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

**담당 교수** : 박성용 교수님

**학번 / 이름** : 20181202 김수미

**개발 기간** : 2021.10.05 ~ 2021.10.31

1. **개발 목표**

- 사용자가 입력하는 system call 명령을 처리하기 위해 입력받은 명령을Parsing하고 Stack에 저장하는 Argument Passing 기능을 구현한다.

- System call 명령어를 처리하는 system call handler를 구현하여 user mode에서 kernel mode의 서비스를 사용할 수 있는 환경을 구축한다.

- Fibonacci 연산, max\_of\_four\_int 연산 등을 수행하는 새로운 API를 정의하고, 해당 API를 user mode에서 사용할 수 있도록 system call 환경을 구축한다.

- User가 memory 영역을 access할 때 잘못된 영역을 참조하지 않는 것을 보장하도록 기존 Pintos 코드를 수정한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

**1) Argument Passing**

사용자가 입력한 명령어를 읽어 띄어쓰기를 기준으로 parsing한 다음, parsing된 조각들을 하나씩 stack에 저장한다.

**2) User Memory Access**

User가 memory 영역을 access 할 때, kernel 영역을 침범하는 등 잘못된 주소를 참조하는 경우 오류를 발생시켜 pintos 프로그램을 강제로 종료한다.

**3) System Calls**

User가 kernel 수준에서 제공하는 서비스를 이용하기 위해 system call 명령어를 입력했을 때, 해당 명령어는 argument passing에 의해 stack에 저장되므로 system call handler를 통해 stack에 저장된 값들을 읽어 user가 요청한 서비스를 실행시킨다.

* 1. **개발 내용**

1. **Argument Passing**

먼저 User가 입력한 명령어를 strtok\_r 함수를 이용해 parsing한다. 그 다음 process.c내에서 기존 stack 공간을 먼저 초기화한 다음, parsing된 조각들(argv)을 esp(stack pointer)를 이용하여 하나씩 stack에 저장한다. Stack에 저장되는 순서는 과제 명세서에 나와있는 것 처럼, 명령어가 ‘A B C’ 세개의 어절로 이루어진 경우 C부터 가장 high 한 주소값에 저장되며, 점차 주소값을 감소시키며 B, A 순서대로 stack에 저장된다(high to low).

Parsing 된 조각들을 모두 stack에 저장한 다음에는 word alignment를 해줘야 한다. Word alignment란 컴퓨터가 메모리 주소 참조를 편리하게 할 수 있도록 4byte 단위로 데이터를 메모리에 저장하는 작업이다. 즉 특정 데이터를 저장했는데 해당 데이터 byte 크기가 4의 배수가 아닌 경우, 의미 없는 값(null 값)을 메모리에 채워 넣음으로써 이를 4의 배수로 맞춰주는 작업을 의미한다. 입력 받은 전체 argv 데이터의 크기를 계산하고, esp가 가리키는 주소값을 필요한 byte 수 만큼 조정해주면 된다.

그 다음에는 Null Pointer Sentinel(경계를 나누는 역할)과 stack에 저장한 parsing조각들의 주소를 stack에 저장하고, 마지막으로 parsing조각들의 총 개수(argc)와 return address까지 stack에 저장하면 Argument Passing이 완료된다.

1. **User Memory Access**

User program이 invalid pointer를 pass해주게 되면 invalid memory access가 발생할 수 있다. 이 때 invalid pointer란 open(NULL)과 같은 NULL 포인터, 또는 unmapped virtual memory, 또는 kernel address space로의 pointer 등을 의미한다. 이런 경우 exception handling이 될 수 있도록 코드를 작성해 주어야 한다.

Pintos는 이러한 문제 상황을 검사할 수 있게 해주는 유용한 함수를 몇가지 제공한다. 먼저 NULL 포인터를 전달받은 경우 해당 pointer의 null 여부를 조사함으로써 쉽게 검사할 수 있으며, unmapped Virtual memory는 pagedir\_get\_page 함수를 이용하여, kernel address space 참조 여부는 is\_user\_vaddr, is\_kernel\_vaddr 함수를 이용하여 검사할 수 있다.

