**Pintos Project 2: User Program (2)**

담당 교수 / 분반 : 박성용 / 1분반

이름 / 학번 : 20171665 / 이선호

개발 기간 : 2022.10.05 ~ 2022.10.26

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

**이번 프로젝트에서는 직전 프로젝트에서 구현하지 못했던 system call handler에서 파일과 관련한 system call을 구현하는 것이 목표이다. 특히, 파일 시스템 구현에 있어서 발생할 수 있는 synchronization 문제를 해결하고, 실행 중인 process가 critical section에 진입했을 때 다른 어떠한 process도 critical section에 진입하지 못하도록 하는 protection을 구현한다.**

1. **개발 범위 및 내용**
   1. **개발 범위**

* 아래 각 항목을 구현해야 하는 이유, 혹은 구현 시 기대되는 결과를 간략히 서술

1. File Descriptor

각 프로세스마다 파일을 열거나 닫을 때 file descriptor를 사용하여 원하는 파일에 접근하여 내용을 읽거나 쓰는데, 이를 위해 thread의 멤버 변수로 file descriptor에 관한 배열을 선언해야 한다.

2. (이번 프로젝트에서 구현해야 하는) System Calls

User program에서 파일과 관련된 system call인 syscall\_create, syscall\_remove, syscall\_open, syscall\_filesize, syscall\_read, syscall\_write, syscall\_seek, syscall\_tell, syscall\_close를 모두 구현하는 것이 목표이다. 파일을 열거나 닫고, 읽거나 쓸 수 있는 기능을 구현해야 하며, 이외 파일의 크기를 구하거나 파일 포인터를 반환하는 부가적인 기능의 구현도 요구된다.

3. Synchronization in Filesystem

어떤 한 프로세스가 파일을 읽거나 쓸 때, 다른 프로세스도 동시에 쓰거나 읽으면 예상치 못한 문제가 발생할 수 있는데, 이처럼 서로 다른 프로세스가 공유하는 데이터에 접근하는 코드 부분인 critical section에 진입해서 발생할 수 있는 문제를 lock와 semaphore를 통해 해결해야 한다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

Pintos의 매뉴얼에 따라서 file descriptor를 최대 128개를 할당할 수 있도록 128개의 file 구조체의 타입을 지니는 files 배열을 thread의 멤버 변수로 선언했다. 배열을 동적할당으로 관리하면 3부터 128까지의 범위에서 중간에 있는 임의의 file descriptor를 free 했을 때 메모리의 관리가 어려워지므로, 원소 개수의 상한선이 정해져 있는 조건을 감안하여 정적 배열을 선언하여 관리하는 것으로 결정했다.

2. System Calls: 구현할 각 system call에 대해 간략히 서술 (하나의 system call 당 최대 3문장으로 간략히 설명; 3문장을 넘길 정도로 길게 작성하지 말 것)

syscall\_create: 파일 이름과 파일의 초기 크기를 인자로 받아서 pintos에서 기존에 구현된 filesys\_create를 통해 파일을 생성하며, 파일 생성이 성공하면 true, 실패하면 false를 반환한다.

syscall\_remove: 파일 이름을 인자로 받아서 파일의 존재 여부와 관계없이 파일 삭제를 수행하는데, 파일 삭제가 성공하면 true, 실패하면 false를 반환한다.

syscall\_open: 파일 이름을 인자로 받아서 해당 이름을 가진 파일을 열고 새로운 file descriptor에 지정해주며, 성공하면 지정한 file descriptor를 반환하고, 실패하면 -1을 반환한다.

syscall\_filesize: file descriptor를 인자로 받아서 해당 file descriptor에 대응되는 파일의 크기를 pintos에 정의된 file\_length의 함수를 사용하여 반환한다.

syscall\_read: file descriptor와 파일의 내용을 읽어 들일 buffer, 그리고 읽을 파일의 데이터 길이를 인자로 받아서 standard input와 같은 file descriptor를 받으면 input\_getc 함수를 사용하고, 그렇지 않으면 file\_read 함수를 사용하여 읽은 데이터의 바이트 수를 반환한다.

syscall\_write: file descriptor와 파일의 내용을 쓸 buffer, 그리고 쓸 파일의 데이터 길이를 인자로 받아서 standard output와 같은 file descriptor를 받으면 input\_getc 함수를 사용하고, 그렇지 않으면 file\_write 함수를 사용하여 쓴 데이터의 바이트 수를 반환한다.

syscall\_seek: file descriptor와 position을 인자로 받아서 파일의 포인터인 다음으로 읽고 쓸 위치를 position으로 받은 위치로 변경한다.

syscall\_tell: file descriptor를 인자로 받아서 해당 file descriptor에 대응되는 파일의 다음으로 읽거나 쓸 위치를 반환한다.

syscall\_close: 파일 이름을 인자로 받아서 해당 이름을 가진 파일을 닫는데, 파일 이름이 없으면 system call을 종료한다.

