**Pintos 프로젝트 2\_1. User Program Basic**

**(설계 프로젝트 수행 결과)**

**과목명 : [CSE4070-02] 임베디드시스템소프트웨어**

**담당교수 : 서강대학교 컴퓨터공학과 김 영 재**

**조원 : 28조 안진우**

**개발기간 : 안진우**

**최 종 보 고 서**

**프로젝트 제목: Pintos 프로젝트 2\_1. User Program Basic**

**제출일: 2017.10.17**

**참여조원: 28조 안진우**

**I. 개발 목표**

User program이 Pintos 내에서 running되도록 하는 것이 개발 목표이다. 이를 위해 system call를 구현하여 user program이 service 요청을 가능하도록 하고, arguments passing을 구현하여 user program이 parameter을 전송받을 수 있도록 한다. 또, parent 와 child process간의 동기화되도록 구현하여 parent가 child의 종료를 기다릴 수 있도록 한다.

**II. 개발 범위 및 내용**

1. **개발 범위**

Userprogram이 pintos OS와 interaction하기 위한 부분을 개발해야 한다. 따라서 User 과 Kernel 사이에 있는 system call part를 개발하고, process가 run할때 실행되는 flow 범위 내에서의 arguments passing, 동기화를 위한 개발이 필요하다.

**나. 개발 내용**

1. arguments passing

process를 run할 때 arguments passing이 stack을 통해 올바른 순서로 이루어지도록 구현해야 한다. 이를 통해 process에서는 무사히 arguments를 stack으로부터 얻을 수 있다.

2. system call

system call을 통해 user에서 OS 에 서비스요청을 할 수 있도록 구현해야 한다. 기본적인 system call이 구현되어야 user program을 실행할 수 있다. 이번 프로젝트에서는 read, write, halt, exit, exec, wait 등 기본적인 6개의 시스템콜을 구현하도록 하였다.

system call에서는 또, user에서 잘못된 영역의 메모리 접근을 차단하도록 개발해야 한다.

3. parent process 와 child process 간 동기화 구현

parent process는 child process가 실행되면 exit될 때 까지 wait해야 한다. 이를 위해서는 process\_wait, thread\_exit함수간에 동기화를 구현해야 한다.

4. 추가 시스템콜 구현

이번 프로젝트에서는 추가적으로 pibonacci, sum\_of\_four\_integers 시스템콜을 구현하여 user program을 통해 실행해야 한다. 특히 sum\_of\_four\_integers함수는 arguments가 4개나 되기 때문에 syscall4를 새로 구현할 필요가 있다.

**III. 추진 일정 및 개발 방법**

**가. 추진 일정**

|  |  |
| --- | --- |
| 10/4 ~ 10/8 | pintos 메뉴얼 분석 및 코드 이해 |
| 10/9 ~ 10/15 | 프로그램 생성 및 구현 |
| 10/16 ~ 10/17 | 테스팅, 에러 처리 및 주석 , 보고서 작성 |

**나. 개발 방법**

이번 프로젝트는 개발하기 앞서서 pintos OS구조를 정확히 이해하고 프로그램이 run될때 실행되는 flow를 코드로 깊숙히 이해하는 것이 매우 중요하다. 이 과정이 선행되어야, 어떻게 구현기에 앞서 추론이 가능해지기 때문이다. 따라서 왠만한 이해가 되기 전에는 섣불리 시작하지 않고 pintos 메뉴얼을 읽으면서 코드를 분석한다. 어느정도의 이해가 선행되면 각 단계를 나눠 구현 전략을 짠다.

1. arguments passing

parameter가 stack을 통해 어떻게 전송되는 지 파악하고, 해당 구현은 process run상에서 어느 위치에서 이루어져야 synchronize하게 프로그램이 돌아가는지 파악한다.

2. system call

기본적인 system call구현 뿐 아니라, user의 잘못된 address 접근 등 예외처리가 어떤 상황이 생길 수 있으며, 어떻게 해결해야 할지 파악한다. 또 전체적으로 system call handler가 어떤 구조로 system call을 호출할지 계획을 잡는다.

