OS PA_0 report

소프트웨어학과 2018314827 차승일

Abstract

Init process를 담당하는 init.c 파일의 코드를 위에서 아래와 같이 수정하였다.

(before)

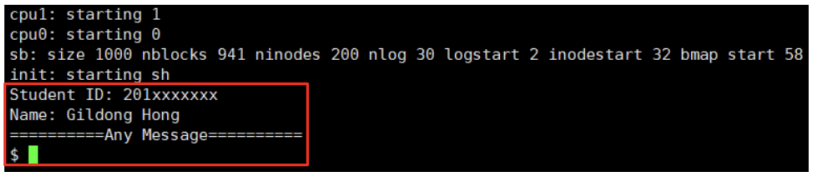
```c
// init: The initial user-level program

#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"

char *argv[] = { "sh", 0 };

int
main(void)
{
  int pid, wpid;

  if(open("console", O_RDWR) < 0){
    mknod("console", 1, 1);
    open("console", O_RDWR);
  }
  dup(0);  // stdout
  dup(0);  // stderr

  for(;;){
    printf(1, "init: starting sh\n");
    pid = fork();
    if(pid < 0){
      printf(1, "init: fork failed\n");
      exit();
    }
    if(pid == 0){
      exec("sh", argv);
      printf(1, "init: exec sh failed\n");
      exit();
    }
    while((wpid=wait()) >= 0 && wpid != pid)
      printf(1, "zombie!\n");
  }
}
```

(after)

```c
// init: The initial user-level program

#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"

char *argv[] = { "sh", 0 };

int
main(void)
{
  int pid, wpid;

  if(open("console", O_RDWR) < 0){
    mknod("console", 1, 1);
    open("console", O_RDWR);
  }
  dup(0);  // stdout
  dup(0);  // stderr

  for(;;){
    printf(1, "init: starting sh\n");
    printf(1, "Student ID: 2018314827\n");
    printf(1, "Name: Seungil Cha\n");
    printf(1, "Gen G fighting~\n");

    pid = fork();
    if(pid < 0){
      printf(1, "init: fork failed\n");
      exit();
    }
    if(pid == 0){
      exec("sh", argv);
      printf(1, "init: exec sh failed\n");
      exit();
    }
    while((wpid=wait()) >= 0 && wpid != pid)
      printf(1, "zombie!\n");
  }
}
```

과제 내용

- **Print your student id, name, and any message(optional) on boot message**



- **Hint : Use grep or other analysis tool**

xv6를 부팅하였을 때, 자신의 학번, 이름, 그리고 아무 메세지 하나를 출력하도록 하는 것이 목표이다.

개발환경

Ji server

문제 분석 및 추론

먼저, 코드를 아무것도 수정하지 않은 채로 xv6를 ***make qemu-nox***로 run시켜 본다.
긴 아웃풋이 출력되고, 예시에 나온 것처럼 마지막 부분에 아래와 같은 output이 출력된다.

```
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
```

그 후 쉘이 시작된다.

마지막 출력에 init 이라는 단어가 들어있으므로, init process(pid 1을 갖는)이라고 추론해볼 수 있다. xv6 디렉토리의 일련의 구성 파일들 중 init이라는 이름의 파일이 있는지 ***ls | grep init*** 으로 찾아본다.

```
_init
init.asm
init.c
initcode
initcode.asm
initcode.d
initcode.o
initcode.out
initcode.S
init.d
init.o
init.sym
```

위와 같은 init이라는 이름을 가진 파일들이 존재한다.

해당 파일들 중 init process를 다루는 코드가 있을 것이고, 해당 코드에 ***init: starting sh***, 즉 init process에서 shell을 시작하는 부분을 찾으면 된다.

init.c에 대해 ***grep "init: starting sh" init.c*** 로 아웃풋을 조사하면 아래와 같은 아웃풋이 나온다.

```
printf(1, "init: starting sh\n");
```

init.c의 코드 구조를 확인해보니, 아래와 같았다.

```
// init: The initial user-level program

#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"

char *argv[] = { "sh", 0 };

int
main(void)
{
  int pid, wpid;

  if(open("console", O_RDWR) < 0){
    mknod("console", 1, 1);
    open("console", O_RDWR);
  }
  dup(0);  // stdout
  dup(0);  // stderr

  for(;;){
    printf(1, "init: starting sh\n");
    pid = fork();
    if(pid < 0){
      printf(1, "init: fork failed\n");
      exit();
    }
    if(pid == 0){
      exec("sh", argv);
      printf(1, "init: exec sh failed\n");
      exit();
    }
    while((wpid=wait()) >= 0 && wpid != pid)
      printf(1, "zombie!\n");
  }
}
```

