운영체제 project #1 report

- ## 학번: 2018044766
- ## 학과: 소프트웨어학부
- ## 이름: 안 요 한
- ## 목차

1.프로젝트 개요

이번 프로젝트의 목표는 프로세스간의 통신을 이용한 간단한 쉘의 구현으로서

사용자는 예외적인 명령어 처리는 하지 않는다.

2.프로젝트 이론

프로젝트를 진행하기전 먼저 이해하고 넘어 가야 할 이론들이 있었다.

가장 먼저 프로세스의 개념부터 짚고 넘어가보도록 한다.

프로세스 란 , 어떠한 프로그램이 주 기억장치에 적재되어 cpu에 의해서 실행과정에 올라간 상태를 말한다. 우리가 흔히 사용하는 맥북 터미널이라든가 하는 모든 프로그램들이 프로세스 상태로 존재 하는 것이다.

본인이 이번 프로젝트를 진행하기 위해서는 프로세스의 생성과 소멸 과정에 대한 이해가 필요 하였다. 프로세스는 실행 되는 동안 여러 개의 새로운 프로세스를 생성할 수 있으며, 이 과정에서 생성하는 프로세스를 '부모프로세스' 그로 인하여 만들어진 새로운 프로세스를 '자식프로세스' 라고 한다. 이러한 프로세스 생성과정은 거대한 프로세스 트리를 형성하며 각각의 프로세스 들은 **PID** 라는 고유한 정수의 식별자를 갖는다.

프로세스가 새로운 프로세스를 생성할 때, **fork()** 시스템콜을 이용하여 자식은 부모의 주소공간에 복사된다. 이러한 프로세스들은 후에 명령어르 실행해 나아가는데, 이 둘의 return 코드가 서로 다르다는 점이다. 부모프로세스 같은 경우 자식프로세스의 식별자가 부모로 return하는 반면에, 새로운 자식 프로세스는 0 return 한다.

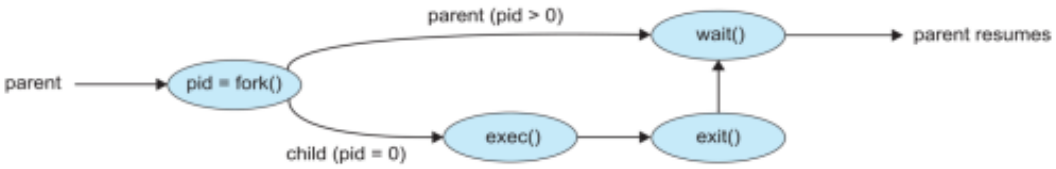


사진 1.1

자식 프로세스가 일련의 명령을 실행하는 동안 부모 프로세스는 **wait()** 이라는 시스템 콜을 호출하여 자식 프로세스가 끝나기를 기다린다. 이는 좀비 또는 고아 프로세스 방지를 위한 과정이며, 부모가 강제로 종료되거나, 자식을 기다리지 않는다면 자식 프로세스는 좀비 또는 고아 프로세스가 된다. 그와 조금은 다른 맥락으로 **background process** 의 경우, 부모 프로세스와 세션이 공유되기 때문에 부모가 받는 signal 의 영향을 받아 부모가 종료된다면 함께 종료된다. 본인은 **background process** 가 좀비 프로세스 가 되는것을 막기 위해 `waitpid(-1,NULL,WNOHANG);`함수를 사용 하였다.

과제 수행을 위해서 프로세스 간 통신에 대한 이해도 필요로 하였다. 현 과제를 수행하기 위하여 **pipe를** 이용한 통신전달자로서 방법을 이용한다. 생산자 소비자 로서의 파이프는 단방향 통신이기 때문에, 한 작업을 수행할 경우 반대 작업을 위한 파이프는 닫아 두어야 한다. 본인은 첫번째 자식의 **output** 을 두번째 자식의 **input** 으로 넣고 두번째 자식의 전체적인 명령어 **output** 을 다시 첫번째 자식의 **input** 으로 넣는 방식으로 **'|' (pipe)** 를 구현하였다. 그 과정에서 `dup2()` 라는 내장함수를 이용하였다.