3. synchronization between parent and child process

parent process가 child process가 종료될 때까지 wait할 수 있도록 하기 위해 어떻게 동기화를 구현하여 lock을 어디에 잡을 지 파악한다.

이후 구현을 진행하고, 테스팅을 통해 지속적으로 디버깅하여 코드의 완성도를 올린다.

**다. 연구원 역할 분담**

1. 핀토스 메뉴얼 분석 및 코드 이해 : 안진우 100%

2. 프로그램 생성 및 구현 : 안진우 100%

3. 테스팅, 에러 처리 및 주석, 보고서 : 안진우 100%

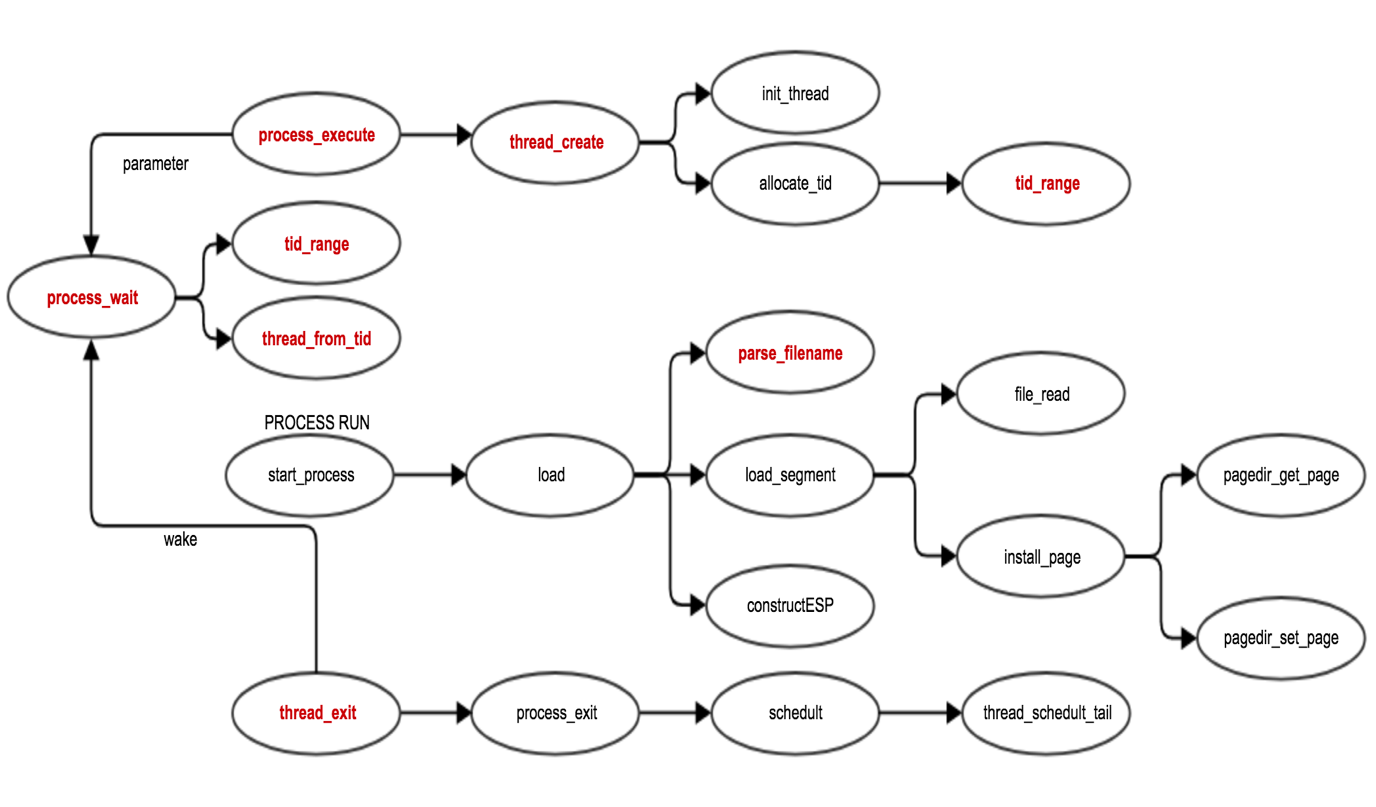
**IV. 연구 결과**

1. 합성 내용:

- 설계 목표에 필요한 내용을 조사 분석한 후 그들을 바탕으로 구성한 전체 프로그램 구성도.

