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

담당 교수 : 문의현

학번 / 이름 : 20180492 / 김범준

개발 기간 : 2022. 10. 05 ~ 2022. 11. 02

1. **개발 목표**

PintOS 프로젝트 1에서는 PintOS의 기본적인 사용자 프로그램을 개발하는 것을 목표로 한다. 구체적으로, 어떠한 명령어를 입력받을 때 해당 명령어를 각각의 단어로 나누고(argument parsing), 나눈 단어들을 유저 스택에 push(argument passing)한다. 다음으로 불법적인 메모리 접근이 일어나지 않도록 차단하는 기능을 구현한다. 대표적으로 사용자가 커널 메모리에 접근하려고 할 때, 해당 접근을 막아주는 기능이 필요하다. 마지막으로, 유저 스택으로 passing된 argument를 바탕으로 간단한 시스템 콜을 구현한다.

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

먼저, argument를 passing하기 전 parsing의 과정이 필요하다. parsing을 통해 명령어를 하나하나의 명령 단위로 구분한 뒤 유저 스택에 하나하나 저장해야 하기 때문이다. parsing된 명령어를 유저 스택에 저장하고, 프로세스 및 쓰레드가 해당 명령어들을 불러올 수 있도록 스택 포인터를 관리하게 된다.

1. User Memory Access

사용자가 커널 메모리에 접근하는 것을 막는다. 사용자가 커널 메모리에 접근하는지 주기적으로 확인하여 커널 메모리로의 불법적 접근이 확인될 경우 프로세스를 강제종료(kill)한다.

1. System Calls

halt, exit, execute, wait, read(stdin), write(stdout)의 시스템 콜을 구현한다. 시스템 콜 핸들러를 통하여 스택 포인터에 접근하고, 스택 포인터를 이용하여 명령어를 커널로 넘겨주어 해당 명령어가 실행되도록 한다.

* 1. **개발 내용**
* **아래 항목의 내용만 서술 (기타 내용은 서술하지 않아도 됨.)**
* Argument Passing

char형 2차원 배열과 strtok\_r 함수를 사용하여 argument를 parsing한다. 다음으로, parsing된 단어들을 스택에 push하고, 스택 포인터는 항상 스택의 top을 가리키도록 조정한다. word alignment를 하여 parsing된 명령어에 더욱 빠르게 접근할 수 있도록 한다. 다음으로 스택에 명령어가 저장된 순으로(배열의 역순) 저장되어 있는 주소값을 스택에 push한다. 마지막으로 parsing되어 생성된 단어들의 개수와 스택의 반환값을 push한다.

* User Memory Access

PintOS에서 invalid memory access는 사용자 모드가 커널 모드의 메모리를 침범하는 경우, 혹은 반대의 경우를 말한다. PintOS는 사용자 메모리에 3GB의 공간을, 커널 메모리에 1GB의 공간을 할당한다. 어떠한 프로세스가 물리 메모리 공간에 직접적으로 할당될 경우 서로의 공간을 침범할 가능성이 높아진다. 이에 대한 대책으로 PintOS는 가상 메모리를 이용한다. 각각의 프로세스가 자신만의 메모리를 가진다는 환상을 주는 것이다. 프로세스가 할당받은 가상 메모리는 페이징 기법을 통하여 물리 메모리 공간과 연결되어 있다. 하나하나의 프로세스들이 연속적으로 메모리 공간을 차지할지 않아도 되기 때문에, 최대한 많은 프로세스들이 다른 모드의 메모리 공간을 침범하지 않는 선에서 동시에 메모리 할당을 받을 수 있게 된다. 하지만 해당 과정에서도 서로의 영역을 침범하는 불법적인 메모리 접근이 발생할 수 있다. 따라서 is\_user\_vaddr 함수를 사용하여 주기적으로 불법적인 메모리 접근이 발생하는지 확인해주어야 한다.