3.프로젝트 진행과정 및 함수 설명

- ### 변수 선언 및 초기화

  처음 구현을 시작하며, `typedef`를 이용하여 operator 검사를 위한 `bool`타입 변수를 새로 선언 해주고. **true,false** 값을 0과1로 선언 해주었고, **pipe**를 위한 `READ_END,WRITE_END`를 선언 해주었다. 아래는 선언한 변수들의 목록이다.

```
typedef int bool;
#define MAX_LINE 80 /* the maximum length command */
#define true 1
#define false 0
#define READ_END 0
#define WRITE_END 1

bool is_right_operator; /* > */
bool is_left_operator;  /* < */
bool ampersand_operator; /* & */
bool is_pipe_operator; /* | */
```

```
/* 변수 선언 */
int fd;
int should_run = 1; /* 프로그램 실행과 종료를 위한 변수 */
char* input;

/* 자식 프로세스의 식별을 위한 pid */
pid_t pid;
pid_t pid2;

int i; /* 배열의 메모리 할당을 위해 전역변수로 초기화 */
int backgournd = 0; /* 백그라운드 프로세스 확인 변수 */

char *args[MAX_LINE/2+1] = {NULL,};  /* 사용자 명령어 저장 배열 */
char *p_args[MAX_LINE/2+1] = {NULL,};  /* 두번째 자식 명령어 저장 배열 */

/* operator 의 존재 유무를 판별하기 위해 처음에 false 로 초기화 */
is_right_operator = false;
is_left_operator = false;
ampersand_operator = false;
is_pipe_operator = false;
```

- **사용자 input 입력 받기 및 예외처리**

그다음 사용자 command 를 받아 해당 명령어를 메모리에 할당 해주는 일과 명령어의 검사였다.

```
/* input 메모리 할당 */
input = (char*) malloc(MAX_LINE*sizeof(char));
printf("osh> ");
fflush(stdout);

/* reading user input */
fgets(input,MAX_LINE,stdin);

/* input의 끝을 선언 */
```

```
    input[strlen(input)-1] = '\0';

    /* 명령어를 띄어쓰기 기준으로 torkenize */
    char* tmp = strtok(input," ");

    /* operator 작동을 위한 파일 네임 선언 */
    char* file_name = NULL;
```

이 과정을 통해 명령어를 저장할 배열을 선언하고, `malloc()` 함수로 input 을 받을 메모리를 할당 해 주었다. 그 후 `fgets()` 함수를 이용하여 사용자 input 을 받아 준다. 이과정에서 사용자가 아무런 명령을 안 치는 경우도 있으니,

`if(input[0] == "\n"){continue};` 를 조건문으로 넣어주어 `while(should_run){}` 이 동작 할 수있게 해주었다.

- **사용자 input 에 대한 operator 검사**

```
while(tmp != NULL){
       /* <,>,& ,| 검사 */
      if(strchr(tmp,'>')){
           is_right_operator = true;
           file_name = strtok(NULL," ");
      }
      else if(strchr(tmp,'<')){
           is_left_operator = true;
           file_name = strtok(NULL," ");
      }

      else if(strchr(tmp,'&')){
           ampersand_operator = true;
      }

      else if(strchr(tmp,'|')){
           i = 0;
           is_pipe_operator = true;
      }

      if(is_pipe_operator){
         if(!strchr(tmp,'|')){
    /* 파이프 오퍼레이터가 명령어 배열에 들어가는 오류를 잡기위해 */
           p_args[i++] = tmp;
         }
      }
      else if(!(is_left_operator || is_right_operator ||  ampersand_operator)){
          if(!strchr(tmp,'|')){
            args[i++] = tmp;
          }
      }
```

```
        tmp = strtok(NULL," ");
    }
```

이제 사용자의 input 검사 인데, 쉘에서 요구하는 operator 들이 있으니 각각이 짤린 문자열에 있는지 확인해준다.

여기서 `p_args[]` 의 경우, 나중에 후술 할 두번째 자식이 수행해야 할 명령어를 저장할 배열로 처음에 선언 하였으므로,

`pipe_operator`가 감지 될시에만 해당 배열에 명령어를 넣어준다. 이 과정에서 "|" 가 명령어 배열에 들어가 명령어가 정상적으로 실행되지 않는 상황을 해결해주기 위하여 "|" 을 날려준 후 문자열을 다시 확인하여 없을때만 명령어를 배열에 넣어 주었다.

나머지 >,<,& 의 경우 주어진 input 에서 해당 operator 가 존재한다면 위에서 초기화 해주었던 operator 관련 변수들의 값을 `TRUE`로 바꾸어 준다.

마지막의 `else if`조건문의 경우 operator 들이 없을때 만 명령어 배열의 명령어를 추가 해주는데,

이는 operator 기준으로 오른쪽에는 파일명 또는 두번째 자식이 수행해야 할 명령어가 오기 때문에 해당 조건을 만족 해야만

명령어 배열에 들어갈수 있도록 해주었다.