3. Synchronization in Filesystem: Lock, Semaphore를 어떻게 이용할 수 있는지 각각에 대해 설명 (다른 방법을 서술해도 되지만 lock과 semaphore는 반드시 포함해야 함)

Lock과 semaphore은 모두 mutual exclusion을 해결하는 방법으로 쓰이는 수단이며, 이를사용하여 이번 파일 시스템에서 어떤 한 프로세스가 파일을 읽거나 쓰고 있을 때 다른 프로세스가 그 파일을 쓰거나 읽으려고 하면 발생할 수 있는 문제를 해결해야 한다.

이를 위해 filesys\_lock라는 lock을 사용하여 여러 프로세스가 동시에 파일에 접근하는 현상을 막고자 했다. 먼저 system call을 초기화하는 syscall\_init 부분에서 lock 변수인 filesys\_lock을 초기화 해준다. 현재 프로세스가 파일을 읽거나 쓸 때 다른 프로세스가 파일에 접근하지 못하도록 Lock\_acquire 함수를 통해 현재 프로세스가 lock에 대한 권한을 갖도록 하고, 요청한 작업을 완료하면 lock\_release 함수를 통해 현재 프로세스가 lock을 스스로 해제함으로써 다른 프로세스도 해당 파일을 읽거나 쓸 수 있도록 한다.

또한 부모 프로세스가 자식 프로세스를 직접 reaping하지 못하고 먼저 종료되는 현상을 막기 위해 자식 프로세스가 process\_exit할 때는 부모 process가 자기 자신을 알아차리는 동시에 부모 프로세스가 자기 자신을 해제하기 전 다른 프로세스가 메모리에 접근하는 것을 막기 위해 두 개의 semaphore를 각각 1만큼 올려주고 내려준다. 또한 부모 process가 자식 프로세스가 다 실행되었는지를 확인하며 모두 실행되었을 때는 자식 process의 실행에 관한 semaphore를 1만큼 내려준 후 프로세스 리스트에서 제거하고 다시 메모리에 대한 semaphore를 1만큼 올려줘서 다른 프로세스가 해제된 자식 프로세스의 메모리 영역을 사용할 수 있도록 한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성
* 10/04 ~ 10/08: 구현에 필요한 개념과 userprogram 작동 flow를 이해하고, 기존에 구현하지 못한 파일 시스템과 관련한 system call을 추가한다.
* 10/09 ~ 10/12: synchronization 문제 해결 코드를 추가하고, 해결되지 않은 테스트 케이스에 관해 점검한다.
* 10/13 ~ 10/16: 부모 process와 자식 process 간의 semaphore 관리 문제를 해결한다.
* 10/16 ~ 10/26: 보고서를 작성하여 마무리하고, 테스트 케이스 외의 에러가 없는지 점검한다.
  1. **개발 방법**
* II. B.의 개발 내용을 구현하기 위해 각각에 대해 다음 사항들을 포함하여 설명
  + 수정해야하는 소스코드
  + 수정하거나 추가해야 하는 자료구조
  + 수정하거나 추가해야 하는 함수

파일과 관련한 system call 추가는 syscall.c에서 진행되었으며, 파일 시스템의 file descriptor 구현을 위해 threads의 thread.h와 thread.c 파일을 수정한다. thread.h에는 file 구조체 타입의 128개의 원소를 가지는 files 배열을 선언하고, 마지막으로 지정한 file descriptor의 번호를 저장하는 변수 fd를 선언한다. thread.c에서는 standard input, standard output, standard error를 제외한 3부터 127까지의 file descriptor number에 대응되는 파일을 null로 초기화하는 작업을 한다.

thread에서 file descriptor를 관리하는 배열을 선언하면서 pintos에 이미 정의된 file 구조체를 import 해 올 수 있도록 기존에 file.c에 있던 file 구조체를 file.h로 옮긴다.

