한다.

1. User Memory Access

: memory에 접근 할 때, address가 user virtual memory에 존재(kernel 영역에 침범하지 않도록)하는 지 확인하고, 잘못넘겨졌을 때(kernel 영역을 침범했을 때) exit system call로 프로그램을 종료한다.

1. System Calls

: halt, exit, exec, wait, read(stdin), write(stdout), fibonacci, max\_of\_four\_int 를 구현하고 적절히 처리한다.

* 1. **개발 내용**
* Argument Passing

입력 받은 argument를 띄어쓰기 단위를 parsing 한 후, parsed[0], parsed[1], …, parsed[argc-1]까지 넣어준다. 그리고 user stack의 pointer인 \*esp = PHYS\_BASE로 초기화한다. argv[argc-1], … ,argv[0], word-align, NULL, argv의 각각의 주소값, argv의 주소값, argc, return address 순으로 esp 포인터 값을 감소 키며 stack을 쌓는다.

* User Memory Access

Pintos 상에서 invalid memory access는 user mode에서 kernel mode의 메모리 영역에 접근하려는 것을 의미한다.

Invalid memory access는 pintos/src/threads/vaddr.h의 함수 is\_user\_vaddr()를 sysall.c에서 system call 함수를 호출하기 전에 선언하였을 때, false라면 exit(-1)을 하여 비정상 종료를 진행하면 막을 수 있다.

* System Calls

시스템 콜이 필요한 이유는 user mode 영역에서 kernel이 제공하는 API를 사용하려고 할 때 이를 가능하게 해준다. 즉, 각각의 영역에서 각자의 process만 실행할 수 있는데 이때, system call을 사용하면 각각의 mode에서 다른 process에 요청을 할 수 있다.

이번 프로젝트에서 halt, exit, exec, wait, read(stdin), write(stdout)와 추가적으로 fibonacci, max\_of\_four\_int system call을 구현하였다.

halt

: pintos를 종료한다. Shutdown\_power\_off() 함수를 사용하였다.

exit

: user program을 종료한다. Kernel에 현재 상태를 반환 하며, thread\_exit()함수를 사용하였다.

exec

: child thread를 생성한다. process\_execute() 함수를 사용하였다.

wait

: parent process와 child process가 존재 할때 child가 종료 될때 까지 parent를 대기시킨다. process\_wait() 함수를 사용하였다.

read

: stdin에 대해서만 다루는데 명령어를 한 char 씩 읽는다. Input\_getc()함수를 사용하였다.

write

: stdin에 대해서만 다루는데 파일에 값을 입력한다. putbuf() 함수를 사용하였다.

fibo

: n의 값에 따라 피보나치 수를 구한다. For문을 사용하여 구현하였다.

max\_of\_four\_int: a, b, c, d 4개의 정수를 입력 받아 그 중 최댓값을 반환한다.

* + 유저 레벨에서 시스템 콜 API를 호출한 이후 커널을 거쳐 다시 유저 레벨로 돌아올 때까지 각 요소를 설명

User level에서 system call을 호출 했을 때 interrupt를 발생시켜 context switch(user -> kernel) kernel mode에 진입한다. Kernel이 요청 받은 system call을 system call number에 따라 처리하고 결과를 반환한다. 이때, 다시 interrupt를 발생 시켜 context switch(kernel -> user) user mode로 복귀하여 결과를 처리한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

23.9.18~23.9.25 : 프로젝트에 필요한 사전 지식 공부

23.9.26~23.9.30 : parsing, passing, user stack 구현

23.9.31~23.10.3 : Invalid memory access, system call(halt~write) 구현

23.10.4~23.10.5: system call(fibo, max\_of\_four\_int) 구현, 오류 수정

23.10.6~23.10.7: 보고서 작성

* 1. **개발 방법**
     1. Argument parsing

수정 경로: pintos/src/userprog/process.c

load 함수에서 page\_directory 할당 및 활성화 이후 입력 받은 file\_name을 command와 parameter로 띄어쓰기 기준으로 parsing한다.

* + 1. Argument passing

수정 경로: pintos/src/userprog/process.c

load 함수에서 setup\_stack 함수 선언 이후로 stack의 포인터인 \*esp를 초기화 해주고 띄어쓰기 기준으로 parse 되었던 argv들을 역순으로 stack에 넣는다. 순서는 argument와 각 argument의 주소, argc, return address다.