- **예외처리 및 background 변수 초기화**

```
    /* exit 명령어시 쉘 종료 */
    if(strcmp(args[0],"exit") == 0){
        break;
        should_run = 0;
    }

    /* 백그라운드 프로세스 */
    if(ampersand_operator){
        backgournd = 1;
    }
```

이후는 예외 처리 과정으로, 사용자 입력이 `exit`이거나, 백그라운드 프로세스 연산자가 들어왔을때 해당 변수를 바꿔주는 조건문이다. 백그라운드 프로세스 같은 경우 후술 할 부모 프로세스의 수행과정에서 조건문에 들어가야 하기 때문에 해당 operator 가 감지 된다면 바꾸어 준다.

- **자식 생성 및 부모 프로세스의 수행 과정 & 파이프 구현**

```
      /* 자식생성 */
    pid = fork();

    /* 포크 에러 */
    if(pid < 0){
        printf("fork Error");
        exit(1);
    }

    /* 자식일때 */
```

```
    if(pid == 0){
        /* 파일을 통한 입출력 처리 */
        if(is_right_operator){
            fd = open(file_name, O_CREAT | O_WRONLY,0755);
            dup2(fd,STDOUT_FILENO);
        }
        else if(is_left_operator){
            fd = open(file_name,O_RDONLY);
            dup2(fd,STDIN_FILENO);
        }
        if(is_pipe_operator){
            int p_fd[2];

            /* 비정상 파이프 */
            if (pipe(p_fd) == -1){
                exit(1);
            }
            /* 두번째 자식 생성 */
            pid2 = fork();

            if(pid2<0){
                printf("Fork Error");
                exit(1);
            }

            if(pid2 == 0){
                dup2(p_fd[READ_END],STDIN_FILENO);
                close(p_fd[WRITE_END]);
                execvp(p_args[0],p_args);
            }
            dup2(p_fd[WRITE_END],STDOUT_FILENO);
            close(p_fd[READ_END]);
        }
        execvp(args[0],args);
    }
```

이제 `fork()` 함수를 통한 자식프로세스를 생성하고 , 쉘의 명령을 수행하는 코드를 구현하였다. 여기서 주목 해보아야 할 것은 `pid == 0` 일 경우이다. **"|"** 파이프 연산자가 없을 경우 첫번째 자식 프로세스는 **">,<,&"** 를 확인한 후 `execvp()` 함수를 실행한다. 이 과정에서 `dup2()` 함수를 사용하여, operator 들의 기능을 작동 시킨다. (파일에 읽고 쓰는 작업을 기능을 넣어줌).

">" 의 경우 fd 라는 file discription 변수를 사용하여 file_name 에 저장된 file 을 열어주고, 해당 파일의 권한 을 선언해주면, `dup2` 함수로 fd 값에 표준 입력 값들을 적어준다.

"<"는 위의 경우와 정 반대 경우인데 , `dup2` 를 사용하여 읽기 전용 파일에서 내용을 표준 출력 으로 바꾸어 출력할 수 있게 해준다.

`execvp()` 같은 exec 계열의 함수의 경우 자식이 명령을 실행하면 파일을 덮어쓰기 때문에 이후 명령은 실행 되지 않는다.

그 다음은 파이프를 위한 `pid2 == 0` 상황인데, 첫번째 자식이 `pid2 = fork()` 하여 두번째 자식을 생성 하고, p_fd[2]를 선언 해줌 으로서 READ WRITE 작업을 가능하게 한다.이제 `pipe()` 함수를 이용하여, 파이프를 연결하여 추가한다.두번째 자식의 부모 (첫번째 자식) 는 실행값을 파이프에 쓰고, 새로운 자식 프로세스가 정상적으로 실행 됐을 시 `dup2()` 를 이용하여 두 번째 자식은 파이프 안에 저장된 값을 input 으로 읽어와 `p_args[]` 에 저장된 명령어를 실행 시킨다. 당연한 이야기 지만, 파이프는 단 방향 통신 수단이기에 안쓰는 파이프라인은 `close` 로 닫아준다.

```
    /* 부모 프로세스 진행 */
        if(pid > 0){
            if(!backgournd){
                printf("[0] %d \n",pid);
                int retval = waitpid(pid,NULL,0);

                if(retval > 0){
                  printf("waiting for child, not a back ground process\n");
                }
                else{
                    printf("Unexpected error\n");
                    exit(0);
                }
                printf("child process complete\n");
        }
        else{
              waitpid(-1,NULL,WNOHANG);
              printf("background process\n");
        }
    }
    backgournd = 0; /* 백그라운드 프로세스 변수 초기화 */
    close(fd); /* 파일 읽기 쓰기를 위해 열었던 file discription 닫기 */
    free(input); /* 지속적인 명령어 파싱을 위한 input 메모리 해체 */
```