1. **System Calls**

Pintos 외에도 거의 모든 운영체제들은 User mode와 Kernel mode로 분리되어 있기 때문에, 사용자는 Kernel mode에서 제공하는 다양한 서비스를 직접 사용할 수는 없고, ‘system call’이라는 것을 사용해 Kernel mode에 서비스 사용을 요청해야 한다. 즉 system call은 User mode와 Kernel mode를 연결하는 매개체 역할을 한다.

Standard input/output, 파일 open/read/write 등 우리가 매우 자주 사용하는 기능들 역시 모두 kernel 차원에서 제공하는 서비스이기 때문에, system call은 컴퓨터를 사용함에 있어서 아주 중요한 부분이라고 할 수 있다.

이번 프로젝트에서 개발할 system call 의 종류는 아래와 같다.  
이때 fibonacci 와 max\_of\_four\_int system call API는 Pintos kermel에서 기본적으로 제공하는 기능이 아니며, 추가적으로 구현해 준 것이다.

1. **halt( )** : shutdown\_power\_off( )함수를 이용하여 Pintos를 종료시킨다.
2. **exit( )** : 현재 user program을 종료시키고 해당 status를 kernel로 return 한다.  
   이 때 exit(0)으로 종료된 경우 아무런 에러 없이 성공적으로 종료되었음을 의미하며, eixt(0) 이외의 값으로 exit되는 경우(exit(1), exit(2)등) 이는 에러에 의해 program이 종료되었음을 의미한다.
3. **exec( )** : Child process를 create 하고, 새롭게 crete된 process의 id를 return 한다. Process\_execute( ) 함수를 참조하여 구성할 수 있다.
4. **wait( )** : Child process가 작업을 끝낼 때 까지 wait하는 기능을 수행한다. Child thread ID가 valid 한지를 검사하고, child thread가 종료될 때 child의 exit status를 return 받는다. Parent process의 wait( )가 child process로부터 status 값을 return 받기 전에 terminate 되는 것을 방지하기 위해 semaphore를 사용했다.
5. **read( )** : Standard input read 기능을 수행하는 system call 이다. STDIN의 file descriptor는 0이며, 실제로 read 한 데이터의 byte 값을 return한다. input\_getc(void) 함수를 사용하면 구현하기 쉽다.
6. **write( )** : Standard output write 기능을 수행하는 system call 이다. STDOUT의 file descriptor는 1이며, 실제로 write 한 데이터의 byte 값을 return 한다. putbuf( ) 함수를 사용하면 구현하기 쉽다.
7. **fibonacci( )** : 정수 n을 입력받으며 fibonnaci 수열의 n번째 값을 return 한다.
8. **max\_of\_four\_int( )** : 4개의 정수값을 입력 받으며, 4개 중 가장 큰 값을 return 한다.

User Level에서 System call API를 호출하면 먼저 하드웨어는 mode bit를 바꿔 User mode에서 Kernel mode로 전환한다. System call 명령은 하드웨어 입장에서 interrupt 이므로(SW interrupt) 커널 내부의 interrupt handler에서 sys\_call interrupt를 처리하게 된다. system call 명령어와 함께 입력된 system call number를 system call table에서 참조하여 User가 어떤 kernel 서비스를 system call로 요청 했는지를 확인하고, 해당 system call function의 주소에 저장되어 있는 서비스를 실행한다. 서비스의 실행이 모두 완료된 후에는 다시 mode bit를 User mode로 전환함으로써 system call을 호출한 User level로 돌아간다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

**10/5 ~ 10/12** : 과제 내용 파악 및 Pintos Manual 숙지, 개발 계획 수립

**10/13~10/26** : (중간고사 기간)

**10/27~10/29** : Argument Passing, User Memory Access, System Call 기능 구현을 위한 코드 작성

**10/30~10/31** : 디버깅 및 코드 간소화 작업 + 보고서 작성

* 1. **개발 방법**

과제 명세서에 기반하여 전체적인 구현 계획을 간략하게 작성하였다.

