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

담당 교수 : 문의현

학번 / 이름 : 20181536 / 엄석훈

개발 기간 : 2022.10.05 ~ 2022.10.31

1. **개발 목표**

* **해당 프로젝트에서 구현할 내용을 간략히 서술.**
* 사용자가 입력한 명령어를 parsing처리하고 스택에 passing하여 저장하는 기능을 구현한다.
* 유저가 메모리를 접근할 때 잘못된 영역의 주소를 진입하지 못하도록 막아준다.
* 원하는 System call을 호출해주는 system\_handler()를 구현하고 각 system call이 호출 되었을 때의 동작을 구현한다.
* 추가적으로 fibonacci()와 max\_of\_four\_int()을 수행하는 system call을 만들어 준다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Argument Passing

사용자가 입력한 명령어를 띄어쓰기 단위로 parsing하고 3.5 80x86 calling convention에 따라서 스택에 passing하여 저장해준다.

1. User Memory Access

유저가 메모리에 접근할 때 커널 영역을 접근 하는 등 잘못된 접근이 발생하면 쓰레드를 강제로 종료해준다.

1. System Calls

유저가 system call을 요청할 때 system\_hander()가 어떤 system call이 호출되었는지 확인한 후 해당 system call함수를 수행시켜준다.

* 1. **개발 내용**
* **아래 항목의 내용만 서술 (기타 내용은 서술하지 않아도 됨.)**
* Argument Passing
  + 커널 내 스택에 argument를 쌓는 과정 설명
  + 유저의 명령어가 들어오면 strtok\_r 함수를 이용하여 띄어쓰기 단위로 명령어를 parsing해준다. 그리고 parsing된 결과를 argv배열에 저장하고 배열의 길이를 argc에 저장해준다. 그리고 stack에 argv배열에서 거꾸로 스택에 쌓아준다. 그리고 word alignment를 이용해서 메모리의 주소를 정리해주고 4byte의 공간을 0으로 구분해준 뒤, 위에서 쌓은 스택의 주소를 다시 스택에 저장해주고, argc값을 스택에 저장하고 return address값을 저장할 공간을 만들어 주는 과정으로 스택에 argument를 쌓는다.
* User Memory Access
  + Pintos 상에서의 invalid memory access 개념을 간략히 설명
  + Pintos 상에서 user program이 주소를 passing받았을 때 접근하면 안되거나 접근할 수 없는 주소를 넘겨받는 경우가 있을 수 있다. 이러한 경우를 invalid memory access라 한다. 이때 NULL값을 전달받거나, 잘못된 가상 주소를 받거나, 커널 공간의 주소를 전달 받을 수 있다.
  + Invalid memory access를 어떻게 막을 것인지 설명
  + Invalid memory access를 막기 위해서는 pintos에서 제공하는 함수들을 활용하면 된다. 주소를 받았을 때 먼저 NULL값인지 확인해주고, 커널 공간의 주소인지 is\_kernel\_vaddr을 통해 확인하고 혹시 몰라 한번 더 is\_user\_vaddr을 통해 유저공간의 주소인지 확인해준다. 그리고 마지막으로 pagedir\_get\_page 함수를 통해서 주소가 유효한 가상 주소인지 확인해주어서 invalid memory access를 막아 준다.
* System Calls
  + 시스템 콜의 필요성에 대한 간략한 설명
  + 보통 운영체제에서 user program이 에러가 나거나 운영체제에 오류를 발생시키려 할 때 이를 막기 위해서 user mode와 kernel mode로 나눠서 운영체제가 동작하도록 한다. 이때 user program이 kernel mode에서 실행되는 다양한 기능을 사용하기 위해서는 시스템 콜을 통해서 요청을 해주어야 한다. 즉, 시스템 콜은 유저 모드와 커널 모드를 연결해주는 가교 역할을 한다.
  + 이번 프로젝트에서 개발할 시스템 콜에 대한 간략한 설명 (하나의 시스템 콜 당 최대 3문장으로 간략히 설명; 3문장을 넘길 정도로 길게 작성하지 말 것)
  + halt – pintos를 종료한다.
  + exit – 현재의 유저 프로그램을 종료하고 해당 상태를 커널로 상태를 돌려준다.
  + exec – 자식 프로세스를 만들고 해당 자식의 pid를 반환한다.
  + wait – 자식 프로세스가 종료될 때 까지 기다리는 역할을 한다. 자식 쓰레드의 id가 유효한지 확인하고 자식 쓰레드가 종료되면 exit status를 넘겨받는다.
  + read – standard input으로 입력을 받는다. 읽은 데이터의 크기를 반환한다.
  + write – standard output으로 출력해준다. 출력한 데이터의 크기를 반환한다.
  + fibo – 입력받은 정수 n번째 피보나치 수를 반환한다.
  + max4 – 입력받은 4개의 정수 중에서 가장 큰 수를 반환한다.
  + 유저 레벨에서 시스템 콜 API를 호출한 이후 커널을 거쳐 다시 유저 레벨로 돌아올 때까지 각 요소를 설명
  + 유저 레벨에서 시스템 콜 API를 호출하면 먼저 유저 모드에서 커널 모드로 전환을 한다. 그리고 커널 입장에서는 인터럽트가 발생하였음으로 인터럽트 핸들러를 수행하고 이는 시스템 콜 핸들러를 호출한다. 그리고 시스템 콜 핸들러는 유저가 원하는 시스템 콜을 수행해주고 결과를 받은 뒤 다시 커널 모드에서 유저 모드로 전환하고 이어서 유저 프로그램을 수행한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