부모 프로세스 의 경우 자식을 `fork()` 하고 나선 단지 자식이 끝날때까지 `wait()` 하면 된다. 단지 백그라운드 프로세스 의 셩우 위에서 예외처리 상황에서 초기화해준 background 변수가 영향을 미치며 해당 operator 가 조건문에 들어가지 않는다면, wait 을 하지 않기 때문에 `waitpid(-1,NULL,WNOHANG);` 을 이용하여 좀비 프로세스를 제거 하여 준다.

- **전체 `main` 코드**

```
#include <stdio.h>
 #include <stdlib.h>
 #include <string.h>
 #include <sys/wait.h>
```

```
#include <sys/fcntl.h>
#include <unistd.h>

typedef int bool;

#define MAX_LINE 80 /* the maximum length command */
#define true 1
#define false 0
#define READ_END 0
#define WRITE_END 1

bool is_right_operator;
bool is_left_operator;
bool ampersand_operator;
bool is_pipe_operator;
```

```
int main(void){

    int fd;
    int should_run = 1; /* 프로그램 실행과 종료를 위한 변수 */
    char* input;

    /* 자식 프로세스의 식별을 위한 pid */
    pid_t pid;
    pid_t pid2;

    int i; /* 배열의 메모리 할당을 위해 전역변수로 초기화 */
    int j;
    int backgournd = 0; /* 백그라운드 확인 변수 */

    while(should_run){
        is_right_operator = false;
        is_left_operator = false;
        ampersand_operator = false;
        is_pipe_operator = false;

        char *args[MAX_LINE/2+1] = {NULL,}; /* 사용자 명령어 저장 배열 */
        char *p_args[MAX_LINE/2+1] = {NULL,}; /* 두번째 자식 명령어 저장 배열 */

        input = (char*) malloc(MAX_LINE*sizeof(char));
        printf("osh> ");
        fflush(stdout);

        /* reading user input */
        fgets(input,MAX_LINE,stdin);

        /* 입력 커맨드가 아무것도 없을때 */
```

```
            if(input[0] == '\n'){
                continue;
            }

            input[strlen(input)-1] = '\0';  /* 해당 input 배열의 종료지점 지정 */

            char* file_name = NULL;
            i = 0;

            /* 명령어를 받아서 띄어쓰기 기준으로 토크나이즈  */
            char* tmp = strtok(input," ");
```

```
            /* 명령어 or 커맨드가 끝나지 않을때까지 */
            while(tmp != NULL){
                /* <,>,& ,| 검사 */
                if(strchr(tmp,'>')){
                    is_right_operator = true;
                    file_name = strtok(NULL," ");
                }
                else if(strchr(tmp,'<')){
                    is_left_operator = true;
                    file_name = strtok(NULL," ");
                }

                else if(strchr(tmp,'&')){
                    ampersand_operator = true;
                }

                else if(strchr(tmp,'|')){
                    i = 0;
                    is_pipe_operator = true;
                    //tmp = strtok(NULL," ");
                }

                if(is_pipe_operator){
                    if(!strchr(tmp,'|')){
                        p_args[i++] = tmp;
                    }
                }
                else if(!(is_left_operator || is_right_operator ||
ampersand_operator)){
                    if(!strchr(tmp,'|')){
                        args[i++] = tmp;
                    }
                }

                tmp = strtok(NULL," ");
```

```
}

/* exit 명령어시 쉘 종료 */
if(strcmp(args[0],"exit") == 0){
    break;
    should_run = 0;
}

/* 백그라운드 프로세스 */
if(ampersand_operator){
    backgournd = 1;
}
```