* + 1. System call

수정 경로: pintos/src/userprog/syscall.c

syscall\_handler 함수에서 command인 system call number를 f->esp를 사용하여 syscall\_no 변수에 저장한다. 각 system call 함수(halt, exit, exec, wait, read(stdin), write(stdout))를 구현한다.이후, switch 문으로 SYS\_HALT, SYS\_EXIT, SYS\_EXEC, SYS\_WAIT, SYS\_READ, SYS\_WRITE로 system call에 따라 나눈다. 이때, is\_user\_vaddr 함수로 invalid memory access를 처리하는데 invalid라면 exit(-1), valid라면 각 함수를 실행한다.

수정 경로: pintos/src/threads/thread.h, process.c

wait 함수에서 사용될 semaphore를 thread 구조체에 추가한다.

* + 1. User memory access

수정 경로: pintos/src/userprog/exception.c

page\_fault() 함수에 kernel에 접근했다면 exit(-1)로 비정상 종료, fault 주소가 kernel의 주소일때도 마찬가지로 exit(-1)로 비정상 종료를 한다.

* + 1. 추가 구현

수정 경로: pintos/src/lib/syscall-nr.h, pintos/src/examples

pintos/ src/lib/user/syscall.c, pintos/ src/lib/user/syscall.h

Fibonacci, max\_of\_four\_int 함수는 추가적 system call로

lib/syscall-nr.h에 SYS\_FIBO, SYS\_MAX를 추가한다.

lib/user/syscall.c에 인자를 4개 받는 max\_of\_four\_int 함수를 위해 syscall4 함수를 정의한다.

lib/user/syscall.h에 fibonacci, max\_of\_four\_int 함수를 선언한다.

examples에 additional.c 파일을 추가하여 추가 함수 구현의 출력을 기재한다. + 같은 경로의 Makefile에도 additionall.c를 처리가 가능하도록 수정한다.