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

2022.10.05 ~ 2022.10.07 강의 영상 시청 및 Argument passing 기능 구현 및 hex\_dump()를 통한 결과 확인

2022.10.26 ~ 2022.10.28 User memory access 기능 구현 및 system calls handler 및 system call별 기능 함수 구현

2022.10.29 코드 테스트 및 오류 제거

2022.10.30 Additional system call 기능 구현

2022.10.31 보고서 작성

* 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)**
* Argument Passing
* userprog/process.c의 load() 함수에 parsing하는 부분과 스택에 쌓는 기능을 추가해야 한다. load함수의 파라미터인 file\_name을 parsing해서 argv에 저장해주고 argc에는 parsing한 갯수를 적어준다. 그리고 \*temp\_file\_name에 parsing한 첫번째 명령어를 적어서 이 명령어대로 filesys\_open이 되도록 해준다. 그리고 위에서 저장한 argv와 argc를 이용해서 스택에 쌓아주어야 하는데 간단하게 3.5 80x86 calling convention에 따라 차곡차곡 스택에 저장해주면 된다.
* User Memory Access
* userprog/syscall.c에서 check\_address()함수를 새로 만들어서 시스템 콜이 일어날 때 먼저 주소를 check\_address()함수를 통해서 확인한 뒤 시스템 콜이 일어나도록 해주었다. 주소값이 들어오면 위에서 말한대로 null인지 확인하고 is\_kernel\_vaddr, is\_user\_vaddr, pagedir\_get\_page 함수를 이용해서 잘못된 메모리 주소에 접근 할 수 없도록 막아주었다.
* System Calls
* userprog/syscall.c에서 syscall\_handler()에서 먼저 호출된 시스템 콜의 숫자에 따라서 switch문을 통해 원하는 시스템 콜이 호출될 수 있도록 해주었다. 각 시스템 콜의 숫자는 syscall\_nr.h에서 확인할 수 있다. 각 시스템 콜을 호출할 때는 위에서 말한대로 user memory access 가능 여부를 확인해주고 시스템 콜을 호출하였다. 각 시스템 콜 함수의 동작은 II. B.에 서술한 대로 구현해주었다.

