Lab 1

Introduction

Lab1의 목표

1. system call interface를 이해한다
2. user program이 어떻게 커널에게 파라미터를 주고 value를 받는지 (-> trap frame?)
3. event handling이 어떻게 일어나는지
4. process structure를 이해하고 수정할 수 있다.

1st thing to do : exec() 살펴보기

Assignment

0. proc.h에 있는 process structure에 terminated된 프로세스의 exit status를 저장하기 위한 field를 추가해야 한다.

1. exit system call signature를 void exit(int status) 바꾸기
   - user.h, defs.h, sysproc.c, proc.c 업데이트
   - exit을 사용하는 모든 user space program들 >> 새로운 prototype에 맞게

>> user와 kernel 스택 사이에서 arg들이 어떻게 pass되는지에 익숙해져야 한다.

>> system call의 프로토타입을 바꿈으로써 발생하는 어려움 이해

>> 이미 존재하는 system call과 PCB를 안전하게 수정

Assignment

2. wait system call을 int wait(int *status)로 바꿔야 한다.

- wait system call은 반드시 현재 process가 자신의 모든 child process가 종료될 때 까지 실행을 하지 않도록 해야 한다
- argument를 통해 종료된 child proc.의 exit status를 반환해야 한다.
- -1 : no child or error / 정상 종료된 경우 child의 pid
- NULL을 argument로 넘겼을 때는 child의 exit status가 discarded되어야 한다 (Q)

>> system call argument와 return value에 대해 익숙해져야 한다 (same as 1.)

Assignment

3. int waitpid (int pid, int *status, int options) 추가하기

- argument pid에 해당하는 process가 종료될 때까지 기다린다 (like wait system call)
- return value : 종료된 process의 pid / -1 (process가 없거나 error)

>> kernel이 현재 프로세스를 blocking (주어진 pid를 가진 프로세스가 종료될 때까지)

4. waitpid가 잘 동작한다는 것을 알 수 있는 example program을 만들고, makefile을 수정해서 그 프로그램이 xv6가 부팅된 후 쉘에서 실행되도록 한다.

Walk through questions to help us

1. hello world program을 xv6 shell에서 실행할 수 있게 어떻게 추가할 것인지?

   - exec 함수의 path로 들어갈 수 있으면 될까?

   - cross compile?

   - edit Makefile (v)

2. gdb를 사용해서 wait system call을 track해보라
   a. children이 없을 때와 있을 때 어떤 일이 일어나는지 설명하라.
   b. system call 동작의 모든 중요한 단계들을 설명할 수 있도록 확실하게!

Deeper Understanding

- xv6 커널에 event가 일어나면, HW가 kernel모드로 전환한다.
- trapasm.S로 jump : label alltraps에서 시작
- build trap frame(1st) , set up memory segments(2nd)
- Trap frame : 중요한 arguments들을 trap handler에게 넘겨주기 위해 사용되는 stack에 있는 data structure
  - ex. user stack을 가리키는 pointers (>> system call이 argument들에 접근할 수 있도록 하기 위해)
  - x86.h에 있음 (간단하게 보기) (602)
- trap()을 부른다 (trap frame을 argument로 사용) (3272)
- trap.c에 trapno(in trap frame)를 기준으로 한 switch문(3353, 3363)이 있는데, 각각의 case가 event의 종류를 represent하고, 따라서 그것을 위한 trap handler를 부른다. (지금은 조금밖에 없다…)
- 첫 번째 case : sanity check 이후에 syscall()을 호출한다.

Deeper Understanding

- syscall() 은 system call들에 대한 top level handler이고, syscall.c에 있음 (3625)
- trapframe에 저장되어 있는 user code의 EAX register로부터 system call number를 얻는다

  >> system call handler table의 index로 사용한다. (system call에 맞는 handler를 부른다)

  >> 모든 handler의 function type은 동일하고, sys_xyz로 불린다 (xyz는 handling이 가능한 syscall 의미)

- system call의 return value도 trap frame의 EAX register에 저장된다. (Linux/x86 convention) >> user에게 return value를 전해주기 위해.

Deeper Understanding

handler들은 sysproc.c (process code), file system code, memory code 등 그 기능에 따라 다른 곳에 구현되어 있다. >> grep을 사용해서 찾아보기 (ex. sys_kill in sysproc.c)

- sys_xyz : sys_kill(), sys_fork(), sys_exit(),, etc
- sys_kill같은 handler들은 kill처럼 그 시스템콜의 실제 implementation이 있는 함수를 call한다.
- argint, argstr (3545-3564) : user stack으로부터의 parameter들을 가져올 수 있도록 돕는 함수들 (trapframe의 정보들을 사용해서)

(textbook) Code : Assembly trap handlers (42p)

xv6는 int instruction을 만났을 때, x86 HW가 무언가를 하도록 setting한다

int instruction: processor가 trap을 발생시키도록 돕는다

System call interrupt: 64번 interrupt (in x86 256 interrupts)

tvinit

- main에서 호출되고, IDT(Interrupt Descriptor Table)의 256개의 entry를 set up 한다. >> i번 interrupt는 vectors[i]에 있는 handler에 의해 handle된다.
- T_SYSCALL(the user system call trap)을 handle : **gate(?)**를 trap type으로 지정 >> SETGATE의 두번째 arg.를 1로 지정함으로써… (interrupt를 enable시킨다. = trap을 handle하는 동안 다른 interrupt들도 허용)
- 커널이 system call gate privilege를 DPL_USER로 지정 >> user program이 int insturction으로 trap을 발생시킬 수 있게 허용

(textbook) Code : Assembly trap handlers (42p)

user mode에서 kernel mode로 protection lv.을 바꿀 때, 커널은 유저 프로세스의 스택을 사용할 수 없다 (not valid).

xv6는 x86 HW가 trap이 일어났을 때 stack switch를 하도록 프로그램한다.