**1) Argument Passing**

pintors/src/userprog/process.c 의 load 함수 내부에 Argument Passing 기능을 수행하는 코드를 작성하면 된다. load 함수는 const char \*file\_name 과 void \*\*esp 변수를 parameter로 가지며, file\_name 문자열은 User가 입력한 명령어를, esp는 stack pointer를 저장하기 때문에 file\_name 변수가 저장하고 있는 문자열을 parsing하여 esp가 pointing하는 stack의 메모리 주소에 저장하는 코드를 작성하면 된다.

**2) User Memory Access**

pintors/src/userprog/syscall.c, pintors/src/userprog/process.c,  
pintors/src/userprog/exception.c 의 코드를 수정하여 user memory access 관련 기능을 구현하면 된다.  
syscall.c 에서는 system call 요청하는 계산을 수행하는 코드들을 포함하고 있으므로 exit, exec, wait 명령어에 대해 넘겨주는 값이 valid한지 검사를 한다. 이 때 is\_user\_vaddr 함수를 사용해주면 된다.  
process.c 에서는 process\_execute 함수 내부에서 filesys\_open 함수의 return 값이 NULL인지 여부를 확인하면 된다.  
exception.c 에서는 page\_fault 함수에서 is\_kernel\_vaddr 함수를 사용하면 된다.

**3) System Calls**

pintors/src/userprog/syscall.c에서 system call 기능을 구현하면 된다. syscall\_handler 함수를 생성하여 그 안에서 switch문을 이용하면 된다. System call number는 intr\_frame \*f 변수를 참조하여 확인할 수 있다. 각 명령어에 맞는 함수를 호출하여 system call을 수행한다. 이때 esp를 참조하여 stack에 저장되어 있는 parsing된 명령어를 읽어오면 되고, 명령 수행에 결과값이 발생하는 경우 이를 eax에 저장하면 된다. pintors/src/lib/syscall\_nr.h에서 system call의 종류를 확인할 수 있다.