* System Calls

시스템 콜은 사용자 모드와 커널 모드를 이어주는 중요한 역할을 한다. 사용자 모드에서는 사용자 프로세스만, 커널 모드에서는 커널 프로세스만 실행이 될 수 있는데, 시스템 콜을 통하여 서로의 프로세스에 대한 요청이 가능해진다. PintOS 프로젝트 1에서 개발하는 시스템 콜은 다음과 같다.

halt

: PintOS를 종료시킨다.

exit

: 프로세스를 종료할 때 출력하는 메시지(thread\_name: exit(num))을 출력하고 프로세스를 종료시킨다.

exec

: 프로세스를 실행한다. 이 때, 존재하지 않는 명령일 경우 에러(TID\_ERROR)를 반환한다.

wait

: 부모 프로세스가 자식 프로세스가 완료되었다는 메시지를 받기 전까지 대기하게 한다.

read

: 파일 디스크립터가 0일 때(stdin) 명령어를 한 글자씩 읽어들인다.

write

: 파일 디스크립터가 1일 때(stdout) 출력한다.

Fibonacci

: 피보나치 수열의 입력받은 수의 위치에 해당하는 수를 출력한다.

max\_of\_four\_int

: 입력받은 네 개의 수 중 가장 큰 수를 출력한다.

유저 모드 API에서 시스템 콜을 호출하면 인터럽트 핸들러가 인터럽트를 발생시켜 context switch가 일어나 커널 모드로 진입한다. 커널 모드에서는 사용자 모드에서의 시스템 콜 호출에서 전달받은 시스템 콜 넘버를 참조하여 시스템 콜 테이블에서 알맞은 명령어를 확인한다. 다음으로 해당 명령어를 실행시키고 다시 인터럽트를 발생시켜 다시 context switch가 일어나 사용자 모드로 전환된다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

2022. 10. 05 ~ 2022. 10. 19 : argument passing, system call의 개념 복습, 동기화 개념 학습

2022. 10. 20 ~ 2022. 10. 25 : 중간시험 기간

2022. 10. 26 ~ 2022. 10. 27 : argument parsing 구현

2022. 10. 28 ~ 2022. 10. 29 : argument passing 구현

2022. 10. 30 ~ 2022. 11. 01 : system call 및 user memory access 구현

2022. 11. 02 : additional system call 구현 및 보고서 작성

* 1. **개발 방법**

process.c

-load

: 스택을 setup하기 전 argument parsing의 기능을 수행하는 코드를 추가하고, 스택이 setup되면 argument를 스택으로 passing한다.

-process\_execute

: 페이지 할당 코드 이후에 함수의 인자로 받은 파일의 이름을 복사본에 옮기고, 명령어를 추출하여 파일을 여는 코드를 추가한다. 해당 파일이 NULL이면 에러를 반환하고, NULL이 아니면 추출된 명령어에 해당하는 쓰레드를 생성한다.

-process\_wait

: 자식 쓰레드가 종료되기를 기다리는 코드를 추가한다. 세마포어를 관리하는 코드 또한 추가한다.

-process\_exit

: 프로세스가 종료되었을 때 세마포어를 관리하는 코드를 추가한다.

thread.h

: 쓰레드 구조체에 동기화와 자식 쓰레드를 관리하기 위한 멤버를 추가한다.

thread.c

-init\_thread

: 생성된 쓰레드의 자식 리스트와 세마포어를 초기화한다.

exception.c

-page\_fault

: 사용자가 불법적인 메모리 접근을 하려고 할 때 차단하는 코드를 추가한다.

userprog/syscall.c

-sysexit

: 쓰레드를 종료시키는 함수를 작성한다. 함수의 마지막에 thread\_exit 함수를 호출한다. 프로토타입은 syscall.h에 추가한다.

-syscall\_handler

: halt, exit, execute, wait, read, write, additional 각각의 시스템 콜에 맞는 코드를 추가한다. 각 시스템 콜마다 사용자가 커널 메모리 영역에 침범하려고 하는지 검사하는 코드를 추가한다.

syscall-nr.h

: additional에 대한 시스템 콜 번호를 추가한다.

