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

담당 교수 : 김영재

이름 : 조현지

개발 기간 : 2024.09.20 ~ 2024.10.03

1. **개발 목표**

* **해당 프로젝트에서 구현할 내용을 간략히 서술.**

: 사용자 프로그램이 시스템 리소스를 사용할 수 있게 halt(), exit(), exec(), wait(), read(), write() 총 6가지의 systemcall 사용자 인터페이스를 제공하여, 커널에 접근할 수 있게 한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Argument Passing

Process에 저장된 파일 이름을 빈칸을 기준으로 파싱하여 개별 인자로 분리한 뒤 스택에 인자를 저장한다. esp 포인터를 이용해 파싱된 내용은 이후 프로그램이 사용자가 입력한 내용을 처리하는 데에 사용된다.

1. User Memory Access

사용자 프로그램이 매핑되지 않은 가상 공간을 가리키는지, 커널 영역을 가리키는지 확인한다. 만약 둘 중 하나에 해당될 경우 유저 가리키는 포인터가 유효하지 않은 포인터임을 파악한 뒤 참조하고 있는 페이지를 가리키지 못하게 만든다. 이를 통해, 커널이 사용자 프로세스 메모리에 잘못 접근하지 않도록 한다.

1. System Calls

사용자 프로그램이 시스템 콜 인터페이스를 통해 전달한 인수를 시스템 콜 번호에 맞춰 처리한다. 이때 유효한 메모리 접근인지 검증이 끝나야 사용자의 요청을 처리한다.

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

Filename을 빈칸을 기준으로 나누기 위해 strtok\_r()함수를 이용해 빈칸을 기준으로 문자열로 들어온 명령어를 인자 단위로 나눈다. 이후, esp 포인터를 이용해 나눠준 인자들을 역순으로 넣는다. 인자들의 전체 크기를 계산한 뒤 word(4byte)단위로 나눠지는지 확인하다. 크기가 맞지 않다면, esp를 이용해 word align을 진행한다. Null pointer sentinel를 추가적으로 스택에 쌓아준다. 이후 스택에 저장된 인자의 주소값을 스택에 저장된 순서대로 넣는다. 사용자가 입력한 인자의 수를 추가해준 뒤, esp가 return 주소를 가리키게 한다.

* User Memory Access
  + Pintos 상에서의 invalid memory access 개념을 간략히 설명
  + Invalid memory access를 어떻게 막을 것인지 설명

? pintos는 virtual memory를 user virtual memory와 kernel virtual memory로 구분한다. 이때 physical\_base를 바탕으로 위쪽은 kernel 영역이고 아래쪽은 user 영역이다. 그리고 kernel 영역은 global memory이다. 사용자가 요청하는 메모리 접근이 유효한 접근인지 확인하기 위해, pagedir\_get\_page()를 통해 해당 페이지를 위해 매핑된 가상 메모리가 맞는지 확인한다. 이후, 접근하려는 주소가 커널 영역이 아니라 유저 영역인지 확인하기 위해 is\_user\_vaddr(), is\_kernel\_vaddr()를 이용한다. 이때 유효하지 않은 접근이라면 사용자 프로그램의 포인터가 현재 가리키는 영역을 참조하지 못하게 만든다.

* System Calls
  + 시스템 콜의 필요성에 대한 간략한 설명
  + 이번 프로젝트에서 개발할 시스템 콜에 대한 간략한 설명 (하나의 시스템 콜 당 최대 3문장으로 간략히 설명; 3문장을 넘길 정도로 길게 작성하지 말 것)
  + 유저 레벨에서 시스템 콜 API를 호출한 이후 커널을 거쳐 다시 유저 레벨로 돌아올 때까지 각 요소를 설명

사용자 프로그램이 직접적으로 시스템 리소스를 사용할 수 있게 되면 리소스 할당 문제로 시스템끼리 충돌이 날 수 있다. 따라서, 시스템 콜을 통해서만 커널의 시스템 리소스에 접근할 수 있게 한다.

1. Halt

Pintos의 시스템 전체를 종료시키는 시스템 콜이다. 호출될 경우, 시스템 상의 모든 프로세스가 종료되며 시스템 리소스인 CPU 또한 종료된다. 이때, 유저 모드로 복귀할 필요는 없다.

1. Exit

현재 실행 중인 프로세스를 종료시키는 시스템 콜이다. 호출될 경우, 넘겨받은 exit status과 실행 중인 프로세스의 이름을 출력한 뒤 process\_exit()를 호출해 프로세스를 종료시킨다. 이때, 종료 상태를 부모 프로세스에 넘긴다.

1. Exec

새로운 프로그램을 실행하기 위한 시스템 콜이다. 인자로 넘겨받은 파일 이름을 process\_execute() 함수에 넘겨 현재 실행 중인 프로세스의 자식 프로세스로 실행된다. 이후, intr\_frame의 eax에 반환값을 저장한 뒤 유저로 복귀한다.