1.1. 프로그램 실행 구성도 / 내가 구현한 부분 (빨간 줄)

1.1.1 process life time 구상도



1.1.2 주요 함수 설명

process\_wait : child process가 exit될 때가지 wait

tid\_range : 현재 할당된 tid의 범위를 리턴

thread\_from\_tid : tid를 가진 프로세스의 struct\_thread(PCB) 리턴

thread\_create : thread생성 및 초기화, 스텍 프레임 초기화

init\_thread : struct\_thread(PCB) 초기화 및 all\_elem(PBC table) 등록

allocate\_tid : tid 할당. 새로운 tid는 기존의 tid에 1을 증가하여 할당한다.

start\_process : 프로세스가 run할때 가장 먼저 호출된다.

load : file로부터 memory로 data를 load하고 page table을 할당한다.

load\_segment: 파일로 부터 data를 읽어오고, page table에 새로운 맵핑정보를 update한다.

install\_page : page table에 맵핑이 안되있을 경우 새로운 맵핑정보를 set한다.

pagedir\_get\_page : page table에 맵핑되는 physical address를 반환한다.

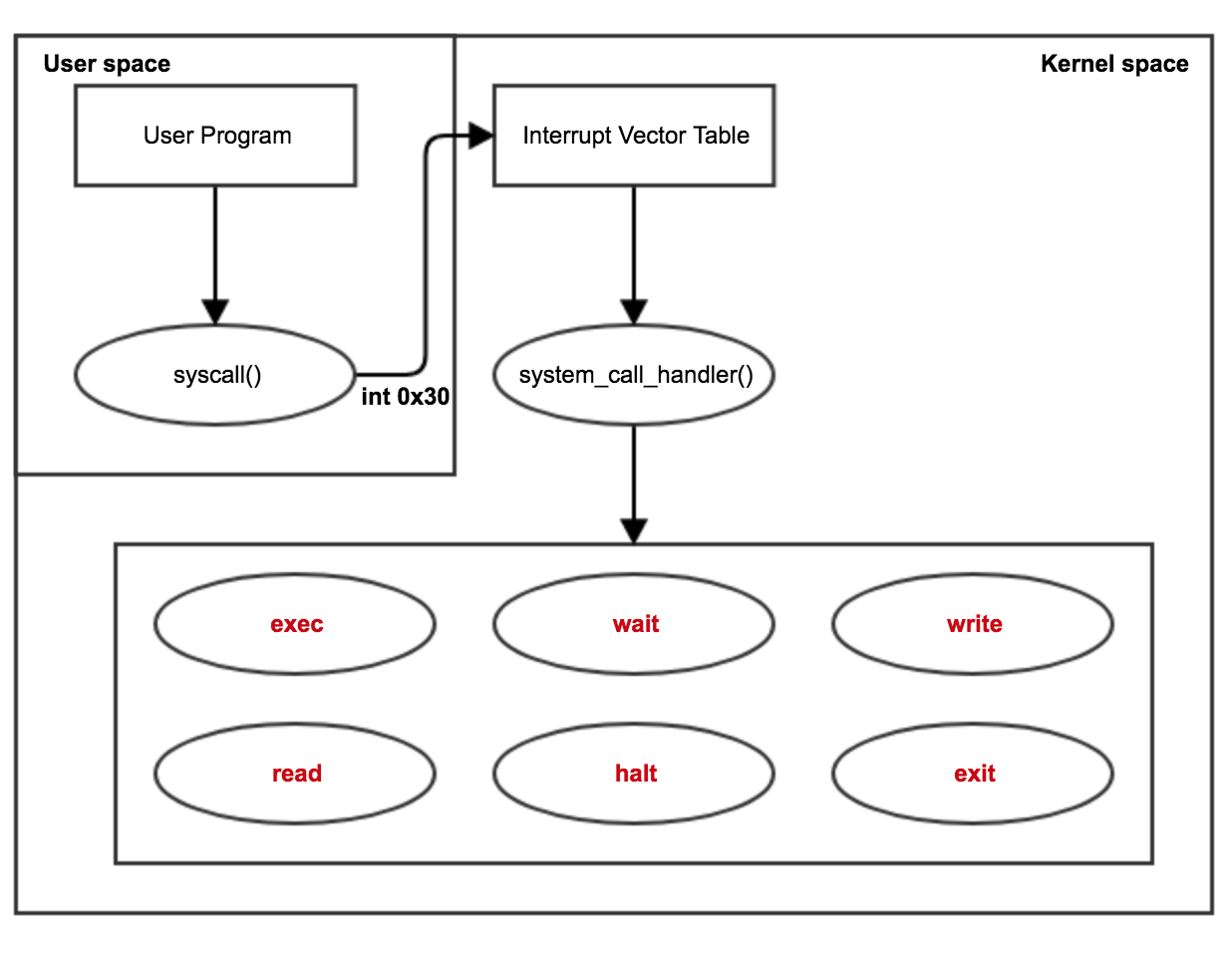
pagedir\_set\_page : page table에 page에 대한 맵핑 정보를 set한다.