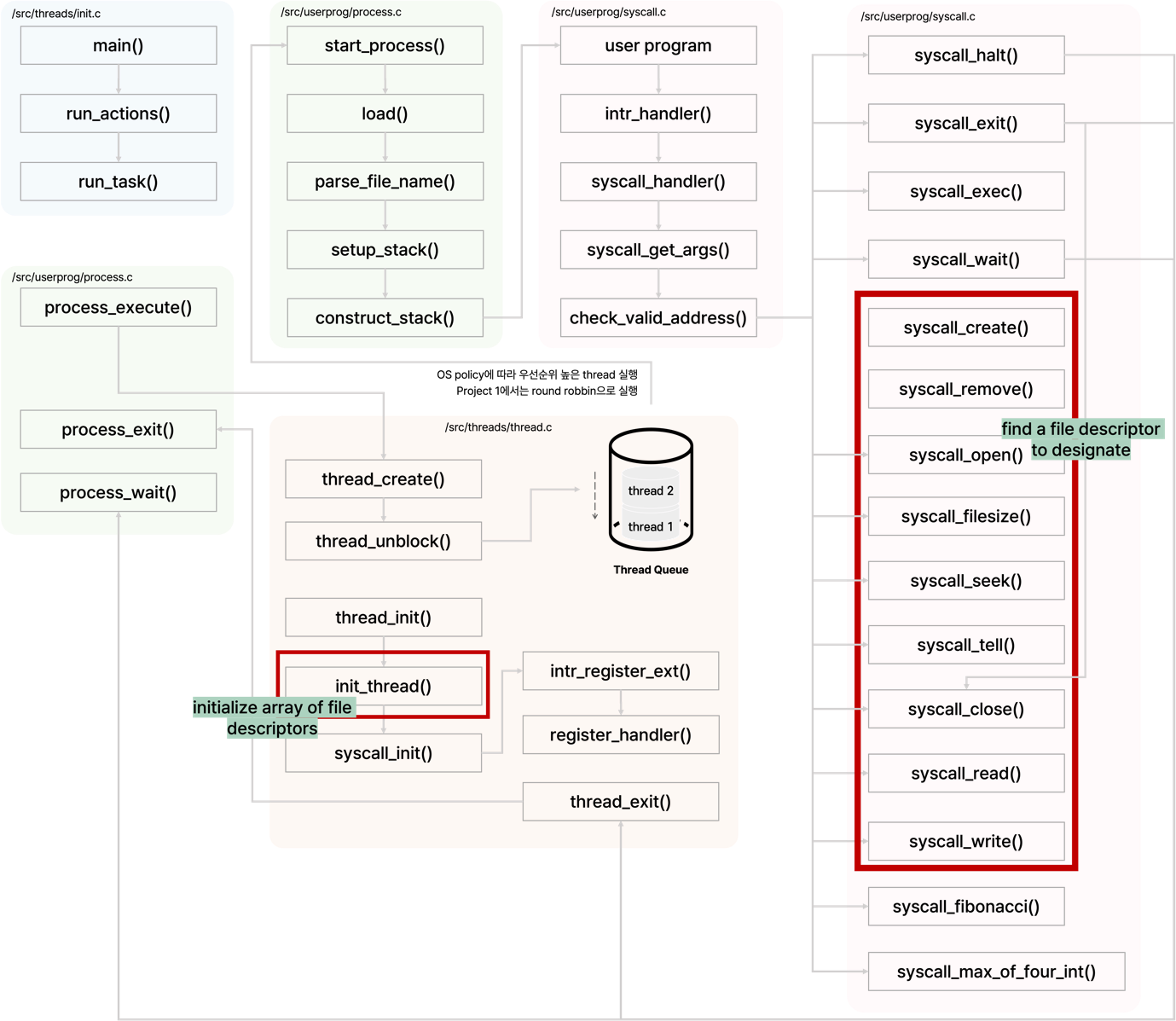
앞서 설명한 system call을 기능별로 구현하기 위해 syscall\_create, syscall\_remove, syscall\_open, syscall\_filesize, syscall\_read, syscall\_write, syscall\_seek, syscall\_tell, syscall\_close 함수를 추가했으며, syscall\_handler에서 interrupt table의 인덱스에 따라서 switch문을 통해 대응되는 system call을 실행한다.

파일을 쓰거나 읽을 때, 그리고 파일을 닫을 때 현재 프로세스가 사용하는 파일을 다른 프로세스가 접근하거나 사용할 수 없도록 lock\_acquire와 lock\_release 함수를 사용하여 ciritical section으로 여러 개의 프로세스가 진입하는 문제를 막는다.

