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

담당 교수 : 김영재 교수님

조 / 조원 : 20211188 방수윤

개발 기간 : 9/20-10/2

1. **개발 목표**

* **해당 프로젝트에서 구현할 내용을 간략히 서술.**

핀토스에서 기존에 제공된 코드에 **argument passing**, **user memory access**, **system call**을 구현하여, 사용자 입력과 출력(I/O) 환경에서 커널과의 통신이 가능하도록 했다. 추가적으로 Fibonacci()와 max\_of\_four\_int() 함수를 구현했다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Argument Passing

프로그램 실행 시 전달된 인자들이 커널 스택에 적절히 저장되고, 프로그램이 시작될 때 이를 올바르게 참조할 수 있도록 하였다. 이를 통해 사용자 프로그램은 실행 시 필요한 인자들을 문제없이 전달받고 처리할 수 있게 되었다.

1. User Memory Access

사용자 프로그램이 접근하려는 메모리 주소가 올바른 사용자 공간에 있는지 확인하는 기능을 추가하였다. 잘못된 메모리 접근 시 커널 패닉이 발생하는 것을 방지하고, 비정상적인 접근이 감지되면 프로그램을 안전하게 종료할 수 있도록 하였다.

1. System Calls

사용자 프로그램이 커널의 기능을 요청할 수 있는 인터페이스를 구현하였다. 파일 시스템 조작, 입출력, 프로세스 관리 등의 시스템 호출이 안정적으로 수행되며, 각 시스템 호출에서 예외 처리를 통해 오류 발생 시 적절한 동작이 이루어지도록 하였다. 이를 통해 사용자 프로그램과 커널 간의 통신이 가능해졌다.

* 1. **개발 내용**
* **아래 항목의 내용만 서술 (기타 내용은 서술하지 않아도 됨.)**
* Argument Passing
  + 커널 내 스택에 argument를 쌓는 과정 설명

프로그램 실행 시 전달된 인자를 strtok\_r로 파싱한 후, 각 인자를 스택에 역순으로 저장한다. 저장된 인자들의 주소를 스택에 다시 저장하여 argv[] 배열을 구성하고, 그 후 argc 값을 스택에 저장한다. 마지막으로 argv의 끝을 NULL로 처리하여 인자 전달을 완료한다.

* User Memory Access
  + Pintos 상에서의 invalid memory access 개념을 간략히 설명

사용자 프로그램이 커널 또는 허용되지 않은 메모리 영역에 접근하려는 시도를 invalid memory access라 한다. 이는 메모리 보호 기법에 위배되며, 커널 패닉이나 비정상적인 동작을 유발할 수 있다.

* + Invalid memory access를 어떻게 막을 것인지 설명

is\_user\_vaddr와 is\_kernel\_vaddr 함수를 사용해 사용자 프로그램이 접근하는 메모리 주소가 올바른지 확인한다. 만약 유효하지 않은 주소에 접근할 경우, 프로그램을 즉시 종료하여 메모리 보호를 구현한다.

* System Calls
  + 시스템 콜의 필요성에 대한 간략한 설명

시스템 콜은 사용자 프로그램이 커널 기능에 안전하게 접근하기 위한 인터페이스로, 파일 시스템 조작, 입출력, 프로세스 제어와 같은 작업을 가능하게 한다. 이를 통해 사용자 프로그램은 직접적으로 하드웨어에 접근하지 않고도 필요한 작업을 수행할 수 있다.

* + 이번 프로젝트에서 개발할 시스템 콜에 대한 간략한 설명 (하나의 시스템 콜 당 최대 3문장으로 간략히 설명; 3문장을 넘길 정도로 길게 작성하지 말 것)
* **EXIT**: 현재 실행 중인 프로세스를 종료하며, 종료 상태를 커널에 전달한다.
* **WRITE**: 지정된 파일 디스크립터에 데이터를 기록하며, 파일이 아닌 경우 표준 출력으로 데이터를 보낸다.
* **READ**: 지정된 파일 디스크립터에서 데이터를 읽어와 버퍼에 저장하며, 파일이 아닌 경우 표준 입력에서 데이터를 받아온다.
* **Halt**: 시스템을 종료하고, 기계를 안전하게 종료한다. 커널에 직접 종료 명령을 내려 시스템의 전원을 끄는 기능을 한다.
* **Exec**: 사용자 프로그램을 실행하며, 프로그램의 파일 이름을 인자로 받아 해당 프로그램을 새로운 프로세스로 실행한다.
* **Wait**: 지정된 프로세스의 종료를 기다리고, 종료 시 그 프로세스의 종료 상태를 반환한다. 프로세스가 종료될 때까지 대기하는 기능을 제공한다.
* **FIBONACCI**: 주어진 숫자의 피보나치 수를 계산하여 반환한다.
* **MAX\_OF\_FOUR\_INT**: 네 개의 정수 중 가장 큰 값을 찾아 반환한다.
  + 유저 레벨에서 시스템 콜 API를 호출한 이후 커널을 거쳐 다시 유저 레벨로 돌아올 때까지 각 요소를 설명