parse\_filename : filename과 arguments를 서로 분리하여 순수한 filename을 획득한다.

parse\_filename : filename+ arguments 로부터 filename을 분리한다.

constructESP : stack에 parameter을 올바른 순서로 할당하여 parameter passing이 가능하게 한다.

1.2. 시스템 콜 수행 구성도



2. 제작 내용: 개발 결과

1. Arguments Passing

|  |
| --- |
| parameter passing은 load함수 수행시 설정되도록 구현하여 stack이 setup된 이후에 stack에 parameter을 올바른 순서로 삽입하였다. 이를 통해 process는 실행이 된 이후 stack을 통해서 parameter을 받아올 수 있었다.  1. parse\_filename  parse\_filename함수는 기존의 filename, arguments가 혼재된 file\_name으로 부터 실제 file name만을 분리하기 위한 함수이다. 이 과정이 선행되어야 load함수내에서는 올바른 file을 open하고 read하여 data를 file system으로부터 memory로 load해올 수 있다.  2. constructESP  constructESP 는 parameter을 stack에 할당하는 함수로, setup\_stack을 통해 stack이 setup된 직후에 호출하여 초기화된 stack에 parameter을 우선적으로 할당하도록 하였다. 여기서 핵심은 올바른 순서로 parameter와 address를 할당하여 process 를 run하면서 보낸 parameter가 process에서 의도대로 받도록 하는 일이었다. 이를 구현하기 위해 hex\_dump함수를 이용하여 디버깅하였다. |