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

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

1. User Memory Access

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

1. System Calls

스크린샷, 텍스트, 흑백, 디자인이(가) 표시된 사진

자동 생성된 설명

* 1. **제작 내용**

1. Argument Passing

수정 경로: pintos/src/userprog/process.c

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

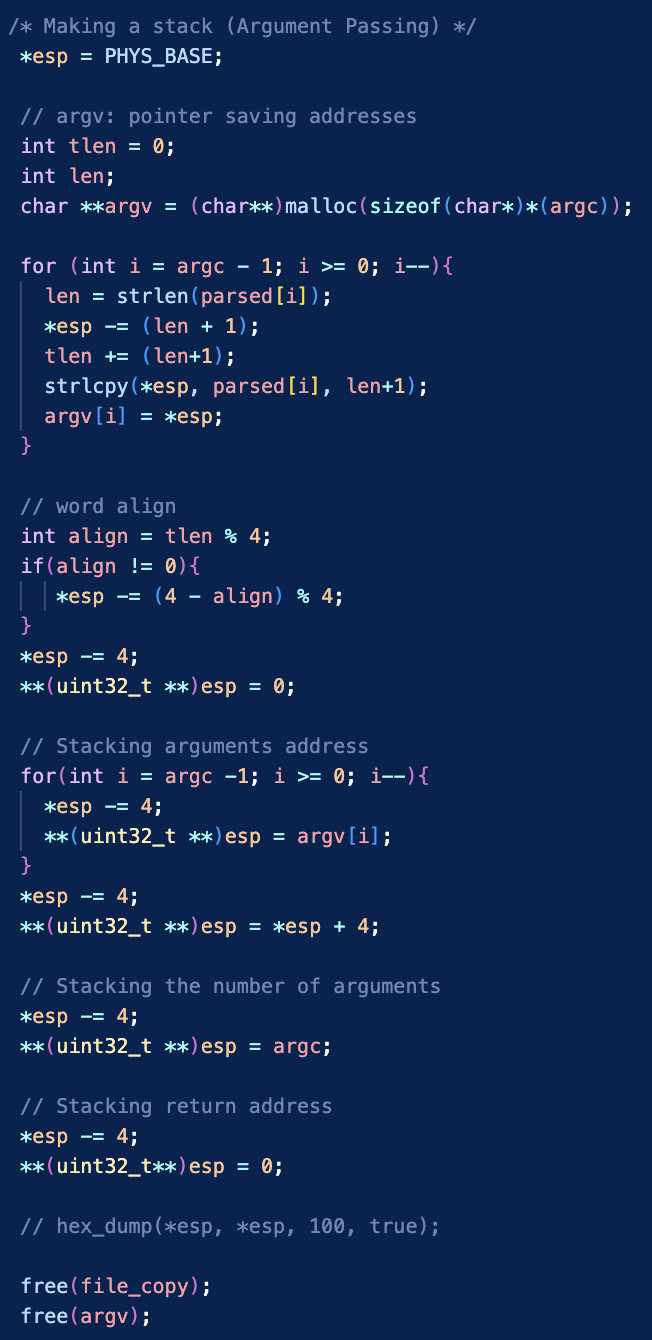
Argument parsing을 위한 변수 선언

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Strlcpy 함수를 사용하여 load 함수의 인자로 받은 file\_name을 file\_copy에 복사한다. file\_name은 command와 parameter를 모두 포함하고 있다. 따라서, 이를 strtok\_r 함수를 이용하여 delim(“ ”)으로 나눈다. temp 가 NULL 값이 될 때까지 즉, 다 나누어질 때까지, 나누어 parsed[100] 배열에 저장한다. 이를 통해 command와 parameter가 저장된 parsed 배열과 argc를 구할 수 있다. 주의할 점은 실행가능한 파일을 열 때,

Filesys\_open(parsed[0])을 해주어야한다. 즉, command가 저장된 parsed[0]을 열어주어야 한다는 것이다.



Stack을 가리키는 포인터인 esp를 PHYS\_BASE로 초기화한다. 그리고 stack 만들기에 필요한 변수들을 선언한다. argv는 위에서 parsing한 인자들의 주소가 들어있는 배열이다. Parsing한 인자들이 들어간 Parsed 배열을 사용하여 stack에 인자들을 쌓는다. 즉, argv[argc-1], … ,argv[0], word-align, NULL, argv의 각각의 주소값, argv의 주소값, argc, return address 순으로 쌓는다. 그리고 malloc으로 메모리 할당을 해주었던 file\_copy, argv를 해제해준다.

1. User Memory Access

수정 경로: pintos/src/userprog/exception.c

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

page\_fault() 함수에 kernel에 접근했다면 exit(-1)로 비정상 종료, fault 주소가 kernel의 주소일때도 마찬가지로 exit(-1)로 비정상 종료를 한다.

수정 경로: pintos/src/userprog/syscall.c

텍스트, 폰트, 스크린샷, 일렉트릭 블루이(가) 표시된 사진

자동 생성된 설명

SYS\_HALT를 제외하고, 나머지 system call number로 들어갈 때 is\_user\_vaddr 함수를 사용하여 user memory 공간인지 확인하여야한다. Kernel 영역을 침범하면 안되기 때문에, kernel 영역을 침범한다면 exit(-1)을 해주면서 비정상적 종료를 진행시킨다.

