Pintos 프로젝트 2_1. User Program Basic

(설계 프로젝트 수행 결과)

과목명 : [CSE4070-01] 운영체제

담당교수 : 서강대학교 컴퓨터공학과 박성용

조원 : 28조 김태훈

개발기간 : 2016. 10. 22. -2016. 10. 28.

최 종 보 고 서

프로젝트 제목: Pintos 프로젝트 2_1. User Program Basic
제출일: 2016. 10. 28.
참여조원: 28조 김태훈

I. 개발 목표

Pintos OS에서 user program의 실행이 가능하도록 하는 다양한 기능을 구현한다. 특히 user program과 OS의 interaction을 가능하게 하는 여러 system call을 구현하는데 중점을 둔다.

II. 개발 범위 및 내용

가. 개발 범위

현재의 Pintos OS에는 가장 기본적인 user program 실행 환경만 구현되어 있다. command-line을 통해서 들어온 argument passing과 user program의 실행을 위해 필요한 system call infrastructure를 구현한다.

나. 개발 내용

1. argument passing을 위해서 필요한 command-line의 parsing과 이후 system call로 argument들을 passing 하기 전 해당 argument들이 잘못된 memory를 access 하여 page fault를 일으키지 않도록 사전에 검사한다.

2. system call을 수행할 system call handler를 구현한다. 이후 system call handler에 read, write, execute, wait, exit, halt 총 6개의 system call이 성공적으로 수행될 수 있도록 관련 함수를 작성한다. 이때 thread 간 synchronization이 필요한 부분에 주의한다.

3. 기존에 구현해야 할 system call 이외에 pibonacci, sum_of_four_integers 라는 두 개의 system call을 추가하고, 이를 test하기 위한 sum.c 파일을 examples directory에 작성한다.

4. user program을 실행하는 과정에서 발생할 수 있는 다양한 error들을 test example 분석을 통해 파악하고 이를 잘 처리하여 이번 프로젝트 관련 test case를 통과한다.

III. 추진 일정 및 개발 방법

가. 추진 일정

10.22-23: Pintos structure 분석 및 개발 내용을 반영할 함수 결정

10.24-25: argument passing과 memory access 관련 문제 해결 및 system call handler 기본 구조 구현