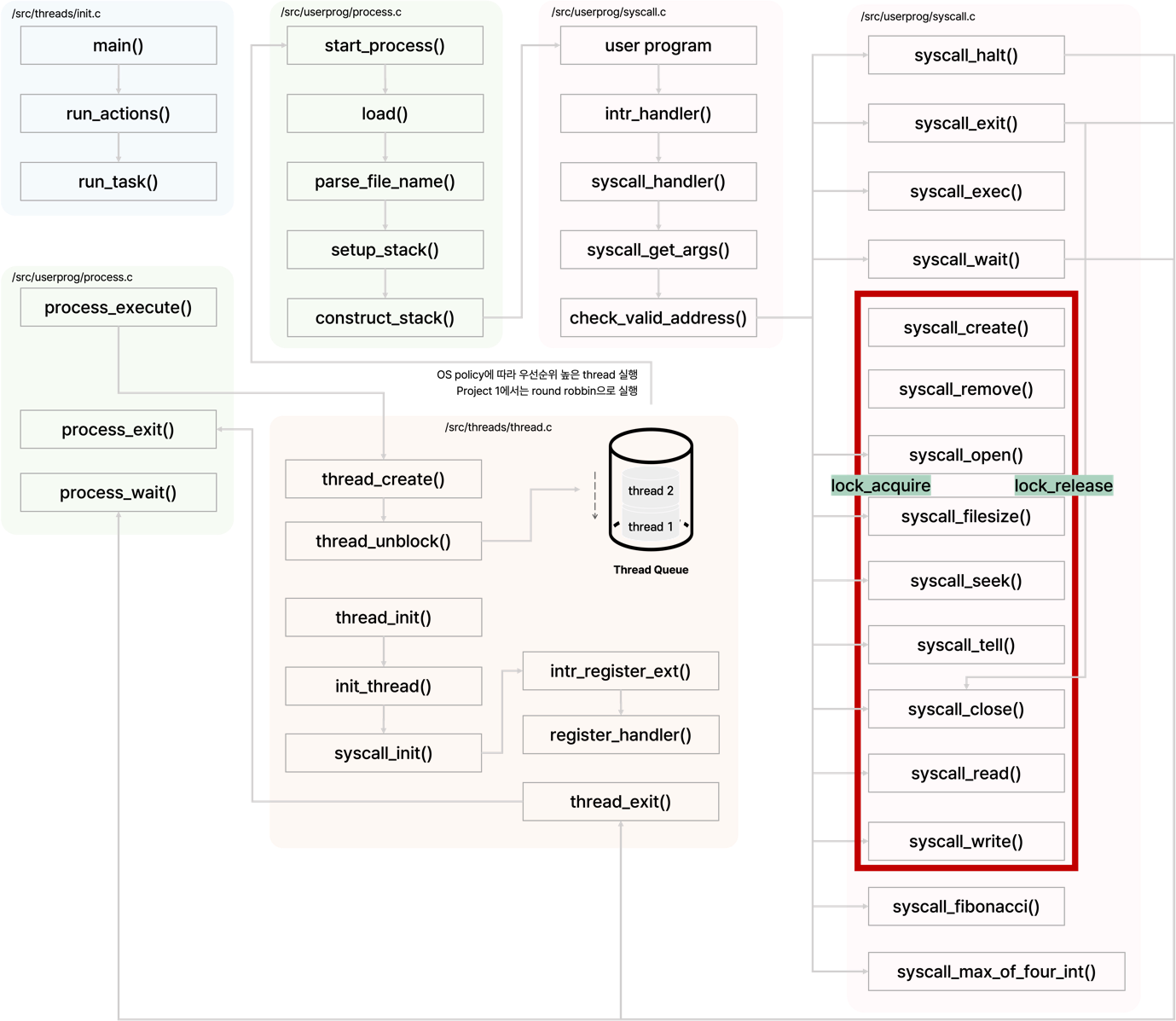
부모 프로세스가 자식 프로세스를 reaping 하는 부분을 구현하기 위해 process\_wait와 process\_exit에서 sema\_up과 sema\_down 함수를 사용해 synchronization을 구현한다.