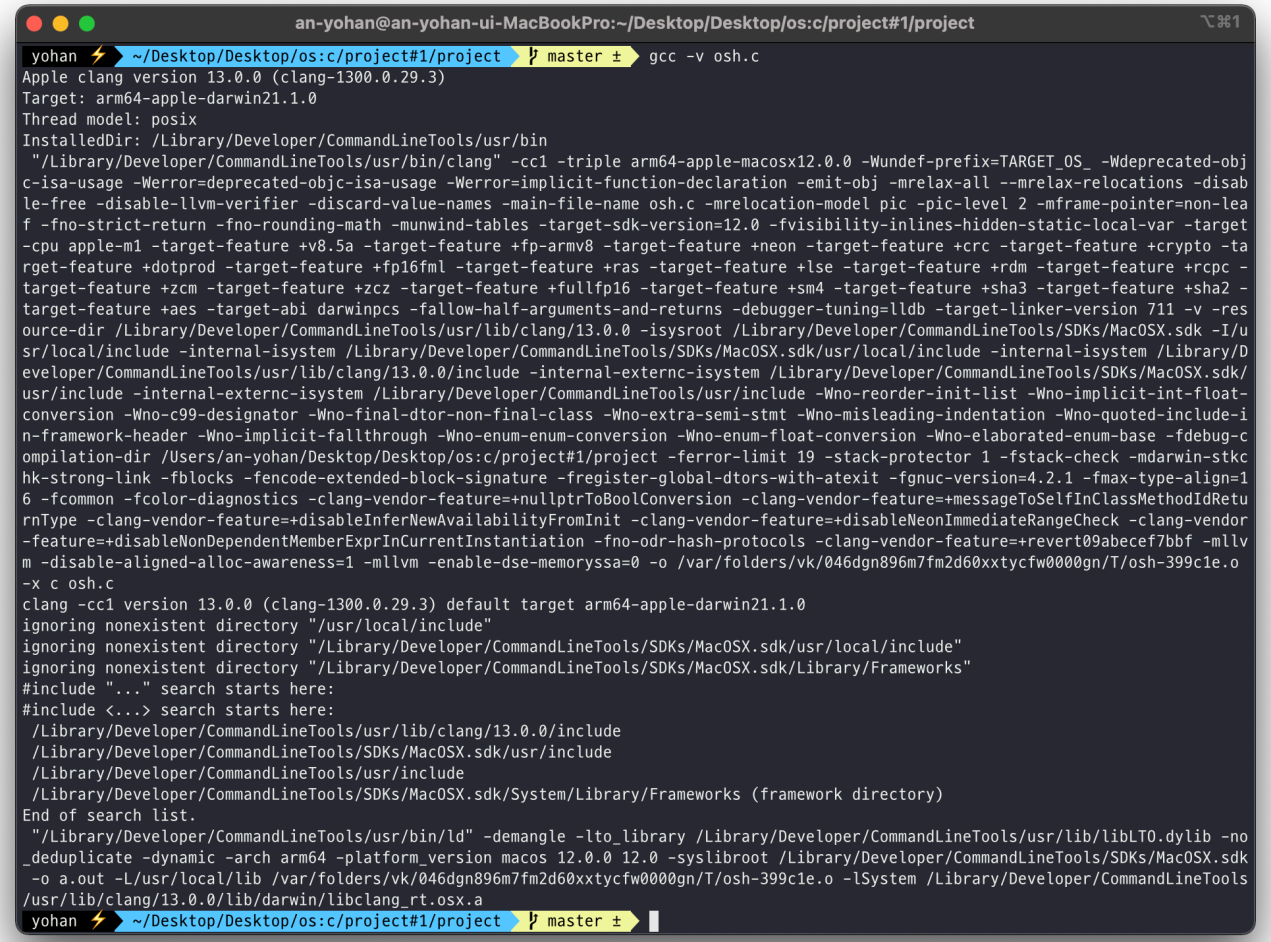
```
/* 자식생성 */
pid = fork();

/* 포크 에러 */
if(pid < 0){
    printf("fork Error");
    exit(1);
}

/* 자식일때 */
if(pid == 0){
    /* 파일을 통한 입출력 처리 */
    if(is_right_operator){
        fd = open(file_name, O_CREAT | O_WRONLY,0755);
        dup2(fd,STDOUT_FILENO);
    }
    else if(is_left_operator){
        fd = open(file_name,O_RDONLY);
        dup2(fd,STDIN_FILENO);
    }
    if(is_pipe_operator){
        int p_fd[2];

        if (pipe(p_fd) == -1){
            exit(1);
        }

        pid2 = fork();

        if(pid2<0){
            exit(1);
        }

        if(pid2 == 0){
            dup2(p_fd[READ_END],STDIN_FILENO);
            close(p_fd[WRITE_END]);
```

```
                    execvp(p_args[0],p_args);
                }
                dup2(p_fd[WRITE_END],STDOUT_FILENO);
                close(p_fd[READ_END]);
            }
            execvp(args[0],args);
```