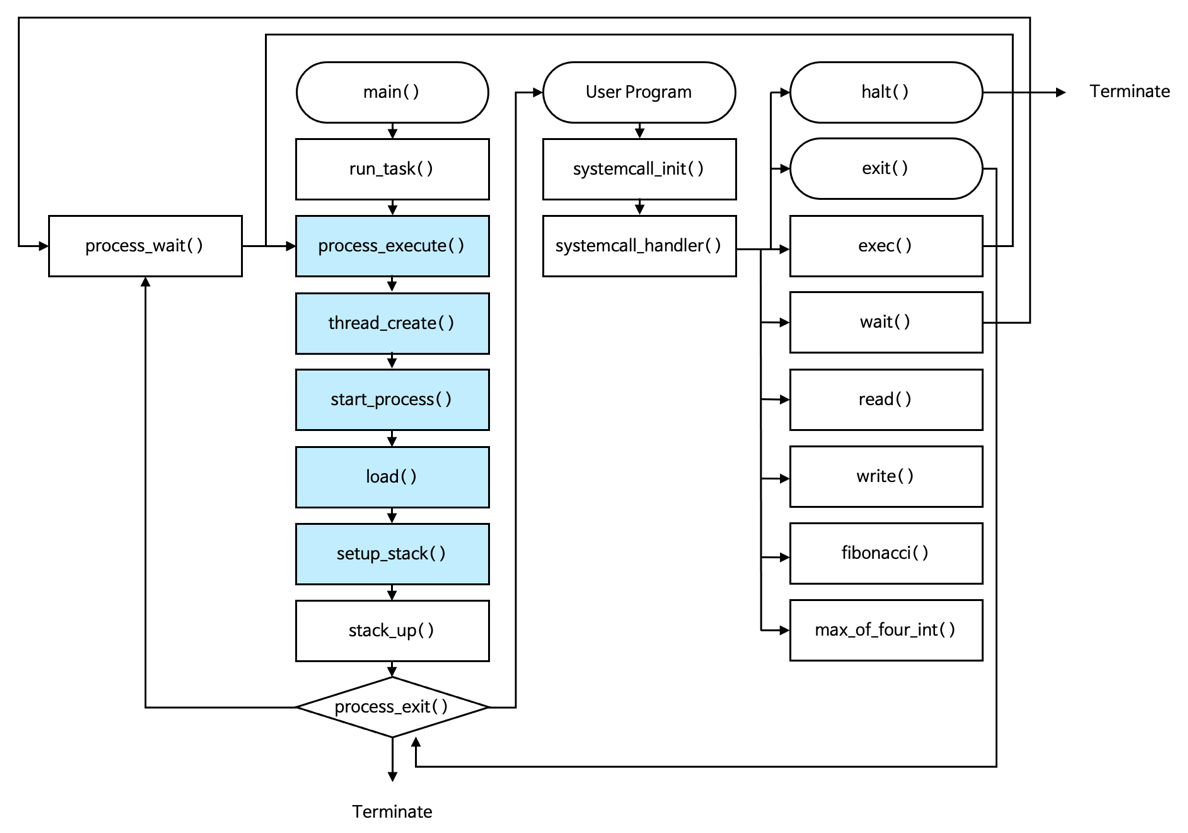
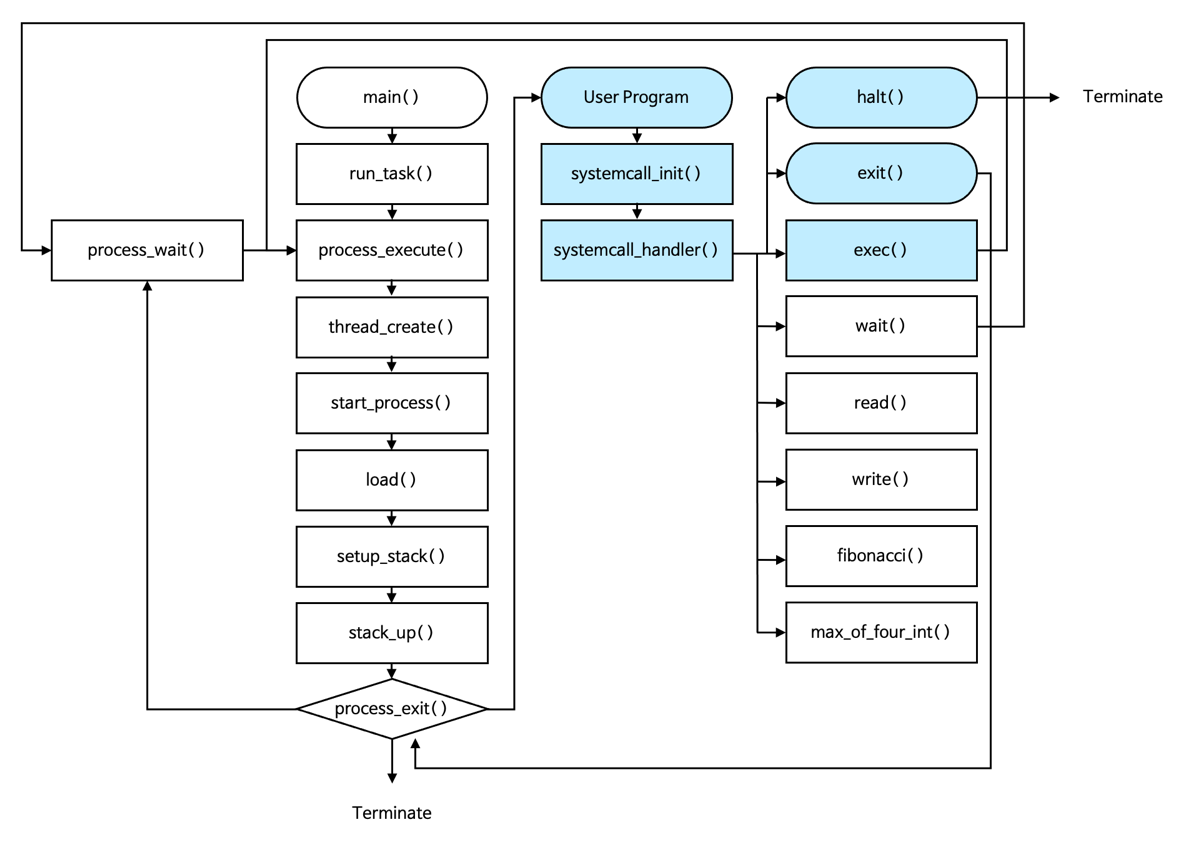