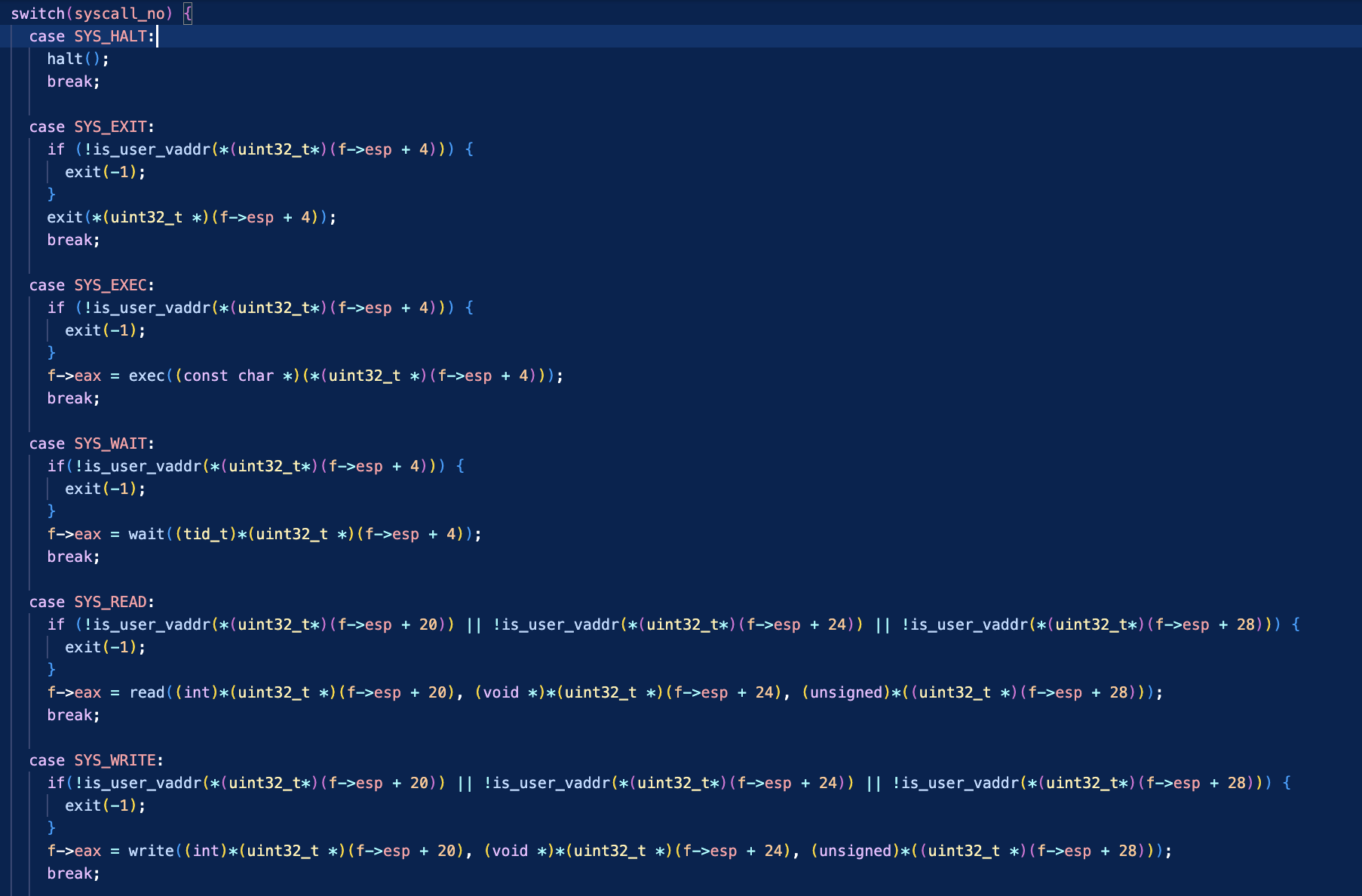
1. System Calls

수정 경로: pintos/src/userprog/syscall.c

텍스트, 폰트, 스크린샷, 일렉트릭 블루이(가) 표시된 사진

자동 생성된 설명

Syscall\_handler 함수에서 system call number를 syscall\_no 변수에 선언한다.



Switch(syscall\_no)문을 선언하여 각 system call number에 따라 처리한다. System call 마다 얼마 만큼의 인자를 받는지 고려하여 주소를 설정한다.

* Halt 함수

텍스트, 폰트, 스크린샷, 일렉트릭 블루이(가) 표시된 사진

자동 생성된 설명

Shutdown\_power\_off() 함수를 사용하여 pintos를 종료한다.

* Exit 함수

텍스트, 스크린샷, 폰트, 일렉트릭 블루이(가) 표시된 사진

자동 생성된 설명

thread\_name과 status를 출력, current thread의 status를 변경해주고 thread\_exit 함수를 호출한다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

exit\_status는 src/thread.h의 thread 구조체에 추가해주었다.

* Exec 함수

텍스트, 폰트, 스크린샷, 일렉트릭 블루이(가) 표시된 사진

자동 생성된 설명

process\_execute 함수를 반환한다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

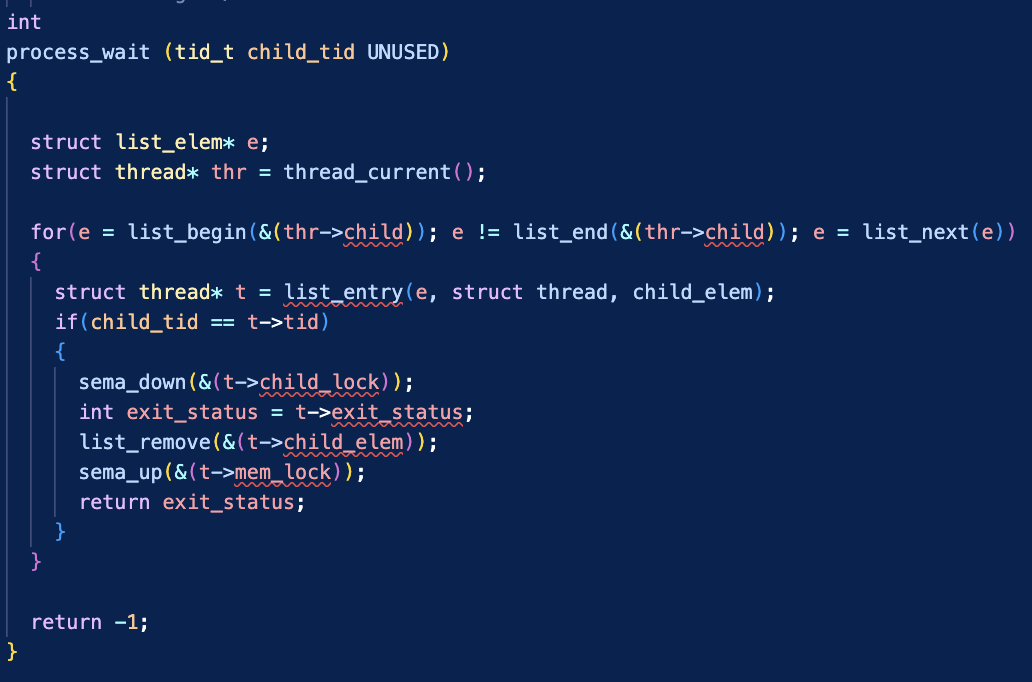
src/userprog/process.c에서 file\_name을 copy할 변수들을 선언해준다 file\_name에 있는 command를 strlcpy, strtok\_r 함수를 사용하여 cmd 변수에 저장하고 filesys\_open(cmd) 값이 NULL이면 return -1로 프로그램을 종료한다. 그리고 thread\_create 함수의 인자로 cmd를 넣어준다. Malloc을 해주었던 file\_copy를 해제해준다.