- stack switch: task segment descriptor를 세팅
  - HW가 stack segment selector, new value for esp (esp가 이제 어디를 가리켜야 하는지)를 load
  - switchuvm : user process의 kernel stack의 top address를 task segment descriptor에 저장한다.

(textbook) Code : Assembly trap handlers (42p)

Trap이 일어나면!

- 만약 processor가 user mode에서 돌아가고 있었다면,
  - task segment descriptor로부터 %esp(stack pointer reg.)와 %ss(stack segment reg.) 를 load >> 새로운 stack에 넣는다.
- 만약 processor가 kernel mode에서 돌아가고 있었다면, 아무 일도 일어나지 X

processor가 eflag, cs, eip register를 push (+ error word)

eip와 cs reg.를 해당하는 IDT entry로부터 load

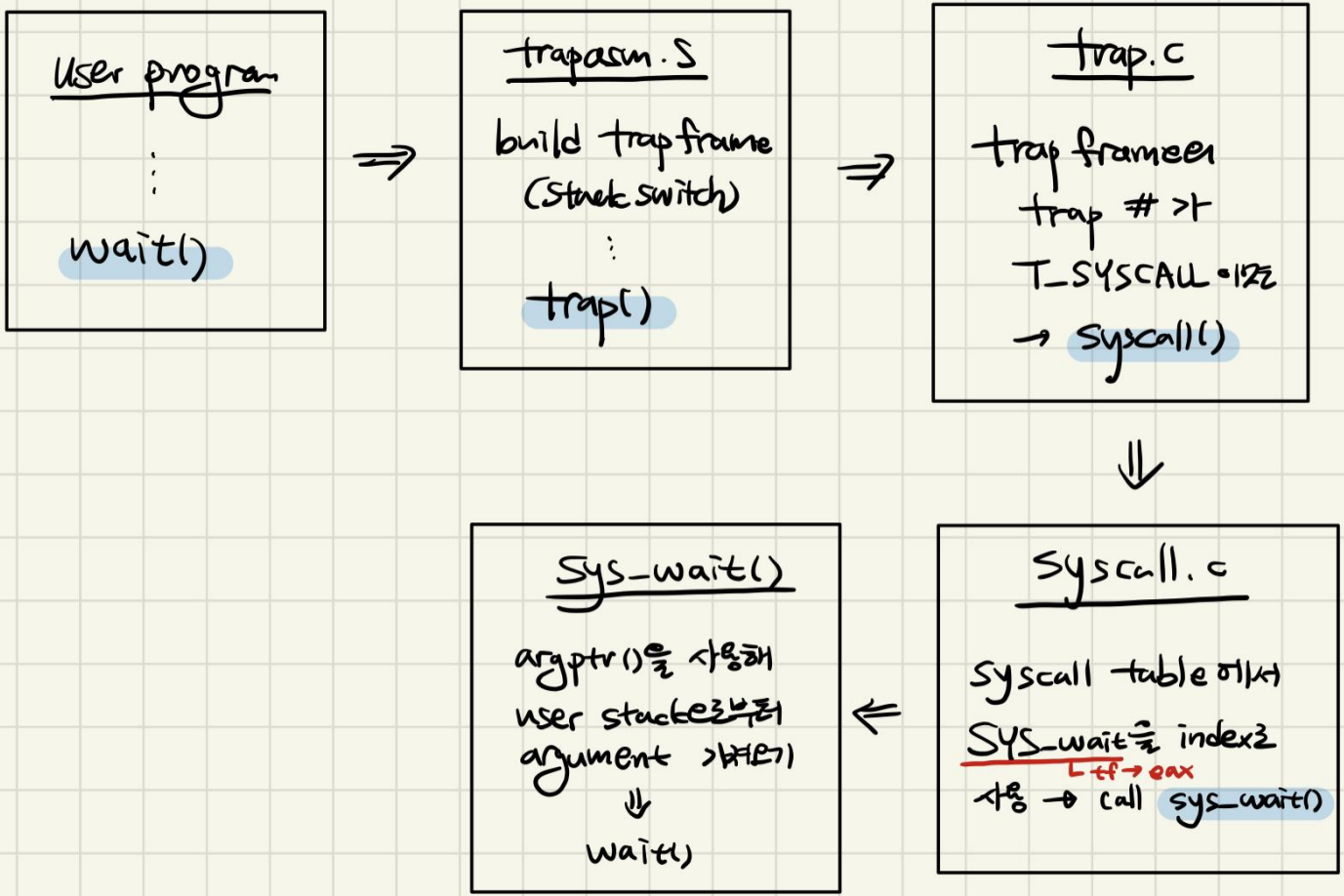
- IDT entry가 가리키는 entry point를 generate하기 위해서 Perl script 사용
- 각 entry가 error code, interrupt number를 push하고, alltraps로 jump

alltraps

- ds, es, fs, gs, 범용 레지스터들을 push

>> 이렇게 함으로써 커널 스택이 trapfrmame을 포함하게 된다.

System Call


User program
wait()
trapasm.S
build trapframe
(Stack switch)
trap()
trap.c
trap frame에
trap # 저장
T_SYSCALL 인지
→ syscall()
syscall.c
syscall table 에서
SYS_wait을 index로
tf→eax
사용 → call sys_wait()
sys_wait()
argptr()을 사용해
user stack으로부터
argument 가져오기
wait()