1. **연구 결과**
   1. **Flow Chart**

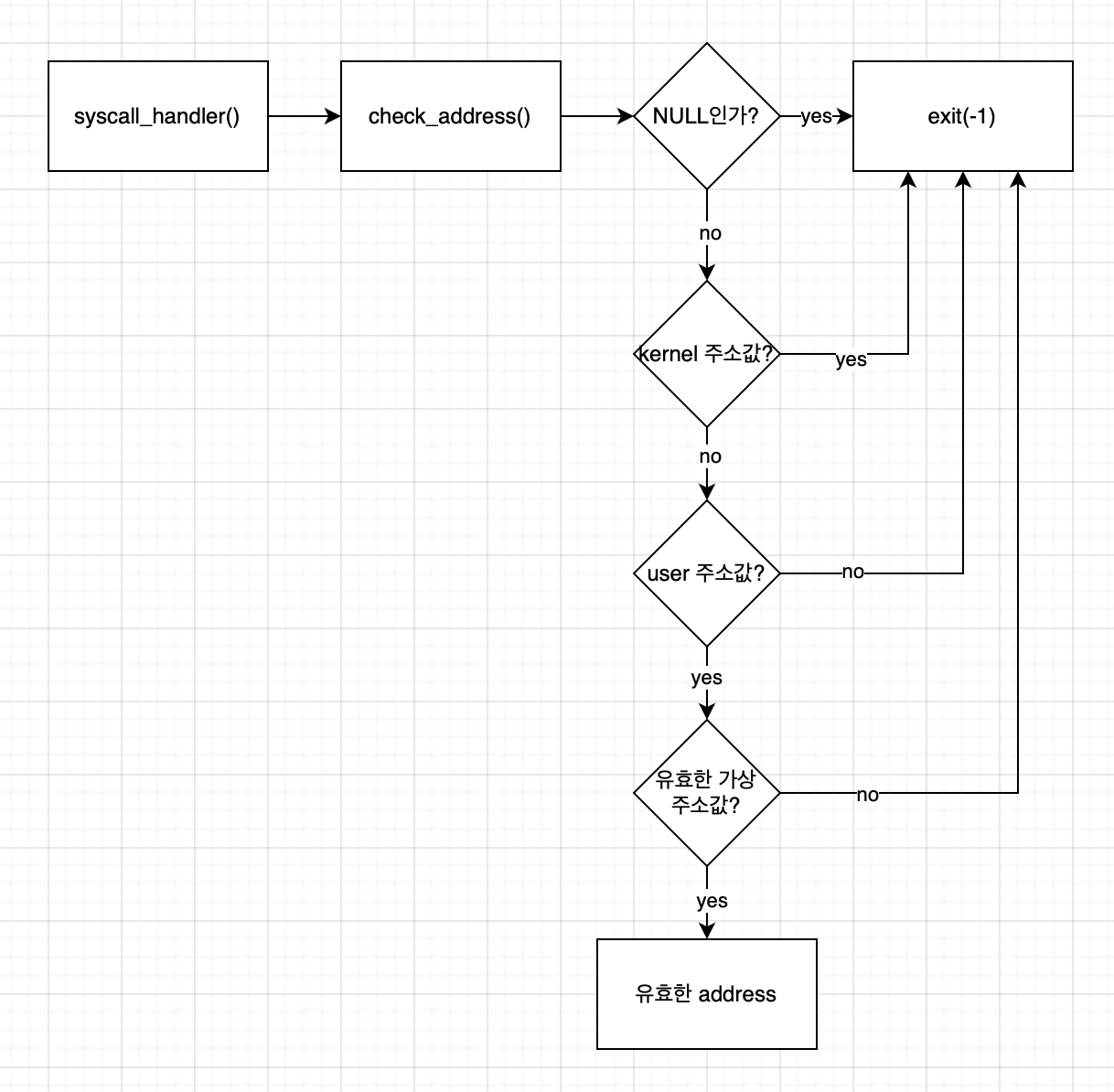
* **II. B. 개발 내용에 대한 Flow Chart를 작성**

1. Argument Passing

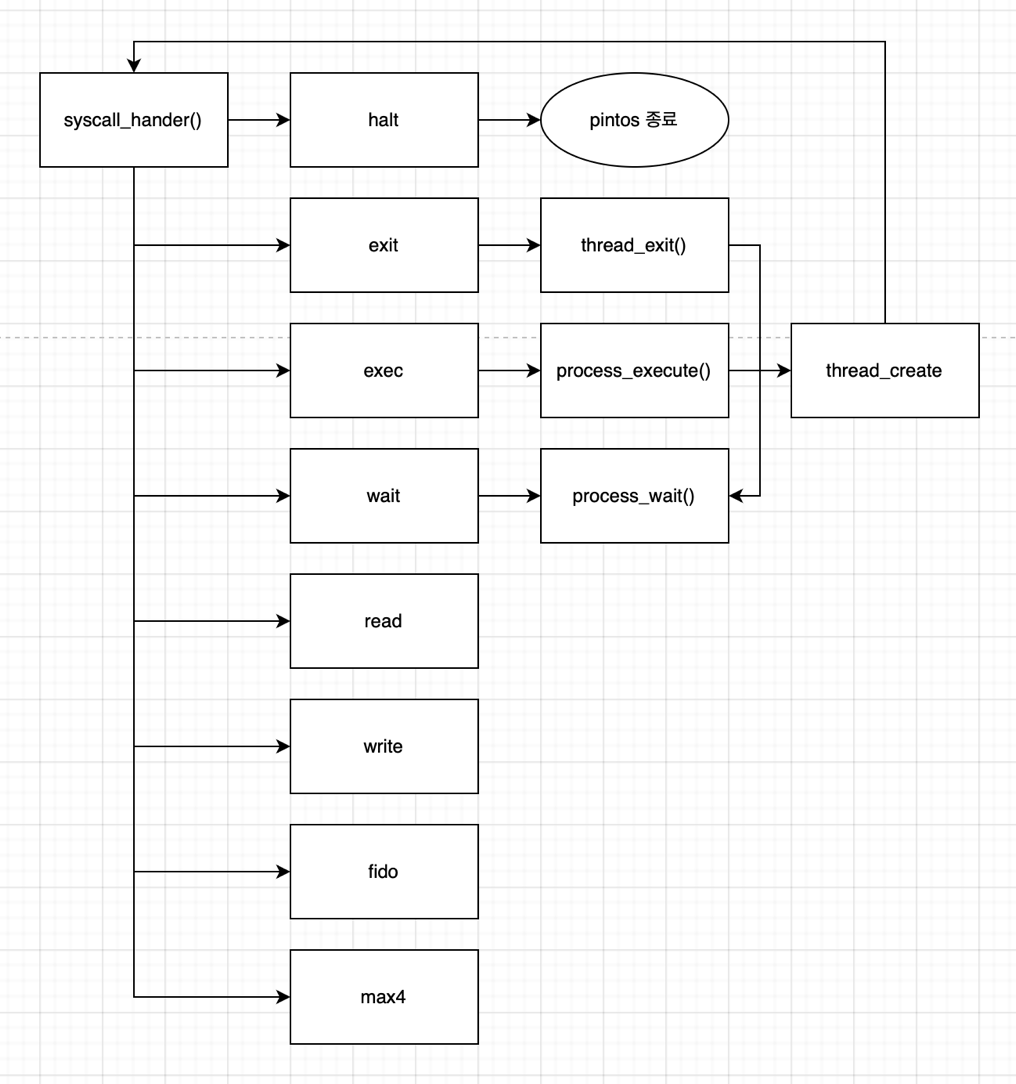
텍스트, 실내, 기기, 주방기기이(가) 표시된 사진

자동 생성된 설명

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

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 화면이(가) 표시된 사진

자동 생성된 설명

가장 위의 사진은 load()함수에서 file\_name을 parse하는 부분의 코드이다. 먼저 file\_name에 저장된 값이 바뀌지 않도록 strlcpy()를 통해 str에 file\_name의 데이터를 복사하고 strtok\_r()을 이용해서 temp\_file\_name에 file\_name의 첫번째 명령어를 parse해서 저장해준다. 그리고 strtok\_r()을 이용해서 계속해서 명령어를 parse한 뒤 argv배열에 parse한 결과를 저장하면서 나눈 명령어의 수를 저장하는 argc의 값을 하나씩 늘려준다.