1. Wait

자식 프로세스가 종료될 때까지 기다리기 위한 시스템 콜이다. 넘겨받은 자식 프로세스의 pid의 유효성을 확인한 뒤, process를 block시킨다. 이때, 자식 프로세스가 실행 중 죽었을 경우 thread\_yield() 함수를 이용한다.

1. Read

사용자가 입력한 내용을 읽어들이기 위한 시스템 콜이다. Input\_getc()를 통해 입력된 문자를 읽고 넘겨받은 buffer에 저장하고 int fd가 0인지 확인한다. 넘겨받은 size만큼 반복한 뒤, intr\_frame의 eax에 저장한다.

1. Write
2. Buffer에 담긴 내용을 출력하기 위한 시스템 콜이다. putbuf()를 통해 buffer에 남긴 데이터를 출력한다. 데이터의 크기를 intr\_frame에 저장한 뒤, 다시 eax 필드에 저장한다.
3. **추진 일정 및 개발 방법**
   1. **추진 일정**

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

|  |  |
| --- | --- |
| 2024.09.20 ~ 2024.09.25 | Argument Parsing |
| 2024.09.26 ~ 2024.09.29 | User Memory Access |
| 2024.09.30 ~ 2024.10.03 | System calls |

* 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 | 변경할 파일  : process.c  수정 내용  : load 함수의 filename 변수에 들어있는 문자열을 빈칸을 기준으로 나눈다. Setup\_stack 함수가 실행된 후, 이전에 parsing해둔 인자들을 esp 포인터를 이용해 stack에 쌓아간다. |
| User Memory Access | 변경할 파일  : vaddr.c  수정 내용  : esp가 가리키는 주소가 유효한 메모리 주소인지 확인하기 위해 pagedir\_get\_page()를 먼저 이용한다. 이때, 주소에 대한 페이지가 실제로 매핑되어있는지 확인한다. 이후, 커널 영역이 아닌 사용자 영역에 접근하는지 확인하기 위해 is\_virtual\_addr(), is\_kernel\_addr() 함수를 이용해 검증한다. |
| System calls | 변경할 파일  : syscall.h, syscall.c  수정 내용  : systcall\_handler()의 struct intr\_frame \*f를 이용해 6가지 시스템 콜(halt(), exit(), exec(), wait(), read(), write())이 작동되게 하기 위해 switch문을 이용한다. 반환값이 없는 halt()와 exit()을 제외한 나머지 시스템 콜의 경우 반환값을 f의 eax에 저장한다. |

1. **연구 결과**
   1. **Flow Chart**

* **II. B. 개발 내용에 대한 Flow Chart를 작성**

1. Argument Passing

|  |  |
| --- | --- |
| 1 | 사용자가 입력한 전체 명령어를 strtok\_r()를 통해 빈칸 기준으로 분리 |
| 2 | 커널 스택에 더블 리스트를 만든 뒤, 분리한 인자를 가리키게 한다. |
| 3 | esp 포인터가 가리키는 위치에 인자를 역순으로 저장한다. |
| 4 | Word align을 맞추기 위해 인자들의 전체 길이를 계산한 뒤, 4의 배수가 되게 align 해준다. |
| 5 | esp 포인터가 가리키는 위치에 인자가 저장된 위치를 스택에 쌓인 순서대로 저장한다. |
| 6 | esp 포인터가 인자의 개수를 가리키게 한다. |
| 7 | esp 포인터가 return address를 가리키게 한다. |

1. User Memory Access

|  |  |
| --- | --- |
| 1 | esp 포인터가 가리키는 주소에 매핑된 가상 메모리가 있는지 pagedir\_get\_page() 함수를 통해 확인한다. |
| 2 | 매핑된 가상 메모리가 없을 경우, 포인터가 해당 주소를 참조하지 못하게 한다. |
| 3 | esp 포인터가 가상 메모리의 커널 영역이 아닌 사용자 영역을 가리키는지 확인하기 위해 is\_virtual\_addr(), is\_kernel\_addr() 함수를 사용해 확인한다. |
| 4 | 커널 영역일 경우, 포인터가 해당 주소를 참조하지 못하게 한다. |

1. System Calls

|  |  |
| --- | --- |
| 1 | 시스템 콜 번호를 시스템 콜에 맞춰 syscall.h에 정의하기 |
| 2 | 시스템 콜 함수를 syscall.h에 정의하기 |
| 3 | Syscall\_handler에 switch문을 넣어 intr\_frame \*f가 가리키는 시스템 콜 번호에 따라 각기 다른 시스템 콜이 수행되게 한다. |
| 4 | 반환값이 있는 시스템의 콜의 경우 반환값을 포인터 f의 eax에 저장한다. |
| 5 | System call handler가 비정상 종료될 경우 exit status를 -1로 세팅한다. |