사용자 프로그램이 시스템 콜 API를 호출하면, 시스템 콜 번호와 인자들이 스택에 저장되고, 소프트웨어 인터럽트를 발생시켜 커널에 진입하게 된다. 커널에 진입한 이후, 시스템 콜 핸들러는 인터럽트를 처리하며 스택에서 시스템 콜 번호와 인자들을 추출한다. 그런 다음, 커널은 해당 시스템 콜 번호에 맞는 적절한 시스템 콜 함수로 작업을 수행하게 된다. 작업이 완료되면, 결과값을 eax 레지스터에 저장하고 유저 레벨로 복귀하여 프로그램이 정상적으로 계속 실행될 수 있도록 한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* **II. A.의 개발 범위를 포함하여 구현 내용에 대한 일정 작성**
  + 9/28: 코드 분석, 보고서 2번 작성
  + 9/29-9/30: argument passing 구현
  + 10/1: system calls 구현
  + 10/2-10/4: system calls 구현
  + 10/5-10/6: 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)**
  + 1. **Argument passing**

**[process.c]**

첫 번째 공백 위치를 찾아 cmd\_name을 추출한다. load 함수에서 strtok\_r 함수를 이용하여 나머지 부분인 argument들을 분리하고, argc를 세어준다. 이후 make\_stack 함수에서 80x86 convention에 맞게 stack에 쌓아준다. esp에 memcpy를 통해 인자들을 복사하고, 추가적으로 4 word alignment를 맞춰준다. 차례로 argument들의 주소, argc, 그리고 return address를 stack에 쌓아준다.

* + 1. **User memory access**

**[exception.c]**

먼저, is\_kernel\_vaddr(fault\_addr)와 user 조건을 통해 커널 메모리에 대한 접근이거나 사용자 모드가 아닌 경우, 이를 잘못된 접근으로 간주하고 EXIT(-1)로 프로세스를 종료한다.

이후, 페이지가 존재하지 않으면서(not\_present), 접근한 주소가 유효한 사용자 모드의 스택 범위 내에 있을 경우(is\_stack\_access), 스택을 확장한다. 스택 확장은 expand\_stack 함수에서 이루어지며, 새로운 페이지를 할당하고 이를 페이지 테이블에 추가하여 스택을 늘린다.

is\_stack\_access 함수는 스택 포인터에서 32바이트 이내로 접근하는 경우를 유효한 스택 접근으로 간주하여, 유효한 범위를 확인한다. 마지막으로, 스택 확장 기능은 palloc\_get\_page를 사용하여 사용자 페이지를 할당하고, 성공적으로 페이지 테이블에 추가하지 못하면 할당된 메모리를 해제한다.

**[process.c]**

이 코드에서는 filesys\_open 함수가 파일을 열지 못한 경우, 즉 반환값이 NULL일 때 printf로 에러 메시지를 출력하고, goto done;을 통해 예외 처리 한다.

**[syscall.c]**

Is\_user\_vaddr을 사용하여 user address가 범위내에 있는지 확인했다.

* + 1. **System calls**

syscall.c에서 syscall\_handler 함수의 switch문을 사용하여 각 시스템 호출 번호에 맞는 작업을 수행하도록 정의했다. 시스템 호출을 처리하기 전에, check\_addr 함수를 통해 사용자 주소가 유효한지 확인하여 메모리 접근 오류를 방지했다. 각 시스템 호출의 결과값은 eax 레지스터에 저장했다. 새로 구현한 FIBONACCI와 MAX\_OF\_FOUR\_INT 시스템 호출에 해당하는 시스템 호출 번호는 syscall-nr.h에 정의했다.

syscall.c에서는 syscall4 함수를 정의하여 MAX\_OF\_FOUR\_INT 시스템 호출을 사용할 수 있도록 했고, syscall3 함수를 통해 FIBONACCI 시스템 호출을 처리했다. additional.c에서는 두 시스템 호출을 테스트하기 위한 코드를 작성했고, Makefile을 수정하여 컴파일 시 additional.c도 함께 빌드되도록 했다.

process.c에서는 thread.h에 추가된 세마포어인 mem\_lock과 child\_lock을 사용하여 프로세스 간 동기화를 구현했고, 부모 프로세스가 자식 프로세스의 종료를 기다릴 수 있도록 했다.