10.24-26: read, write, execute, wait, exit, halt system call 추가 및 에러 처리

10.27-28: Project 마무리 및 보고서 작성

나. 개발 방법

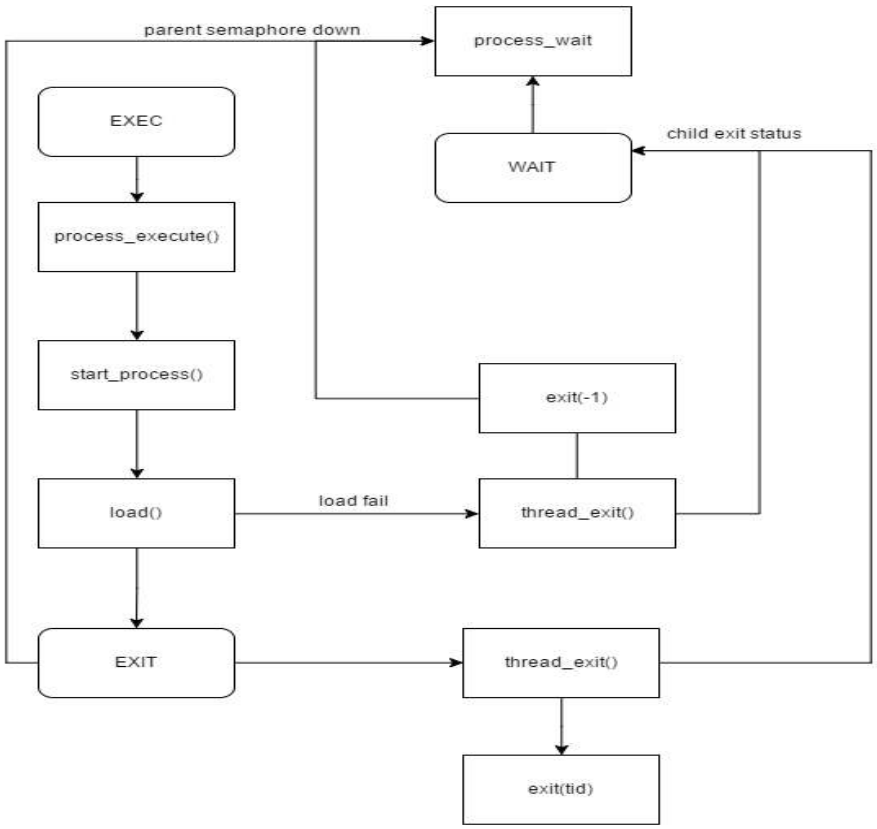
Pintos manual과 강의 자료, Operating Systems 교재를 참고하여 관련 사항을 학습한다. 특히 synchronization의 구현을 위한 semaphore에 관한 내용은 수업 시간에 배우지 않았으므로 주의 깊게 살펴본다. semaphore를 0으로 initialize한 상태에서 down을 하면 해당 thread는 semaphore가 다시 up될 때까지 wait 상태에 머물게 된다. sema_up을 해당 thread와 synchronize 할 thread에서 semaphore를 up하면 down 되어 있던 thread는 다시 execution을 시작한다. 이를 바탕으로 up, down을 실행할 함수를 결정하고 이를 잘 적용하여 synchronization을 잘 구현한다. 프로젝트 목표가 제대로 완수 되었는지 확인하기 위해서 Pintos OS 내부 test tool을 활용한다.

다. 연구원 역할 분담

혼자서 진행하는 프로젝트이므로 충분한 시간을 두고 철저한 계획 하에 프로젝트를 진행한다. 일정 부분 이상 진행되지 않으면 중간 결과를 확인 할 수 없으므로 구현 사항에 우선 순위를 정하여 그 순서대로 프로젝트를 진행 한다.

Ⅳ. 연구 결과

1. **합성 내용:** 전체적인 흐름도는 다음과 같다.



2. **제작 내용:**

(1) argument passing: command line으로 들어온 내용을 parsing하여 stack에 들어가도록 process.c의 load()를 변경하였다. 이를 위해서 strtok_r()을 활용하였다. 이를 stack이 set up된 다음 해당 프로세스의 stack에 push하였다. stack에 push 할 때는 manual에 있는 아래 그림을 참고 하여 해당 순서대로 push한다.

Address	Name	Data	Type
0xbffffffc	argv[3][...]	'bar\0'	char[4]
0xbffffff8	argv[2][...]	'foo\0'	char[4]
0xbffffff5	argv[1][...]	'-l\0'	char[3]
0xbfffffed	argv[0][...]	'/bin/ls\0'	char[8]
0xbfffffec	word-align	0	uint8_t
0xbfffffe8	argv[4]	0	char *
0xbfffffe4	argv[3]	0xbffffffc	char *
0xbfffffe0	argv[2]	0xbffffff8	char *
0xbfffffdc	argv[1]	0xbffffff5	char *
0xbfffffd8	argv[0]	0xbfffffed	char *
0xbfffffd4	argv	0xbfffffd8	char **
0xbfffffd0	argc	4	int
0xbfffffcc	return address	0	void (*) ()

(2) user memory access: system call handler가 user memory를 access하고자 할 때는 access 불가능한 영역에 access를 시도하여 page fault가 뜨지 않도록 사전에 막아야 한다. 이를 위하여 syscall.c에 check_address()를 추가하여 address가 null인 경우, address가 kernel 영역을 참조하는 경우 현재 thread를 강제 종료 하도록 하였으며, 만약 현재 thread의 page가 아닌 영역을 참조하려는 경우는 pagedir.c의 pagedir_get_page()를 사용하여 이를 판단한 뒤에 현재 thread를 강제 종료하도록 하였다.