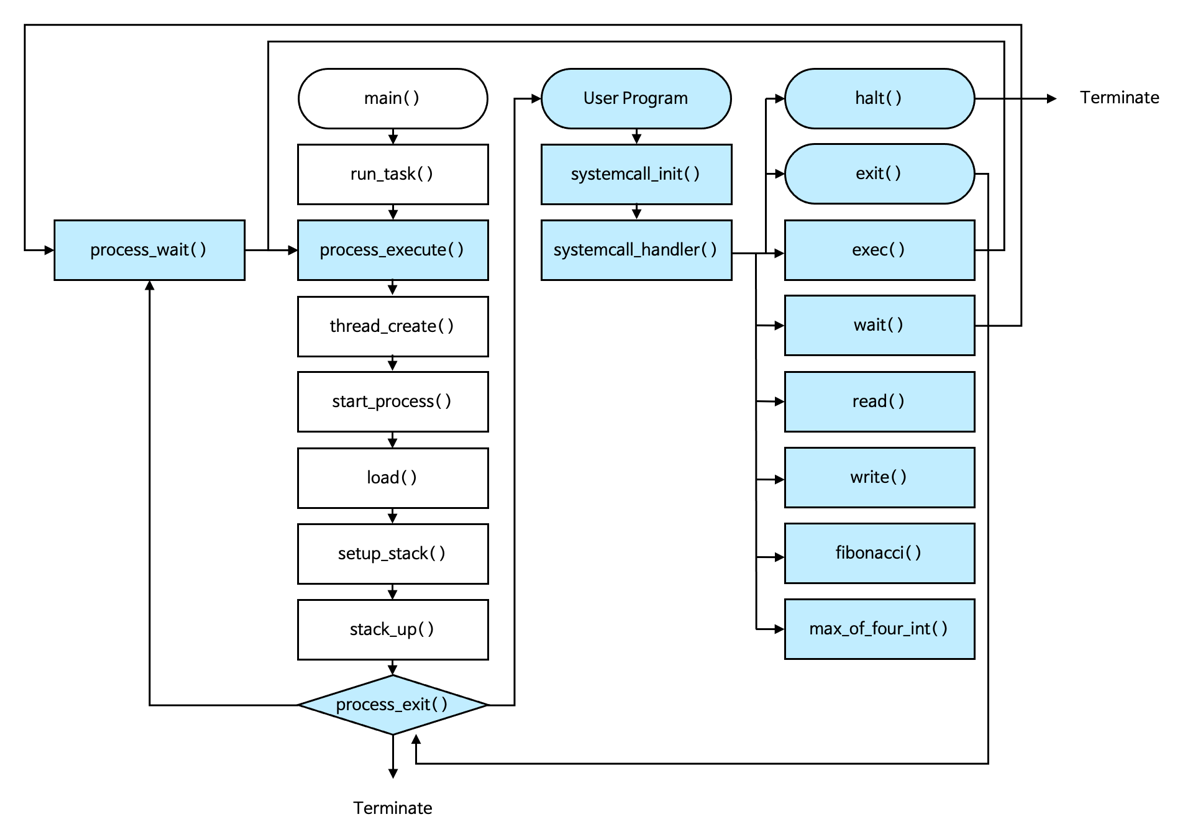
**4) 기타**