1. **연구 결과**
   1. **Flow Chart**

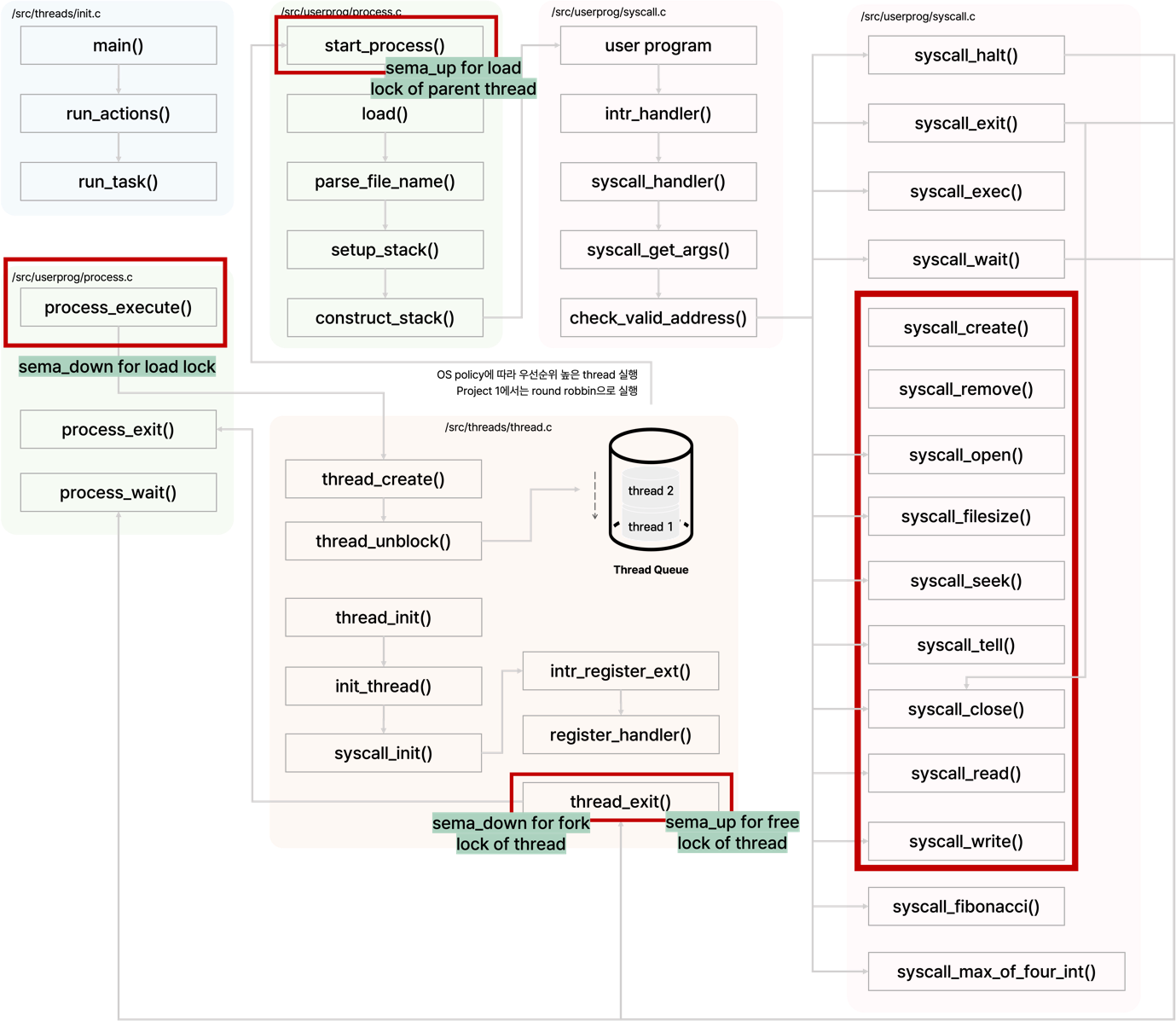
* II. B. 개발 내용의 각 3가지 항목에 대하여 Flow Chart 작성
* 1) File Descriptor