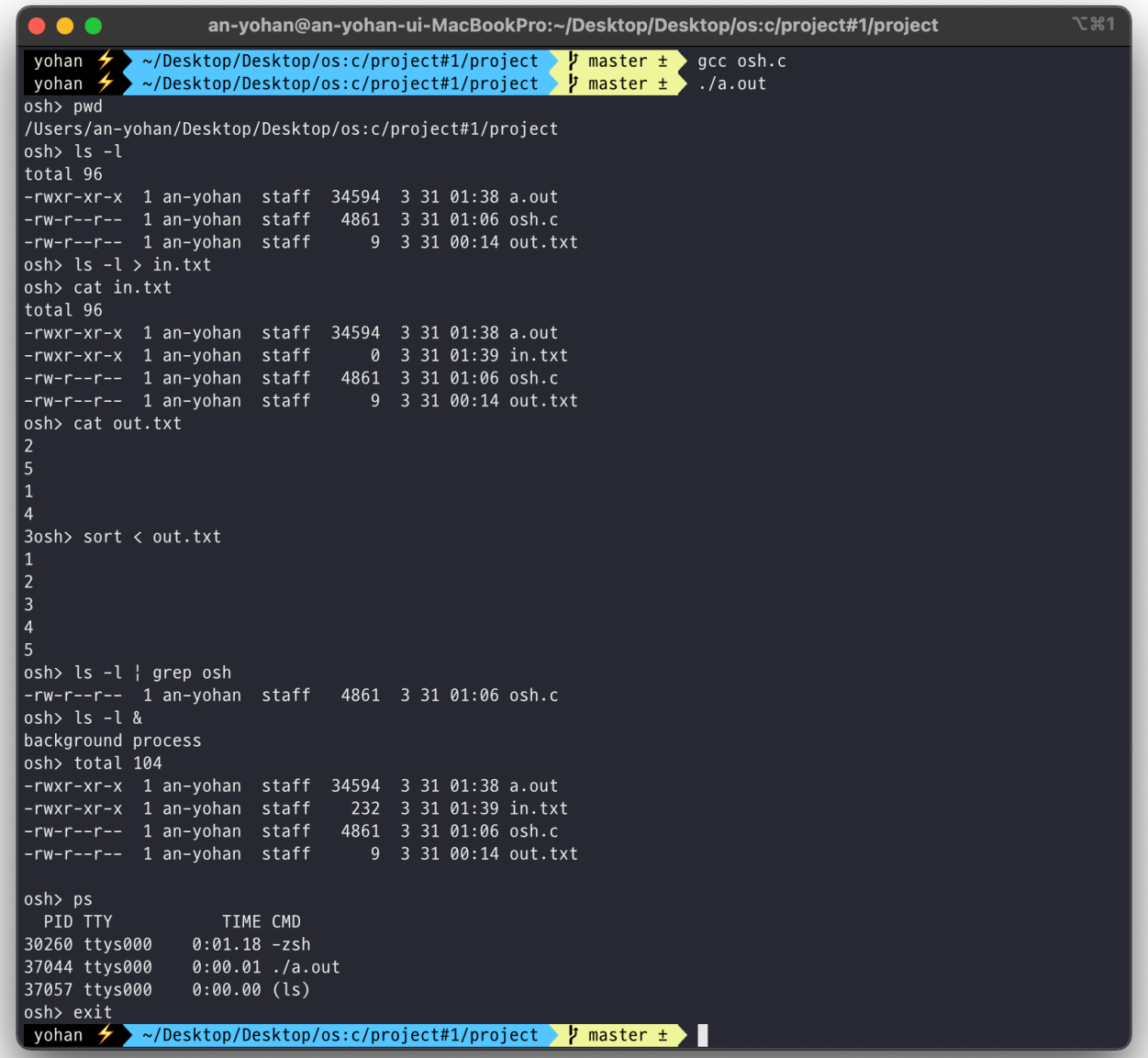
```
        }

        /* 부모 프로세스 진행 */
        if(pid > 0){
            if(!backgournd){
                //printf("[0] %d \n",pid);
                int retval = waitpid(pid,NULL,0);

                if(retval > 0){
                    //printf("waiting for child, not a back ground process\n");
                }
                else{
                    printf("Unexpected error\n");
                    exit(0);
                }
                //printf("child process complete\n");
            }
            else{
                waitpid(-1,NULL,WNOHANG);
                printf("background process\n");
            }
        }
        backgournd = 0;
        close(fd);
        free(input);
    }
    return 0;
}
```