* 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

|  |  |
| --- | --- |
| 인자 개수 세기 | Filename을 strlcpy()로 복사한다.  Filename의 전체 길이를 구한다.  복사한 filename을 strtok\_r()함수를 통해 빈칸을 기준으로 나누면서 인자의 전체 개수 파악한다.  인자의 전체 개수를 바탕으로 더블 리스트를 동적으로 생성한다. |
| 인자 파싱 | 다시 한번 strtok\_r() 함수를 통해 빈칸을 기준으로 인자를 나눈 뒤 이를 더블 리스트에 넣는다. |
| 스택에 인자 저장 | 더블 리스트에 넣어둔 인자를 역순으로 스택에 넣는다.  스택에 넣고 나서, esp가 가리키는 주소를 인자 대신 넣는다.  사전에 구한 filename의 전체 길이를 바탕으로 4의 배수로 esp가 가리키는 주소가 나눠떨어질 수 있게 word align을 진행한다.  Null pointer sentinel 값을 추가한다.  더블 리스트에 넣어둔 인자가 저장된 주소를 역순으로 스택에 넣는다.  사전에 구한 인자의 전체 개수를 스택에 넣는다.  esp가 return address를 가리키게 한다. |

|  |  |
| --- | --- |
| 문제점 | strcpy()와 strtok()함수를 사용하려고 했지만, 빌드에서 스레드 안전성을 이유로 경고가 떴다. |
| 해결책 | 해당 함수들이 static 변수를 사용하여 발생한 문제라, 스레드 독립적으로 함수가 처리되는 strlcpy()와 strtok\_r() 함수를 사용해주었다. |

1. User Memory Access

|  |  |
| --- | --- |
| 매핑 확인 | esp 포인터가 가리키는 주소에 매핑된 메모리가 있는지 확인한다. 이때, pagedir\_get\_page() 함수를 이용한다.  만약 매핑된 메모리가 없다면, 포인터가 해당 주소를 참조하지 못하게 한다. |
| 사용자 영역인지 확인 | esp 포인터가 가리키는 주소가 PHYSICAL\_BASE를 기준으로 사용자 영역에 있는지 확인한다. 이때, is\_virtual\_addr()과 is\_kernel\_addr() 함수를 이용한다.  만약 커널 영역을 가리키고 있다면, 포인터가 해당 주소를 참조하지 못하게 한다. |

1. System Calls

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

|  |  |
| --- | --- |
| 시스템 콜 정의 | Syscall.h에 시스템 콜 번호와 시스템 콜을 정의한다. 시스템 콜 번호와 시스템 콜의 이름은 sycall-nr.h에 저장된 내용을 참고하였다. |
| 시스템 콜 핸들러 | Intr\_frame \*f가 가리키는 시스템 콜을 처리하기 위해 switch문을 만들고, 시스템 콜 번호에 따라 각기 다른 시스템 콜이 호출된다.  이때, 반환값이 있는 시스템 콜의 경우 포인터 f의 eax에 반환값을 저장하여 사용자 모드로 전환이 됐을 때 값을 읽을 수 있게 한다.  -halt() : shutdown\_power\_off() 함수 호출  -exit() : 현재 실행 중인 프로세스의 이름과 status를 출력한다. 이때 프로세스 이름은 thread\_name() 함수를 통해 가져오고, status는 exit 함수에서 넘겨받은 status 값이다. 이후, process\_exit() 함수를 실행시킨다. Process\_exit() 함수 안에는 thread\_exit() 함수도 포함되어 있다.  -exec() : 넘겨받은 program 이름을 process\_execute() 함수의 인자로 넘겨준다. 이때, process\_execute 함수가 자식 프로세스를 생성시키고 실행한다.  -wait() : 현재 프로세스의 자식 프로세스의 tid가 TID\_ERROR인지 확인하기 위해 thread\_pid() 함수를 실행한다. while문을 통해 주기적으로 자식 프로세스가 pid가 죽었는지 thread\_pid()를 통해 확인한다. 만약 죽었다면, 자식 프로세스에 저장된 exit status를 가져온다. 그리고 thread\_yield()를 통해 CPU 리소스를 반환한다.  -read() : fd가 0인지 확인하고 아니라면 -1을 반환한다. 0이라면 넘겨받은 size만큼 input\_getc() 함수를 반복한다. 이때, 반복할 때 buffer에 읽어들인 내용을 계속해서 저장한다. 이후, 반복한 횟수를 반환한다.  -write() : fd가 1인지 확인한다. 1이 아니라면 -1을 반환한다. Putbuf 함수를 이용해, buffer에 담긴 내용을 size만큼 출력한다. 이후, size 값을 반환한다. |

1. Additional System calls

* **새로운 시스템 콜(fibonacci, max\_of\_four\_int)을 구현하기 위해 수정하거나 작성한 코드에 대해 서술**
  1. **시험 및 평가 내용**
* **fibonacci 및 max\_of\_four\_int 시스템 콜 수행 결과를 캡처하여 첨부.**