그리고 다음 두 사진의 코드는 parse한 명령어를 스택에 저장해주는 코드이다. Argc-1의 값에서 부터 하나씩 인덱스를 줄여가며 argv에 들어있는 데이터를 스택에 저장해준다. 이때 스택의 주소를 temp\_argv에 저장하고 total\_len에 스택에 저장한 데이터의 값을 저장해둔다. 그리고 total\_len을 이용해서 남은 데이터만큼 word-alignment를 해주고 4byte의 0을 넣어주어 경계선을 만들어 준다. 그 다음 temp\_argv에 저장해두었던 스택의 주소를 스택에 다시 쌓아주고 argc값을 저장하고 return address를 스택에 저장하면 끝이 난다.

1. User Memory Access

텍스트이(가) 표시된 사진

자동 생성된 설명

syscall.c에 check\_address()함수를 직접 만들어서 시스템 콜이 일어날 때 주소를 확인하였다. 간단하게 if문 4개를 통과시켜주었다. 확인하고 싶은 주소값을 파라미터 addr로 받아서 addr == NULL이면 exit(-1)을 해주고 해당 주소가 커널 영역인지 확인하는 함수인 is\_kernel\_vaddr(addr)을 이용해서 커널 영역이라면 똑같이 exit(-1)을 하였다. 그리고 해당 주소가 유저 영역인지 확인하는 is\_user\_vaddr(addr)을 통해서 user영역이면 true값이 나올 테니 함수를 실행해서 0값이 반환되면 exit(-1)을 해주었고 해당 주소에 상응하는 물리적 주소가 실제로 있는지 확인하는 pagedir\_get\_page합수를 이용하여 NULL값이 나오면 exit(-1)을 해주었다.

1. System Calls

* **이번 프로젝트에서 개발한 시스템 콜을 구현 관점에서 상세히 서술.**

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

가장 먼저 src/userprog/syscall.c에서 시스템 콜이 호출되었을 때 호출된 시스템 콜을 호출할 수 있도록 syscall\_handler()함수를 만들어 주었다. 시스템 콜이 호출되면 syscall\_number에 시스템 콜 번호를 받은 뒤 switch(syscall\_number)를 통해서 각 시스템 콜이 호출되도록 조건문들 만들어 주었다. 각 시스템 콜에 대해서 check\_address()를 통해서 접근하고자 하는 주소를 확인하고 halt(), eixt(), exec(), wait(), read(), wirte()함수고 호출되도록 구성하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

먼저 halt()의 경우는 간단하게 shutdown\_power\_off()함수를 호출하여 pintos가 종료되도록 해주었다. 다음으로 exit()함수는 thread가 종료되었다고 pintos에서 출력되도록 printf()함수를 먼저 호출한다. 그리고 thread의 exit\_status를 exit()을 호출하면서 넘긴 파라미터 값으로 할당해주고 thread\_exit()을 호출하고 다시 process\_exit()을 호출하여 프로세스가 종료되도록 해주었다. 다음으로 exec()함수는 process\_execute()함수를 호출하여 자식 프로세스가 생성되도록 하였다. 다음으로 wait()함수는 process\_wait()함수를 호출하여 자식 프로세스가 종료될 때 까지 기다리도록 구성하였다. 위에서 이야기 한 process\_exit()와 process\_execute(), process\_wait()는 뒤에서 더 자세히 설명할 예정이다. 그리고 read()는 STDIN으로 fd가 0인 경우에만 input\_getc()함수를 이용하여 사이즈 만큼 입력을 받도록 하였다. 그리고 읽어드린 size만큼 리턴 해주었다.. 그리고 write()함수는 STDOUT으로 fd가 1인 경우에만 putbuf를 이용해서 buffer의 내용을 size만큼 출력하고 출력한 size를 리턴 해주었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