4.컴파일 &실제 실행 모습

- 컴파일 과정


an-yohan@an-yohan-ui-MacBookPro:~/Desktop/Desktop/os:c/project#1/project
yohan ~/Desktop/Desktop/os:c/project#1/project master ± gcc -v osh.c
Apple clang version 13.0.0 (clang-1300.0.29.3)
Target: arm64-apple-darwin21.1.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
"/Library/Developer/CommandLineTools/usr/bin/clang" -cc1 -triple arm64-apple-macosx12.0.0 -Wundef-prefix=TARGET_OS_ -Wdeprecated-objc-isa-usage -Werror=deprecated-objc-isa-usage -Werror=implicit-function-declaration -emit-obj -mrelax-all --mrelax-relocations -disable-free -disable-llvm-verifier -discard-value-names -main-file-name osh.c -mrelocation-model pic -pic-level 2 -mframe-pointer=non-leaf -fno-strict-return -fno-rounding-math -munwind-tables -target-sdk-version=12.0 -fvisibility-inlines-hidden-static-local-var -target-cpu apple-m1 -target-feature +v8.5a -target-feature +fp-armv8 -target-feature +neon -target-feature +crc -target-feature +crypto -target-feature +dotprod -target-feature +fp16fml -target-feature +ras -target-feature +lse -target-feature +rdm -target-feature +rcpc -target-feature +zcm -target-feature +zcz -target-feature +fullfp16 -target-feature +sm4 -target-feature +sha3 -target-feature +sha2 -target-feature +aes -target-abi darwinpcs -fallow-half-arguments-and-returns -debugger-tuning=lldb -target-linker-version 711 -v -resource-dir /Library/Developer/CommandLineTools/usr/lib/clang/13.0.0 -isysroot /Library/Developer/CommandLineTools/SDKs/MacOSX.sdk -I/usr/local/include -internal-isystem /Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/local/include -internal-isystem /Library/Developer/CommandLineTools/usr/lib/clang/13.0.0/include -internal-externc-isystem /Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include -internal-externc-isystem /Library/Developer/CommandLineTools/usr/include -Wno-reorder-init-list -Wno-implicit-int-float-conversion -Wno-c99-designator -Wno-final-dtor-non-final-class -Wno-extra-semi-stmt -Wno-misleading-indentation -Wno-quoted-include-in-framework-header -Wno-implicit-fallthrough -Wno-enum-enum-conversion -Wno-enum-float-conversion -Wno-elaborated-enum-base -fdebug-compilation-dir /Users/an-yohan/Desktop/Desktop/os:c/project#1/project -ferror-limit 19 -stack-protector 1 -fstack-check -mdarwin-stkchk-strong-link -fblocks -fencode-extended-block-signature -fregister-global-dtors-with-atexit -fgnuc-version=4.2.1 -fmax-type-align=16 -fcommon -fcolor-diagnostics -clang-vendor-feature=+nullptrToBoolConversion -clang-vendor-feature=+messageToSelfInClassMethodIdReturnType -clang-vendor-feature=+disableInferNewAvailabilityFromInit -clang-vendor-feature=+disableNeonImmediateRangeCheck -clang-vendor-feature=+disableNonDependentMemberExprInCurrentInstantiation -fno-odr-hash-protocols -clang-vendor-feature=+revert09abecef7bbf -mllvm -disable-aligned-alloc-awareness=1 -mllvm -enable-dse-memoryssa=0 -o /var/folders/vk/046dgn896m7fm2d60xxtycfw0000gn/T/osh-399c1e.o -x c osh.c
clang -cc1 version 13.0.0 (clang-1300.0.29.3) default target arm64-apple-darwin21.1.0
ignoring nonexistent directory "/usr/local/include"
ignoring nonexistent directory "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/local/include"
ignoring nonexistent directory "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/Library/Frameworks"
#include "..." search starts here:
#include <...> search starts here:
/Library/Developer/CommandLineTools/usr/lib/clang/13.0.0/include
/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include
/Library/Developer/CommandLineTools/usr/include
/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/System/Library/Frameworks (framework directory)
End of search list.
"/Library/Developer/CommandLineTools/usr/bin/ld" -demangle -lto_library /Library/Developer/CommandLineTools/usr/lib/libLTO.dylib -no_deduplicate -dynamic -arch arm64 -platform_version macos 12.0.0 12.0 -syslibroot /Library/Developer/CommandLineTools/SDKs/MacOSX.sdk -o a.out -L/usr/local/lib /var/folders/vk/046dgn896m7fm2d60xxtycfw0000gn/T/osh-399c1e.o -lSystem /Library/Developer/CommandLineTools/usr/lib/clang/13.0.0/lib/darwin/libclang_rt.osx.a
yohan ~/Desktop/Desktop/os:c/project#1/project master ±