printf(1, "init: starting sh\n"); 가 무한루프 안에 있었다. Shell을 실행하기 전 메세지를 출력하는 부분이다. 이 부분에 원하는 메세지를 추가하면 될 것이다.(***init: starting sh***에서도 첫번째 인자를 1로 주어 stdout으로 지정하였으므로 똑같이 따라갔다)

```
printf(1, "init: starting sh\n");
printf(1, "Student ID: 2018314827\n");
printf(1, "Name: Seungil Cha\n");
printf(1, "Gen G fighting~\n");
```

코드를 수정하고 다시 ***make qemu-nox***로 run시켜 본다.

```
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
Student ID: 2018314827
Name: Seungil Cha
Gen G fighting~
```

잘 출력된다. 이제 init.c 코드를 분석해 해당 과제가 어떤 의미를 가지고 있는지 공부한 점을 기록하겠다.

분석

제일 먼저, *argv 부분이 눈에 들어왔다.

```
char *argv[] = { "sh", 0 };
```

argv는 프로세스가 실행될때 argument를 전달하기 위해 사용한다. 즉, shell을 실행하기 위한 프로세스에게 전달할 것이라고 추론할 수 있다.(0은 NULL 포인터)

init: starting sh을 출력한 후 아래와 같은 코드가 있다.

```
pid = fork();
if(pid < 0){
  printf(1, "init: fork failed\n");
  exit();
}
if(pid == 0){
  exec("sh", argv);
  printf(1, "init: exec sh failed\n");
  exit();
}
while((wpid=wait()) >= 0 && wpid != pid)
  printf(1, "zombie!\n");
```

현재 프로세스(init process)에서 fork()를 하여 자식 프로세스를 만든다.

```
if(pid < 0){
  printf(1, "init: fork failed\n");
  exit();
}
```

for 함수가 실패하면 pid가 0보다 작은 -1이 return 되므로 해당 상황을 핸들링 하는 코드인 것으로 판단된다.

```
if(pid == 0){
  exec("sh", argv);
  printf(1, "init: exec sh failed\n");
  exit();
}
```

자식 프로세스는 fork()의 리턴값으로 0을 받으므로 해당 코드는 자식 프로세스가 적용받는 코드이다.
exec() 시스템 콜은 자식한테 부모가 자신과는 다른 일을 시킬 때 address space, register을 날려버리고 새로운 프로세스로 만들 때 쓰므로, ***exec("sh", argv);*** 로 shell을 실행함을 추론할 수 있다.

정상적으로 exec이 성공한다면 아예 다른 프로세스로 전환이 된 것이므로, ***print(1, "init: exec sh failed\n");*** 는 출력이 되면 안되고, 만약에 출력이 된다면 exit()을 하도록 되어있다.

```
while((wpid=wait()) >= 0 && wpid != pid)
  printf(1, "zombie!\n");
```

pid 변수가 -1이나 0이 아닌 경우에 실행되는 블록이므로, 부모 프로세스가 적용받는 코드이다.
wait()는 wait(NULL)과 동일하게 동작하므로, 자식 프로세스들 중 제일 먼저 종료되는 것의 pid를 받는다.
먼저 부모 프로세스인 init 프로세스는 ***wait()***를 통해 자식 프로세스가 끝날때까지 블록킹된다. 그 다음 쉘 프로세스가 된 자식 프로세스가 완료되면, 자식 프로세스가 있었는지 확인 하고(***wpid >= 0*** 인지), 생성했던 자식과 pid와 같은지 비교 후, 아니라면 부모 프로세스는 좀비를 발견했다고 출력 후 다시 wait()를 하도록 한다. 어떤 의미를 지닌 것인지는 xv6 코드를 더 분석해 확인할 필요가 있어보인다.

해당 과제의 목표는 "xv6를 부팅하였을 때, 자신의 학번, 이름, 그리고 아무 메세지 하나를 출력하도록 하는 것"인데, 이를 달리 말하면, "컴퓨터를 부팅할 때 init process에서 shell을 실행하기 전 출력 메세지를 수정해라"와 같은 말이라 판단된다. 따라서 이 과제의 진정한 목표 및 의미는 컴퓨터가 부팅될때의 과정을 분석하고, xv6에서 init process를 담당하는 곳을 찾아가 분석해 코드를 수정하는 것이라 생각된다.