(3) system call handler infrastructure: 여러 system call을 처리하기 위한 switch 문 구조와 각종 함수들을 이용한 read, write, execute, wait, exit, halt 총 6개의 system call을 구현하였다. read, write, halt는 이미 구현된 관련 함수를 사용하였으며, parent와 child 간의 synchronization이 필요한 execute, wait, exit에 해당하는 함수들 사이에는 semaphore를 활용하였다.
synchronization을 위해 struct thread에 새로 추가할 내용은 struct sync_tool이라는 구조체로 묶었다. 여기에 list_elem elem을 추가하여 list의 다양한 기능을 활용할 수 있도록 하였다.

```
struct sync_tool
{
    struct semaphore wait;
    struct semaphore exec;
    int exit_status;
    tid_t parent;
    struct list child_list;
    struct list_elem elem;
};
struct thread
{
    struct sync_tool sync;
    /* Owned by thread.c. */
    tid_t tid;                          /* Thread identifier. */
    enum thread_status status;          /* Thread state. */
    .....
};
```

(4) synchronization: child의 load가 끝나기 전에 parent가 exit에 들어가지 말아야 하며, child의 exit가 완전히 마무리 될 때까지 parent는 wait 상태에 머물러야 한다. 이를 위해서 semaphore를 사용하였다. 각 thread는 exec semaphore와 wait semaphore를 가지고 있으며, child의 load가 끝나기 전까진 parent의 exec semaphore를 down 시켜두었고, child의 load가 끝나면 다시 up하여 parent가 수행될 수 있도록 하였다. wait의 경우에는 parent의 wait semaphore를 down 시킨다. 이 semaphore는 child가 exit를 마치면서 다시 up 시킨다. 이때 parent가 child의 exit status를 가져와 이를 return 한다.

(5) trial and error: test case를 돌리는 과정에서 다양한 예외 처리 항목을 발견했다. argument로 적합하지 않은 내용이 들어올 경우 해당 thread를 종료하도록 하였으며, parent가 wait 상태에서 child의 exit_status를 가져오기 전에 child가 먼저 exit 해버리는 상태를 해결하기 위해 child가 parent의 wait semaphore를 up 한 이후 child의 exec semaphore를 down하여 parent가 child의 exit status를 가져올 수 있도록 synchronize하였다. 이후 parent는 child의 exec semaphore를 up 시켜 child의 exit process가 다시 진행되도록 한다.

(6) pibonacci(), sum_of_four_integer(): system call handler에 n번째 피보나치 수열의 값을 계산하는 pibonacci와 4개 수의 합을 계산하는 sum_of_four_integer system call을 추가하였다. 이를 위해 system call list에 두 system call을 추가하고 관련 기능 수행을 위한 함수들을 추가하였다. 이를 test 하기 위한 user program인 sum도 examples directory에 작성하였다.
sum의 실행 예시는 다음과 같다.

```
cse20121577@cspro1:~/pintos/src/userprog/build$ pintos-mkdisk filesys.dsk --filesys-size=2
cse20121577@cspro1:~/pintos/src/userprog/build$ pintos -p ../../examples/sum -a sum -- -f -q run 'sum 5 3 4 2'
Prototype mismatch: sub main::SIGVTALRM () vs none at /sogang/under/cse20121577/pintos/src/utils/pintos line 936.
Constant subroutine SIGVTALRM redefined at /sogang/under/cse20121577/pintos/src/utils/pintos line 928.
Copying ../../examples/sum to scratch partition...
qemu -hda /tmp/fqBNHFHzbK.dsk -hdb filesys.dsk -m 4 -net none -nographic -monitor null
Could not access KVM kernel module: No such file or directory
failed to initialize KVM: No such file or directory
Back to tcg accelerator.
PiLo hda1
Loading.........
Kernel command line: -f -q extract run 'sum 5 3 4 2'
Pintos booting with 4,096 kB RAM...
383 pages available in kernel pool.
383 pages available in user pool.
Calibrating timer...  52,275,200 loops/s.
hda: 1,008 sectors (504 kB), model "QM00001", serial "QEMU HARDDISK"
hda1: 174 sectors (87 kB), Pintos OS kernel (20)
hda2: 85 sectors (42 kB), Pintos scratch (22)
hdb: 5,040 sectors (2 MB), model "QM00002", serial "QEMU HARDDISK"
hdb1: 4,096 sectors (2 MB), Pintos file system (21)
filesys: using hdb1
scratch: using hda2
Formatting file system...done.
Boot complete.
Extracting ustar archive from scratch device into file system...
Putting 'sum' into the file system...
Erasing ustar archive...
Executing 'sum 5 3 4 2':
5 14
sum: exit(0)
Execution of 'sum 5 3 4 2' complete.
```