- 실제 실행 장면



- `pwd` 실행 프로세스의 해당 경로를 보여준다
- `ls -l` 디렉토리 안의 폴더 및 파일을 자세하게 출력한다
- `ls -l > in.txt` ls 명령어의 실행결과를 in.txt 파일을 만들어 넣는다
- `cat in.txt` 위 명령이 잘 실행 되었는지 확인.
- `cat out.txt` 미리 만들어둔 out.txt 를 확인.
- `sort < out.txt` out.txt 안의 내용을 sort 명령어로 실행
- `ls -l | grep osh` osh 를 포함하는 파일 및 폴더 읽기
- `ls -l &` 백그라운드에서 ls -l 실행
- `ps` 백그라운드에서 명령어가 잘 돌아가는지 확인
- `exit` 쉘 종료.

5.느낀점 및 어려웠던점

- **운영체제 전반적인 이론에 대한 이해**

  부모 프로세스가 자식을 `fork()`하여, 자식을 생성하고 각자의 프로그램을 실행하는 프로세스 흐름은 쉽게 이해가 되었다. 하지만 파이프 기능을 구현 할 때 골머리를 앓았던 점은 커널영역의 열린 파이프에 대한 직관적인 이해가 부족했다. 단지 파이프의 배열 인덱스에 값이 저장 되는 것 인지 알았지만, 파이프에 쓰고 읽는 작업을 위한 것 이였다는 것을 늦게 깨달았다. 또한 첫번째 자식이 파이프에 값을 쓰고 그 값을 두번째 자식이 가져와 읽는 과정이 실제로 코드로 구현하는 것이 쉽지 않았다. 오퍼레이터 구현에 있어 `dup2()`함수를 사용해 파일을 복제하고 그 파일에 값을 읽고 쓰는것은 이해가 되었지만 이것을 커널영역에 있는 파이프에 적용한다는 것이 쉽게 떠오르지 않았다.

  운영체제 이론에대한 전반적인 이해를 하였다고 생각하였지만 막상 실제로 겪어 보는 프로프래밍은 그렇지 않아서 조금 더 세심하고 싶은 이해가 필요하다고 느꼈다.

- **시스템 프로그래밍 숙련도 이슈**

  이론을 이해했다고 해서 바로 코드로 구현 할 수 있는 것이 아니었다. 처음 유저 인풋을 받아 메모리를 할당하는 것부터 다시 공부를 해야했다.또한 포인터 에대한 공부도 다시 하였다. 또한 c 자체가 다른 언어 와는 달리 시스템 자원에 직접 접근 제어를 할 수 있어서,이번 쉘을 구현하는 데 있어서도 세심한 주의를 필요로 하였다.그리고 c 언어의 내장함수의 특징들도 숙지하고 있지 않았는데 이 때문에 문자열을 나누고 오퍼레이터들을 판단할때, 파일의 골머리를 앓았다. 파이프 오퍼레이터나 >,< 등이 자꾸 엉뚱하게 명령어 배열에 들어가 조건문을 고치는 일도 허다하여 많은 시간을 쏟았다. 하나를 해결하면 하나가 안되고 그런 상황의 연속이였지만 그걸 해결 해 나아가면서 조금은 더 시스템 프로그래밍에 대한 이해도 쌓여가는 기분 이였다.

- 느낀점

  이번 프로젝트를 진행하며 느낀점은 c프로그래밍에 대한 공부가 필요하다는 점이였다. 이론적인 부분에서도 충분한 공부가 필요 하겠지만 내가 이해한것을 코드로 구현 하는것이 쉬운일이 아니였다. 그렇다고 해서 이론적인 부분이 쉬웠다는 말은 전혀 아니지만 어느 한 기능을 구현하는데 있어서 본인의 직관적인 이해만 필요한게 아니라는 것을 느꼈다.

  그리고 처음으로 프로세스라는 개념과 프로세스간의 생성과 소멸을 공부하면서 조금 더 컴퓨터에 대해 알게 되었는데,

  본인이 기존의 생각하던 쉘의 구동, 또는 컴퓨터 프로그램의 구동이 그렇게 쉽게 돌아가지만은 않는다는 것을 새삼 깨닫게 되었다. 또한 운영체제가 그러한 프로세스를 어떤식으로 통제 하는지 공부를 통해 자세하게 알수 있었고, 꽤나 어려운 과제였지만 배운 것은 정말 많은 과제였다고 생각한다. 이번 과제를 주춧돌 삼아 앞으로의 과제, 수업등 더 흥미 있게 수강 할 수 있을 것 같다.