2. System call

|  |
| --- |
| System call은 유저가 커널에 서비스를 요청하기 위한 함수이다. system call handler에서는 효율적으로 system call 을 수행하기 위해서 function array를 사용하였다. 즉 function array에 수행할 여러가지 system call 함수를 등록하고, index로 system call number을 부여하였다. 이를 통해, 시스템콜 핸들러함수에서는 user로부터 받은 system call number을 switch문등을 사용하지 않고 index로 direct하게 system call을 호출할 수 있었다. 이번 과제에서는 총 6개의 시스템콜 함수를 구현하였다.  1. sys\_exec()  process를 초기화하여 실행하기 위한 system call이다. 이를 위해 process\_execute함수를 호출하며 tid를 리턴한다. 여기서는 예외처리를 위해 dir\_lookup\_file함수를 따로 구현하여, parameter로 받은 file name이 실제로 존재하는지 확인하였다.  .2 sys\_wait()  child process tid를 parameter로 받아 child가 종료될 때까지 기다리기 위한 system call이다. exit\_status를 리턴한다.  3. sys\_exit()  process를 exit하기 위한 함수로, exit\_status를 유저가 parameter로 보낸다. 이 exit\_status를 parent process의 process\_wait함수가 return할때까지 synchronize하게 보관하는 것이 매우 까다로웠는데, 이를 위해 semaphore를 구현하고 struct\_thread에 exit\_status 변수를 추가하여 exit\_status를 보관하였다.  4. sys\_halt()  system을 shutdown하기 위한 함수로 shutdown\_power\_off 함수를 호출한다.  5. read()  file또는 표준입력을 user에 보내기 위한 system call이다. 이번 과제에서는 표준 입력만 받도록 구현하였으며 이를 위해 input\_getc 함수를 이용하였다.  6. write()  file또는 표준출력을 user로부터 보내기 위한 system call이다. 이번 과제에서는 표준 출력만 받도록 구현하였으며 이를 위해 putbuf 함수를 이용하였다.  7. 기타  7.1 get\_argv  system call handler에 stack을 통해 전달된 parameter를 받아오기 위한 함수이다.  7.2 check\_address  user access memory에서의 user가 보낸 invalid pointer, kernel memory로의 잘못된 접근, NULL pointer 등이 아닌지 체크하기 위한 함수이다. 잘못된 접근일 경우 terminate\_process를 호출하여 unlock 및 동적 할당 메모리를 free시키고 프로세스를 종료한다.  7.3 terminate\_process  프로세스가 잘못된 메모리 접근 등에 대한 예외처리를 위한 함수로, unlock 및 동적 할당 메모리를 free시키고 프로세스를 종료한다. |

3. synchronization issue (process\_wait, thread\_exit)

|  |
| --- |
| pintos OS에서는 parent process는 child process가 종료될때까지 wait해야한다. 이를 위해서는 process\_wait함수와 thread\_exit 함수가 synchronize하게 작동하도록 구현하는 것이 핵심인데, struct\_thread에 변수로 할당한 semaphore가 조금이라도 잘못 구현하면 exit함수를 통해 아예 free되버리기 때문에 매우 까다로웠다. 또한, user가 exit() system call요청을 통해 child process를 exit시킬 경우 parameter로 보낸 exit\_status를 parent process의 wait에서 받도록 하는 부분 역시 매우 까다로웠다.  1. issue 1 : parent process wait until child process dies  이를 구현하기 위해서 struct thread에 sema\_wait이라는 이름의 semaphore를 구현하였다. process\_wait함수에서는 우선 sema\_wait의 value를 0으로 초기화하고 바로 lock을 걸어 process를 wait하도록 하였다.  이를 thread\_exit에서 unlock시켜주었는데, 여기서 완전히 exit이 되버린 후에 unlock시키려 한다면 이미 struct thread가 free되어 semaphore을 찾을 수 없게 되어 문제가 생긴다. 따라서, thread가 dying 상태로 바뀌기 직전에 unlock시켜 synchroinze하게 돌아가도록 하였다.  2. issue 2 : parent should receive exit\_status from child process  exit\_status를 child로부터 parent로 보내기 위해서는 thread\_struct에 변수를 생성하여 보관하는 것이 가장 바람직하다. 하지만 문제는 process\_wait이 thread\_exit으로부터 wake하여 exit\_status를 받기 전에 thread\_exit함수가 thread\_struct를 free시킬 수 있다는 점이다.  따라서 이를 해결하기 위해서는 또하나의 semaphore을 이용해 이번에는 child process의 thread\_exit에서 lock을 걸어, process\_wait이 무사히 exit\_status를 thread\_struct로부터 받을 때 까지 기다려야 한다.  이를 위해 thread\_struct에 sema\_exit이라는 이름의 semaphore을 생성하였다. thread\_exit에서는 sema\_wait을 unlock 시키자마자 이 sema\_exit의 value를 0으로 초기화시키고 lock을 걸어 wait한다. wake된 process\_wait에서는 exit\_status를 thread\_struct로부터 받고 sema\_eixt이 lock이 될때까지 while문을 통해 잠시 기다리다가, lock이 걸리면 sema\_exit을 unlock시킨다. 이러면 synchronize하게 나머지 exit이 진행된다. |