1. **연구 결과**
   1. **Flow Chart**

* **II. B. 개발 내용에 대한 Flow Chart를 작성**

1. Argument Passing

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. User Memory Access

텍스트, 도표, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

1. System Calls

텍스트, 도표, 스케치, 평면도이(가) 표시된 사진

자동 생성된 설명

* 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

[process.c]

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

process\_execute() 함수에서 명령어의 첫 번째 공백을 기준으로 파일 이름 (cmd\_file)을 분리하였다. 처음 구현 시에는 파일 이름을 추출한 후 문자열의 끝에 '\0'을 추가하지 않아, 이후 문자 처리에서 명령어를 제대로 인식하지 못하는 문제가 있었다. 이 문제는 cmd\_file[len] = '\0';를 추가하여 문자열이 올바르게 끝나도록 함으로써 해결되었다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

file\_name을 copy\_file\_name에 복사한 후, 공백을 기준으로 명령어를 파싱하여 각 인자를 argv 배열에 저장하고, 인자 개수를 argc로 세어준다. 마지막으로 argv[0]에 저장된 첫 번째 인자를 실행할 파일 이름으로 사용한다.

텍스트, 스크린샷, 소프트웨어, 운영 체제이(가) 표시된 사진

자동 생성된 설명

make\_stack 함수는 명령어 인자들을 스택에 올바르게 배치하기 위한 역할을 한다. 이 함수는 argv[] 배열에 저장된 인자들을 역순으로 스택에 복사하고, 각각의 인자의 주소를 기록한 후 해당 주소를 스택에 다시 저장하는 방식으로 동작한다.

먼저, 각 인자의 길이를 구한 후 스택 포인터를 감소시켜 해당 위치에 인자를 복사한다. 이때, 인자들의 주소도 별도의 배열에 저장한다. 모든 인자를 복사한 후, 스택 포인터가 4바이트의 정렬을 맞추도록 패딩을 추가한다. 그 다음, 저장해 둔 인자의 주소들을 스택에 차례대로 복사하여 \*\*argv[]\*\*의 포인터 배열을 구성한다. 이어서, argv 배열의 주소를 스택에 저장하고, 인자의 개수(argc)를 기록한 후 마지막으로 0을 가리키는 NULL 포인터를 삽입하여 스택 설정을 마무리한다.

이 과정을 통해 프로세스가 실행될 때 스택에 인자들이 올바르게 배치되어, 프로그램이 인자를 읽어들일 수 있게 된다.

1. User Memory Access

[process.c]

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

load() 함수에서 filesys\_open()이 파일 열기에 실패하여 NULL을 반환하면, 에러 메시지를 출력하고 goto done;으로 예외 처리를 진행한다. 이로 인해 파일 로드를 중단하고, 파일을 닫거나 필요한 후처리를 수행한 후 실패를 반환하게 된다.

[exception.c]

스크린샷, 텍스트, 폰트이(가) 표시된 사진

자동 생성된 설명

이 코드는 페이지 폴트가 발생했을 때 메모리 접근의 유효성을 검사하고 처리하는 부분이다. 먼저, fault\_addr가 커널 메모리 주소이거나 user 모드가 아닌 경우 EXIT(-1)을 호출하여 프로세스를 종료한다. 이후, 페이지가 존재하지 않으면서(not\_present) fault\_addr가 유효한 사용자 주소(is\_user\_vaddr)이고 스택 확장 조건(is\_stack\_access)을 만족하는 경우, expand\_stack()을 호출하여 스택을 확장한다.

1. System Calls

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

[syscall.c]

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명

스택에서 시스템 호출 번호를 가져와 syscall\_num 변수에 저장한다. 만약 시스템 호출 번호가 음수라면, 오류 메시지를 출력하고 thread\_exit()을 호출해 프로세스를 종료한다. 이후 switch문을 통해 시스템 호출 번호에 따라 각 기능을 처리한다.

SYS\_HALT는 HALT()를 호출하여 시스템을 종료한다. SYS\_EXIT는 스택에서 인자를 가져와 EXIT()를 호출해 프로세스를 종료한다. SYS\_EXEC는 프로그램 이름을 인자로 받아 EXEC()를 호출해 프로그램을 실행하며, 그 결과를 f->eax에 저장한다. SYS\_WAIT는 특정 프로세스가 종료될 때까지 대기하기 위해 WAIT()를 호출하고, 그 결과를 f->eax에 저장한다.

SYS\_READ는 파일 디스크립터, 버퍼, 크기를 인자로 받아 READ()를 호출해 데이터를 읽고, 읽은 바이트 수를 f->eax에 저장한다. SYS\_WRITE는 파일 디스크립터, 버퍼, 크기를 인자로 받아 WRITE()를 호출해 데이터를 기록하고, 기록한 바이트 수를 f->eax에 저장한다.