process\_execute()에서 추가한 코드이다. 자식 프로세스가 생성된 뒤 이의 상태를 세마포어에 저장하기 위해서 find\_thread\_with\_tid라는 새로운 함수를 이용해서 자식 프로세스의 tid를 찾은 뒤 해당 자식 프로세스를 현재의 쓰레드의 자식으로 저장하고 자식이 호출되었다는 의미에서 load\_sema를 하나 줄여주었다.

텍스트이(가) 표시된 사진

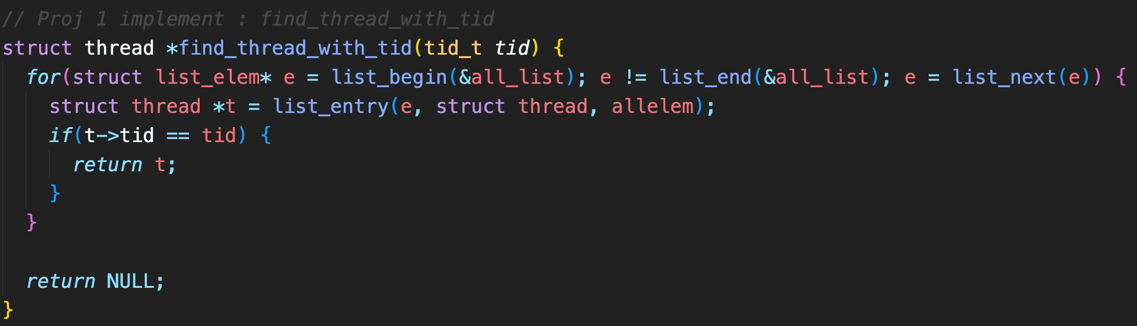
자동 생성된 설명

process\_exit()함수에서도 코드를 추가하였다. 현재 스레드의 wait\_sema 세마포어를 늘려주고 모든 자식 프로세스의 wiat\_sema 세마포어는 감소시키고 exit\_sema 세마포어는 증가시켜서 모든 자식 프로세스를 종료시키고 자신의 exit\_sema 세마포어를 줄여서 현재의 프로세스 까지 종료하도록 해주었다.