이 외에도 코드를 작성하다 보면 추가적으로 구현해 주어야 할 부분들이 있다.

먼저 process\_wait 부분이 구현되어 있지 않기 때문에  
pintors/src/userprog/process.c, pintors/src/threads/thead.c 에서 이를 구현해 주어야 한다.

이를 위해 pintors/src/threads/synch.h 에 선언되어 있는 semaphore 구조체를 활용할 수 있다. process.c에서는 process wait, process exit 사이에서 semaphore를 이용해 parent는 child의 exit를 기다리고 child의 는 parent가 remove를 할 때까지 기다리도록 wait 기능을 구현해 줄 수 있다.

또한 fibonnaci와 max\_of\_four\_int system call을 구현하기 위해  
pintos/src/lib/user의 syscall\_nr.h, syscall.c 코드를 수정해주면 된다.  
먼저 4개의 인수를 받아서 처리하는 syscall4가 없기 때문에 이를 syscall.c에 구현해 주어야 하며 fibonacci( ), max\_of\_four\_int( ) 함수도 새로 구현해준다. 위 3번의 System calls에서 서술한 switch문에도 이 두가지를 추가해주면 된다. 또한 계산 수행 결과 출력을 위해 pintos/src/example 폴더 내부에 additional.c 파일을 만든 다음 실행 파일 생성을 위해 Makefile에 additional.c 관련 부분을 추가해주면 된다.

1. **연구 결과**
   1. **Flow Chart**
2. **Argument Passing**
3. **User Memory Access**
4. **System Calls**
   1. **제작 내용 (자세한 내용은 코드 주석 참조)**

**1) Argument Passing**

1. pintos/src/userprog/process.c

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명  
텍스트이(가) 표시된 사진

자동 생성된 설명

먼저 process.c 의 load 함수 내부에 argument passing 부분을 구현한 것이다. User가 입력한 명령어를 strtok\_r 함수를 이용하여 parsing 한 뒤 esp를 이용해 stack에 하나씩 저장한다. 저장하는 과정에서는 word alignment와 Null Pointer Sentinel 역시 빠트리지 않고 구현해 두었다.

1. pintos/src/userprog/process.h

텍스트이(가) 표시된 사진

자동 생성된 설명

process.c 에서 새로 만든 stack\_up 함수를 process.h 헤더파일에도 추가해 주었다.

**2) User Memory Access**

1. pintos/src/userprog/process.c

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

NULL 포인터의 전달로 invalid memory access가 발생하지 않는지 체크하는 부분.

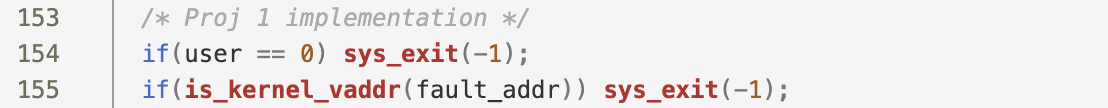
1. pintos/src/userprog/syscall.c

텍스트이(가) 표시된 사진

자동 생성된 설명

is\_user\_vaddr 함수를 사용하여 exit, exec, wait system call이 실행될 때 user virtual address의 침범이 일어나지 않는지 check 하는 부분이다.

1. pintos/src/userprog/exception.c



is\_kernel\_caddr 함수를 사용하여 kernel virtual address의 침범이 일어나지 않는지 , 혹은 user가 0은 아닌지를 check 하는 부분이다. 해당 함수는 현재 fault\_addr이 kernel 공간을 가리키고 있는지를 검사한다.

**3) System Calls**

1. pintos/src/userprog/syscall.c

테이블이(가) 표시된 사진

자동 생성된 설명  
텍스트이(가) 표시된 사진

자동 생성된 설명  
텍스트이(가) 표시된 사진

자동 생성된 설명  
텍스트이(가) 표시된 사진

자동 생성된 설명

system call 기능을 구현한 부분이다. stack에 저장되어 있는 명령어 값들을 읽어들여 syscall number를 확인하고, 해당 number에 해당하는 서비스를 실행시켜주는 역할을 한다. 추가적으로 구현한 API인 fibonacci와 max\_of\_four\_int system call도 동일하게 구현되어 있다.

1. pintos/src/userprog/syscall.h

텍스트이(가) 표시된 사진

자동 생성된 설명

syscall.c 에서 필요에 따라 작성한 함수는 syscall.h 헤더파일에도 추가해 주었다.

1. pintos/src/threads/thread.h

텍스트이(가) 표시된 사진

자동 생성된 설명

여기서부터는 wait와 signal을 다루는 부분이다. 먼저 process가 wait, exit 등을 할 수 있도록 만들기 위해 thread구조체에 semaphore 부분을 추가해 주었다. child\_list는 child thread 이고 child\_semaph 와 memory\_semaph는 각각 child와 memory 상태에 대한 lock을 수행하는 semaphore 구조체 변수이다. child\_element는 chiild의 elem 값이고 exit\_stat는 exit 당시의 status 값을 저장하는 변수이다.