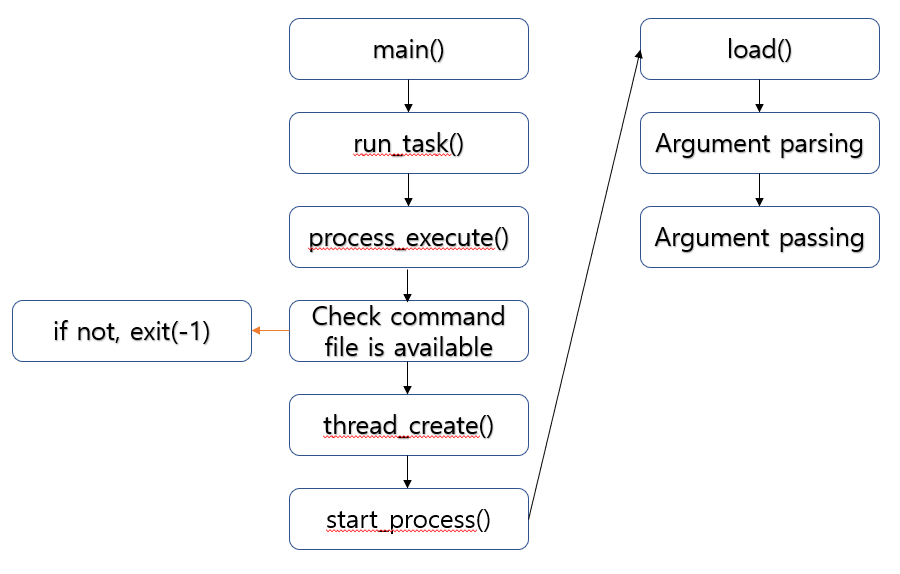
/lib/user/syscall.c

: 인자가 네 개 들어오는 경우의 어셈블리 코드를 추가한다. additional에 대한 함수도 추가한다. 프로토타입은 syscall.h에 정의한다.