* Wait 함수

텍스트, 폰트, 스크린샷, 일렉트릭 블루이(가) 표시된 사진

자동 생성된 설명

process\_wait 함수를 반환한다.



userprog/process.c에서 semphore을 사용하여 child thread가 수행을 끝날 때까지 기다린 후 thread의 exit\_status를 반환한다. child thread의 수행이 끝나지 않았을 경우 semphore는 for문에서 수행이 끝나지 않은 자식 process와 부모 process가 남아있게 해준다. 모든 child thread가 종료된경우 -1을 리턴하고 parent thread도 종료시킨다.

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

Process.c 의 process\_exit 함수의 마지막 부분에 process 종료시의 처리를 위한semaphore를 선언해주어야한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Semaphore의 경우 src/threads/thread.h의 thread 구조체에 semaphore mem\_lock, child\_lock 그리고 list child, list\_elem child\_elem도 선언해주었다.

* Read 함수

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Input\_getc() 함수를 사용하여 Stdin을 읽어온다.

* Write 함수

텍스트, 스크린샷, 폰트, 일렉트릭 블루이(가) 표시된 사진

자동 생성된 설명

fd가 1인 경우, putbuf 함수를 사용하여 Stdout에 출력한다.

1. Additional System calls

수정경로: Src/lib

폰트, 스크린샷, 그래픽, 일렉트릭 블루이(가) 표시된 사진

자동 생성된 설명

Src/lib/user/syscall.h에 fibonnaci, max\_of\_four\_int 선언

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Src/lib/user/syscall.c에 fibonnaci, max\_of\_four\_int 선언

폰트, 텍스트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Src/lib/syscall-nr.h에 두 system call number 추가

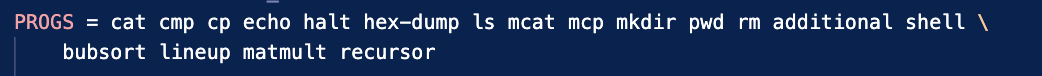
max\_of\_four\_int는 argument로 4개의 인자를 받으므로 syscall4를 선언해주어야한다.

수정경로: Src/examples/additional.c

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Examples directory에 additional.c 함수를 추가하여 Fibonacci, max\_of\_four\_int 두 함수의 처리를 할 수 있도록 한다.



****

같은 경로에 있는 makefile에 additional.c의 처리를 하도록 추가한다.

수정 경로: src/userprog/syscall.h, syscall.c

폰트, 스크린샷, 텍스트, 일렉트릭 블루이(가) 표시된 사진

자동 생성된 설명

두 함수를 선언한다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

fibonacci 함수와 max\_of\_four\_int 함수를 구현한다. Fibonacci는 재귀를 돌지 않고 for문으로 처리해주었다. max\_of\_four\_int의 경우에도 maximum 값을 a로 잡고 다음 인자가 a보다 크면 max 값을 업데이트 크지 않으면 a를 유지 시키는 방법으로 진행하였다.

* 1. **시험 및 평가 내용**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

위의 결과 10번째 피보나치 수로 55를, 입력한 10 20 62 40 중 제일 큰 수인 62를 출력한 결과를 볼 수 있다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

위의 결과 5번째 피보나치 수로 5를, 입력한 5 100 2 31 중 제일 큰 수인 100을 출력한 결과를 볼 수 있다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

위의 결과 1번째 피보나치 수로 1을, 입력한 1 0 30 1중 제일 큰 수인 30을 출력한 결과를 볼 수 있다.