SYS\_FIBONACCI는 피보나치 수를 계산하는 FIBONACCI()를 호출해 그 결과를 f->eax에 저장한다. SYS\_MAX\_OF\_FOUR\_INT는 네 개의 정수 중 최대값을 계산하는 MAX\_OF\_FOUR\_INT()를 호출해 그 결과를 f->eax에 저장한다. 마지막으로, 알 수 없는 시스템 호출 번호가 들어오면 오류 메시지를 출력하고 thread\_exit()을 호출해 프로세스를 종료한다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

시스템 종료를 위해 shutdown\_power\_off() 함수를 호출하여 시스템을 종료한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

현재 스레드의 종료 상태를 출력하고, 해당 스레드의 종료 상태를 저장한 후 thread\_exit()을 호출하여 프로세스를 종료한다.

텍스트, 스크린샷, 폰트, 디스플레이이(가) 표시된 사진

자동 생성된 설명

주어진 프로그램을 실행하기 위해 process\_execute()를 호출하고, 실행에 실패하거나 파일을 찾지 못하면 -1을 반환한다.

텍스트, 폰트, 스크린샷, 친필이(가) 표시된 사진

자동 생성된 설명

특정 프로세스가 종료될 때까지 대기하기 위해 process\_wait()를 호출하여 해당 프로세스의 종료를 기다린다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

파일 디스크립터가 표준 입력(0)인 경우, 입력을 받아 버퍼에 저장하고 읽은 바이트 수를 반환한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

파일 디스크립터가 표준 출력(1)인 경우, 버퍼의 데이터를 출력하고 출력한 바이트 수를 반환한다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

이 함수는 부모 스레드가 자식 스레드의 종료를 기다리는 역할을 한다. 먼저, 자식 스레드 리스트에서 child\_tid에 해당하는 자식을 찾고, 자식이 종료될 때까지 sema\_down()으로 대기한다. 자식이 종료되면 종료 상태를 반환하고, 자원의 정리 작업을 수행한 후 종료 상태를 반환한다.

[threads.h]

mem\_lock은 자식 프로세스의 자원을 안전하게 해제하기 위해 부모와 자식 간의 자원 접근을 동기화한다. child\_lock은 부모가 자식 프로세스의 종료를 기다리도록 동기화하여 종료 여부를 안전하게 처리한다.

[threads.c]

이 코드는 자식 스레드를 위한 초기화 작업을 수행한다. child\_lock과 mem\_lock을 각각 0으로 초기화하여 부모-자식 간 동기화를 준비하고, 자식 스레드 리스트를 초기화한 후 현재 스레드의 자식 리스트에 새로 생성된 자식 스레드를 추가한다.

1. Additional System calls

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

[syscall-nr.h]

추가함수를 실행시킬 수 있도록 systeamm call number를 추가로 작성하였다.

[lib/user/syscall.c]

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

이 코드는 syscall4 매크로로, 시스템 호출을 위한 어셈블리 코드를 정의하고 있다. 인자 4개를 스택에 푸시하고, 시스템 호출 번호를 설정한 후 int 0x30으로 시스템 호출을 수행한다. 호출 후 반환값은 retval에 저장되며, 마지막에 반환된다.

[syscall.c]

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

이 함수는 주어진 n에 대해 피보나치 수열의 n번째 값을 계산한다. 입력이 0이면 0을 반환하고, 1이나 2일 경우 1을 반환하며, 그 외의 양수인 경우 반복문을 통해 n번째 피보나치 값을 계산한다. 음수 입력 시에는 오류 메시지와 함께 -1을 반환한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

이 함수는 네 개의 정수 중 최대값을 찾는다. 먼저, 첫 번째 두 수(a, b) 중 최대값을 계산하고, 두 번째 두 수(c, d) 중 최대값을 계산한 후, 두 최대값을 비교하여 최종 최대값을 반환한다.

[additional.c]

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

이 프로그램은 명령행 인자로 4개의 숫자를 받아 첫 번째 숫자의 피보나치 수와 네 숫자 중 최대값을 계산하여 출력한다. argc가 5가 아니면 오류 메시지를 출력하고 종료한다. 이후, FIBONACCI()와 MAX\_OF\_FOUR\_INT() 함수를 호출하여 결과를 출력한다.

[examples/Makefile]

Additional.c가 examples에서 make되도록 추가해주었다.

* 1. **시험 및 평가 내용**
* **fibonacci 및 max\_of\_four\_int 시스템 콜 수행 결과를 캡처하여 첨부.**
* 스크린샷, 패턴, 텍스트, 스티치이(가) 표시된 사진

  자동 생성된 설명
* **텍스트, 스크린샷이(가) 표시된 사진

  자동 생성된 설명**
* **Project1의 evaluation항목을 모두 만족시키는 모습을 볼 수 있다.**
* **텍스트, 전자제품, 스크린샷, 소프트웨어이(가) 표시된 사진

  자동 생성된 설명추가로 구현한 함수들도 잘 작동하고 있다.**