1. pintos/src/threads/thread.c

텍스트이(가) 표시된 사진

자동 생성된 설명

thread 구조체 변수를 수정했기 때문에 이를 사용하기 전에 먼저 init 작업이 필요하다. 따라서 init\_thread 함수 내부에서 구조체 내부 필드 변수들을 초기화해주었고, child\_element는 running\_thread의 child\_list에 list\_push\_back 해주었다.

1. pintos/src/userprog/process.c

텍스트이(가) 표시된 사진

자동 생성된 설명

process\_wait을 구현한 부분이다. parent process는 child\_process가 죽을 때까지 wait 해야 하고, 죽으면 child process의 exit\_status를 return받게 된다. 따라서 while문을 사용하여 child\_semaph의 sema\_down을 통해 child process가 종료할 때 까지 wait 하게 되고, meory\_semaph의 sema\_up을 통해 메모리를 remove 전까지 남겨둔다.  
텍스트이(가) 표시된 사진

자동 생성된 설명

process\_exit 에서는 semaphore를 process\_wait과 반대로 적용해 주면 된다.  
즉 child\_semaph를 sema\_up 해주고, memory\_semaph를 각각 sema\_down 해주면 된다.

**4) Additional System calls**

1. pintos/src/lib/syscall\_nr.h

텍스트이(가) 표시된 사진

자동 생성된 설명

syscal\_nr.h 에 fibonacci와 max\_of\_four\_int에 대한 system call enum을  
추가해 주었다.

1. pintos/src/lib/user/syscall.c

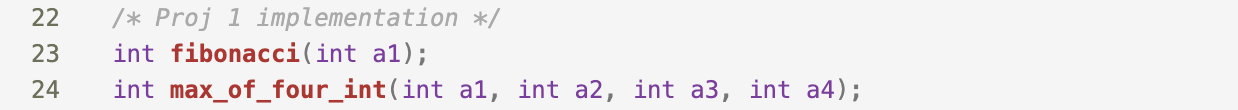
텍스트이(가) 표시된 사진

자동 생성된 설명  
텍스트이(가) 표시된 사진

자동 생성된 설명

4개의 인자를 받아 system call을 수행하는 syscall4 매크로 와 fibonacci, max\_of\_four\_int의 system call을 수행하는 함수를 추가해 주었다.

1. pintos/src/lib/user/syscall.h



syscall.c 에서 새롭게 추가한 함수는 syscall.h 헤더파일에도 추가해 주었다.

1. pintos/src/userprog/syscall.c

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

이미 위에서 한번 언급한 적 있는 부분이다. userprog/syscall.c 부분에 함수의 계산 기능을 수행하는 함수를 만들었고, system call number를 받아 이에 해당하는 fucntion을 call 하는 switch문에도 fibonacci와 max\_of\_four\_int 부분을 추가해 주었다.

1. pintos/src/userprog/syscall.h

텍스트이(가) 표시된 사진

자동 생성된 설명

fibonacci와 max4에 대해 새롭게 추가한 함수는 syscall.h 헤더파일에도 추가했다.

1. pintos/src/examples/additional.c

텍스트이(가) 표시된 사진

자동 생성된 설명

추가적으로 구현한 두가지 API의 실행 결과를 print하는 additional.c 파일을 example 디렉토리 안에 구현해 주었다.

1. pintos/src/examples/Makefile

텍스트이(가) 표시된 사진

자동 생성된 설명

example 디렉토리 안에서 additional.c 파일을 새로 추가했기 때문에, example 디렉토리의 Makefile이 additional.c에 대한 실행파일도 생성할 수 있도록 내용을 수정해 주었다.

* 1. **시험 및 평가 내용**



pintos/src/userprog 에서 make check 를 실행한 결과의 일부와

pintos –filesys-size=2 -p ../examples/additional -a additional -- -f -q run ‘additional 10 20 62 40’ 을 실행한 결과 화면을 캡처한 것이다.

과제에서 요구하는 21개의 test를 모두 pass 하며 additional 10 20 62 40의 입력에 대해 55(피보나치 수열 10번째 값), 62(4개 정수 10, 20, 62, 40 중 최댓값)의 결과값을 잘 출력하는 것을 확인할 수 있다.