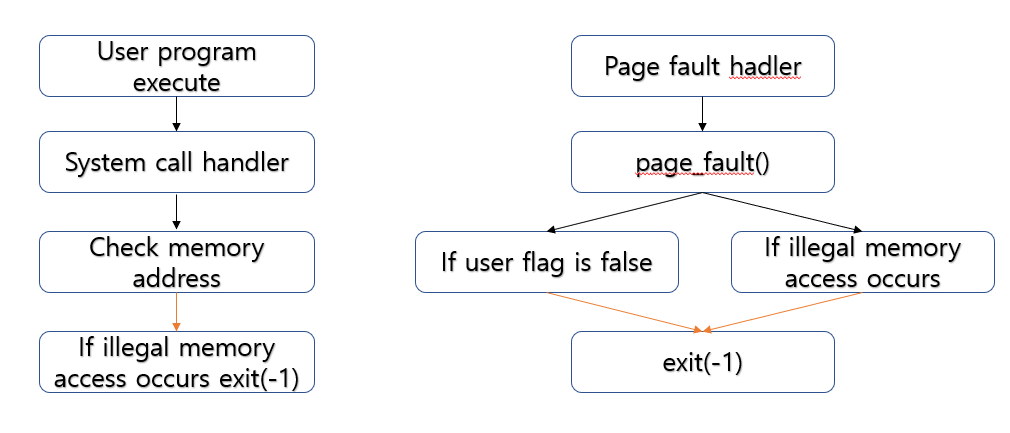
1. **연구 결과**
   1. **Flow Chart**

* **II. B. 개발 내용에 대한 Flow Chart를 작성**

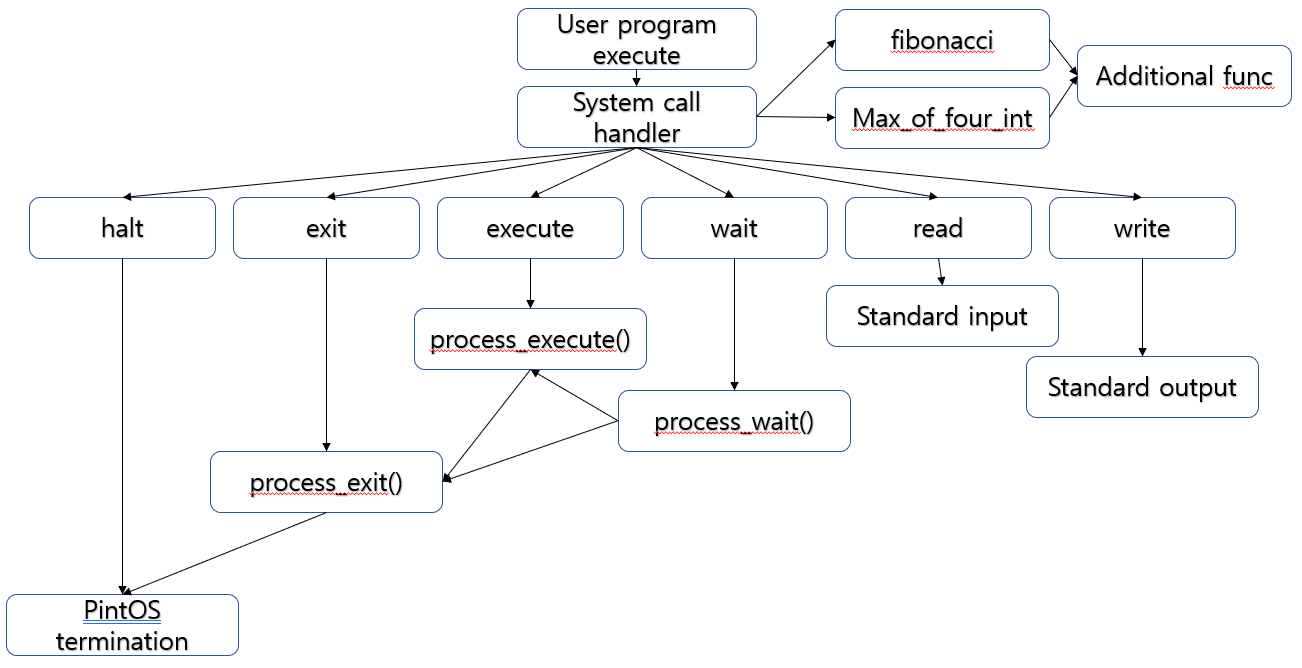
1. Argument Passing



1. User Memory Access



1. System Calls



* 1. **제작 내용**

1. Argument Passing

process.c

-load

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

: argument parsing을 위한 변수 및 배열을 정의한다. 인자의 개수를 나타내는 int argc, parsing한 명령어를 저장하는 배열 char \*argv[256]을 정의한다. 다음으로 strtok\_r 함수를 사용하여 입력받은 명령어를 공백을 기준으로 나누어 argv에 저장한다. parsing이 끝나면 stack이 setup되고, 스택 포인터 exp가 빈 스택의 top을 가리킨다. parsing된 명령어의 스택 주소값을 저장하는 배열 char \*argv\_mem[256], word alignment를 위한 변수 word\_align을 선언한다. parsing된 명령어가 저장된 argv배열을 top\_down 방식으로 스택에 push하고, 각각의 주소값, 인자 개수 또한 스택에 push한다. 처음에 메모리 크기를 고려하지 않고 무조건적으로 \*\*(uint8\_t\*\*)esp에 메모리를 할당하였으나 후에 수정했다.

1. User Memory Access

page\_fault

텍스트이(가) 표시된 사진

자동 생성된 설명

: 사용자가 커널 메모리에 접근하려고 하는지 검사하고 불법적 접근이 일어날 경우 sysexit(-1)을 호출한다.

syscall.c의 switch문



: 각각의 모든 시스템 콜마다 커널 메모리에 접근하려고 하는지 검사하고 불법적 접근이 일어날 경우 sysexit(-1)을 호출한다.

User Memory Access를 구현할 때 vaddr.h에 정의된 is\_user\_vaddr 함수를 사용하여 커널 메모리로의 접근이 일어나는지 검사하였다.

1. System Calls

syscall.c

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

: 시스템 콜 핸들러는 다음과 같이 구현하였다. switch문을 이용하여 각 시스템 콜 숫자에 맞는 코드가 실행되도록 구성하였다. 시스템 콜 구현과 동기화를 위하여 쓰레드 구조체에 다음과 같은 멤버를 추가하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

자식 프로세스에 점근할 수 있는 리스트, 종료 시의 상태를 출력하는 exit\_status, signal과 wait 세마포어를 추가하였다. 세마포어의 사용은 threads 디렉토리의 synch.h와 synch.c를 참조하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

다음은 thread.c의 init\_thread함수 마지막 부분에 추가한 코드이다. 생성된 쓰레드 구조체의 리스트를 초기화하고, 세마포어를 초기화한다. 처음에 세마포어를 초기화하지 않아서 페이지 폴트가 계속하여 발생하였으나 디버깅 후 수정하였다.