4. 새로운 system call 구현 (pibonacci(), sum\_of\_four\_integers())

|  |
| --- |
| 기존 pintos에 pibonacci와 sum\_of\_four\_integers 를 구현하여, 실제 프로그램을 작성하여 해당 system call을 호출하였다. 실제 프로그램은 pintos/user/example 디렉토리에 sum이라는 이름으로 생성하였다.  1. syscall4()  sum\_of\_four\_integers 는 4개의 파라미터를 받기 때문에 syscall4 함수를 새로 추가하였다. syscall 함수는 stack에 system call number 과 4개의 parameter을 삽입하고 interrupt를 걸어 kernel 영역으로 전환하여 system call handler함수가 호출되도록 한다.  2. /lib/syscall-nr.h에 system call number 등록  새로운 시스템콜에 대한 system call number을 등록하였다.  3. system call handler 내 system call 구현  pibonacci()와 sum\_of\_four\_integers()의 system call 함수를 실제 구현하여 동작하도록 하였다.  4. user level program & testing  실제 user level program을 구현하여 pibonacci와 sum\_of\_four\_integers을 호출하였다. 실험결과 성공적으로 함수가 호출되고, 함수내에서는 올바른 계산이 이루어졌다. |

3. 시험 및 평가 내용:

-평가 방법

1. 21개의 테스트 케이스 수행하기 (make clean 상태일때)

1. $cd pintos/src/thread
2. $make
3. $cd pintos/src/userprog
4. $make
5. $cd build
6. $make check
7. 76개의 테스트케이스중 21개가 구현되었나 확인한다.
8. cf)pintos/src/userprog 에서 make\_and\_test.sh 쉘을 참고한다.

2. ./sum 실행파일로 추가한 pibonacci, sum\_of\_four\_integers 시스템 콜 수행하기 (make clean 상태일때)

1. $cd pintos/src/thread
2. $make
3. $cd pintors/src/example (sum.c 프로그램이 있는 경로)
4. $make (sum 실행파일 생성)
5. $cd pintos/src/userprog
6. $make
7. $cd build
8. $pintos-mkdisk filesys.dsk --filesys-size=2 (file system 생성)
9. $pintos -f -q (formatting)
10. pintos -p ../../example/sum -a sum -- -q (sum실행파일 파일시스템디스크에 옮기기)
11. pintos -q run 'sum 5 5 1 4'
12. cf)pintos/src/userprog 에서 make\_and\_exe.sh 쉘을 참고한다.

-보건 및 안정, 생산성과 내구성

이번 pintos에서의 안정성 및 보건성에서 핵심으로 둘 부분은 synchronization이다. 복잡한 운영체제가 여러 thread로 분기하여 도는 환경에서 서로의 synchronization을 정밀하게 맞추지 않는다면 OS는 먹통이 되기 때문이다. 가장 까다로운 부분은 parent process와 child process간의 synchronization을 맞추는 부분이었다. 특히, process\_wait 함수에서는 thread\_struct의 변수에 넣은 exit\_status를 받아야 하는데, child process가 exit되고 나면 free되어 받을 수 없다는 문제가 있었다.

이를 해결하기 위해 semaphore을 2개를 생성하였다. sema\_wait은 parent가 child의 exit을 기다리도록 lock을 거는 semaphore이다. 다만, semaphore가 thread\_struct 내에 있기 때문에 실제 exit이 다 종료될때 까지 기다리면 안되고 exit 과정에서 초반부에 wake 하여 수행되어야 한다. 따라서 chlid가 exit이 되는 한 순간에서는 parent process와 child process가 동시에 돌게 된다.

sema\_exit은 child process가 parent process가 exit\_status를 받기까지 기다리기 위한 semaphore이다. child process와 parent process가 exit 때 동시에 돌기 때문에 parent 가 exit\_status를 받기 전 child 가 struct thread를 free시킬 수 있기 때문에 child는 exit\_sema를 통해 wait하게 된다. 이렇게 두개의 semaphore를 구현하여 각 parent, child에게 wait하도록 함으로써 synchronization을 sparse하게 맞출 수 있었다.