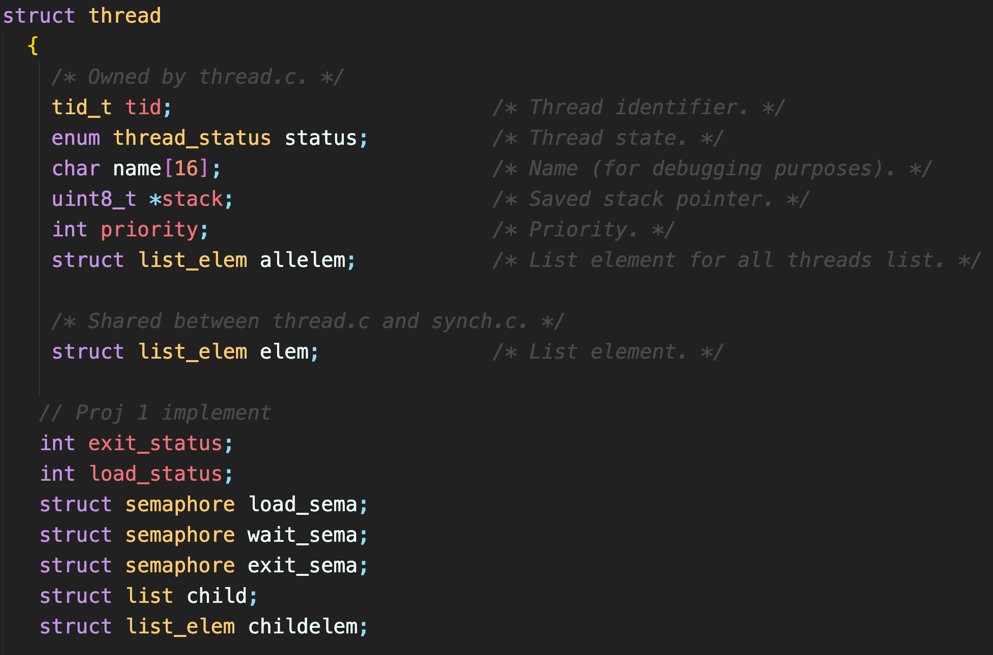
텍스트이(가) 표시된 사진

자동 생성된 설명

process\_wait()함수도 코드를 짜주었다. 먼저 모든 프로세스에 대해서 for문을 돌면서파라미터로 받은 tid와 같은 자식 프로세스를 찾아준다. 그리고 해당 자식 프로세스의 wait\_sema 세마포어가 감소하고 exit\_sema 세마포어를 증가시키고 자식이 가지고 있던 exit\_status를 받아서 return해준다.



새로 짜준 함수로 find\_thread\_with\_tid()함수이다. 원하는 tid값을 가진 자식 쓰레드를 찾아주기 위해서 짠 함수로 tid값이 들어오면 모든 쓰레드가 포함된 list를 돌면서 주어진 tid와 해당 프로세스의 tid와 값이 같으면 그 쓰레드의 주소값을 리턴해준다.



그리고 thread의 구조체에 여러가지 변수들을 추가하였다. exit와 load의 상태를 저장하는 exit\_status와 load\_status가 추가되었고 load, wait, exit할 때 사용할 세마포어 3가지를 선언하였다. 그리고 해당 쓰레드의 자식 쓰레드를 저장하는 child와 childelem을 추가로 저장할 수 있게 해주었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그리고 마지막으로 init\_thread()함수에 위에서 선언한 변수들을 초기화 할 수 있도록 코드를 짜주었다.

1. Additional System calls

* **새로운 시스템 콜(fibonacci, max\_of\_four\_int)을 구현하기 위해 수정하거나 작성한 코드에 대해 서술**

텍스트이(가) 표시된 사진

자동 생성된 설명

새로운 시스템 콜을 구현하기 위해서 src/lib/syscall-nr.h 파일에 다른 시스템 콜과 같이 새로운 시스템 콜의 번호를 만들어주었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그리고 src/lib/user/syscall.c에 4개의 파라미터 대한 시스템 콜이 수행 가능하도록 syscall4를 구현해주었다. 위에 3개의 파라미터에 대한 시스템 콜 수행이 가능하도록 해둔 syscall3을 참고하여 구현하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그리고 방금 만든 syscall4와 원래 있던 syscall1을 이용해서 fibonacci함수와 max\_of\_four\_int함수에 대한 시스템 콜 호출이 가능하도록 코드를 구현하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그리고 src/userprog/syscall에 새로운 시스템 콜의 번호가 호출되었을 때 해당 시스템 콜 함수가 수행되도록 syscall\_handler 함수를 수정해주었다. 먼저 주소가 접근 가능한지 확인하고 시스템 콜이 수행되도록 fibo와 max4함수를 호출하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

그리고 각 fibo함수와 max4함수가 호출되었을 때 n번째 피보나치 수를 구하는 함수와 4개의 파라미터에서 최대값을 구하는 함수를 만들어주었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

마지막으로 이러한 시스템 콜의 수행을 pintos상에서 돌려볼 수 있도록 src/examples/additional.c 파일을 만들어서 해당 기능을 수행하고 결과를 출력하도록 예제 프로그램을 만들어 주었다.

* 1. **시험 및 평가 내용**
* **fibonacci 및 max\_of\_four\_int 시스템 콜 수행 결과를 캡처하여 첨부.**

**텍스트이(가) 표시된 사진

자동 생성된 설명**