****

* 2) System Call

****

* 3) Synchronization

****

* 1. **제작 내용**
* II. B. 개발 내용의 각 3가지 항목에 대하여 실직적으로 구현한 코드의 관점에서 작성 (구현 내용, 알고리즘 등을 명확히 서술할 것)
* 구현에 있어 Pintos에 내장된 라이브러리나 자체 제작한 함수를 사용한 경우 이에 대해서도 설명
* 개발 중 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결한 방식에 대해 설명
* syscall\_open에서는 인자로 온 파일 이름을 가지고 파일을 열어주는데, 파일을 여는 데 성공하면 앞서 정의한 thread의 fd 변수에 1을 더한 값으로 file descriptor를 대응시켜준다. 단, fd에 대응되는 file descriptor가 이미 사용 중이거나 file fd가 127을 넘어가면 thread가 지니고 있는 file descriptor의 배열을 하나씩 탐색하며 비어 있는 file descriptor를 확인하여 지정해준다. 만약 모두 다 사용 중이면 -1을 반환한다.
* 파일을 open하는 syscall\_open에서는 매번 3부터 127까지의 모든 file descriptor를 탐색하면서 비어 있는 file descriptor를 찾는 것은 비효율적이다. 그래서 가장 마지막에 지정한 file descriptor를 저장하여 다음에 새로운 파일을 open 할 때는 이전에 저장한 file descriptor에서 1을 더한 값을 새로운 file descriptor로 지정한다. 대신에 1을 더한 값이 file descriptor의 최댓값을 넘어가거나 3 미만이면 모든 file descriptor를 for loop으로 탐색하면서 비어 있는 가장 첫 번째 file descriptor를 찾아서 반환해준다. 만약 비어 있는 file descriptor가 없으면 -1을 반환한다. 대신에 file open 중에 다른 프로세스가 해당 파일 리소스에 접근하려고 하면 mutual exclusion을 지키지 못하므로 파일을 열기 전에 lock\_acquire 함수로 lock 권한을 얻고, file descriptor를 반환하기 직전에 lock\_release 함수로 lock을 해제하여 다른 프로세스에게 알린다.
* 파일을 close하는 syscall\_close에서는 인자로 들어온 file descriptor 값을 pintos에 내장된 flie\_close 함수의 인자로 넘기고, file 구조체 배열의 index의 내용을 null로 초기화한다. 만약 현재 마지막으로 열었던 file descriptor이면 이를 1만큼 감소시킨다.
* 파일을 읽는 syscall\_read에서도 synchronization을 지키기 위해 lock\_acquire 함수를 통해서 lock 권한을 얻고, buffer의 메모리 주소가 유효한지를 검사하는 check\_valid\_address 함수를 통해 buffer의 내용을 읽어도 되는지 확인한다. 인자로 받은 file descriptor 값이 0이면 standard input이므로 input\_getc를 통해서 buffer에 있는 내용을 단순히 읽고, file descriptor 값이 3 이상 128 미만이면 file\_read 함수를 통해 buffer에 있는 내용을 읽는다. 읽는 작업이 다 끝나면 lock\_release를 한다.
* 파일을 쓰는 syscall\_write에서도 synchronization과 유효한 주소인지를 검사하는 작업은 동일하다. 인자로 받은 file descriptor 값이 1이면 standard output이므로 putbuf 함수를 통해 buffer에 file descriptor로 지정된 파일의 내용을 읽고, file descriptor 값이 3 이상 128 미만이면 file\_write 함수를 통해 buffer에 내용을 작성한다. 단, 현재 쓰려고 하는 파일의 deny\_write boolean 값이 true이면 file\_deny\_write 함수를 통해 파일 작성을 거부한다.
* syscall\_tell은 file\_tell 함수를, syscall\_seek은 file\_seek 함수를 실행한다. 단, 인자로 온 file descriptor가 가리키는 파일이 null이면 syscall\_exit을 통해 잘못된 요청임을 알린다.
* syscall\_create와 syscall\_remove에서는 인자로 넘어 온 파일 이름이 null이면 syscall\_exit을 통해 잘못된 요청임을 알리고, 그렇지 않으면 각각 filesys\_create와 filesys\_remove 함수를 실행한다.
* 부모 프로세스의 실행이 자식 프로세스보다 빨리 끝나서 자식 프로세스의 load를 실패했을 때, 죽어버리는 자식 프로세스에 관해 -1을 출력하지 못하는 문제를 해결해야 했다. 그래서 process\_execute로 프로세스를 실행할 때 메모리 load에 관한 semaphore를 1만큼 감소시키고, start\_process 함수에서 자식 프로세스 기준으로 load가 잘 진행된 이후이므로 부모 프로세스의 메모리 load에 관한 semaphore를 1만큼 올려주는 작업을 구현했다.
  1. **시험 및 평가 내용**
* make check 수행 결과를 캡처하여 첨부

텍스트이(가) 표시된 사진

자동 생성된 설명