두번째로 내구성을 보장하기 위한 핵심은 예외처리였다. user의 잘못된 memory접근에 대해서는 kernel영역인지 체크하는 함수(is\_kernel\_vaddr)를 이용하고, unvalid pointer인지 체크하는 함수(pagedir\_is\_mapped)를 직접 생성하였다.

또, exec함수에 존재하지 않는 file name이 들어온걸 체크하기 위하여 dir\_lookup\_file 함수를 생성하여 파일시스템의 root 디렉토리 내에 해당 파일이 존재하는지 체크하도록 하였다.

또 process\_wait 함수에서는 parameter로 온 tid가 현재 OS내 실존하는 tid인지 체크하기 위한 tid\_range함수를 만들었고, struct\_thread에 exit\_soon변수를 생성하여 child process가 이미 한번 exit이 되진 않았는 지 체크하도록 하였다. exit\_soon 변수는 process\_execute에서 thread가 생성될때 false로 초기화되며, thread\_exit에서 thread가 exit될때 true로 바뀐다. 따라서 process\_wait에서 exit\_soon이 true가 되어있으면 예외처리를 해야한다.

**V. 기타**

1**. 연구 조원 기여도:**

안진우: 100%

2. 기타 본 설계 프로젝트를 수행하면서 느낀 점을 요약하여 기술하라. 내용은 어떤 것이든 상관이 없으며, 본 프로젝트에 대한 문제점 제시 및 제안을 포함하여 자유롭게 기술할 것.

본 프로젝트는 서강대학교에 입학한 이래로 가장 어려운 프로젝트였다. 기존의 프로젝트들은 주로 시간을 무작정 투자하면 노력만으로도 꾸역 꾸역 진행이 가능하였지만, 핀토스 프로젝트는 OS에 대한 확실한 이해와 분석, 코드를 짜기 전 전략 수립을 하지 않으면 코드를 짜는 것 자체가 무의미 하였다. 또, 왠만큼 짜지 않으면 프로그램이 돌아가는 지 확인하기가 매우 어렵다는 점도 진행을 더디게 하였다. 따라서 커널 내에서 무슨 일이 일어나는 지를 순수한 추론에 의존해야 하는 경우가 많았으며, 이는 역설적으로 실력 향상 및 추론력 향상에 매우 큰 도움이 되었다고 생각한다. 또, 시스템 쪽에 큰 흥미를 가지고 있었기에, 어렵지만 스스로 난관을 해쳐 나가면서 굉장한 뿌듯함과 자신감을 얻을 수 있었다.

필자의 경우에는 운영체제를 듣기 전 수강한 임베디드 시스템과목 덕택에 커널에 대한 기본적인 이해가 밑바탕되어 비교적 프로젝트에 수월하게 적응할 수 있었다. 하지만, 커널 자체에 대한 개념이 없는 3학년 수강생이 수행하기에는 프로젝트 난이도가 지나치게 높다는 문제점이 있는 것 같다. 반대로 임베디드 시스템 역시 난이도가 높은 과목이지만, 커널의 기본기를 배우고 그에 맞춘 실습 및 과제가 진행되기에 적응 할 수 있다. 따라서 학교 현행 규정상 핀토스를 하는 운영체제가 임베디드 시스템과목의 선행 과목이라는 점은 아쉬운 점이라고 생각한다. 임베디드 시스템을 선행으로 듣고, 커널의 기본적인 이해가 밑바탕이 된 상태에서 OS를 수강하며 핀토스 프로젝트를 진행한다면, 반대의 경우보다 수강생 입장에서 훨씬 강력한 시스템 소프트웨어의 이해와 추론 및 구현능력을 갖출 수 있을 것이라 생각한다.