각각의 시스템 콜에 대한 세부 설명은 다음과 같다.

-halt

텍스트이(가) 표시된 사진

자동 생성된 설명

shutdown\_power\_off 함수를 호출하여 PintOS를 종료시킨다.

-exit

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

현재 쓰레드의 이름과 exit\_status를 출력한 뒤 쓰레드를 종료시킨다. 현재 쓰레드의 exit\_status에 s의 값을 할당하는 코드를 빠뜨렸다가 디버깅 후 수정하였다.

-execute

텍스트, 실내이(가) 표시된 사진

자동 생성된 설명

커널로 접근하기 위해 f->eax에 스택에 저장된 명령어를 인자로 받는process\_execute 함수의 반환값을 할당하였다. process.c의 process\_execute에는 다음과 같은 코드를 추가하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

char \*cp[100]을 선언하고 함수의 인자로 받은 파일 이름의 메모리를 cp로 옮긴다. strtok\_r 함수를 이용하여 명령어를 분리시키고, 해당 명령어에 해당하는 파일을 연다. 만약 해당 파일이 NULL이면 TID\_ERROR를 반환한다. NULL이 아닐 경우, tid = thread\_create(file\_name, PRI\_DEFAULT, start\_process, fn\_copy)의 첫 인자를 분리한 명령어로 교체한다. 처음 작성시에 cp를 만들지 않고 바로 cmdline에 strtok\_r(file\_name, “ “, &tmp)와 같이 구현하였다가 exec-arg 케이스를 통과하지 못하여 디버깅 후 수정하였다.

-wait

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

f->eax에 process\_wait 함수를 호출하여 반환값을 할당한다. process\_wait 함수에서는 while문을 이용하여 자식이 모두 종료될 때 까지를 기다린다. 만약 자식 프로세스가 종료되면 -1을 반환하여 부모 프로세스를 종료하고, 그렇지 않다면 세마포어를 이용하여 계속 메모리에 해당 프로세스가 남아있게 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

다음은 세마포어 관리를 위해 process.c의 process\_exit 함수 마지막에 추가한 코드이다. 초기에 해당 코드의 순서가 뒤바뀌어 무한루프에 빠지는 상황이 발생했으나 디버깅 후 수정하였다.

-read

텍스트이(가) 표시된 사진

자동 생성된 설명

유저 스택의 세 부분에 접근해야 하므로 메모리 접근 검사를 3번 진행하였다. 파일 디스크립터가 0이 아닐 경우 f->eax에 -1의 값을 할당하였고, 그렇지 않은 명우에 문자열이 끝나거나 모든 문자를 다 읽을 때 까지 input\_getc 함수를 사용하여 한 글자씩 읽으면서 세고, 센 숫자를 f->eax에 할당하였다.

-write

텍스트이(가) 표시된 사진

자동 생성된 설명

유저 스택의 세 부분에 접근해야 하므로 메모리 접근 검사를 3번 진행하였다. 파일 디스크립터가 1일 경우 putbuf 함수를 사용하여 출력한 뒤 f->eax에 출력한 크기를 할당하였다. 파일 디스크립터가 1이 아닐 경우 f->eax에 -1을 할당하였다.

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

시스템 콜에서 fibonacci와 max\_of\_four\_int를 다음과 같이 구현하였다. fibonacci는 다이나믹 프로그래밍 기법을 활용하였고, max\_of\_four\_int는 각 수를 모두 비교하였다. max\_of\_four\_int의 경우 유저 스택의 네 부분에 접근해야 하므로 메모리 접근 검사를 4번 하였다.

시스템 콜 넘버를 syscall-nr.h에 다음과 같이 구현하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

커널에서의 시스템 콜은 다음과 같이 구현하였다. 어셈블리 프로그래밍 경험이 부족하여 시간을 오래 소모하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

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

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**텍스트이(가) 표시된 사진

자동 생성된 설명**