3. 시험 및 평가 내용:

Project 결과에 대한 평가는 make check의 pass/fail을 기준으로 진행하였다.
평가는 cspro.sogang.ac.kr에서 진행하였으며 simulator로 qemu를 사용하였다.
그 결과는 다음과 같다.

총 21개의 test를 모두 pass 하였다.

(1) 다음 11개의 test를 통해 생산성을 test하였다.
pass tests/userprog/args-none
pass tests/userprog/args-single
pass tests/userprog/args-multiple
pass tests/userprog/args-many
pass tests/userprog/args-dbl-space
pass tests/userprog/halt
pass tests/userprog/exit
pass tests/userprog/exec-once
pass tests/userprog/exec-arg
pass tests/userprog/exec-multiple
pass tests/userprog/wait-simple
system call을 문제없이 수행하여 의도한 결과를 출력하여 생산성을 만족하였다.

(2) 다음 11개의 test를 통해 내구성을 test 하였다.
pass tests/userprog/exec-missing
pass tests/userprog/sc-bad-sp
pass tests/userprog/sc-bad-arg
pass tests/userprog/sc-boundary
pass tests/userprog/sc-boundary-2
pass tests/userprog/exec-bad-ptr
pass tests/userprog/wait-twice
pass tests/userprog/wait-killed
pass tests/userprog/wait-bad-pid
pass tests/userprog/multi-recurse
각종 예외 상황을 적절하게 대처하여 Pintos OS의 비정상적인 종료를 방지하였다. 이로서 내구성을 만족하였다.

(3) 보건 및 안정성 test
OS가 마주하는 예외 사항은 다양하다 . 이에 대해 적절하게 대처하지 못하면 사용자의 불편을 초래하게 된다. 따라서 page fault를 방지하기 위해 모든 예외 상황을 파악하고 page fault가 발생하기 전에 해당 thread를 강제로 종료 시켰다. system call의 argument로 잘못된 값이 들어오는 경우는 다음과 같은 함수를 사용하여 안정성을 높였다.

```
void check_address(void* address){
    struct thread *cur=thread_current();
    if (address==NULL){
        thread_current()->sync.exit_status=-1;
        thread_exit();//exit(-1);
```

```
        }
        else if(is_kernel_vaddr(address)){
            thread_current()->sync.exit_status=-1;
            thread_exit();//exit(-1);
        }
        else if(pagedir_get_page(cur->pagedir,address)==NULL){
            thread_current()->sync.exit_status=-1;
            thread_exit();//exit(-1);
        }
    }
```

V. 기타

- 기타 관련 내용을 기술할 것.

1. 연구 조원 기여도: 김태훈 (100%)

2. 해당 프로젝트를 진행한 cspro1.sogang.ac.kr은 서버의 낙후화로 다양한 문제점들이 발생하였다. bochs simulator를 사용할 수 없어 qemu simulator만을 사용하여야 했으며, git이 설치되어 있지 않아 source code version control에 어려움이 있었다. 하지만 느린 서버 환경이 다양한 synchronization 문제를 발생시켜 이를 해결하며 Pintos OS 자체의 성능을 향상하는데 도움이 되었다.